IBM® Information Management Software

DB2 for z/OS and WebSphere Integration for Enterprise Java Applications

Understand Java drivers usage for workload balancing and failover

Tune DB2 and WebSphere on z/OS for best performance

Extend security and accounting to your clients

Paolo Bruni
Zhen Hua Dong
Josef Klitsch
Maggie Lin
Rajesh Ramachandran
Bart Steegmans
Andreas Thiele

Redbooks

ibm.com/redbooks

International Technical Support Organization

DB2 for z/OS and WebSphere Integration for Enterprise Java Applications

August 2013

SG24-8074-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xxvii.

First Edition (August 2013)

This edition applies to IBM DB2 Version 10.1 for z/OS (program number 5605-DB2) and IBM WebSphere Application Server for z/OS Version 8.5 (program number 5655-W65).

Contents

Figures

Tables

Examples

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

- AIX®
- CICS®
- DB2®
- DB2 Connect™
- developerWorks®
- Distributed Relational Database Architecture™
- DRDA®
- DS8000®
- Enterprise Storage Server®
- eServer™
- FICON®
- FlashCopy®
- GDPS®
- Geographically Dispersed Parallel Sysplex™
- HiperSockets™
- IBM®
- IMS™
- iSeries®
- Language Environment®
- MVS™
- OMEGAMON®
- Optim™
- OS/390®
- Parallel Sysplex®
- pureQuery®
- pureXML®
- RACF®
- Rational®
- Redbooks®
- Redbooks (logo) ®
- Resource Measurement Facility™
- RMF™
- System Storage®
- System z®
- System z9®
- Tivoli®
- VIA®
- VTAM®
- WebSphere®
- z/Architecture®
- z/OS®
- z/VM®
- z/VSE®
- z9®
- zEnterprise®
- zSeries®

The following terms are trademarks of other companies:

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

IBM® DB2® for z/OS® is a high-performance database management system (DBMS) with a strong reputation in traditional high-volume transaction workloads that are based on relational technology. IBM WebSphere® Application Server is web application server software that runs on most platforms with a web server and is used to deploy, integrate, execute, and manage Java Platform, Enterprise Edition applications. In this IBM Redbooks® publication, we describe the application architecture evolution focusing on the value of having DB2 for z/OS as the data server and IBM z/OS as the platform for traditional and for modern applications.

This book provides background technical information about DB2 and WebSphere features and demonstrates their applicability presenting a scenario about configuring WebSphere Version 8.5 on z/OS and type 2 and type 4 connectivity (including the XA transaction support) for accessing a DB2 for z/OS database server taking into account high-availability requirements.

We also provide considerations about developing applications, monitoring performance, and documenting issues.

DB2 database administrators, WebSphere specialists, and Java application developers will appreciate the holistic approach of this document.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Paolo Bruni is an ITSO Project Leader that is based in the Silicon Valley Lab in San Jose, CA. Since 1998, Paolo authored Redbooks publications about DB2 for z/OS, IBM IMS™, and related tools and has conducted workshops worldwide. During his many years with IBM in development and in the field, Paulo’s work has been mostly related to database systems on IBM System z®.

Zhen Hua Dong is an IBM Advisory Software Engineer for DB2 for z/OS Worldwide Level 2 support, China Development Lab. He has worked in the DB2 for z/OS team for six years. His primary focus is the RDS component in DB2 for z/OS, including SQL processing, optimizer, CCSID, and so on. He also participated in several projects for local customers, such as DB2 migration and core-banking system implementation.

Josef Klitsch is a Senior IT Specialist for z/OS Problem Determination Tools with IBM Software Group, Switzerland. After he joined IBM in 2001, he provided DB2 consultancy and technical support to Swiss DB2 for z/OS customers and worked as a DB2 subject matter expert for IBM China and as a DB2 for z/OS technical resource for IBM France in Montpellier. Before his IBM employment, Josef worked, for more than 15 years, for several European customers as an Application Developer, Database Administrator, and DB2 Systems Programmer with a focus on DB2 for z/OS and its interfaces. His preferred area of expertise in DB2 for z/OS is stored procedures programming and administration. He co-authored the IBM Redbooks publications *DB2 9 for z/OS: Deploying SOA Solutions*, SG24-7663 and *DB2 10 for z/OS Technical Overview*, SG24-7892.

Maggie Lin is an IBM Advisory Software Engineer for DB2 for z/OS Development, Silicon Valley Lab. She has worked in DB2 for z/OS team for 12 years with four years of experience in DB2 for z/OS native system team and 8 years in DB2 for z/OS distributed development. Her primary focus is distributed connectivity for DB2 for z/OS. She works closely with other IBM products such as IBM Data Server drivers, IBM DB2 Analytics Accelerator, Data Studio tools, and IBM OMEGAMON® for DB2 Performance Expert for z/OS.

Rajesh Ramachandran is an Executive IT Specialist in IBM Software Group Solution Services as an Application and Integration Middleware Software Enterprise Architect. Raj has 17 years of experience in application development on various platforms, which includes z/OS, UNIX, and Linux using COBOL, Java, IBM CICS®, and Forte.

Bart Steegmans is a Consulting DB2 Product Support Specialist from IBM Belgium, currently working remotely for the Silicon Valley Laboratory in San Jose, providing technical support for DB2 for z/OS performance problems. Bart was on assignment as a Data Management for z/OS Project Leader at the ITSO, San Jose Center, 2001 - 2004. He has over 23 years of experience in DB2. Before joining IBM in 1997, Bart worked as a DB2 system administrator at a banking and insurance group. His areas of expertise include DB2 performance, database administration, and backup and recovery.

Andreas Thiele is a freelancing Infrastructure Consultant from Hamburg, Germany. He has over 25 years of experience in various IT fields. Before being self employed, he worked with a large German bank as a Systems Programmer for 15 years. He then joined an international consulting company as Chief IT Architect where he started to work with IBM WebSphere Application Server mainly on z/OS. As an independent consultant he worked with clients in the banking and pharmaceutical industry as well as for public administration in Germany and Switzerland for the last 10 years.

Thanks to the following people for their contributions to this project:

Richard Conway
Bob Haimowitz
Linda Robinson
International Technical Support Organization

Maria Sueli Almeida
Madhavi Amirneni
Sigi Bigelis
Bill Bireley
Tom Brooks
Shu Li Kragness
Jim Pickel
Hugh Smith
Derek Tempongko
Tom Toomire
Maryela Weihrauch
IBM Silicon Valley Lab

Mark Rader
ATS Dallas, IBM US

Gareth Jones
IBM UK

David Follis
z/OS WebSphere Development, Poughkeepsie, IBM US

Don Bagwell
ATS Tucson, IBM US

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Application development with DB2 for z/OS

This chapter describes DB2 for z/OS integration with the System z platform and its capabilities as a data server for mission critical applications.

The value proposition of System z, z/OS is centered around efficient sharing of resources. Benefits can are derived by running on the platform or direct exploitation of platform qualities and attributes by the code under the specification interfaces.

This chapter covers the following topics:

- Mainframe and DB2 for z/OS
- The System z platform
- Programming languages
- Integrated application and database on z/OS
- The synergy between z/OS and DB2 for z/OS

1.1 Mainframe and DB2 for z/OS

Today, mainframe computers play a central role in the daily operations of most of the world's largest corporations, including many Fortune 1000 companies. Although other forms of computing are used extensively in business in various capacities, the mainframe occupies a coveted place in today's e-business environment. In banking, finance, healthcare, insurance, utilities, government, and a multitude of other public and private enterprises, the mainframe computer continues to form the foundation of modern business.

The long-term success of mainframe computers is without precedent in the information technology (IT) field. Today, as in every decade since the 1960s, mainframe computers and the mainframe style of computing dominate the landscape of large-scale business computing.

The mainframe owes much of its popularity and longevity to its inherent reliability and stability, a result of continuous technological advances since the introduction of the IBM System/360 in 1964. No other computer architecture in existence can claim as much continuous, evolutionary improvement, while maintaining compatibility with existing applications.

The term mainframe has gradually moved from a physical description of the IBM larger computers to the categorization of a style of computing. One defining characteristic of the mainframe has been continuing compatibility.

One key advantage of mainframe systems is their ability to process terabytes of data from high-speed storage devices and produce valuable output. For example, mainframe systems make it possible for banks and other financial institutions to perform end-of-quarter processing and produce reports that are necessary to customers (for example, quarterly stock statements or pension statements) or to the government (for example, financial results).

Mainframe workloads fall into one of two categories: Batch processing or online transaction processing, which includes web-based applications:

- With mainframe systems, retail stores can generate and consolidate nightly sales reports for review by regional sales managers. The applications that produce these statements are batch applications.
- In contrast to batch processing, transaction processing occurs interactively with the user. Typically, mainframes serve a vast number of transaction systems. These systems are often mission-critical applications that businesses depend on for their core functions. Transaction systems must be able to support an unpredictable number of concurrent users and transaction types. Most transactions are ran in short time periods (fractions of a second in some cases).

The IBM relational database management system (RDBMS) offered by System z is DB2 for z/OS. It is a member of the DB2 family of databases and uses the strengths of that family and the strength of the System z platform.

DB2 for z/OS data can be accessed in various ways, such as:

- Transactions from IMS TM or CICS
- Application servers using SQLJ or JDBC (such as WebSphere Application Server)
- IBM Distributed Relational Database Architecture™ (IBM DRDA®) protocol

1.2 The System z platform

Infrastructure simplification is key to solving many IT problems. Simplification can be achieved by resource sharing among servers. It is all about sharing data, sharing applications, and simplified operational controls. The System z platform, along with its highly advanced operating systems, provides standard format, protocols, and programming interfaces that enable resource sharing among applications that are running on the mainframe or a set of clustered mainframes.

Resource sharing is intended to help reduce redundancy that often comes from maintaining multiple copies of duplicate data on multiple servers. Sharing can also improve privacy management by enabling better control and enforcing privacy regulations for data sources. Sharing data can help simplify disaster recovery scenarios because fewer servers are being deployed; therefore, sharing data means that less data must be protected during periodic back-up operations (for example, daily or weekly maintenance) compared to having multiple copies. But most of all, infrastructure simplification helps a business assess its entire computing capabilities to determine the best directions and strategy for overall, integrated workflow, and in doing so, helps to better take advantage of existing assets and drive higher returns on IT investments.

1.2.1 Using System z technology to reduce complexity

System z servers offer capabilities that can help reduce the size and the complexity of a modern IT infrastructure. The ability to “scale up,” or add processor power for additional workloads, is a traditional mainframe strength. Today’s System z servers are available with up to 54 processors in a single footprint. Businesses can order a System z server that is configured with less than the maximum amount of processor power, and upgrade it on demand, which means using a customer-initiated procedure to add processing power when it is needed to support new applications or increased activity for existing applications, without waiting for a service representative to call.

Processing power can also be turned on (or activated) when needed and turned off when it is no longer needed. This is useful in cases of seasonal peaks or disaster recovery situations.

Adding processing power and centralizing applications represents one strategy to help control the cost and complexity of an infrastructure. This approach can also provide a highly effective way to maximize control while minimizing server sprawl, in essence, reducing the number of single-application servers that are operating in uncontrolled environments. A number of single-application servers can typically be deployed to support business processes in both production and supporting test environments. Hot stand-by failover servers, quality assurance servers, backup servers, and training, development, and test servers are some of the types of resources that are required to support a given application. A System z server can help reduce the numbers of those servers by its ability to *scale out*.

The term “scale out” describes how the virtualization technology of the System z server lets users define and provision virtual servers that have all of the characteristics of distributed servers, except they do not require dedicated hardware. They coexist, in total isolation, sharing the resources of the System z server.

Virtual servers on System z can communicate between each other, using inter-server communication that is called IBM HiperSockets™. This technology uses memory as its transport media without the need to go out of the server into a real network, simplifying the need to use cables, routers, or switches to communicate between the virtual servers.

1.2.2 Business integration and resiliency

We have seen how the need to be flexible and responsive drives businesses. If the site is not up or responsive to its clients or employees when they need it, the more likely it loses customers, or it takes the employees more time to do their jobs. A resilient infrastructure and integrated applications are also critical to the success of any business.

Availability

One of the basic requirements for today's IT infrastructure is to provide continuous business operations in the event of planned or unplanned disruptions. The availability of the installation's mission-critical applications, which are based on a highly available platform, directly correlates to successful business operations.

System z hardware, operating systems, and middleware elements have been designed to work together closely, providing an application environment with a high level of availability. The System z environment approaches application availability with an integrated and cohesive strategy that encompasses single-server, multi-server, and multi-site environments.

The System z hardware itself is a highly available server. From its inception, all of the hardware elements have always had an internal redundancy. Starting with the energy components and ending with the central processors, all of these redundant elements can be switched automatically in the event of an error. As a result of this redundancy, it is possible to make fixes or changes to any element that is down without stopping the machine from working and providing support for the customers.

The System z operating system that sits on top of the hardware has traditionally provided the best protection and recovery from failure. For example, z/OS, the flagship operating system of the System z platform, was built to mask a failure from the application. In severe cases, z/OS can recover through a graceful degradation rather than end in a complete failure. Operating system maintenance and release change can be done in most cases without stopping the environment.

Middleware running on z/OS is built to take advantage of both the hardware and operating system availability capabilities. IBM middleware such as IBM DB2 for z/OS, IBM CICS products, IBM WebSphere Application Server, and IBM IMS can provide an excellent solution for an available business application.

The IBM Parallel Sysplex® architecture on System z allows clustered System z servers to provide resource sharing, workload balancing, and data sharing capabilities for the IT, delivering ultimate flexibility when supporting different middleware applications. Although System z hardware, operating systems, and middleware have long supported multiple applications on a single server, Parallel Sysplex clustering enables multiple applications to communicate across servers, and even supports the concept of a large, single application that spans multiple servers, resulting in optimal availability characteristics for that application.

Parallel Sysplex is a cluster solution that is implemented from the IBM hardware to the middleware layer and, as a consequence, does not have to be designed and developed in the application layer.

With Parallel Sysplex and its ability to support data sharing across servers, IT architects can design and develop applications that have a single, integrated view of a shared data store. System z shared databases also provide high-quality services to protect data integrity.

This single-view database simplicity helps remove management complexity in the IT infrastructure. And simpler IT infrastructures help reduce the likelihood of errors while allowing planned outages to have a smaller impact across the overall application space.

Figure 1-1 shows the System z high availability family solution, from single system to the IBM Geographically Dispersed Parallel Sysplex™ (IBM GDPS®).

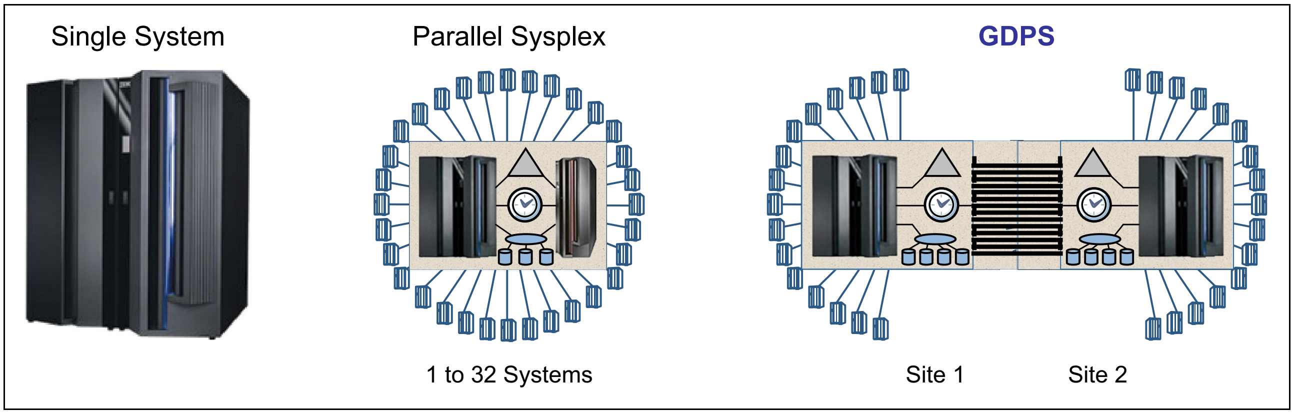


Figure 1-1 System z availability

GDPS technology provides a total business continuity solution for the z/OS environment. GDPS is a sysplex that spans multiple sites, with disaster recovery capability, which is based on advanced automation techniques. The GDPS solution allows the installation to manage remote copy configuration and storage subsystems, automate Parallel Sysplex operation tasks, and perform failure recovery from a single point of control.

GDPS extends the resource sharing, workload balancing, and continuous availability benefits of a Parallel Sysplex environment. It also significantly enhances the capability of an enterprise to recover from disasters and other failures, and to manage planned exception conditions, enabling businesses to achieve their own continuous availability and disaster recovery goals.

Hardware and software synergy

System z Operating Systems were designed to use central processors (CP). The vital connection between the hardware and the software resulted in the development of new instructions for the central processor that over time were able to respond to new application demands. The System z platform database product DB2 for z/OS also uses the specialized instructions to speed up some basic database calculations.

IBM has introduced several “specialty engines”: Processors that can help users expand the use of the mainframe for new workloads, while helping to lower cost of ownership.

- The System Assist Processor (SAP) is standard on IBM System z servers and is a dedicated I/O processor to help improve efficiencies and reduce the impact of I/O processing of every IBM System z logical partition regardless of the operating system (z/OS, IBM z/VM®, Linux, IBM z/VSE® and z/TPF).
- The IBM Integrated Facility for Linux (IFL) is another processor that enables the Linux on System z operating system to run on System z hardware.
- The IBM System Integrated Information Processor (zIIP) is designed to help improve resource optimization for running database workloads in z/OS. DB2 for z/OS can reroute queries, DRDA activity, utilities, and asynchronous I/O to the zIIP engines.
- The IBM System z Application Assist Processor (zAAP) is used by the z/OS Java virtual machine. z/OS can shift Java workloads to this new zAAP, letting the CP focus on other non-Java workloads. zAAP can also be used for XML parsing.

Processors such as zAAP and zIIP can lower the software cost of the platform, making it more cost effective.

1.2.3 Managing the System z platform to meet business goals

When new workloads are added to a System z server, they are not simply added randomly. Usually a workload is distinguished by its importance to the business. Some workloads, such as those associated with customer ordering and fulfillment, tend to have a higher degree of importance than applications used internally. Making resources available to mission-critical applications when they need them is a priority for System z hardware and software designers.

System z servers running a single z/OS image or z/OS images in Parallel Sysplex can take advantage of the Workload Manager (WLM) function. The overall mission of these advanced workload management technologies is to use established policy and business priorities to direct resources to key applications when needed. These policies are set by the user based on the needs of the individual business. These time-tested workload management features provide the System z environment with the capability to effectively operate at average usage levels exceeding 70% and sustained peak usage levels of 100% without degradation to high-priority workloads.

Figure 1-2 shows the effect of processor sharing on a System z server with multiple and different workloads running concurrently. In an environment that is not constrained for CPU, the response time for each application is not affected by the other applications running at the same time.

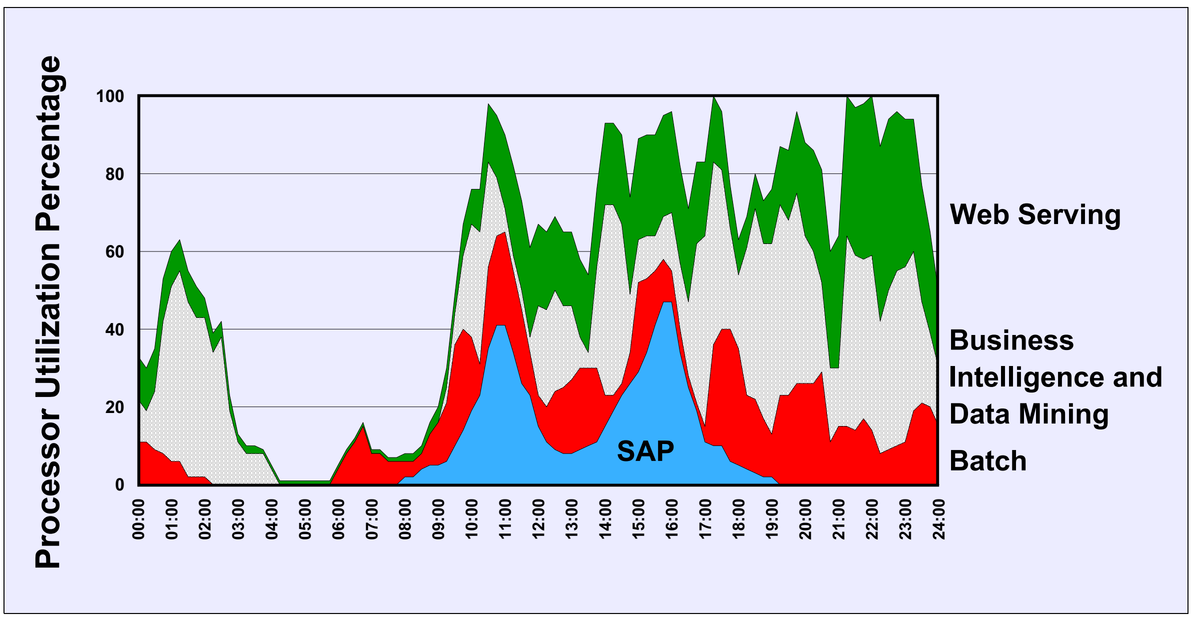


Figure 1-2 Mixed workloads on the System z platform

The higher degree of workload management represents a key System z advantage. Workload management can start at the virtual server level and drill down to the transaction level, enabling the business to decide which transaction belonging to which customer has a higher priority over others.

The Intelligent Resource Director (IRD) is a technology that extends the WLM concept to virtual servers on a System z server. IRD, a combination of System z hardware and z/OS technology that is tightly integrated with WLM, is designed to dynamically move server resources to the systems that are processing the highest priority work.

1.2.4 Security

For a business to remain flexible and responsive, it must be able to give access to its systems to existing customers and suppliers as well as to new customers, while still requiring the correct authorization to access e-commerce systems and data. The business must provide access to the data that is required for the business transaction, but also be able to secure other data from unauthorized access. The business must prevent rogue data from being replicated throughout the system and to protect the data of the trusted partners. In summary, the business must be open and secure at the same time.

The System z environment, as with its previous mainframe models, has the security concept that is deeply designed in the operating system. The ability to run multiple applications concurrently on the same server demands isolating and protecting each application environment. The system must be able to control access, allowing users to get to only the applications and data that they need, not to those that they are not authorized to use.

Hardware components, such as those for the cryptographic function that is implemented on each central processor, deliver support to the System z platform for encryption and decryption of data, and for scaling up the security throughput of the system.

In addition, other security components such as IBM RACF® (Resource Access Control Facility) provide centralized security functions such as user identification and authentication, access control to specific resources, and the auditing functions that can help provide protection and meet the business security objectives.

1.3 Programming languages

System z platform offers all tools that are needed for implementing industry-standard software engineering methodologies.

1.3.1 Language Environment

IBM Language Environment® for z/OS and z/VM (Language Environment) provides a single runtime environment for C, C++, COBOL, Fortran, PL/I, and assembler applications. See Figure 1-3. The Language Environment common library includes common services such as messages, date and time functions, math functions, application utilities, system services, and subsystem support. All of these services are available through a set of interfaces that are consistent across programming languages. You can either call these interfaces yourself or use language-specific services that call the interfaces. All of this provides consistent and predictable results for your applications, independent of the language they are written in.

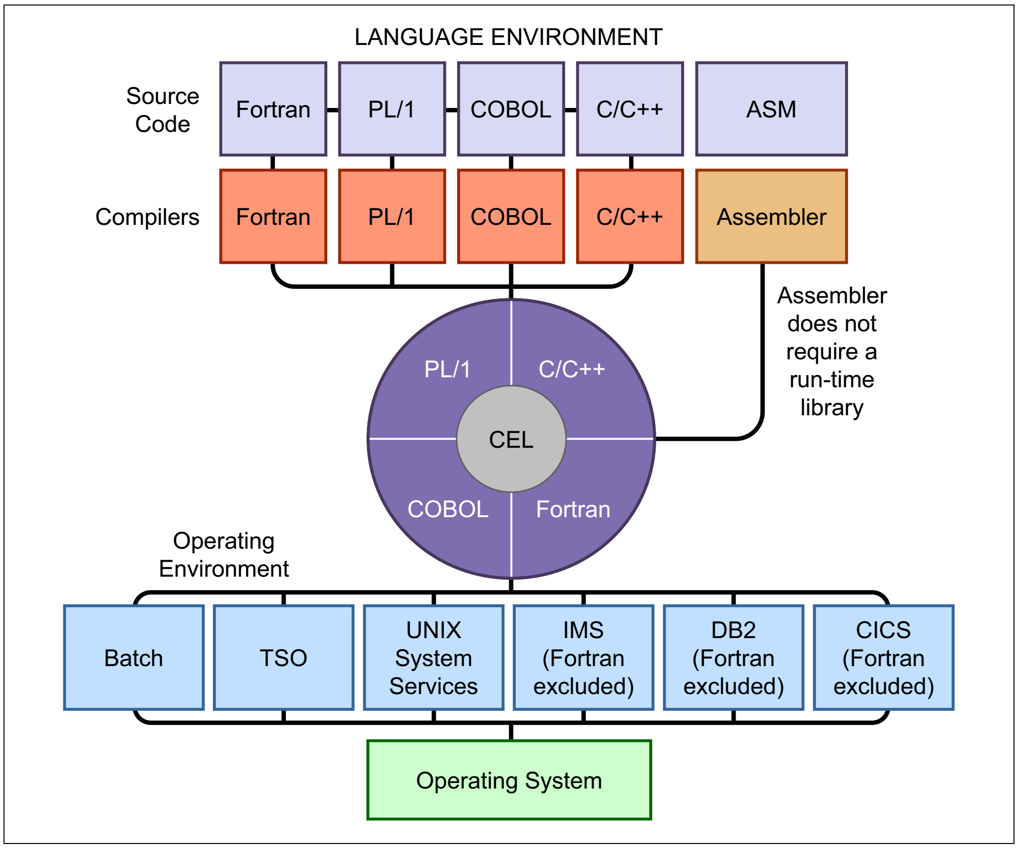


Figure 1-3 Language Environment

1.3.2 Java

Java is an object-oriented programming language that is developed by Sun Microsystems Inc. Java can be used for developing traditional mainframe commercial applications as well as Internet and intranet applications that use standard interfaces.

Java is an increasingly popular programming language that is used for many applications across multiple operating systems. IBM is a major supporter and user of Java across all of the IBM computing platforms, including z/OS. The z/OS Java products provide the same, full-function Java APIs as on all other IBM platforms. In addition, the z/OS Java licensed programs have been enhanced to allow Java access to z/OS unique file systems. Programming languages such as Enterprise COBOL and Enterprise PL/I in z/OS provide interfaces to programs written in Java.

The various Java Software Development Kit (SDK) licensed programs for z/OS help application developers use the Java APIs for z/OS, write or run applications across multiple platforms, or use Java to access data that is on the mainframe. Some of these products allow Java applications to run in only a 31-bit addressing environment. However, with 64-bit SDKs for z/OS, pure Java applications that were previously storage-constrained by 31-bit addressing can run in a 64-bit environment. System z processors support zAAP for running Java applications. Using a zAAP engine adds capacity to the platform without increasing software charges. Java programs can be run interactively through z/OS UNIX or in batch.

1.3.3 Business application languages

The Java platform offers many attractive characteristics for building modern software systems. Programmers that are already experienced with object-oriented languages typically find Java relatively easy to learn and use. But developers familiar with procedural programming, fourth-generation languages (4GLs), and other traditional development technologies often find Java complex—so much so that they resist opportunities to use it. They instead continue developing with the programming technologies (such as COBOL, PL/I, Assembler, C/C++) with which they are most comfortable.

Enterprise Generation Language (EGL) is designed to help the traditional developer take advantage of all of the benefits of Java and COBOL, yet avoid learning all of its details. EGL is a simplified high-level programming language that enables you to quickly write full-function applications that are based on Java and modern web technologies. For example, developers write their business logic in EGL source code, and from there, the EGL tools generate Java or COBOL code, along with all runtime artifacts needed to deploy the application to the wanted execution platform.

EGL hides the details of the Java and COBOL platform and associated middleware programming mechanisms. This frees developers to focus on the business problem rather than on the underlying implementation technologies. Developers who have little or no experience with Java and web technologies can use EGL to create enterprise-class applications quickly and easily.

IBM Rational® COBOL Generation Extension for System z provides the ability to continue reaping the benefits of the highly scalable, 24x7 availability of the System z platform by enabling procedural business developers to write full-function applications quickly while focusing on the business aspect and logic and not the underlying technology, infrastructure, or platform plumbing.

Built on open standards, Rational COBOL Generation for System z adds valuable enhancements to the IBM Software Development Platform so you can:

- Provide an alternative path to COBOL adoption.
- Construct first-class services for the creation and consumption of web Services for service-oriented architecture.
- Hide middleware and runtime complexities.
- Achieve the highest levels of productivity.
- Migrate from existing technologies to a modern development platform.
- Deliver applications that are based on industry standards that interoperate with existing systems.

- Easily retrain procedural business programmers to be highly productive in the Java Platform, Enterprise Edition world.
- Use visual programming techniques for web development and code automation capabilities for rapid development of application business logic.

1.4 Integrated application and database on z/OS

The applications that run directly on the System z platform under z/OS take advantage of the benefits of locating applications and data in the same technical environment. Communications between the application servers and the database manager use an efficient cross-memory mechanism.

The operations team has a single environment to manage.

The classic applications that are written in languages such as COBOL or PL/I run within a classic z/OS transaction manager such as CICS or IMS.

The new applications can also be written in Java and run within CICS or IMS as well or benefit from WebSphere Application Server for z/OS, a certified Java Platform, Enterprise Edition application server running on the System z platform.

1.4.1 Data consolidation on the System z platform

Many organizations have many applications and the data that those applications manipulate is scattered in many places. Data consolidation on the System z platform without having to change the applications can bring many benefits.

Data consolidation on the System z platform helps reduce:

- The number of data copies, and hence the risk of disparate data
- The cost and complexity of backup and recovery
- The network traffic
- The amount of storage that is needed through centralization and efficient hardware data compression
- The database administration and management tasks
- The risk that is associated with distributed privacy, security, and audit policies

With data consolidation, customers take advantage of System z technology through:

- The use of Parallel Sysplex clustering for scalability, availability, and performance
- Data-sharing capabilities that allow for them to get a single view of data
- Centralized backup, recovery, privacy, security, and audit policies

Figure 1-4 illustrates data consolidation on the System z platform.

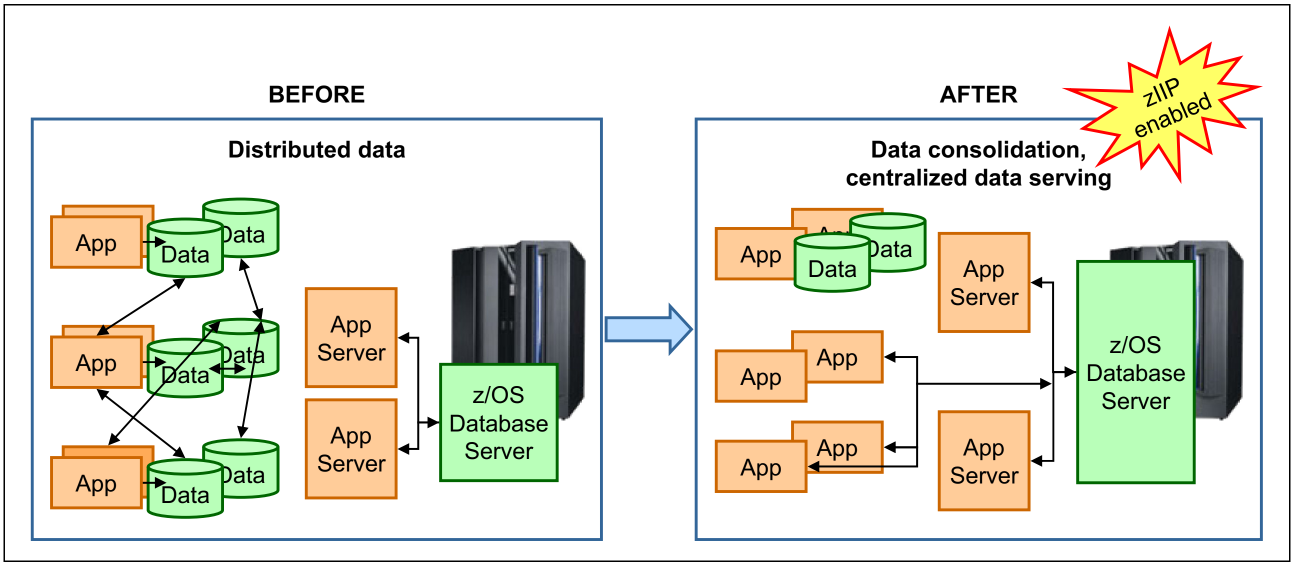


Figure 1-4   Consolidating data on z/OS

With the change in virtual storage in DB2 10, more work can run in one DB2 subsystem, allowing a consolidation of LPARs as well as DB2 members, and storage monitoring is also reduced. The net result for this virtual storage constraint relief is reduced cost, improved productivity, easier management, and the ability to scale DB2 much more easily.

DB2 10 increases the limits for CTHREAD, MAXDBAT, IDFORE, IDBACK, MAXOFILR threads. Specifically, the improvement allows a 10 times increase in the number of these threads (meaning 10 times the current supported value at your installation, not necessarily 10 times 2000). So, for example, if in your installation you can support 300-400 concurrently active threads that are based on your workload, you might now be able to support 3000-4000 concurrently active threads.

1.4.2 Data consolidation and integration of the applications on z/OS

From a technical point of view, the solution that brings the most value to the enterprise is having the data consolidated on the z/OS environment with the applications are running there as well. Figure 1-5 shows a before-and-after illustration of this data consolidation and application integration.

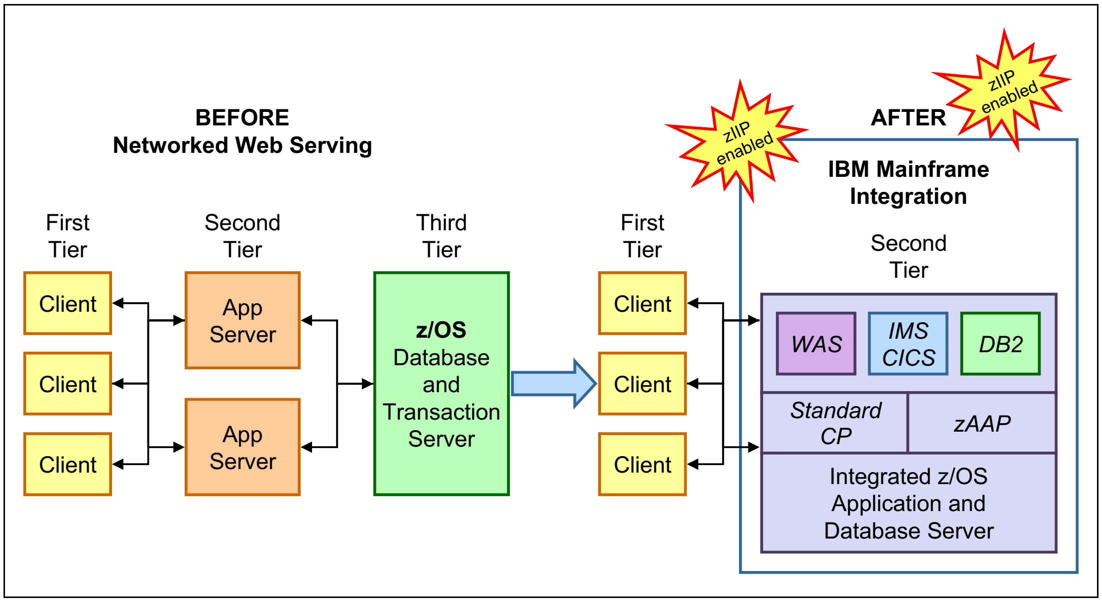


Figure 1-5 Consolidating data and integrating applications on z/OS

This situation is obvious for customers who are already running applications on the z/OS platform and must extend them. It represents a good move in other cases where enterprises can benefit from the portability of Java Platform, Enterprise Edition distributed applications to WebSphere Application Server on z/OS.

This solution increases the benefits that we already stated, and adds new ones:

- In this environment, the management of identities is more consistent, and the solution enhances auditability.
- The z/OS system is optimized for efficient use of the resources it is allowed to use.
- Transaction processing and batch work can be done at the same time on the same data: it improves availability and versatility.
- If an issue occurs, the integrated problem determination and diagnosis tools quickly help solve it.
- Automatic recovery and rollback ensure a superior level of transactional integrity.

The Java workload that are created by Java Platform, Enterprise Edition applications can benefit from the specialty processor System z Application Assist Processor (zAAP).

1.5 The synergy between z/OS and DB2 for z/OS

IBM DB2 for z/OS is the leading relational database for the System z platform.

The requirements of mission-critical environments can best be achieved through deep integration of the data server with the hardware, operating system, middleware, and tools.

As a result, DB2 for z/OS delivers important benefits that are not possible from other relational database management systems on other platforms. It is this integration that enables System z servers to provide the highest levels of availability, reliability, scalability, security, and utilization capabilities as seen by the application users. That solid foundation is critical for data servers because they are at the center of enterprise applications. Any weaknesses in the underlying infrastructure are reflected all the way through the applications to users.

1.5.1 How DB2 for z/OS uses the System z platform

DB2 for z/OS builds on the System z platform and drives some of the requirements for its evolution.

DB2 for z/OS Version 8, available since March 2004, was redesigned to take advantage of the 64-bit virtual addressing capabilities that are provided by the architecture of the System z hardware platform since 2000 and of z/OS since IBM OS/390® Version 10. It benefits from a much larger virtual storage. The internal management tasks for large databases have been modified to take advantage of this enhanced virtual storage to again improve scalability and availability.

Parallel Sysplex

The advanced clustering functions of the System z platform, the Parallel Sysplex, are based on the concept of “share everything” by opposition to other clustering environments that are based on the “share nothing” approach. In this latter approach, some processing power is tied to a fraction of the data. In Parallel Sysplex systems, all of the DB2 data included in a DB2 group can be accessible by all of the system images participating in the cluster.

This approach is backed up by efficient locking mechanisms that allow data that is accessed by several instances of an application running in different operating system images to be read or modified consistently.

DB2 data sharing support allows multiple DB2 subsystems within a sysplex to concurrently access and update shared databases. DB2 data sharing uses the coupling facility to efficiently lock data to ensure consistency, and to buffer shared data. DB2 serializes data access across the sysplex through locking. DB2 uses coupling facility cache structures to manage the consistency of the shared data. DB2 cache structures are also used to buffer shared data within a sysplex for improved sysplex efficiency.

Accessibility

- Unicode handling

  To handle the peculiarities of the different languages of the world (accented letters, special characters, and so forth) computer users use different sets of characters named code pages. It creates many difficulties to exchange data internationally.

  Unicode (`http://www.unicode.org`) is a set of standards that provides a consistent way to encode multilingual plain text.

  DB2 for z/OS understands Unicode, and users do not have to convert existing data. DB2 can integrate newer Unicode data with existing data and handle the translations. The synergy between DB2 and z/OS Unicode Conversion Services helps this process to be high performing.

  IBM z/Architecture® instructions exist that are designed just for Unicode conversions. There have been significant Unicode functional and performance enhancements in the System z platform starting with z/OS 1.4, z990, and DB2 Version 8.

- Multiple encoding schemes

  In addition to the EBCDIC support, support for ASCII tables was added in DB2 for z/OS V5, and Unicode was added in Version 7. DB2 V8 completed the integration of multiple encoding schema support by enabling SQL access to EBCDIC, ASCII, and Unicode in the same SQL statement. The majority of the DB2 catalog tables has been converted to Unicode. Key DB2 processes such as program preparation and SQL parsing are done in Unicode.

- XML support

  The IBM pureXML® feature on DB2 offers sophisticated capabilities to store, process, and manage XML data in its native hierarchical format. By integrating XML data intact into a relational database structure, users can take full advantage of DB2 relational data management features.

  DB2 pureXML starts the z/OS XML system services for XML parsing. As a result, the XML parsing request becomes 100% zIIP- or zAAP-eligible, depending on whether the parsing or schema validation request is driven by DRDA through a database access thread (DBAT) or through an allied DB2 thread.

  DB2 9 for z/OS provided expanded support of XML data type, native storage of XML documents, integration of the XPath language, and catalog extensions to support definitions of XML schemas. Utilities support creation and maintenance of XML data. DB2 10 expanded the support with XQuery, binary format for Java, and engine managed document verification.

  For details, see *Extremely pureXML in DB2 10 for z/OS*, SG24-7915.

Networking capabilities

The System z platform supports the TCP/IP V6 standard, which is the new de facto standard for interactions between nodes in a network. This capability strengthens the role of this platform as a data serving hub.

Specialty processor for data serving

The IBM System z9® Integrated Information Processor (zIIP) is designed so that a program can have all or a portion of its enclave Service Request Block (SRB)[1] dispatched work that is directed to the zIIP. z/OS, acting on the direction of the program running in SRB mode, controls the distribution of the work between the general-purpose processor (CP) and the zIIP. Using a zIIP can help free up capacity on the general-purpose processor.

DB2 for z/OS uses the zIIP processor starting from z/OS V1R6.

The following types of workloads are eligible for the zIIP processor:

- Network-connected applications

  An application (running on UNIX, Linux, Intel, Linux on System z, or z/OS) might access a DB2 for z/OS database that is hosted on a System z. Eligible work that can be directed to the zIIP is portions of those requests that are made from the application server to the host, through SQL calls through a DRDA over TCP/IP connection (like that with IBM DB2 Connect™).

  DB2 for z/OS gives z/OS the necessary information to direct portions of the eligible work to the zIIP. Examples of workloads that might be running on the server that is connected through DRDA over TCP/IP to the System z9 can include Business Intelligence, ERP, or CRM application serving.

[1] An enclave is a specific “business transaction” without address space boundaries. It is dispatchable by the operating system. It can be of system or sysplex scope.

Database workloads such as CICS, IMS, WebSphere for z/OS with local JDBC type 2 access, stored procedures, and batch have become increasingly efficient and cost effective on the mainframe and are not concerned with zIIP. One key objective with the zIIP is to help bring the costs of network access to DB2 for z/OS more closely in line with the costs of running similar workloads under CICS, IMS, or Batch on the System z platform.

Figure 1-6 illustrates the way zIIP helps reduce the workload of general processors on the System z platform for eligible workloads.

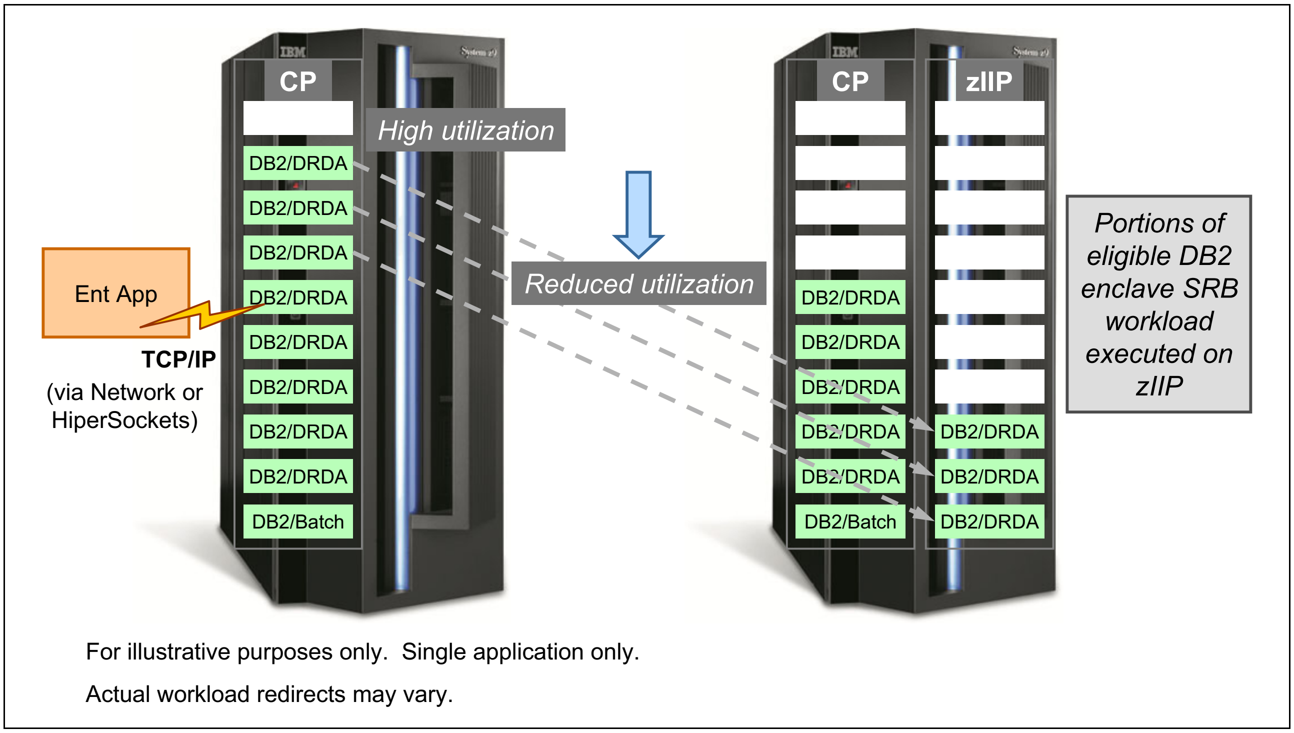


Figure 1-6 Using zIIP for enterprise applications

- Data warehousing applications

  Applications can run queries to a DB2 for z/OS database that is hosted on a System z9. Eligible work that can be directed to the zIIP is portions of requests that use complex star schema parallel queries. DB2 for z/OS gives z/OS the necessary information to direct portions of these queries to the zIIP. Examples of these applications can include Business Intelligence (BI) applications.

- Utility functions

  Some DB2 for z/OS utility functions (Load, Reorg, and Rebuild Index) are written in SRB mode. They are performing processes related to maintenance of index structures. Those portions of those utility functions that run in SRB mode are eligible as work that can be directed to the zIIP. DB2 for z/OS gives z/OS the necessary information to direct a portion of these functions to the zIIP.

- Asynchronous I/O

  Starting with DB2 10, asynchronous I/O ran by buffer pool prefetch engines and deferred write engines is 100% zIIP eligible. Buffer pool prefetch includes dynamic prefetch, list prefetch, and sequential prefetch activities. Buffer pool prefetch activities are asynchronously initiated by the database manager address space (DBM1) and are ran in a dependent enclave. Redirection to zIIP can be even more significant with index compression and insert processing with index I/O parallelism.

Workload management

z/OS includes policy-driven workload management functions that benefit all subsystems that are based on it, especially DB2. These functions grant workloads the correct priority access to key technical resources to meet business goals. Workload Manager (WLM) and Intelligent Resource Director (IRD) monitor the system to adapt to changes in both workload and configuration to meet the defined goals.

Synergy with disk hardware architecture

Disk hardware has evolved significantly since IBM introduced its first direct access storage device (DASD) back in 1956, the IBM 350. Over the years, newer disk hardware resulted in the advantages of more space per device, a smaller footprint, faster throughput of data, and improved functionality such as automatic data encryption. DB2 has made many changes to keep pace and use the disk improvements. DB2 integrates with the storage management software and continues to deliver synergy with IBM FICON® (fiber connector) channels and disk storage features.

Because I/O rates are increasing, existing applications must perform according to SLA expectations. To support existing SLA requirements in an environment of rapidly increasing data volumes and I/O rates, DB2 for z/OS uses features in the Data Facility Storage Management Subsystem (DFSMS) that help to benefit from performance improvements in DFSMS software and hardware interfaces:

- DB2 uses Parallel Access Volume and Multiple Allegiance features of the IBM TotalStorage Enterprise Storage Server® (ESS) and IBM System Storage® DS8000®.
- IBM FlashCopy® on ESS and DS8000 increases the availability of your data while running DB2 utilities.
- DB2 integrates with z/OS to deliver solutions applicable to recovery, disaster recovery, or environment cloning needs.
- Larger control interval sizes help performance with table space scans, and resolve some data integrity issues.
- The MIDAW function, improves FICON performance by reducing channel utilization and increasing throughput for parallel access streams.
- Support for solid-state drives and row level sequential detection algorithm help to reduce the need for Reorgs.
- Higher processor capacity requires greater I/O bandwidth and efficiency. High Performance FICON (zHPF) enhances the IBM z/Architecture and the FICON interface architecture to provide greater I/O efficiency. zHPF is a data transfer protocol that is optionally employed for accessing data from an IBM DS8000 storage subsystem. Both the DS8800 and the zHPF provide great improvements when used with DB2 for z/OS.
- DB2, in combination with z/OS and System z functions, can use Extended Address Volumes for all types of data sets, and by using Extended Addressability for the SMS-managed catalog, can allocate DSSIZE greater than 4 GB.

Shared memory and distributed connections

Distributed connections to DB2 for z/OS benefit from z/OS V1R7 changes. Its distributed communication processes (the distributed address space) access data directly from the database manager address space, instead of moving the data. The distributed address space also uses 64-bit addressing, as the database manager and lock manager address spaces do today with V8.

This internal change benefits new and existing workloads, where distributed communications are configured with another logical partition (LPAR) or to an application running on the System z platform.

Security synergy with Security Server for z/OS

DB2 for z/OS has strong and granular access control. It controls access to its objects by a set of privileges. Default access is none. Until access is granted, nothing can be accessed. This is called discretionary access control (DAC).

DB2 has extensive auditing features. For example, you can answer such questions as, "Who is privileged to access which objects?" and "Who has accessed the data?"

The catalog tables describe the DB2 objects, such as tables, views, table spaces, packages, and plans. Other catalog tables hold records of every granted privilege or authority. Every catalog record of a grant contains information such as name of the object, type of privilege, IDs that receive the privilege, ID that grants the privilege, and time of the grant.

The audit trace records changes in authorization IDs, changes to the structure of data, changes to values (updates, deletes, and inserts), access attempts by unauthorized IDs, results of GRANT and REVOKE statements, and other activities that are of interest to auditors.

You can use the System z platform Security Server (also know as Resource Access Control Facility (RACF)) or equivalent to:

- Control access to the DB2 environment
- Facilitate granting and revoking to groups of users
- Ease the implementation of multilevel security in DB2 (see details below)
- Fully control all access to data objects in DB2

DB2 defines sets of related privileges, called administrative authorities. You can effectively grant many privileges by granting one administrative authority.

Security-related events and auditing records from RACF and DB2 can be loaded into DB2 databases for analysis. The DB2 Instrumentation Facility Component can also provide accounting and performance-related data. This kind of data can be loaded into a standard set of DB2 tables (definitions provided). Security and auditing specialists can query this data easily to review all security events.

For regulatory compliance reasons (for example, Basel II, Sarbanes-Oxley, EU Data Protection Directive), and other reasons such as accountability, audit ability, increased privacy, and security requirements, many organizations focus on security functions when designing their IT systems. DB2 10 for z/OS provides a large set of options that improve and further secure access to data held in DB2 for z/OS to address these challenges.

- Separating the duties of database administrators from security administrators
- Protecting sensitive business data against security threats from insiders, such as database administrators, application programmers, and performance analysts
- Further protecting sensitive business data against security threats from powerful insiders such as SYSADM by using row-level and column-level access controls
- Using the RACF profiles to manage the administrative authorities

- Auditing access to business sensitive data through policy-based SQL auditing for tables without having to alter the table definition
- Auditing the efficiency of existing security policies using policy-based auditing capabilities Benefitting from security features that were introduced recently by z/OS Security Server, including support for RACF password phrases (z/OS V1R10) and z/OS identity propagation (z/OS V1R11)

For details about DB2 security functions, see *Security Functions of IBM DB2 10 for z/OS*, SG24-7959.

Data encryption

System z servers have implemented leading-edge technologies such as high-performance cryptography, large-scale digital certificate support, continued excellence in Secure Sockets Layer (SSL) performance, and advanced resource access control function.

DB2 ships a number of built-in functions that enable you to encrypt and decrypt data. IBM offers an encryption tool that is called the IBM Data Encryption for IMS and DB2 Databases, program number 5799-GWD. This section introduces both DB2 encryption and the IBM Data Encryption tool. It also describes recent hardware enhancements that improve encryption performance.

Data encryption has several challenges. These include changing your application to encrypt and decrypt the data, encryption key management, and the performance impact of encryption.

DB2 encryption is available at the column level and at the row level.

Security and networking: SSL sessions

The System z platform provides an efficient mechanism to support secure communications over the SSL protocol.

Security and external media storage encryption

Data administrators often think a lot about securing active data. Access is not granted to everyone and data can be encrypted as seen above.

However, the removable media storage, such as cartridges, that are used for back-up copies often contain enterprise data in readable format. If these media are stolen, enterprise data is at risk.

The System z platform provides efficient ways to secure external media storage based on hardware and software facilities.

Security certifications

The data-serving environment that is based on the System z platform benefits from the use of the following security certifications.

The reference information is available at:

`http://www.ibm.com/security/standards/st_evaluations.shtml`

Java applications

The Java programming language is the language of choice for portable applications that can run on multiple platforms. The System z platform has been optimized to provide an efficient Java virtual machine.

The IBM Data Server Driver for JDBC and SQLJ is a single driver that includes JDBC type 2 and JDBC type 4 behavior. When an application loads the IBM Data Server Driver for JDBC and SQLJ, a single driver instance is loaded for type 2 and type 4 implementations.

The driver has a common code base for Linux, UNIX, Windows, and z/OS. This largely improves DB2 family compatibility. For example, it enables users to develop on Linux, UNIX, and Windows, and to deploy on z/OS without having to make any change.

IBM Data Server Client Packages are available from:

http://www.ibm.com/support/docview.wss?uid=swg21385217

2

Accessing DB2 for z/OS from WebSphere applications

In this chapter, we describe the structure of enterprise Java applications accessing DB2 data through WebSphere Application Server.

This chapter covers the following topics:

- Application server infrastructure
- Core concepts of WebSphere Application Server
- Server configurations
- Clusters and high availability
- Database access from WebSphere Application Server
- WebSphere Application Server - DB2 high availability configuration options

For more information, refer to *WebSphere Application Server V8.5 Concepts, Planning, and Design Guide*, SG24-80222.

2.1 Application server infrastructure

WebSphere Application Server provides the environment to run your solutions and to integrate them with every platform and system. The core component in WebSphere Application Server is the application server runtime environment. An application server provides the infrastructure for running the applications that run your business. It insulates the infrastructure from the hardware, operating system, and network (Figure 2-1).

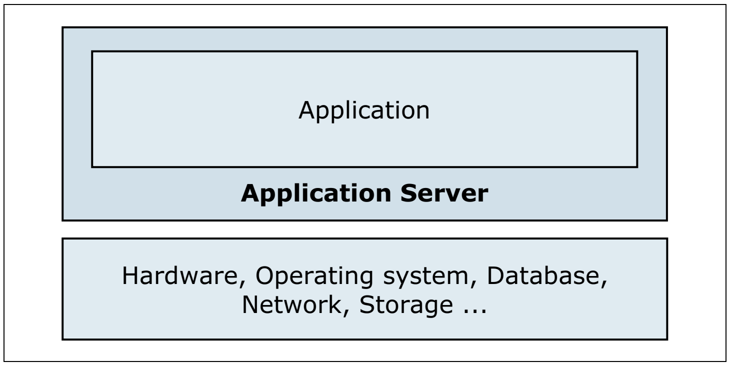


Figure 2-1 Basic presentation of an application server and its environment

An application server provides a set of services that business applications can use, and serves as a platform to develop and deploy these applications. The application server acts as middleware between back-end systems and clients. It provides a programming model, an infrastructure framework, and a set of standards for a consistent designed link between them. As business needs evolve, new technology standards become available. Since 1998, WebSphere Application Server has grown and adapted itself to new technologies and to new standards. It provides an innovative and cutting-edge environment so that you can design fully integrated solutions and run your business applications.

WebSphere Application Server is a key SOA building block, providing the role of the business application services (circled in Figure 2-2) in the SOA reference architecture.

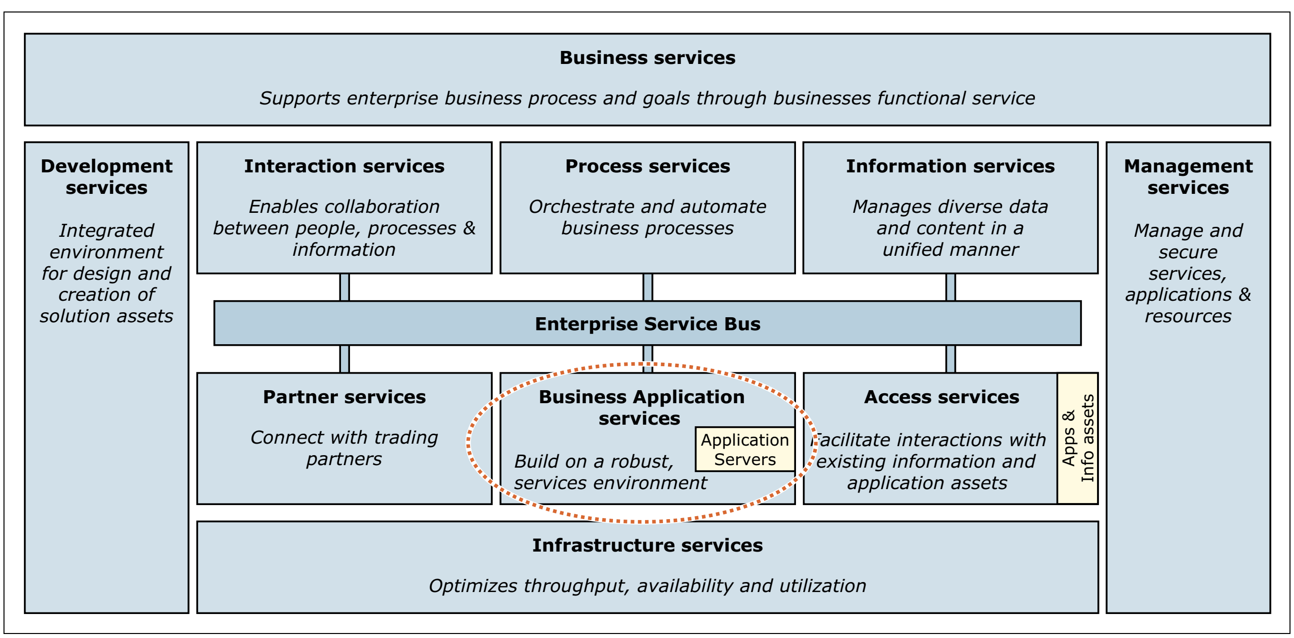


Figure 2-2 Position of business application services in an SOA reference architecture

From an SOA perspective, you can perform the following functions with WebSphere Application Server:

- Build and deploy reusable application services quickly and easily
- Run services in a secure, scalable, highly available environment
- Connect software assets and extend their reach
- Manage applications effortlessly
- Grow as your needs evolve, reusing core skills and assets

WebSphere Application Server is available on a range of platforms and in multiple packages to meet specific business needs. By providing an application server to run specific applications, it also serves as the base for other WebSphere products and many other IBM software products.

The packaging options available for WebSphere Application Server provide a level of application server capabilities to meet the requirements of various application scenarios. Although these options share a common foundation, each provides unique benefits to meet the needs of applications and the infrastructure that supports them. At least one WebSphere Application Server product fulfills the requirements of any particular project and its supporting infrastructure. As your business grows, the WebSphere Application Server family provides a migration path to more complex configurations.

The following packages are available:

- WebSphere Application Server—Express V8.5
- WebSphere Application Server—Base V8.5
- WebSphere Application Server for Developers V8.5
- WebSphere Application Server Network Deployment V8.5
- WebSphere Application Server for z/OS V8.5

Figure 2-3 summarizes various WebSphere Application Server packaging options.

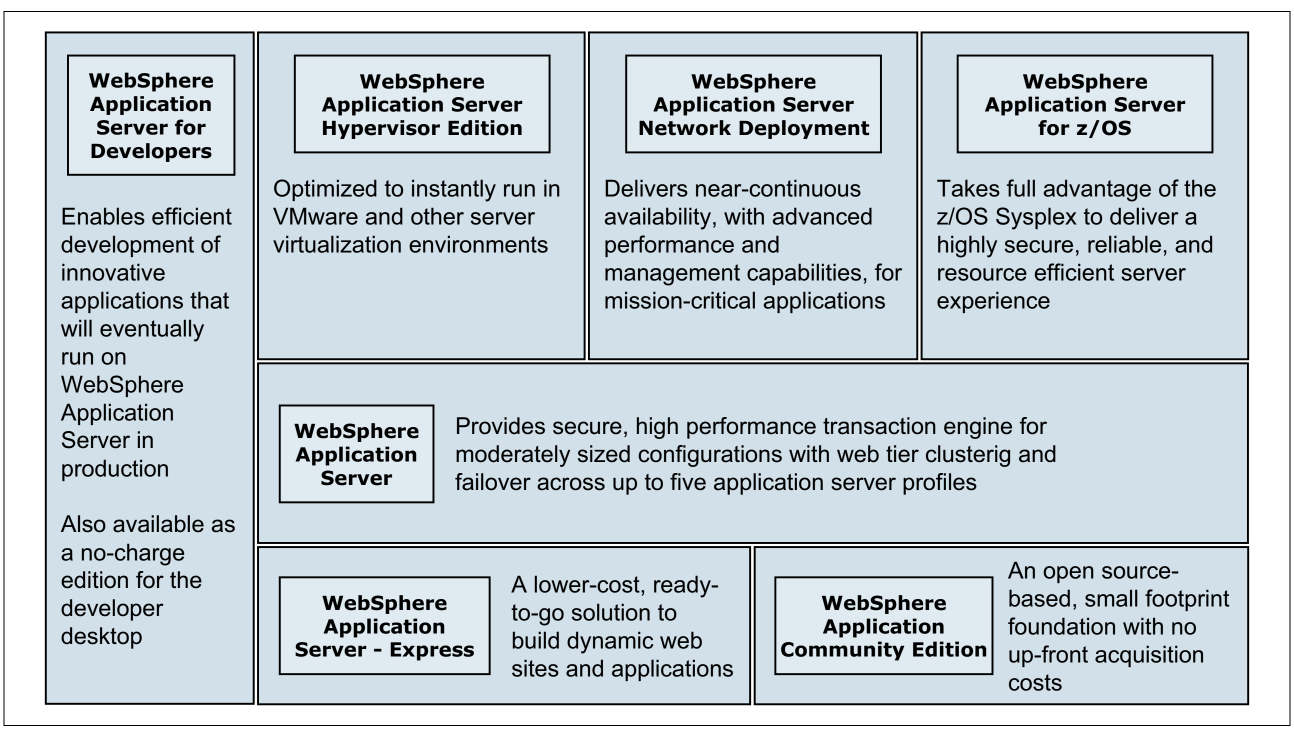


Figure 2-3 WebSphere Application Server editions

Figure 2-4 summarizes the main components that are included in each WebSphere Application Server package.

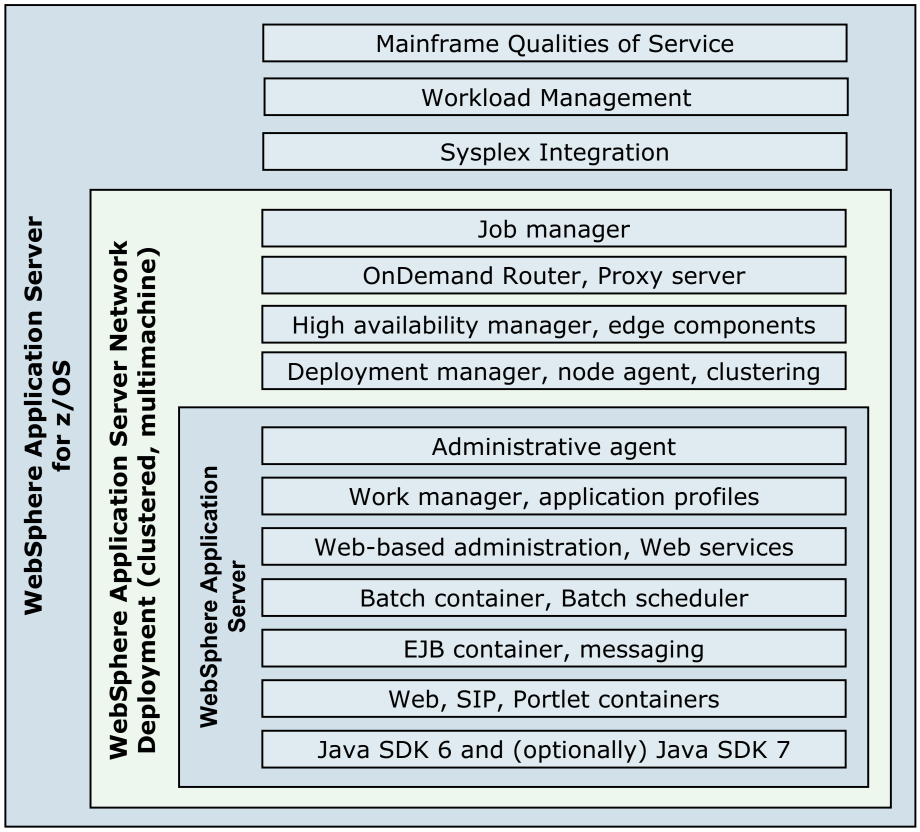


Figure 2-4 Packaging Structure WebSphere Application Server V8.5

2.2 Related products

IBM offers complementary software products for WebSphere Application Server that provide a simplified development process, enhanced management features, and a high performance runtime environment. This section provides information about the following related products:

- WebSphere Application Server Community Edition
- WebSphere eXtreme Scale
- Rational Application Developer for WebSphere Software V8.5

2.2.1 WebSphere Application Server Community Edition

WebSphere Application Server Community Edition is a lightweight single-server Java EE application server that is built on Apache Geronimo, which is the open source application server project of the Apache Software Foundation. This edition of WebSphere Application Server is based on open source code and is available for download at no charge.

> **Product information:** The code base of WebSphere Application Server Community Edition is different from the code base for WebSphere Application Server. WebSphere Application Server Community Edition is not a different packaging option for WebSphere Application Server. It is a separate product.

WebSphere Application Server Community Edition is a powerful alternative to open source application servers and has the following features:

- Brings together the best related technologies across the broader open source community to support Java EE specifications such as the following examples:
  - Apache Aries
  - Apache MyFaces
  - Apache OpenEJB
  - Apache Open JPA
  - Apache ActiveMQ
  - TranQL
- Includes support for Java EE 6 and Java SE 6
- Supports the JDK from IBM and Oracle
- Can be used as a run time for Eclipse with its plug-in
- Includes an open source Apache Derby database, which is a small-footprint database server with full transactional capability
- Contains an easy-to-use administrative console application
- Supports product binary files and source code as no-charge downloads from the IBM website
- Provides optional fee-based support for WebSphere Application Server Community Edition from IBM Technical support teams
- Can be included in advanced topologies and managed with the Intelligent Management functionality of WebSphere Application Server V8.5

For more information and the option to download WebSphere Application Server Community Edition, see:

http://www.ibm.com/software/webservers/appserv/community/

2.2.2 WebSphere eXtreme Scale

WebSphere eXtreme Scale provides the technology to enhance business by extending the data-caching concept with advanced features. With WebSphere eXtreme Scale, business applications can process large volumes of transactions with efficiency and linear scalability. WebSphere eXtreme Scale operates as an in-memory data grid that dynamically caches, partitions, replicates, and manages application data and business logic across multiple servers. It provides transactional integrity and not apparent fail over to ensure high availability, high reliability, and consistent response times.

For more information about WebSphere eXtreme Scale, see:

http://www.ibm.com/software/webservers/appserv/extremescale/

2.2.3 Rational Application Developer for WebSphere Software V8.5

Rational Application Developer for WebSphere Software is a full-featured Eclipse-based IDE that includes a comprehensive set of tools to improve developer productivity. It is the only Java IDE tool that you must design, develop, and deploy your applications for WebSphere Application Server.

Rational Application Developer for WebSphere Software adds functions to Rational Application Developer Standard Edition (Figure 2-5).

Figure 2-5 Rational development tools

Rational Application Developer for WebSphere Software includes the following functions:

- Concurrent support for Java Platform, Enterprise Edition 1.2, 1.3, 1.4, Java EE 5, and Java EE 6 specifications and support for building applications with JDK 5 and JRE 1.6
- EJB 3.1 productivity features
- Visual editors such as:
  - Domain modeling
  - UML modeling
  - Web development
- Web services and XML productivity features
- Portlet development tools
- Relational data tools
- WebSphere Application Server V6.1, V7, V8, and V8.5 test servers
- Web 2.0 development features for visual development of responsive Rich Internet Applications with Ajax and Dojo
- Integration with the Rational Unified Process and the Rational tool set, which provides the end-to-end application development lifecycle
- Application analysis tools to check code for coding practices

  Examples are provided for preferred practices and issue resolution.
- Enhanced runtime analysis tools, such as memory leak detection, thread lock detection, user-defined probes, and code coverage
- Component test automation tools to automate test creation and manage test cases
- WebSphere Adapters support, including CICS, IBM IMS, SAP, Siebel, JD Edwards, Oracle, and PeopleSoft
- Support for Linux and Microsoft Windows operating systems.

For more information about Rational Application Developer for WebSphere Software V8, see:

http://www.ibm.com/software/awdtools/developer/application/

2.3 Core concepts of WebSphere Application Server

The following concepts are central to understanding the architecture of WebSphere Application Server V8.5:

- Applications
- Containers

- Application servers
- Profiles
- Nodes, node agents, and node groups
- Cells
- Deployment manager

A person in an administrative role must understand these concepts to manage WebSphere Application Server regularly. Understanding these concepts and how they apply to your environment facilitates designing and troubleshooting.

This section provides information about these concepts. You can find additional concepts about WebSphere Application Server that build on these core concepts in 2.4, "Server configurations" on page 41.

2.3.1 Applications

At the heart of WebSphere Application Server is the ability to run *applications*, including the following types:

- Enterprise
- Business-level
- Middleware

Figure 2-6 illustrates the applications that run in the Java virtual machine (JVM) of WebSphere Application Server.

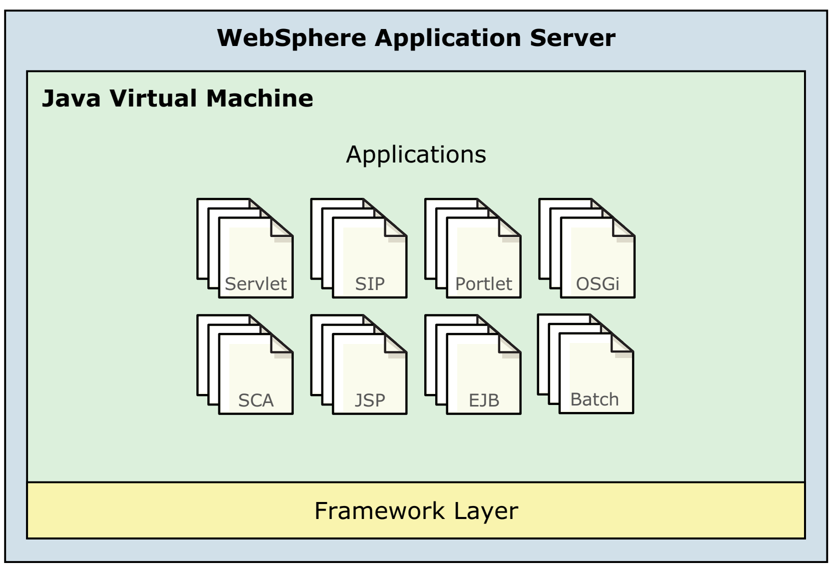


Figure 2-6 Applications running in WebSphere Application Server

Java Platform, Enterprise Edition applications

Java Platform, Enterprise Edition (Java EE) is the standard for developing, deploying, and running enterprise applications.

WebSphere Application Server V8.5 supports the Java EE 6 specification. New and existing enterprise applications can take advantage of the capabilities added by Java EE 6. If you decide not to use the Java EE 6 capabilities, portable applications continue to work with identical behavior on the current version of the platform.

Version note: IBM WebSphere SDK Java Technology Edition V6.0 is installed by default with WebSphere Application Server V8.5. Optionally, you can install IBM WebSphere SDK Java Technology Edition V7.0 in addition to the default Java version by using IBM Installation Manager. In WebSphere Application Server V8.5, you can select between Java SDK V6 and V7.

The Java EE programming model has the following types of application components:

- Enterprise JavaBeans (EJB)
- Servlets and JavaServer Pages (JSP) files
- Application clients (Java Web Start Architecture 1.4.2)

The primary development tool for WebSphere Application Server Java EE 6 applications is IBM Rational Application Developer for WebSphere V8.5. It contains tools to create, test, and deploy Java EE 6 applications. Java EE applications are packaged as enterprise archive (EAR) files.

For more information about Java EE 6 supported specifications, see the JSR page on the Java Community Process website at:

http://jcp.org/en/jsr/detail?id=316

For more information about web application specifications, see the following resources:

- JSR 154, 53 and 315 (Java Servlet 3.0 specification)

  http://jcp.org/en/jsr/detail?id=315

- JSR 252 and127 (Apache MyFaces JSF 2.0 specification)

  http://jcp.org/en/jsr/detail?id=314

- JSR 318 (EJB 3.1 specification)

  http://jcp.org/en/jsr/detail?id=318

For more information, see the following resources:

- Reference information about developing enterprise OSGi applications for WebSphere Application Server:

  http://www.ibm.com/developerworks/websphere/techjournal/1007_robinson/1007_robinson.html

- IBM Education Assistance an online presentation about developing modular and dynamic OSGi applications:

  http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/topic/com.ibm.iea.was_v8/was/8.0/ProgramingModel/WASV8_OSGi_part1/player.html

- Preferred practices for working with OSGi applications:

  http://www.ibm.com/developerworks/websphere/techjournal/1007_charters/1007_charters.html

- Supported specifications for OSGi applications:

  http://www.osgi.org/Release4/HomePage

2.3.2 Containers

Containers are specialized to run specific types of applications and can interact with other containers by sharing session management, security, and other attributes. Figure 2-7 illustrates applications that run in different containers inside the JVM. *Containers* provide runtime support for applications.

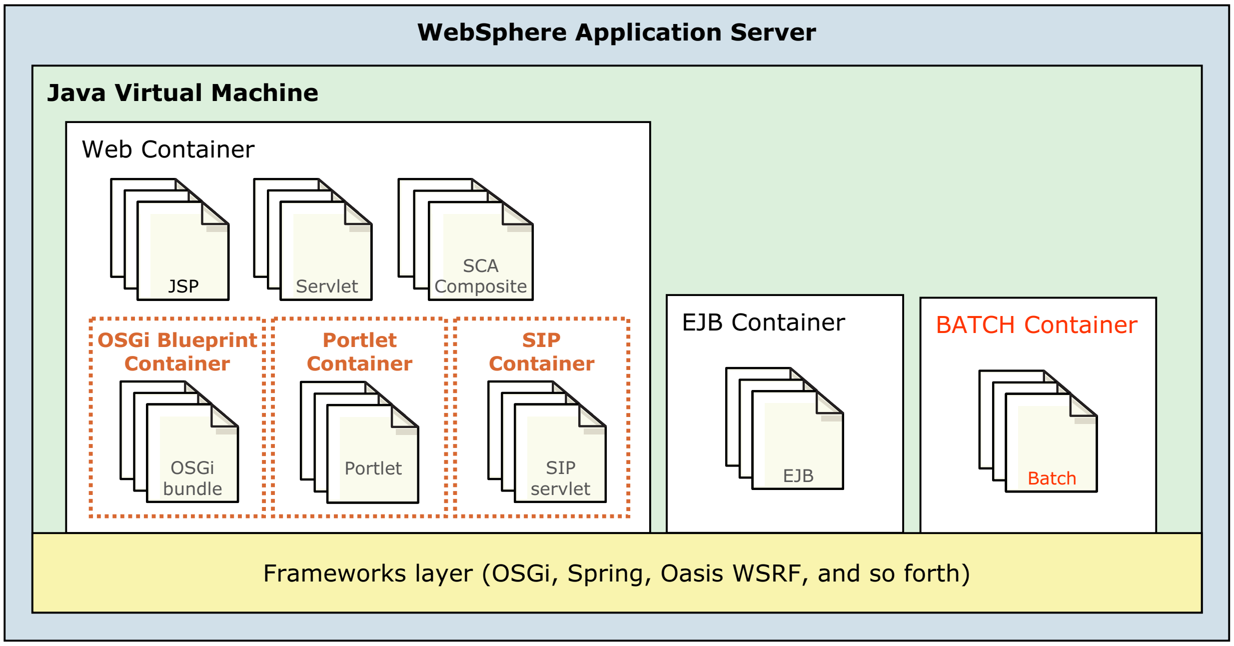


Figure 2-7 WebSphere Application Server V8.5 container services

WebSphere Application Server V8.5 includes the following logical containers:

- The *web container* processes servlets, JSPs, and other types of server-side objects.

  Each application server run time has one logical web container. Requests are received by the web container through the web container *inbound transport chain*. The chain consists of a Transmission Control Protocol (TCP) inbound channel that provides the connection to the network, an HTTP inbound channel that serves HTTP 1.0 and 1.1 requests. It also includes a web container channel over which requests for servlets and JSPs are sent to the web container for processing. Requests for HTML and other static content that are directed to the web container are served by the web container inbound chain.

- The *Enterprise JavaBeans (EJB) container* provides all of the runtime services that are needed to deploy and manage enterprise beans.

  This container is a server process that handles requests for both session and entity beans. The container provides many low-level services, including transaction support. From an administrative viewpoint, the container manages data storage and retrieval for the contained enterprise beans. A single container can host more than one JAR file.

- The *Batch container,* new in WebSphere Application Server V8.5*,* is where the job scheduler runs jobs that are written in XML job control language (xJCL).

  The batch container provides an execution environment for the execution of batch applications that are based on Java EE. Batch applications are deployed as EAR files and follow either the transactional batch or compute-intensive programming models.

The following containers are logical extensions of the web container main function:

- The *portlet container* provides the runtime environment to process JSR 286-compliant portlets. A simple portal framework is built on top of the web container to render a single portlet into a full browser page.
- The *SIP container* processes applications that use at least one SIP servlet that are written to the JSR 289 specification. It provides network services over which it receives requests and sends responses. It determines which applications to start and in what order. The container supports the UDP, TCP, and TLS/TCP protocols.
- The *OSGi Blueprint container* processes OSGi applications that are based on the OSGi framework. The OSGi Blueprint is separate from Java EE technology. However, they can be combined to deploy modular applications that use both Java EE 6/7 and OSGi R4 V4.2 technologies.

2.3.3 Application servers

At the core of each product in the WebSphere Application Server family is an *application server*. The *application server* is the platform on which Java language-based applications run (Figure 2-8). It provides services that can be used by business applications, such as database connectivity, threading, and workload management.

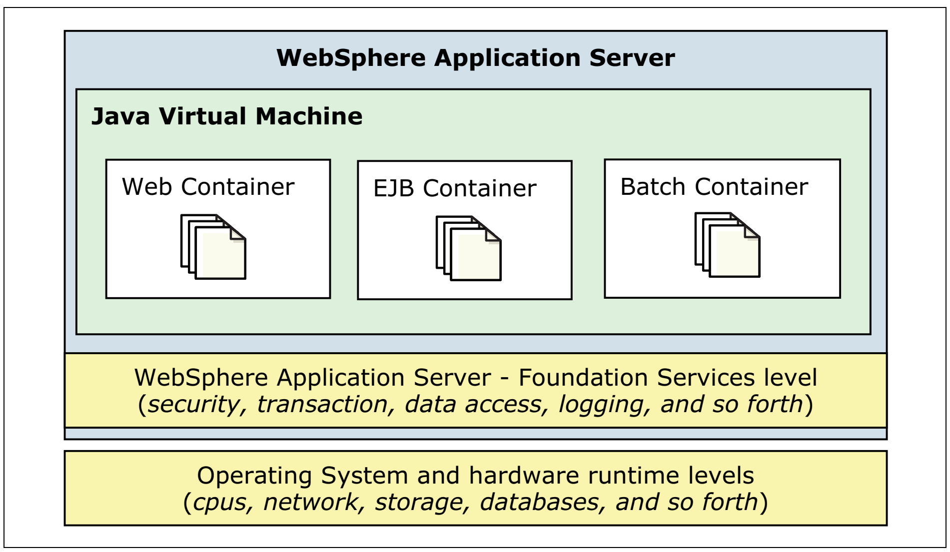


Figure 2-8 Relationship between applications and WebSphere Application Server

The following packaging options of the WebSphere Application Server family are presented in this section:

- IBM WebSphere Application Server Express V8.5, referred to as *Express*
- IBM WebSphere Application Server V8.5, referred to as *Base*
- IBM WebSphere Application Server Network Deployment V8.5, referred to as *Network Deployment* or *ND*
- IBM WebSphere Application Server Hypervisor Edition V7, referred to as *Hypervisor Edition*
- IBM WebSphere Application Server for z/OS V8.5, referred to as *WebSphere Application Server for z/OS*

Each member has essentially the same main architectural structure that is shown in Figure 2-9. They are built on a common code base. The difference between the options involves licensing terms and platform support.

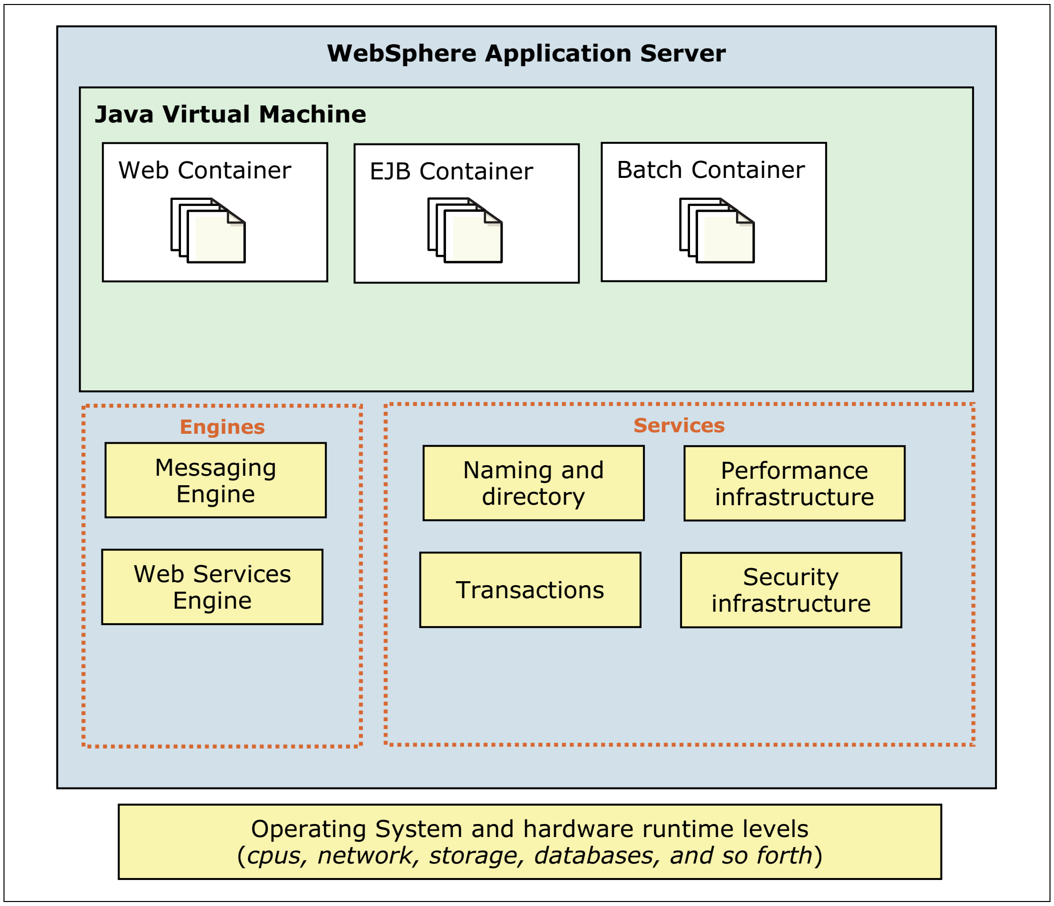


Figure 2-9 WebSphere Application Server architecture for Base and Express

The Base and Express platforms are limited to stand-alone application servers. With the Network Deployment configuration (Figure 2-10), more advanced topologies provide the following advantages:

- Workload management
- Scalability
- Near-continuous availability
- Central management of multiple application servers

These advantages are important for mission-critical applications. You can also manage multiple base profiles centrally, but you do not have workload management and the same capabilities for those base profiles.

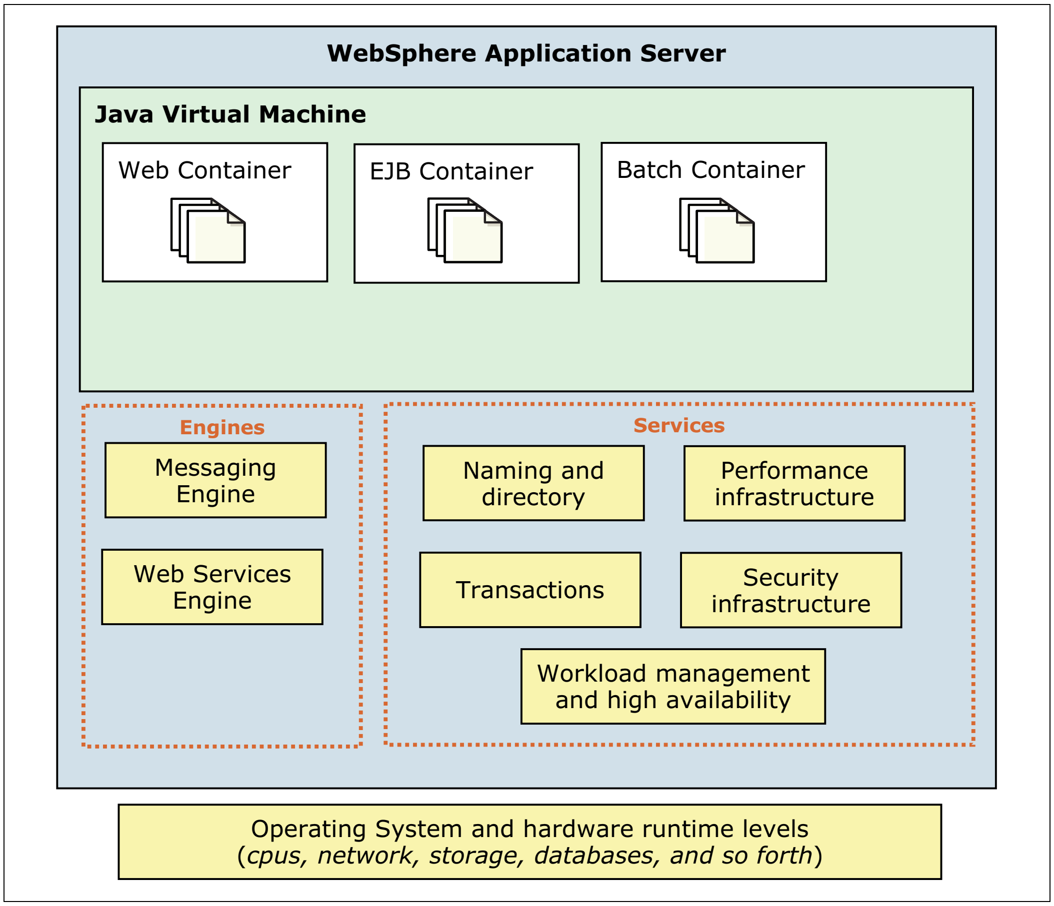


Figure 2-10 WebSphere Application Server architecture - Network Deployment configuration

Stand-alone application servers

All WebSphere Application Server packages support a single stand-alone server environment. With a stand-alone configuration, each application server acts as a unique entity, functioning independently from other application servers. An application server runs one or more applications, and provides the services that are required to run these applications. Each stand-alone server is created by defining an application server profile (Figure 2-11).

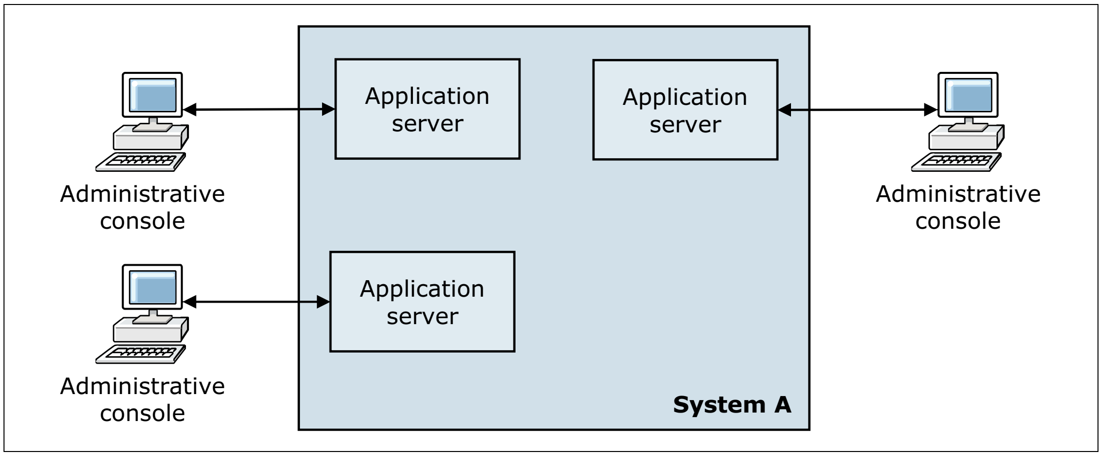


Figure 2-11 Stand-alone application server configuration

A stand-alone server can be managed from its own administrative console. You can also use the **wsadmin** scripting facility in WebSphere Application Server to perform every function that is available in the administrative console application.

Multiple stand-alone application servers can exist on a system. You can either use independent installations of the WebSphere Application Server product binary files, or create multiple application server profiles within one installation. However, stand-alone application servers do not provide workload management or fail over capabilities. They are isolated from each other.

With WebSphere Application Server for z/OS, you can use workload load balancing and response time goals on a transactional base. You can also use balancing on a special clustering mechanism, the *multiple servant regions*, with a stand-alone application server.

> **Remember:** With WebSphere Application Server V8.5, you can manage stand-alone servers from a central point by using administrative agents and a job manager.

Distributed application servers

With Network Deployment, you can build a distributed server configuration to enable central administration, workload management, and fail over. In this environment, you integrate one or more application servers into a cell that is managed by a central administration instance, a *deployment manager*. For more information, see 2.3.7, “Deployment manager” on page 41. The application servers can be on the same system as the deployment manager or on multiple separate systems. Administration and management are handled centrally from the administration interfaces of the deployment manager (GUI or scripting) as illustrated in Figure 2-12.

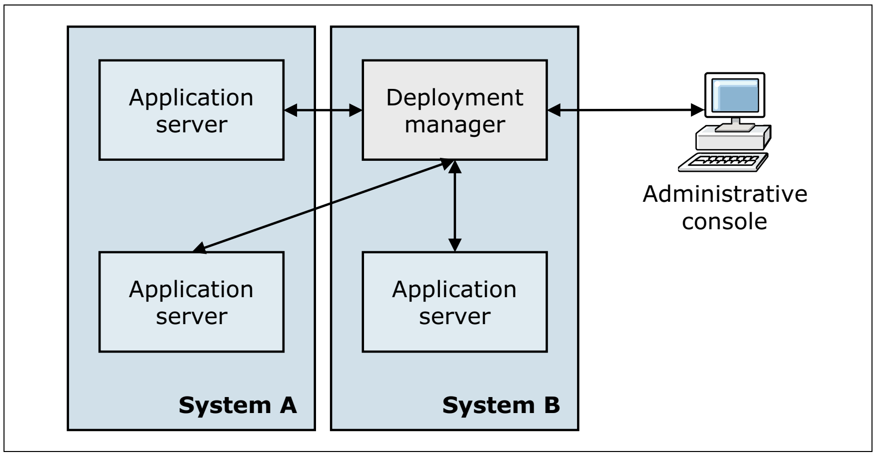


Figure 2-12 Distributed application servers with WebSphere Application Server V8.5

With a distributed server configuration, you can create multiple application servers to run unique sets of applications, and manage those applications from a central location. More importantly, you can cluster application servers to allow for workload management and fail over capabilities. Applications that are installed in the cluster are replicated across the application servers. The cluster can be configured so when one server fails, another server in the cluster continues processing. Workload is distributed among containers in a cluster by using a weighted round-robin scheme.

Tip for z/OS: The weighted round-robin mechanism is replaced by the integration of WebSphere Application Server for z/OS in the Workload Manager (WLM). The WLM is a part of the operating system. Requests can be dispatched by using this configuration to a cluster member according to real-time load and regardless of whether the member reaches its defined response time goals.

Application servers types

WebSphere Application Server V8.5 provides the following server types, which can be defined and configured by using the administrative console:

- WebSphere Application Server
- Generic server
- On-demand router
- PHP server
- WebSphere proxy server
- WebSphere MQ server
- Community Edition server
- Web server

With the mixed server environment and mixed node definitions, other existing server types can be added and administered. These types include external WebSphere application servers, Apache Server, and Custom HTTP Server.

2.3.4 Profiles

WebSphere Application Server runtime environments are built by creating set of configuration files, named *profiles,* that represent a WebSphere Application Server configuration. The following categories of WebSphere Application Server files are available, as illustrated in Figure 2-13:

- *Product files* are a set of read-only static files or product binary files that are shared by any instances of WebSphere Application Server.
- *Configuration files (profiles)* are a set of user-customizable data files. This file set includes WebSphere configuration, installed applications, resource adapters, properties, and log files.

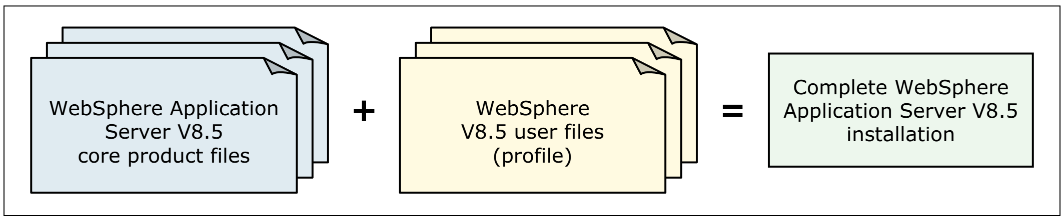


Figure 2-13 Anatomy of a profile

The Customization Toolbox allows you to create separate environments, such as for development or testing, without a separate product installation for each environment. Different profile templates are available in WebSphere Application Server V8.5 through the Customization Toolbox Profile Management Tool (PMT):

- Cell

  A cell template contains a federated application server node and a deployment manager.
- Deployment manager

  The Network Deployment profile provides the necessary configuration for starting and managing the deployment manager server.
- Default profile (for stand-alone servers)

  This server default profile provides the necessary configuration file for starting and managing an application server, and all the resources that are needed to run enterprise applications.
- Administrative agent

  This profile is used to create the administrative agent to administer multiple stand-alone application servers.
- Default secure proxy

  This profile is available when you install the DMZ secure proxy server feature.

- Job manager

  This profile coordinates administrative actions among multiple deployment managers, and administers multiple stand-alone application servers. It also asynchronously submits jobs to start servers, and completes various other tasks.

- Custom

  This profile, also known as *Empty Node* because it has no application server inside, can be federated to a deployment manager cell later. It is used to host application servers, clusters, an on-demand router, and other Java processes.

> **The Liberty profile:** Do not confuse the Liberty profile with the concept of a profile that is created by the PMT in previous versions of WebSphere Application Server. The Liberty profile provides a composable and dynamic application server runtime environment on WebSphere Application Server V8.5. The Liberty profile is a subset of base functions of the WebSphere Application Server, which is installed separately.
>
> You can create compressed files that contain all or subsets of the Liberty profile server installation. You can then extract these files on other target hosts as a substitute for the product installation.
>
> With a simpler configuration model based on XML, you do not need to create a profile by using the PMT to create Liberty profile application servers.

Each profile contains files that are specific to that run time (such as logs and configuration files). You can create profiles during and after installation. After you create the profiles, you can perform further configuration and administration by using WebSphere administrative tools.

Each profile is stored in a unique directory path (Figure 2-14), which is selected by the user when the profile is created. Profiles are stored in a subdirectory of the installation directory by default, but can be located anywhere.

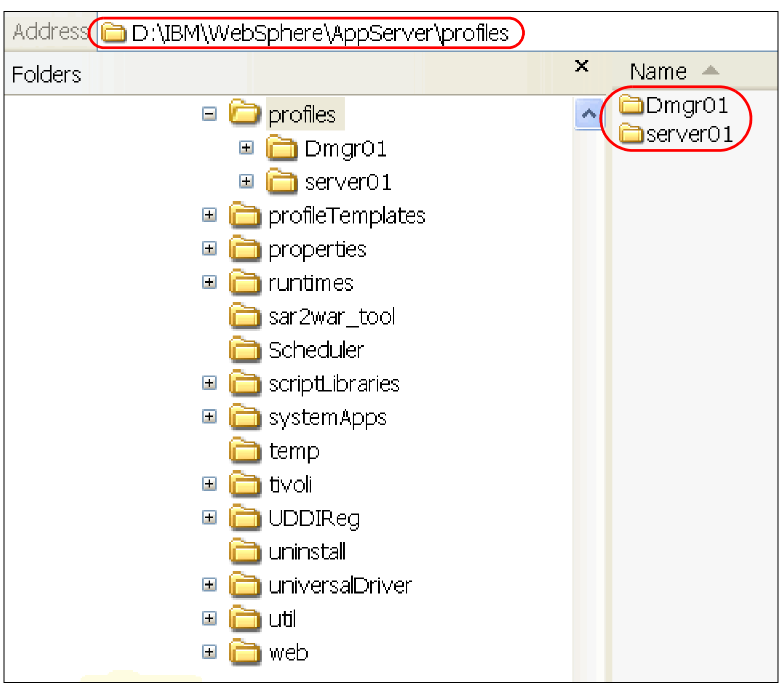


Figure 2-14 Profiles directory structure of WebSphere Application Server V8.5 on a Windows system

By creating various profiles, you can create a distributed server configuration by using one of the following methods:

- Create a deployment manager profile to define the deployment manager, and then create one or more custom node profiles. The nodes that are defined by each custom profile can be federated into the cell that is managed by the deployment manager. You can federate these nodes during profile creation, or manually later. The custom nodes can exist inside the same operating system image as the deployment manager or in another operating system instance. You can then create application servers by using the administrative console or **wsadmin** scripts.

  This method is useful when you want to create multiple nodes, multiple application servers on a node, or clusters.

- Create a deployment manager profile to define the deployment manager. Then, create one or more application server profiles, and federate these profiles into the cell that is managed by the deployment manager. This process adds both nodes and application servers into the cell. The application server profiles can exist on the deployment manager system or on multiple separate systems or z/OS images.

  This method is useful in development or small configurations. Creating an application server profile gives you the option of having the sample applications installed on the server. When you federate the server and node to the cell, any installed applications can be carried into the cell with the server.

- Create a cell profile. This method creates both a deployment manager profile and an application server profile. The application server node is federated to the cell. Both profiles are on the same system.

  This method is useful in a development or test environment. Creating a single profile provides a simple distributed system on a single server or z/OS image.

2.3.5 Nodes, node agents, and node groups

This section provides details about the concepts of nodes, node agents, and node groups.

Nodes

A *node* is an administrative grouping of application servers for configuration and operational management within one operating system instance. You can create multiple nodes inside one operating system instance, but a node cannot leave the operating system boundaries. A stand-alone application server configuration has only one node. With Network Deployment, you can configure a distributed server environment that consists of multiple nodes that are managed from one central administration server.

From the administrative console, you can also configure middleware nodes (defined into a generic server cluster) to manage middleware servers by using a remote agent.

Figure 2-15 illustrates nodes that are managed from a single deployment manager.

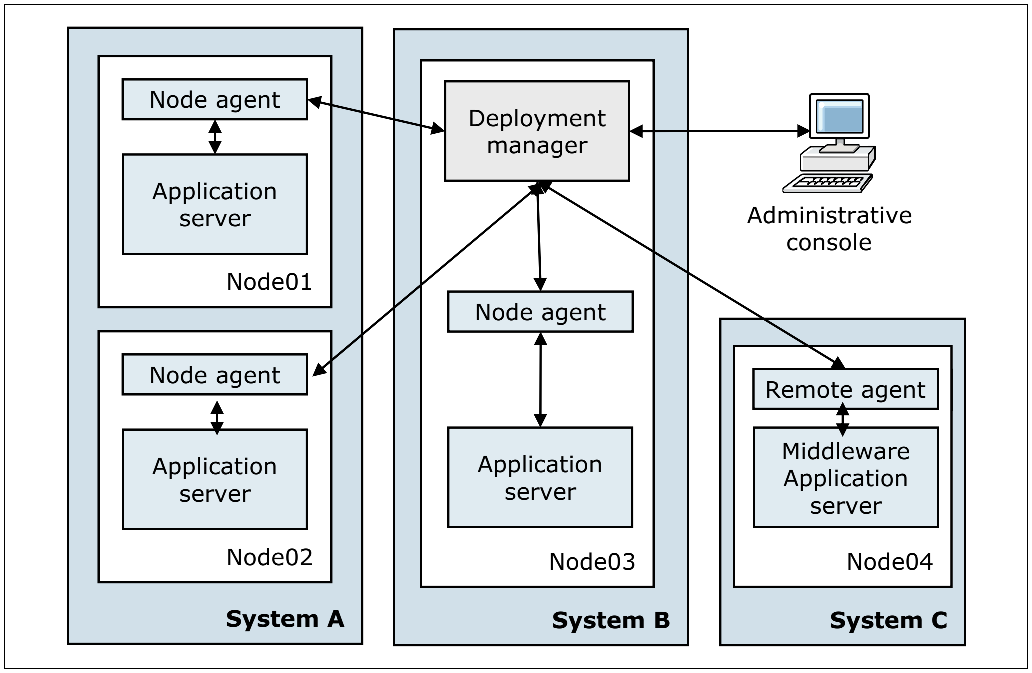


Figure 2-15 Node concept - WebSphere Application Server Network Deployment configuration

Node agents

In distributed server configurations, each node has a *node agent* that works with the deployment manager to manage administration processes. A node agent is created automatically when you add (federate) a stand-alone application server node to a cell. Node agents are not included in the Base and Express configurations because a deployment manager is not needed in these architectures. In Figure 2-15, each node has its own node agent that communicates directly or remotely with the deployment manager. The node agent is an administrative server that runs on the same system as the node. It monitors the application servers on that node, routing administrative requests from the deployment manager to those application servers.

Node groups

A *node group* is a collection of nodes within a cell that have similar capabilities in terms of installed software, available resources, and configuration. A node group is used to define a boundary for server cluster formation so that the servers on the same node group host the same applications.

A node group validates that the node can run certain functions before allowing them. For example, a cluster cannot contain both z/OS nodes and non-z/OS nodes. In this case, you can define multiple node groups, one for the z/OS nodes and one for non-z/OS nodes. A DefaultNodeGroup is created automatically. The DefaultNodeGroup contains the deployment manager and any new nodes with the same platform type. A node can be a member of more than one node group.

Sysplex on z/OS: On the z/OS platform, a node must be a member of a system complex (*sysplex*) node group. Nodes in the same sysplex must be in the same sysplex node group. A node can be in one sysplex node group only. A sysplex is the z/OS implementation of a cluster. This technique uses distributed members and a central point in the cluster. It uses a *coupling facility* for caching, locking, and listing. The coupling facility runs special firmware, the Coupling Facility Control Code (CFCC). The members and the coupling facility communicate with each other by using a high-speed InfiniBand memory-to-memory connection of up to 120 Gbps.

Figure 2-16 shows a single cell that contains multiple nodes and node groups.

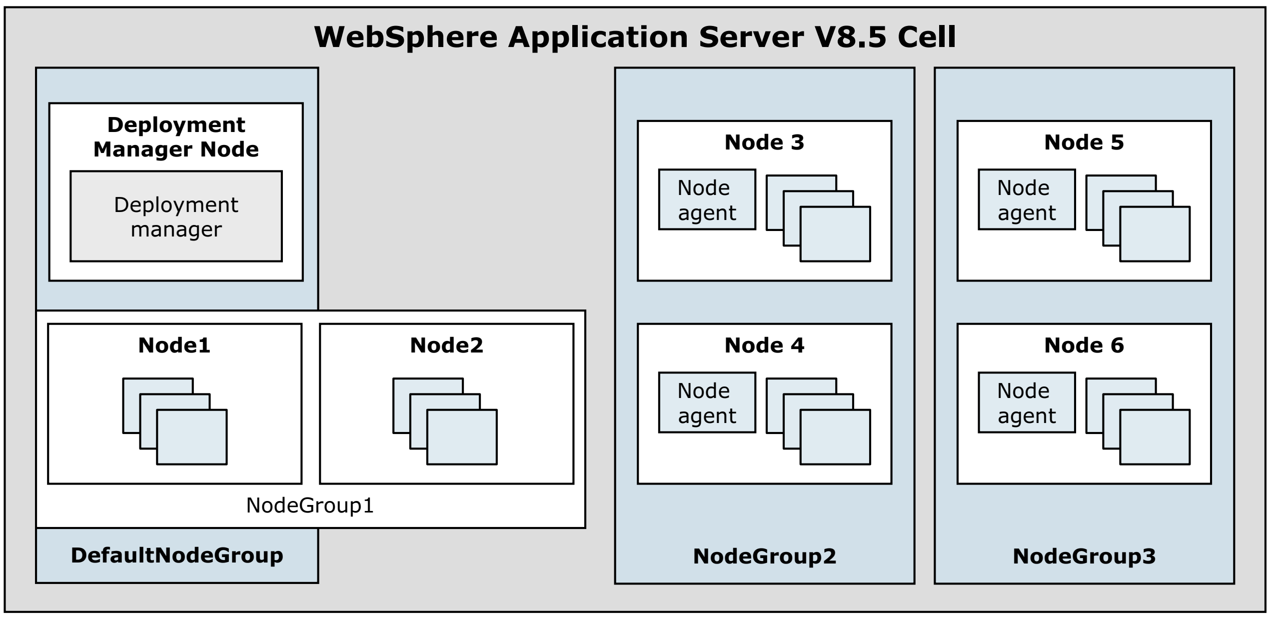


Figure 2-16 Examples of a node and node group

2.3.6 Cells

A *cell* is a grouping of nodes into a single administrative domain. A cell encompasses the entire management domain. In the Base and Express configurations, a cell contains one node, and that node contains one server. The left side of Figure 2-17 on page 40 illustrates a system with two cells that are each accessed by their own administrative console. Each cell has a node and a stand-alone application server.

In a Network Deployment environment (the right side of Figure 2-17), a cell can consist of multiple nodes and node groups. These nodes and groups are all administered from a single point, the deployment manager. Figure 2-17 shows a single cell that spans two systems that are accessed by a single administrative console. The deployment manager is administering the nodes.

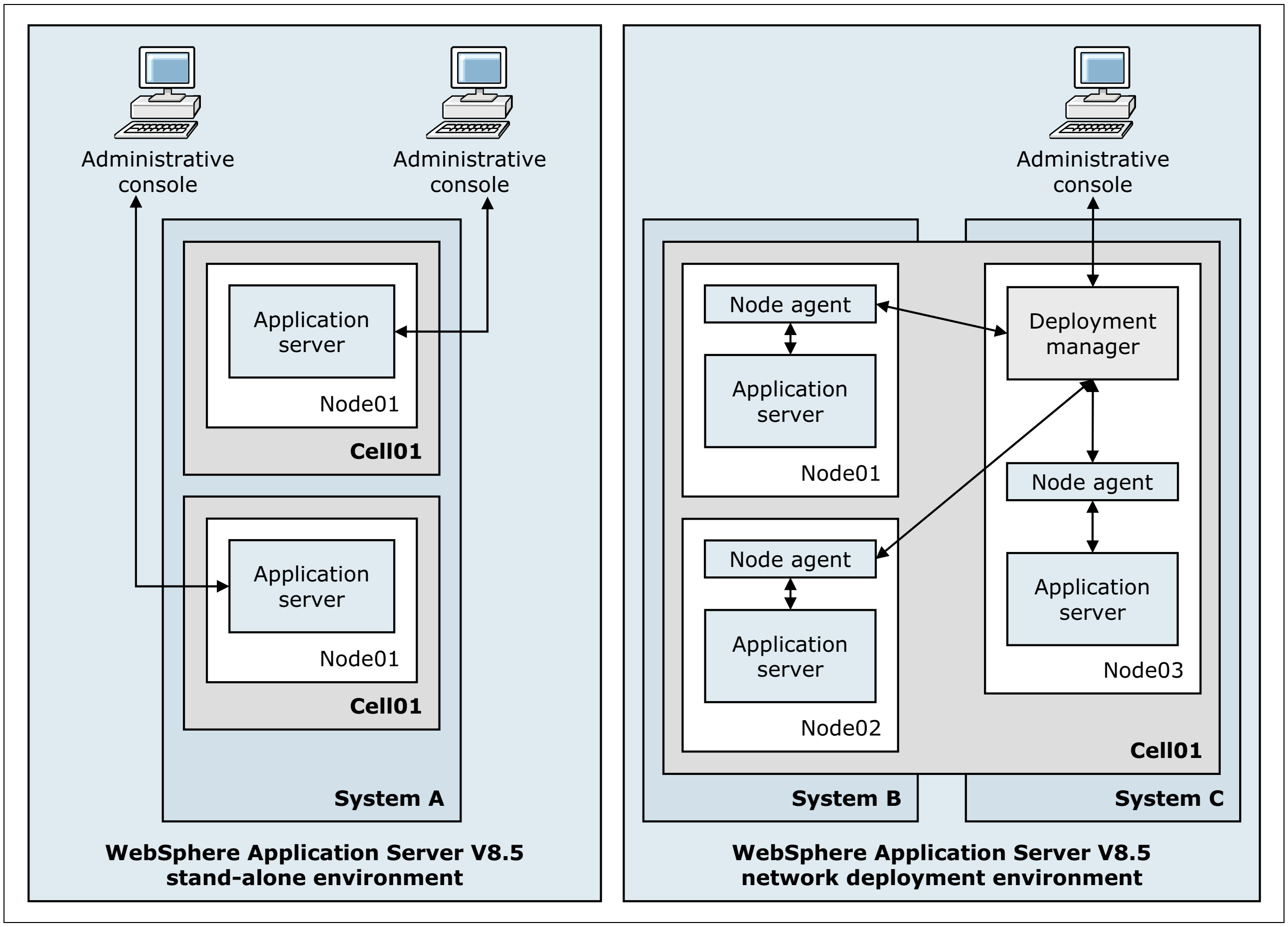


Figure 2-17 Cells representation in stand-alone and network deployment environments

A cell configuration that contains nodes that are running on the same platform is called a *homogeneous cell*.

It is also possible to configure a cell that consists of nodes on mixed platforms. With this configuration, other operating systems can exist in the same WebSphere Application Server cell. Cells can span z/OS sysplex environments and other operating systems. For example, z/OS nodes, Linux nodes, UNIX nodes, and Windows system nodes can exist in the same WebSphere Application Server cell. This configuration is called a *heterogeneous* cell. A heterogeneous cell requires significant planning.

Figure 2-18 shows a heterogeneous cell, where node groups are defined for different operating systems.

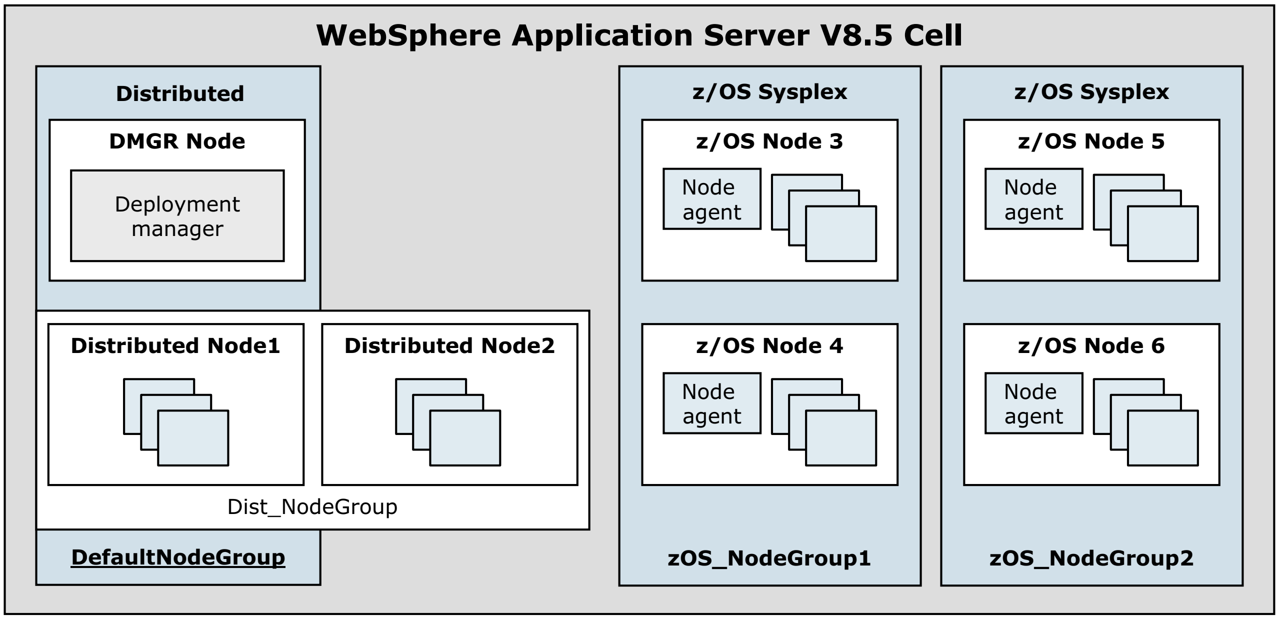


Figure 2-18 A heterogeneous cell with the coexistence of distributed and z/OS nodes

2.3.7 Deployment manager

The *deployment manager* is the central administration point of a cell that consists of multiple nodes and node groups in a distributed server configuration. It is similar to the configuration shown in Figure 2-15 on page 38. The deployment manager communicates with the node agents of the cell that it is administering to manage the applications servers within the node. The deployment manager provides management capability for multiple federated nodes, and can manage nodes that span multiple systems and platforms. A node can be managed by a single deployment manager, and the node must be federated to the cell of that deployment manager.

The configuration and application files for all nodes in the cell are centralized into the *master repository*. This centralized repository is managed by the deployment manager and regularly synchronized with local copies that are held on each of the nodes. If the deployment manager is not available in the cell, the node agents and the application servers cannot synchronize configuration changes with the master repository. This limitation continues until the connection with deployment manager is reestablished.

> **Version note**: A high availability deployment manager is available in WebSphere Application Server V8.5. You can configure a hot-standby deployment manager to recover failures of the currently active deployment manager.

2.4 Server configurations

With WebSphere Application Server, you can build various server environments that consist of single and multiple application servers that are maintained from central administrative points.

A *system* is defined as one of the following types:

- A server system (a physical machine) that contains only one operating system
- An operating system virtual image where the host server system contains multiple operating system images
- A z/OS image

With WebSphere Application Server, you can create two types of configurations in a single cell environment:

- Single system configurations
- Multiple systems configurations

Single system configurations

With the Base, Express, and Network Deployment packages, you can create a cell that contains only a single node with a single application server (Figure 2-19).

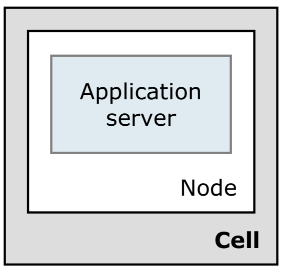


Figure 2-19 Single cell configuration in Base and Express packages

Single system is the only configuration option with Base and Express. The cell is created when you create the stand-alone application server profile.

A node agent at each node is the contact point for the deployment manager during cell administration. A single system configuration in a distributed environment includes all processes in one system as illustrated in Figure 2-20.

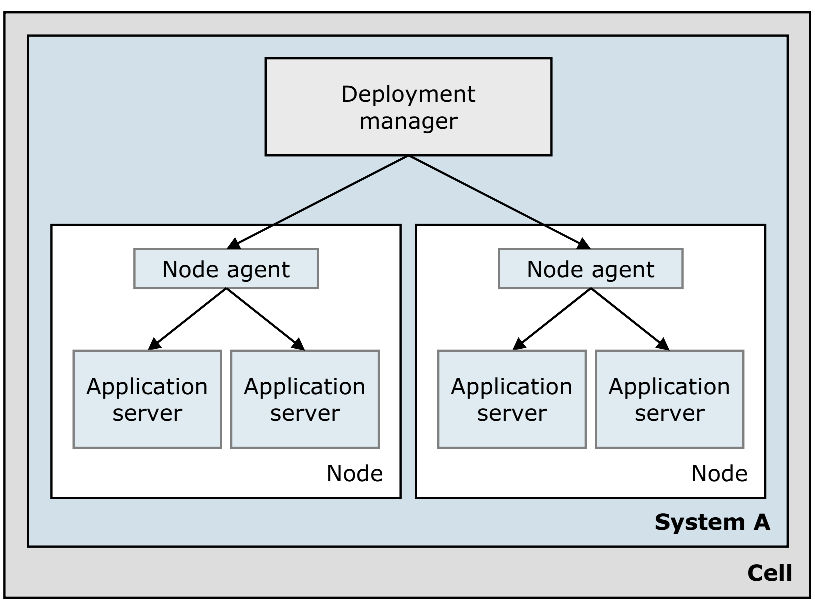


Figure 2-20 Cell configuration option in Network Deployment: Single system

Multiple system configurations

A Network Deployment environment allows you to install the WebSphere Application Server components on systems and locations that suit your requirements. With the Network Deployment package, you can create multiple systems configurations.

Figure 2-21 shows the deployment manager that is installed on one system (System A) and each node on a different system (System B and System C). The servers can be mixed platforms or the same platform. In this example, System A can be an IBM AIX® system, System B can be a Windows operating system, and System C can be a z/OS image.

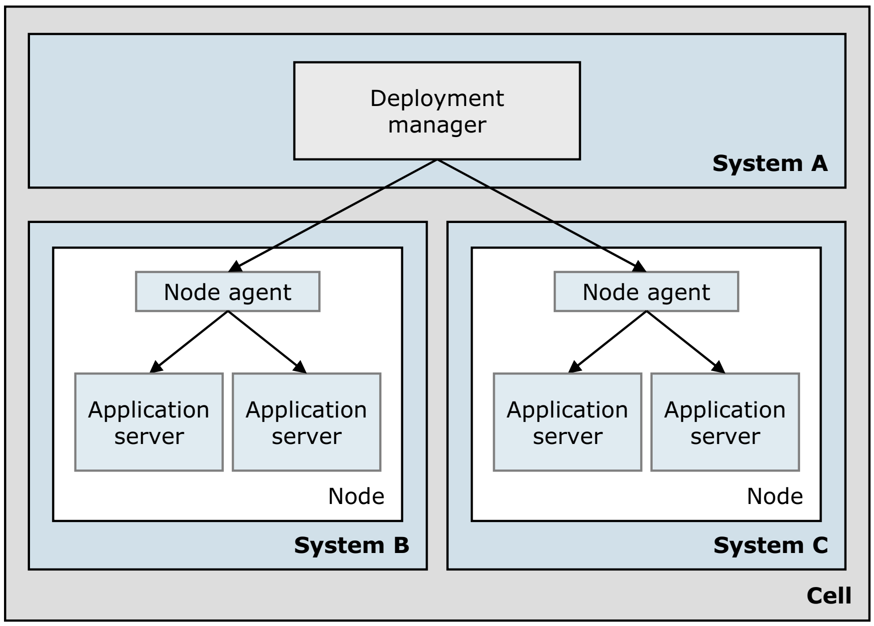


Figure 2-21 Cell configuration option in Network Deployment: Multiple systems

Using the same logic, other combinations can be installed. For example, you can install the deployment manager and a node on one system with additional nodes installed on separate systems.

2.5 Clusters and high availability

A *cluster* is a collection of servers that are managed together. With clusters, enterprise applications can scale beyond the amount of throughput that can be achieved with a single application server. Also, enterprise applications are made highly available because requests are automatically routed to the running servers in the event of a failure. The servers that are members of a cluster can be on different host systems. A cell can include no clusters, one cluster, or multiple clusters.

WebSphere Application Server provides clustering support for the following types of servers:

- Application server clusters
- Proxy server clusters
- Generic server clusters
- Dynamic clusters

An *application server cluster* is a logical collection of application server processes that provides workload balancing and high availability. It is a grouping of application servers that run an identical set of applications that are managed so that they behave as a single application server (parallel processing). WebSphere Application Server Network Deployment or WebSphere Application Server for z/OS is required for clustering.

Application servers that are a part of a cluster are called *cluster members*. When you install, update, or delete an application, the updates (changes) are distributed automatically to all cluster members. By using the rollout update option, you can update and restart the application servers on each node. This process can be done one node at a time, providing continuous availability of the application to the user.

Application server clusters have the following important characteristics:

- A cluster member can belong to only a single cluster.
- Clusters can span server systems and nodes, but they cannot span cells.
- A cluster cannot span from distributed platforms to z/OS.
- A node group can be used to define groups of nodes that have enough in common to host members of a cluster. All cluster members in a cluster must be in the same node group.

2.5.1 Vertical cluster

When cluster members are on the same system, the topology is known as *vertical scaling* or *vertical clustering*. Figure 2-22 illustrates a simple example of a vertical cluster.

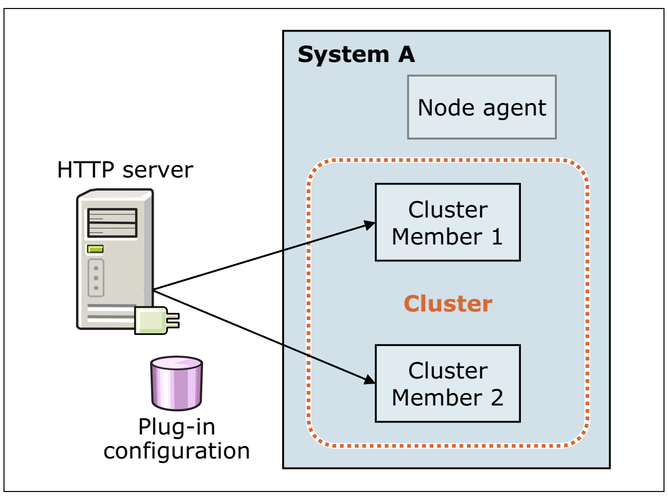


Figure 2-22 Vertical cluster

Vertical clusters offer fail over support within one operating system image, provide processor level fail over, and increase resource usage.

2.5.2 Horizontal cluster

Horizontal scaling or *horizontal clustering* refers to cluster members that are spread across different server systems and operating system types (Figure 2-23). In this topology, each system has a node in the cell that is holding a cluster member. The combination of vertical and horizontal scaling is also possible.

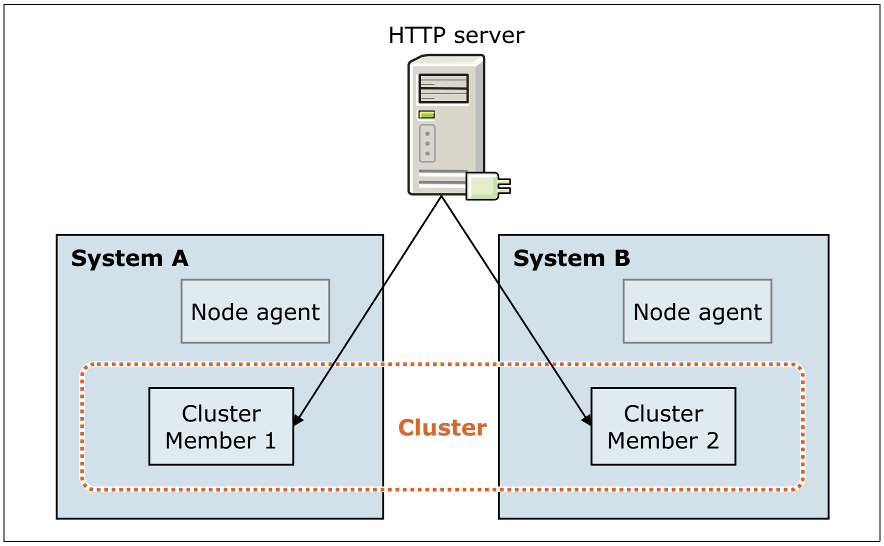


Figure 2-23 Horizontal cluster

Horizontal clusters increase availability by removing the bottleneck of using only one physical system and increasing the scalability of the environment. Horizontal clusters also support system fail over.

2.5.3 Mixed cluster

Figure 2-24 illustrates a cluster that has four cluster members and combines vertical and horizontal clustering. The cluster uses multiple members inside one operating system image (on one system) and that are spread over multiple physical systems. This configuration provides a mix of fail over and performance.

Cluster members cannot span cells.

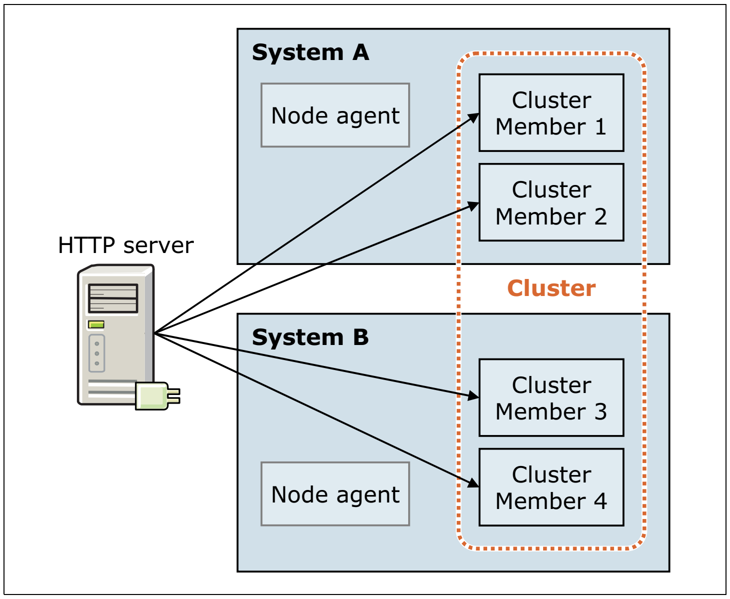


Figure 2-24 Vertical and horizontal clustering

2.5.4 Mixed-node versions in a cluster

A WebSphere Application Server Network Deployment V8.5 cluster can contain nodes and application servers from WebSphere Application Server V7 and V8. The topology that is illustrated in Figure 2-25 on page 47 contains mixed version nodes within a cluster. You can upgrade any node in the cell and leave the other nodes at a previous release level. Consider using this feature only for migration scenarios.

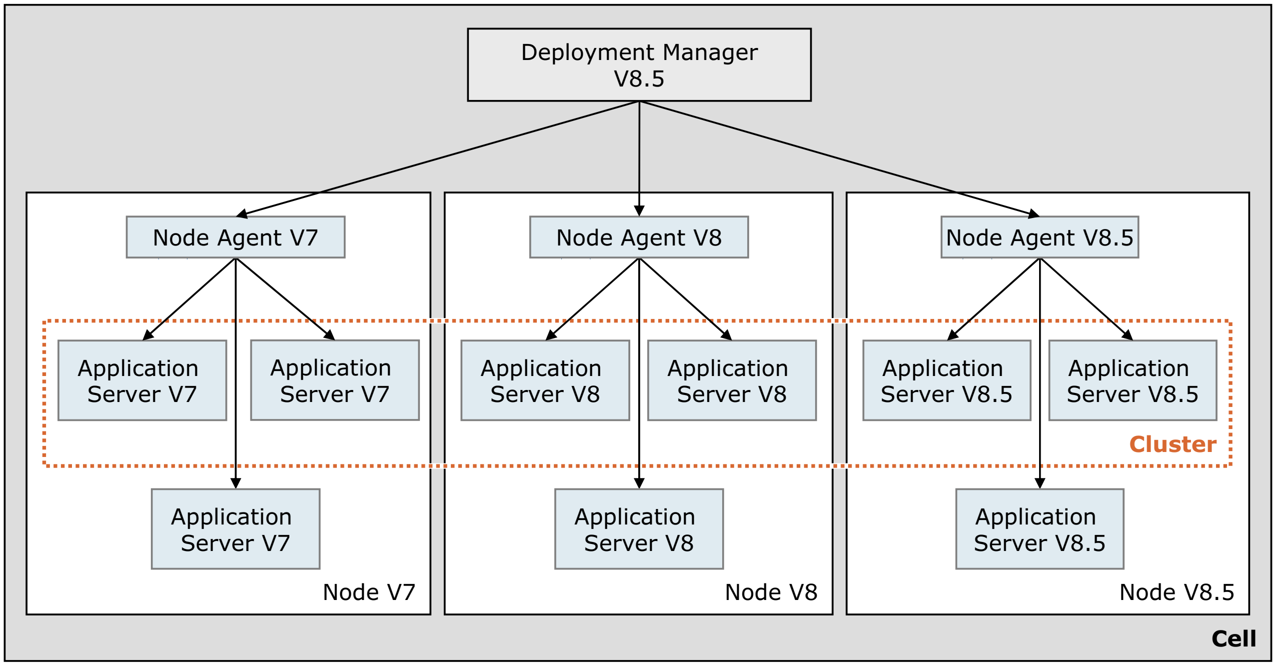


Figure 2-25 Mixed version nodes that are clustered in a cell

2.5.5 Dynamic cluster

Dynamic clusters are application deployment targets that operate at the application layer virtualization. Dynamic clusters provide capabilities to better manage dynamic workload by using the on-demand router server.

Keep in mind the following key points about dynamic clustering:

- Dynamic clusters grow and shrink depending on the workload demand.
- Dynamic clusters work closely with the on-demand router to ensure even distribution of workload among the cluster members.

2.5.6 Cluster workload management

This section highlights cluster workload management on distributed systems. It also addresses considerations for the z/OS platform.

Cluster workload management on distributed systems

Workload management, which is implemented by the use of application server clusters, optimizes the distribution of client processing requests. WebSphere Application Server can handle the workload management of servlet and EJB requests. HTTP requests can be workload-managed by using tools similar to a load balancer.

Using an HTTP traffic-handling device, such as IBM HTTP Server and the web server plug-in, is a simple and efficient way to front end the WebSphere HTTP transport.

WebSphere Application Server implements a server-weighted round-robin routing policy to ensure a balanced routing distribution. The policy is based on the set of server weights that is assigned to members of a cluster. In horizontal clustering, where each node is on a separate server system, the loss of one server system does not disrupt the flow of requests. Instead, requests are routed to cluster members on other nodes. In a horizontal cluster, the loss of the deployment manager has no impact on operations and primarily affects configuration activities. You can still use administration scripts to manage the WebSphere Application Server environment.

Cluster workload management consideration on z/OS

Workload management for EJB containers that run on z/OS can be performed by configuring the web container and EJB containers on separate application servers. Multiple application servers with the EJB containers can be clustered, enabling the distribution of enterprise bean requests between EJB containers on different application servers.

Instead of using a static round-robin procedure, workload management on the z/OS platform introduces a finer granularity and the use of real-time performance data. You can use these features to determine which member to process a transaction on.

> **Remember:** Workload management is achieved by using the WLM subsystem in combination with the Sysplex Distributor (SD) component of z/OS. The Sysplex Distributor receives incoming requests through a Dynamic Virtual IP address and prompts WLM to indicate to which cluster member the request should be transmitted. WLM tracks how well each cluster member is achieving its performance goals in terms of response time. Therefore, it chooses the one that has the best response time to process the work.

You can classify incoming requests according to their importance. For example, requests that come from a platinum-ranked customer can be processed with higher importance (and therefore faster) than a silver-ranked customer.

When resource constraints exist, the WLM component can ensure that the member that processes a higher prioritized request gets additional resources. This system protects the response time of your most important work.

> **WLM changes:** The WLM component can change the amount of processor, I/O, and memory resources that are assigned to the different operating system processes (the address spaces). To decide whether a process is eligible for receiving additional resources, the system checks whether the process meets its defined performance targets, and whether more important work is in the system. This technique is run dynamically so that there is no need for manual interaction after the definitions are made by the system administrator (the system programmer).

2.6 Database access from WebSphere Application Server

Java Platform, Enterprise Edition components that are deployed in a WebSphere Application Server often require access to data stored in databases such as DB2 for z/OS.

Accessing data out of a Java EE environment (which WebSphere Application Server is) involves a key concept that is important to understand -- *the specifics of the actual data system are hidden from the application; they are hidden behind a standardized layer of abstraction.* What the Java EE specification provides is a standardized API for accessing data, with the vendor of the actual data system responsible for implementing the code behind the API layer. The implementation that is offered by the vendors is called a "connector" as shown in Figure 2-26.

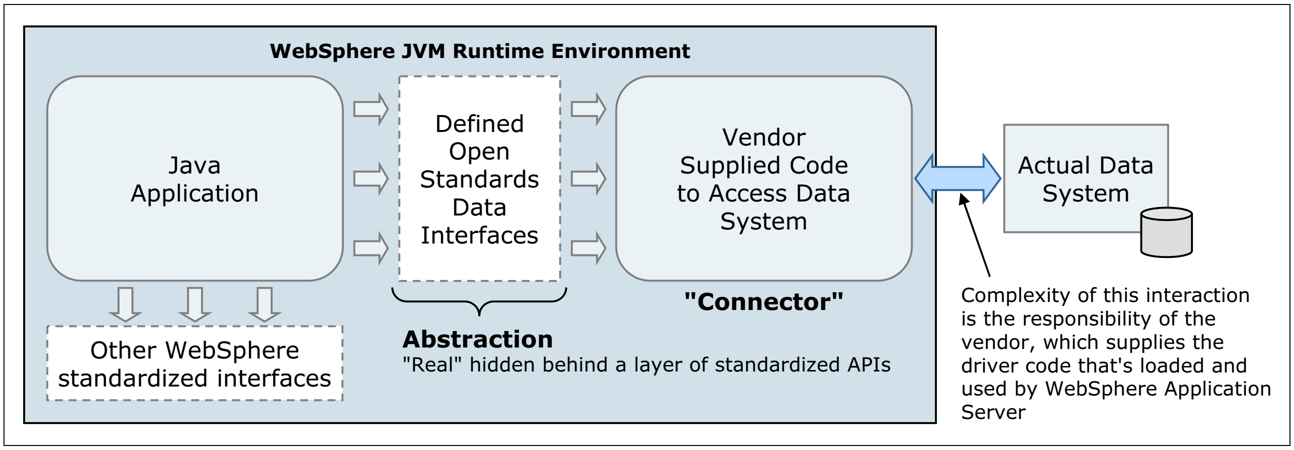


Figure 2-26 Accessing data from Java applications on WebSphere

The standardized API for relational databases is defined by what is called Java Database Connectivity Specification (JDBC).

2.6.1 JDBC driver types

The role of the JDBC driver is to implement the objects, methods, and data types that are defined in the JDBC specification. Currently DB2 for z/OS supports the following driver types:

Type 2

Type 2 drivers are written partly in the Java programming language and partly in native code. These drivers use a native client library specific to the data source to which they connect. JDBC type 2 connectivity should be used only when running Java applications - whether stand-alone Java applications or applications running in WebSphere Application Server on z/OS - on z/OS accessing DB2 data on z/OS in the same LPAR. This type of connectivity is recommended when the applications are deployed in WebSphere Application Server on z/OS accessing data on DB2 for z/OS on the same LPAR.

Type 4

Type 4 drivers are written in pure Java and implement the database protocol for a specific data source. The client connects directly to the data source. DRDA is the protocol that is used when connecting to a DB2 system as a data source. The type 4 driver is fully portable because it is written purely in Java.

The IBM implementation of these drivers is called IBM Data Server Driver for JDBC and SQLJ. For details, see 3.2, "IBM Data Server Drivers and Clients" on page 87.

2.6.2 Concept of JDBC providers

JDBC provider is way to define to WebSphere Application Server - irrespective of platform - the location of the Java classes that must be used by the application server when connecting to the database. There are two types of providers that WebSphere Application Server supports when it comes to JDBC access to DB2 for z/OS. They are the

- DB2 Universal JDBC Driver Provider
- DB2 Universal JDBC Driver Provider (XA)

The DB2 Universal JDBC Driver Provider (XA) should be used only if applications running in WebSphere Application Server meet the two following criteria:

- Require global transaction support
- Want to use JDBC type 4 access to DB2 for z/OS

The DB2 Universal JDBC Driver Provider should be used only if applications running in WebSphere Application Server meet the either of the following criteria:

- Applications running on WebSphere Application Server on z/OS and access DB2 for z/OS on the same LPAR using JDBC type 2 access. This provider supports both 1-phase and 2-phase commit processing
- Applications running on WebSphere Application Server (irrespective of platform) that need access to DB2 for z/OS and do not require global transaction support

For details, see 5.2, “Configuring WebSphere Application Server for JDBC type 4 XA access” on page 209 and 5.3, “Configuring WebSphere Application Server for JDBC type 2 access” on page 222.

2.6.3 Concept of data sources

Applications that run in WebSphere Application Server look at a DB2 server through a data source object, which is logically addressable through a name, recommended to be in the form jdbc/data-name called the JNDI name. The purpose of the data source is to define to WebSphere Application Server, the connection information of the database such as the database name, the type of connection to use (type 2 or type 4), IP address where the database is located, the port number the database uses to receive connections and the user ID and password to use when the application server establishes a connection to the database. The data source definition in WebSphere Application Server also allows users to define the following important settings:

- Connection Pool configuration
- Prepared Statement Cache setting
- IBM Data Server Driver for JDBC and SQLJ custom properties, such as current schema

Data sources are defined to JDBC providers in WebSphere Application Server.

For details, see 5.2, “Configuring WebSphere Application Server for JDBC type 4 XA access” on page 209 and 5.3, “Configuring WebSphere Application Server for JDBC type 2 access” on page 222.

2.6.4 WebSphere Application Server connection pooling

Connection pooling can improve the response time of any application requiring connections to access a data source, especially web-based applications. To avoid the impact of acquiring and closing connections, WebSphere Application Server provides connection pooling for connection reuse (caching of JDBC connections). WebSphere Application Server enables administrators to establish a pool of database connections that can be reused. They are defined with the panel shown in Figure 2-27.

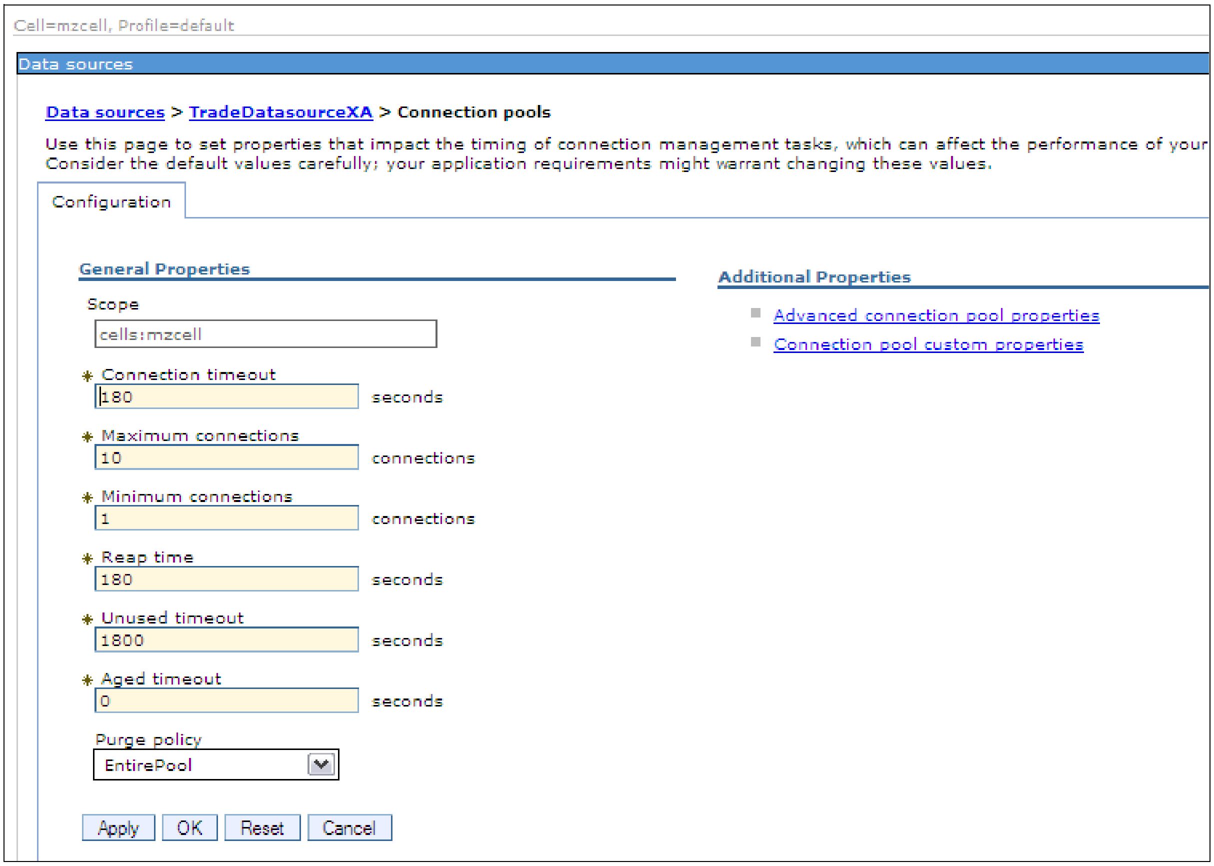


Figure 2-27 WebSphere Application Server database connections

The following is a brief description of the properties:

Connection Timeout	How long to attempt connection creation before timeout
Max Connections	Maximum number of connections from JVM instance
Min Connections	Lazy minimum number of connections in the pool
Reap Time	How often cleanup of pool is scheduled in seconds
Unused Timeout	How long to let a connection sit in the pool unused
Aged Timeout	How long to let a connection live before recycling
Purge Policy	After StaleConnection, does the entire pool get purged or only individual connection

Information about these properties can be found at the following websites:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/cdat_conpool.html
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/udat_conpoolset.html

To get the most out of connection pooling, consider the following items:

- If an application creates a resource, the application should explicitly close it after the resource is no longer being used

  All JDBC resources that have been obtained by an application should be explicitly closed by the same application. These include connections, CallableStatements, PreparedStatements, ResultSets, and others. Be sure to close resources even in the case of a failure. For example, each PreparedStatement object in the cache can have one or more result sets associated with it. If a result set is opened and not closed, even though you close the connection, that result set is still associated with the prepared statement in the cache. Each of the result sets has a unique JDBC cursor that is attached to it. This cursor is kept by the statement and is not released until the prepared statement is cleared from the WebSphere Application Server prepared statement cache.

- Obtaining and closing the connection in the same method.

  When possible, we recommend that an application obtains and closes its connection in the same method in which the connection is requested. This keeps the application from holding resources that are not being used, and leaves more available connections in the pool for other applications. Additionally, it removes the temptation to use the same connection for multiple transactions. There might be times in which this is not feasible, such as when using BMP.

- Do not reuse the statement handle without closing it first.

  To prevent resource leakage, close prepared statements before reusing the statement handle to prepare a different SQL statement with the same connection.

- Set WebSphere Application Server connection Unused Timeout to a value smaller than DB2 for z/OS idle thread timeout to avoid stale connection conditions.

- Consider setting min connections to 0 (zero)

- Consider setting WebSphere Application Server “aged timeout” to less than 5 min, recommended 120 sec to reduce exposure of long living threads

Heterogeneous connection pooling

WebSphere Application Server also provides what is called a heterogeneous connection pool. Heterogeneous pooling is the ability to share one data source and hence one connection pool among different applications that are deployed in the same WebSphere Application Server that try to access the data in DB2 for z/OS. This feature helps customers address the impact of having to manage connections at individual data sources and also helps them avoid a proliferation of data source definitions in WebSphere Application Server going to the same DB2 for z/OS. There are some rules that must be followed for this to work

- Applications must use resource references when they look up a data source

- Defer to each application definition of application-specific non-core data source properties such as

  – currentSchema
  – currentFunctionPath

- The core properties have to be identical such as
  - Username
  - Host and port

The following link in WebSphere Application Server information center explains the extended properties that can be set for each application

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/tdat_heteropool.html?resultof=%22%68%65%74%65%72%6f%67%65%6e%6f%75%73%22%20%22%68%65%74%65%72%6f%67%65%6e%22%20

The benefits of heterogeneous connection pooling are:

- Reduction in memory consumption in WebSphere.
- Creating one data source versus multiple data sources.
- Reduction in memory consumption in DB2 (threads, connection) and hence improved performance.

2.6.5 WebSphere connection pooling combined with sysplex workload balancing for JDBC type 4 connectivity

The DB2 primary approach for scalability and high availability is to use the clustering capabilities of the System z Parallel Sysplex. In a DB2 data sharing environment, multiple instances of DB2 data sharing member can all access the same databases. Workload can be spread across the different members of the data sharing groups based on factors such as:

- Current workload of the DB2 member
- Current health and state of the DB2 member,
- Current capacity of the LPAR in which DB2 is running

Applications that are deployed in WebSphere Application Server that use IBM Data Server Driver for SQLJ and JDBC and use a JDBC type 4 connection to DB2 for z/OS can be enabled to be sysplex aware.

In Figure 2-28, we can see the logical connections are managed by WebSphere Application Server while the transports, which are the actual connections to DB2 for z/OS, are managed by the IBM Data Server Driver for JDBC and SQLJ. As shown in the figure below, the logical connections are disassociated from the transports at commit/rollback boundaries (transaction boundaries). This allows for transaction level workload balancing.

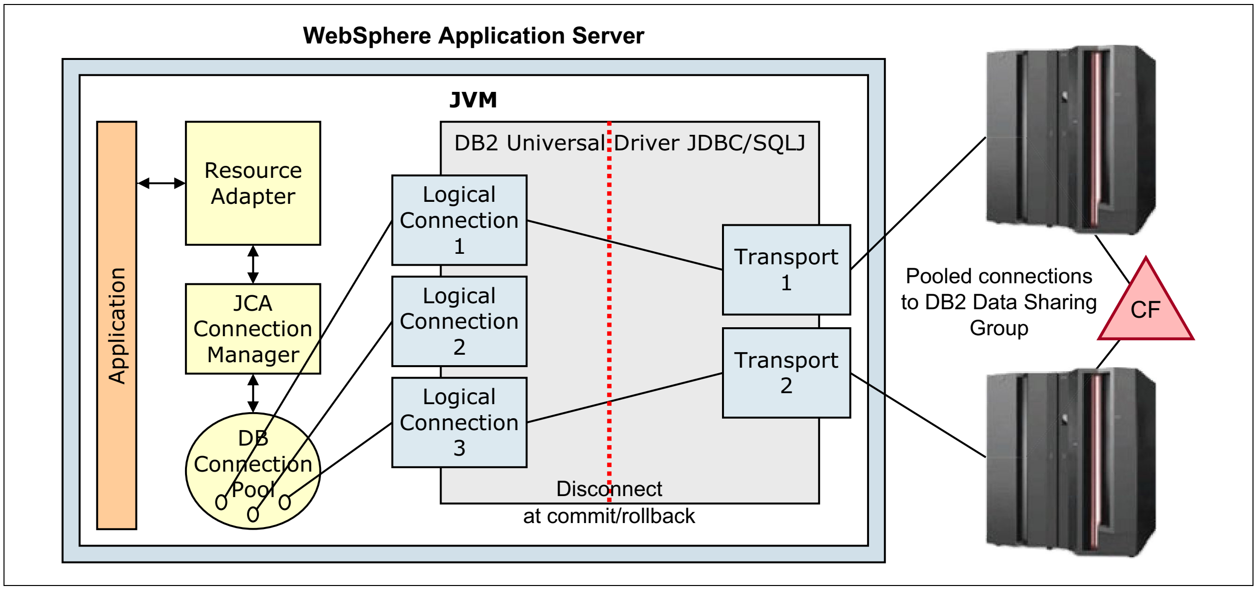


Figure 2-28 Logical connections and Transports

To enable sysplex workload balancing, the following items must configured

- Data source custom property enablesysplexWLB must be added with a value of true.
- DB2 for z/OS data sharing must be set up following the preferred practice recommendation
- The group DVIPA address of the DB2 data sharing group must be specified as the server host name in the data source definition

The following is a brief description of how sysplex workload balancing works

1. The initial connection to DB2 is made by using the group dynamic virtual IP address. This resolves to any available member based on WLM
2. The initial connection returns a member IP list of all available DB2 data sharing group members with WLM weights
3. Any following connection uses client sysplex workload balancing algorithm to determine which member to use if reuse is OK. Transports that have the following are not eligible for reuse:
   a. Transports have open WITH HOLD cursor
   b. Declared global temporary tables must not exist. Declared global temp tables must be explicitly or implicitly dropped
   c. Packages that are bound with KEEPDYNAMIC YES

When connection errors happen, the following behavior occurs:

- If first SQL stmt in transaction fails and reuse OK
  - No errors reported back to application
  - SET statements that are associated with the logical connection are replayed with first SQL on another transport
- If subsequent SQL fails and reuse OK
  - 30108 reuse error returned to application (transaction is rolled back and reconnected).
  - SET statements are replayed on another transport to recover connection state
  - Up to application to retry transaction
- If subsequent SQL and reuse not OK

  30081 connection failed error returned to application.
  Connection returned to initial (default) state
  Application needs to reestablish connection state and retry transaction
- If all members in the member list are tried and none seems to be available, the initial data source group DVIPA address is retried to make sure that really no member is available.

Considerations

We list here considerations for best practices:

- Ensure that the application handles the above mentioned SQL codes and take appropriate action
- WebSphere Application Server provides a feature that pretests a connection by running a SQL statement. This is to avoid stale connections. If sysplex workload balancing is used, it is recommended to disable this feature as the IBM Data Server Driver for JDBC and SQLJ ensures that a valid connection is returned. This pretest can be disabled as shown in Figure 2-29 by leaving the boxes for validation cleared.

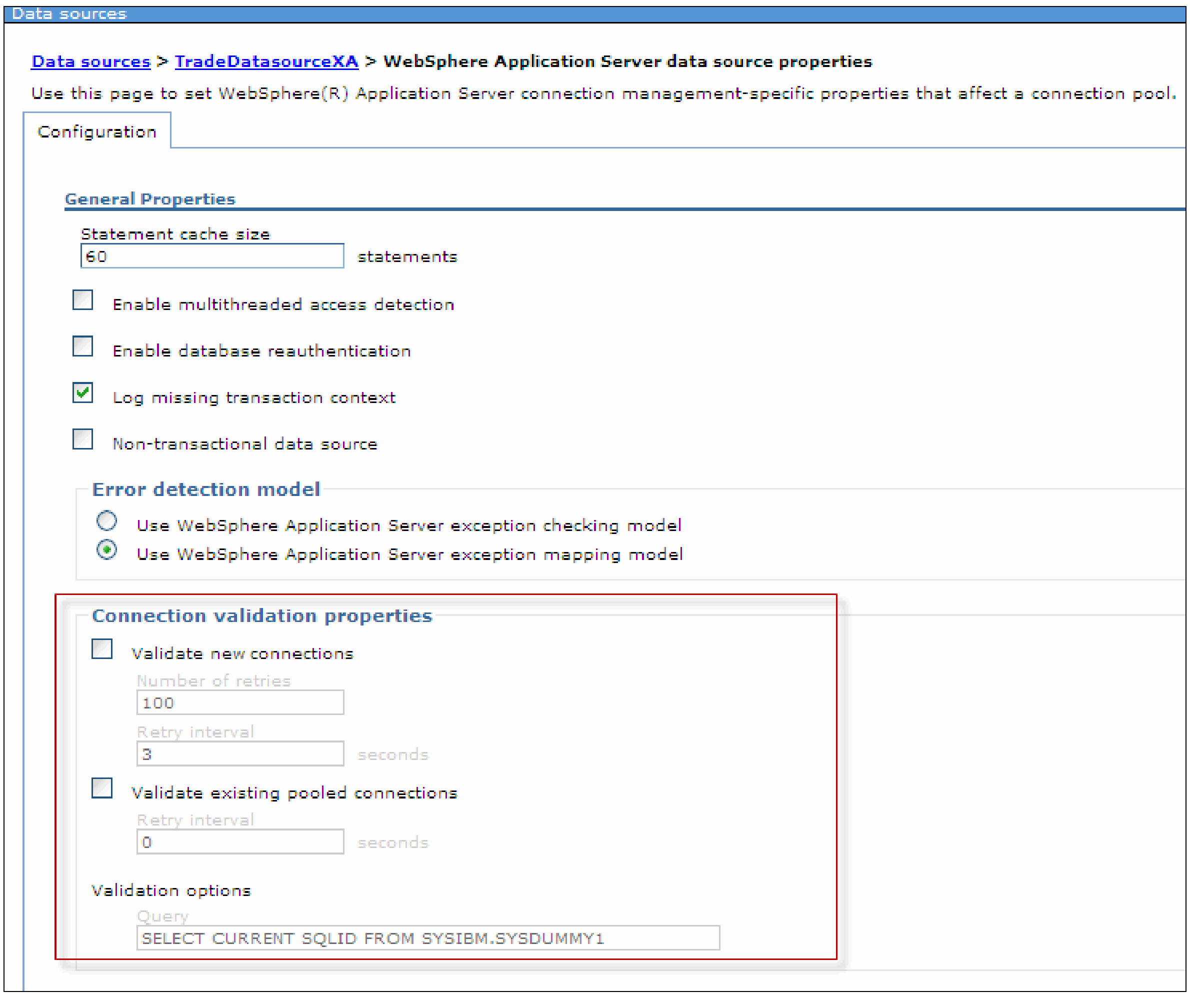


Figure 2-29 Disabling pretest connection

- If sysplex workload balancing is exploited, then it is recommended to disable the WebSphere Application Server connection pool properties Reap Time, Unused Timeout and Aged Timeout by setting the values to zero as shown in Figure 2-30 on page 57. The reason being that the actual physical connections are handled by the IBM Data Server Driver for JDBC and SQLJ and the connections that are handled by WebSphere Application Server are logical connections.

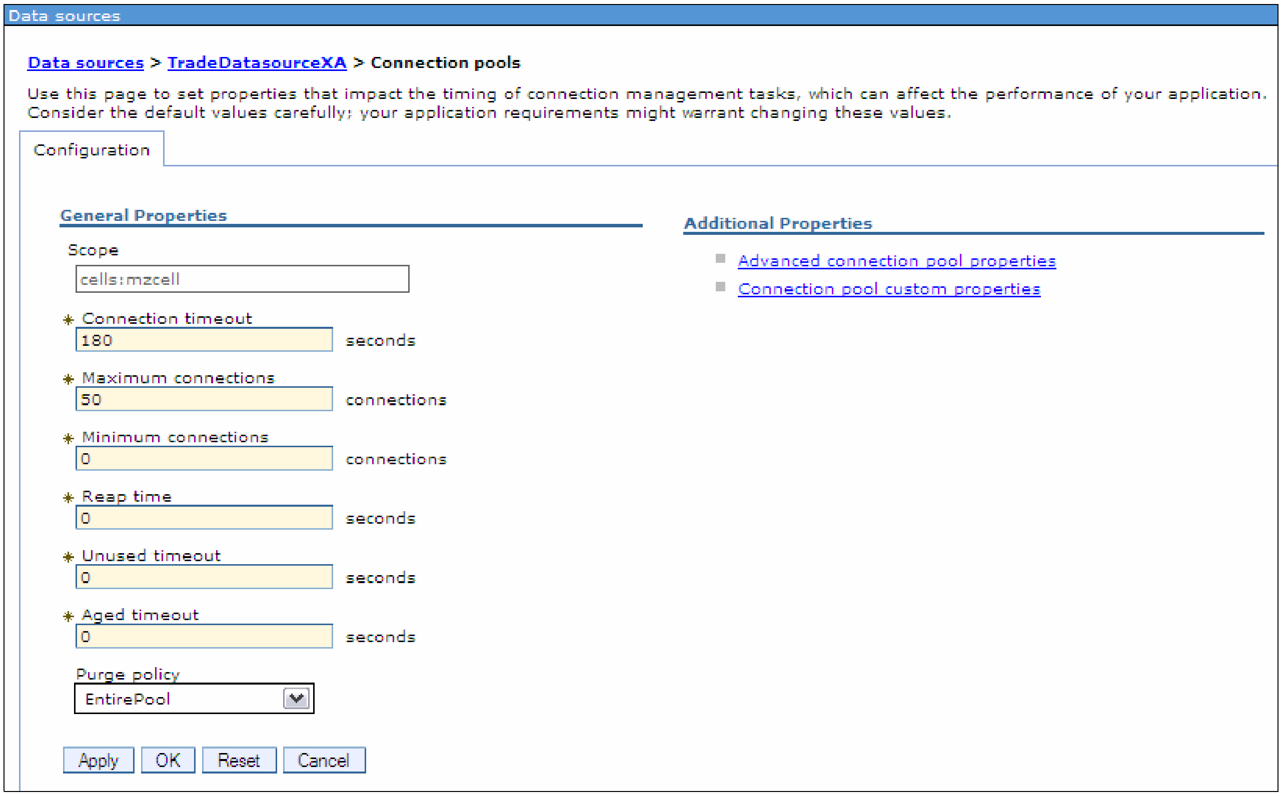


Figure 2-30 Disabling properties Reap Time, Unused Timeout, and Aged Timeout

2.6.6 WebSphere Application Server prepared statement cache

WebSphere Application Server manages a cache of previously created prepared statement objects at a connection level. When a new prepared statement is requested on a connection, by the application, the cached prepared statement object is returned if it is available on that connection. Creating a new prepared statement object is costly in Java. WebSphere Application Server prepared statement cache does not store any DB2 specific information. The cache is solely used by WebSphere to reduce processor consumption for creating a Java object. Customers should monitor the number of distinct SQL statements that are used by the application and then come up with a number to define the size of the statement cache. This is at a connection level and hence has an impact on WebSphere Application Server heap size.

WebSphere Application Server prepared statement cache with DB2 Dynamic Statement cache

The WebSphere Application Server statement cache function works together with DB2 dynamic statement caching. When the prepared statements are cached in the DB2 for z/OS (DSNZPARM CACHEDYN=YES), recalculation of the access path can be avoided if the statement in the DB2 dynamic statement cache can be reused by a subsequent execution. This saving is in addition to saving the JDBC precompiled SQL cost that WebSphere Application Server statement caching (caching the prepared statement object) offers. This is depicted in Figure 2-31.

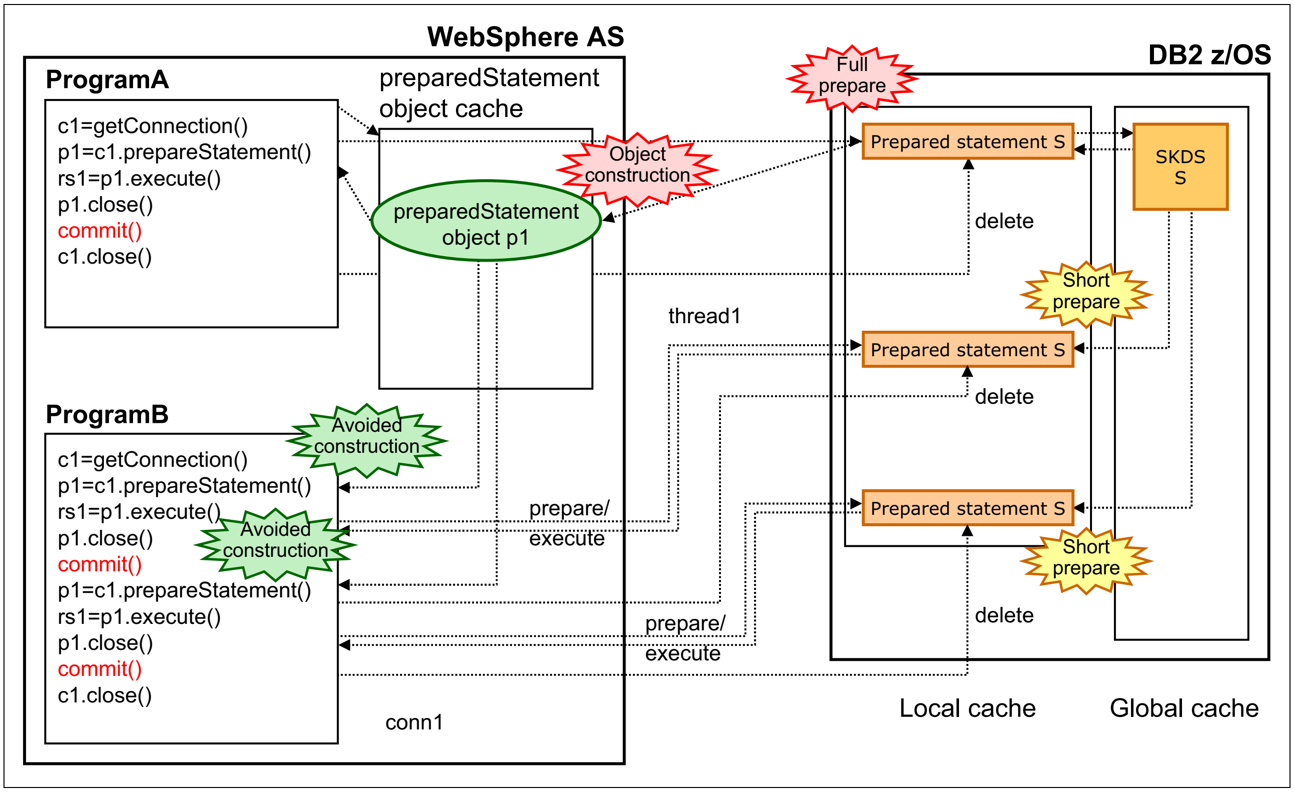


Figure 2-31 WebSphere Application Server: caching the prepared statement object

When the application runs the prepareStatement() JDBC API, WebSphere Application Server looks for existence of the Java preparedStatement object in the statement cache that exists in WebSphere Application Server. This cache is unique to each connection in the connection pool. It must be remembered that it is a Java preparedStatement object and has nothing to do with the prepare that happens in DB2 for z/OS. In this case, because this statement is prepared for the first time, WebSphere Application Server cannot find it in the cache. It creates a Java preparedStatement object and stores it in its cache.

- When using JDBC type 2 connectivity to DB2 for z/OS, the driver will immediately send the SQL statement to DB2 to be prepared. DB2 will first look in "local cache" to see whether it can find the SQL statement. In this case, it does not exist. DB2 then looks for the statement in global dynamic statement cache. In this case, the statement is not found in the global dynamic statement cache. DB2 does what is called a "full prepare" during which it checks the validity of the SQL, determines the access path to be used, and so on.

If the SQL statement is valid, the DB2 then stores the statement in the global statement cache called “global cache” in Figure 2-31 on page 58. DB2 also stores information about the prepared statement in thread storage that is created in DB2. Then, it returns to WebSphere Application Server, which then returns the Java prepared statement object back to the application. The application then runs the statement and then issues a commit. When the commit is issued, the prepared statement artifacts that are stored in the DB2 thread storage that is known as “local cache” are also deleted and the DB2 thread is ready for reuse

- When using JDBC type 4 connectivity to DB2 for z/OS, the driver by default will not send the SQL statement immediately to DB2 for z/OS. Instead, the WebSphere Application Server returns a Java preparedStatement object to the application. This behavior is controlled by a JDBC property called “deferPrepares”. By default this property is to true and is only valid for the JDBC type 4 connectivity to DB2 on z/OS. This helps to optimize the number of trips to DB2 on z/OS over the network. When the application issues the preparedtStaement.execute command, the JDBC driver will then send the SQL statement to DB2 on z/OS. DB2 will look for the statement in global dynamic statement cache. In this case, the statement is not found in the global dynamic statement cache. DB2 does what is called a “full prepare” during which it checks the validity of the SQL, determines the access path to be used, and so on.

  If the SQL statement is valid, DB2 then stores the statement in the global statement cache called “global cache” in Figure 2-31 on page 58. DB2 also stores information about the prepared statement in thread storage that is created in DB2. This is called “local cache”. DB2 then runs the SQL statement and then returns control back to the WebSphere Application Server and to the application. The application then issues a commit. When the commit is issued, the prepared statement artifacts that are stored in the DB2 thread storage (“local cache”) are also deleted and the DB2 thread is ready for reuse

Now, the next application thread comes along and the same code to prepare the SQL statement is ran. WebSphere Application Server, looks in the statement cache in WebSphere Application Server. In this case, it finds the preparedStatement object in the cache and the object construction is avoided

- When using JDBC type 2 connectivity to DB2 on z/OS, the driver immediately sends the SQL statement to DB2 to be prepared. DB2 first looks in “local cache” to see whether it can find the SQL statement. In this case, it does not exist. DB2 then looks for the statement in global dynamic statement cache. In this case, the statement is found in the global dynamic statement cache. DB2 does what is called a “short prepare” during which it actually copies the artifacts from the global statement cache to the thread (“local cache”). Then, it returns to WebSphere Application Server, which then returns the Java prepared statement object back to the application. The application then runs the statement and then issues a commit. When the commit is issued, the prepared statement artifacts that are stored in the DB2 thread storage that is known as “local cache” are also deleted and the DB2 thread is ready for reuse.
- When using JDBC type 4 connectivity to DB2 for z/OS, the driver by default does not send the SQL statement immediately to DB2 for z/OS. Instead, the WebSphere Application Server returns a Java preparedStatement object to the application. This behavior is controlled by a JDBC property called “deferPrepares”. By default this property is set to true and is only valid for the JDBC type 4 connectivity to DB2 for z/OS. This helps to optimize the number of trips to DB2 on z/OS over the network. When the application issues the preparedtStaement.execute command, the JDBC driver will then send the SQL statement to DB2 for z/OS. DB2 first looks in “local cache” to see whether it can find the SQL statement.

In this case, it does not exist. DB2 then looks for the statement in global dynamic statement cache. In this case, the statement is found in the global dynamic statement cache. DB2 does what is called a “short prepare” during which it actually copies the artifacts from the global statement cache to the thread known as “local cache”. DB2 then runs the SQL statement and then returns control back to the WebSphere Application Server and to the application. The application then issues a commit. When the commit is issued, the prepared statement artifacts that are stored in the DB2 thread storage that is known as “local cache” are also deleted and the DB2 thread is ready for reuse.

This behavior of copying artifacts from global dynamic statement cache to local cache is also followed when using static SQL applications. Instead of copying from global statement cache, the artifacts are copied from the static application packages in the EDM pool to thread storage.

This shows the benefits of having a prepared statement cache in WebSphere Application Server and also how it works with DB2 global dynamic statement cache.

WebSphere Prepared Statement Cache and DB2 KEEPDYNAMIC option

In the previous section, we talked about WebSphere Prepared Statement cache. In that section, we mentioned “local cache”. In this section, we talk about the “local cache” and how it is useful and how best to use it.

The local cache is associated with an individual thread in DB2. When an application runs a SQL statement, the contents of the global dynamic statement cache are copied into the local cache in DB2 thread storage. This cache is “destroyed” when the application issues a commit. Then, the application runs the SQL statement again, the process of copying is repeated and after the application issues a commit, the “destroy” is also repeated. This copying of contents from the global dynamic statement cache into local cache becomes expensive (CPU time) if this happens over and over again.

To keep the local cache across commit boundaries, the following steps must be completed:

- DB2 provides a bind option called KEEPDYNAMIC. The JDBC/SQLJ packages that are provided by IBM must bound with KEEPDYNAMIC(YES) bind option. Typically you should bind these packages to a different collection than the ones used for applications that do not use the KEEPDYNAMIC option. For example, let the collection name be “MYCOLL1”.
- If you use SQLJ / IBM pureQuery® applications, then those application packages also must be bound with KEEPDYNAMIC option to a different collection name. For example, let the collection name be “MYCOLL2”.
- If the application is using JDBC type 4 connectivity, the above collection names must be specified as part of the “currentPackagePath” data source property. For example, the value specified in the “currentPackagePath” property looks like “MYCOLL1.*,MYCOLL2.*”.
- If the application is using JDBC type 2 connectivity, then the above collection names must be specified as part of the “pkList” data source property. For example, the value specified in the “pkList” property looks like “MYCOLL1.*,MYCOLL2.*”.
- Specify the “keepDynamic” property in the data source custom property and set the value to 1

After the above steps are completed, then WebSphere prepared statement cache, DB2 “local cache” (which is nothing but thread storage in DB2) and the DB2 global dynamic statement cache work together as shown in Figure 2-32.

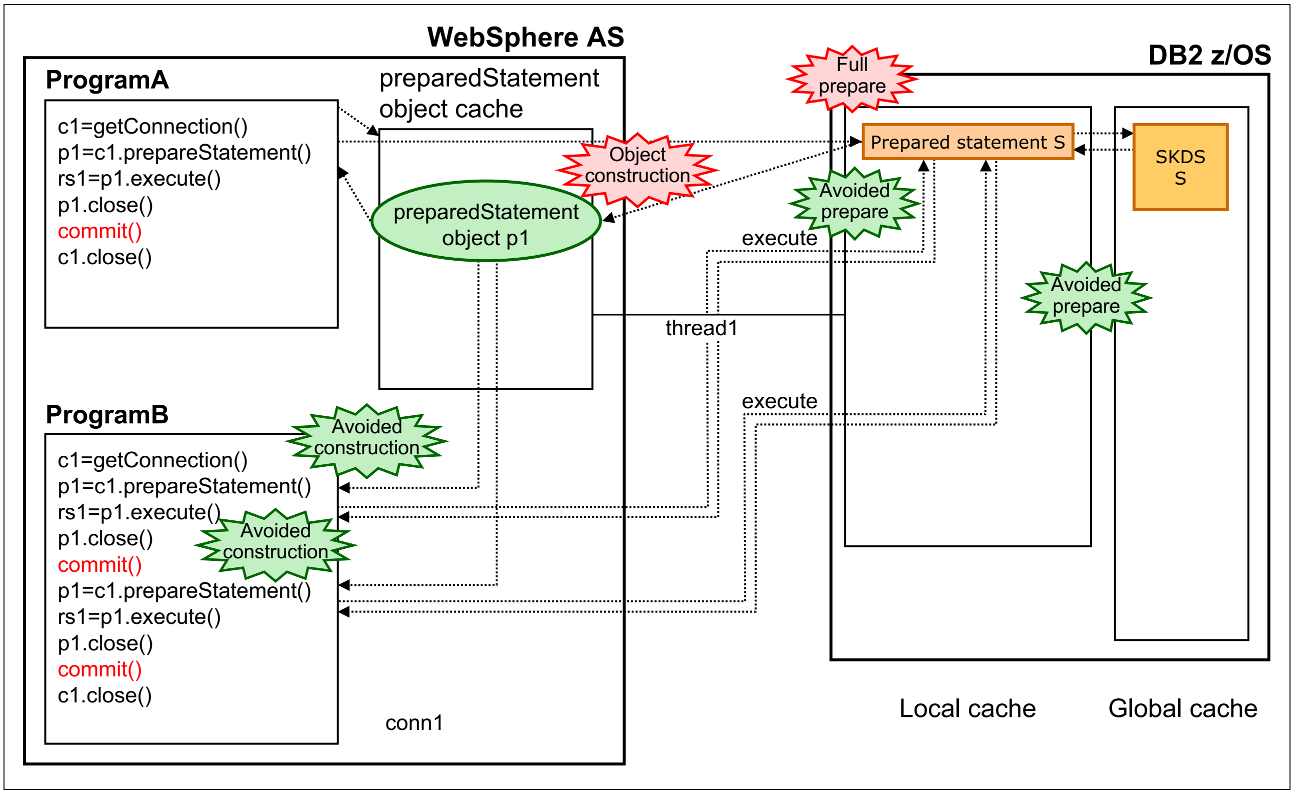


Figure 2-32 preparedStatement object loaded in cache

As we can see, in the first case the application issues the prepareStatement() JDBC API. WebSphere Application Server looks for the Java preparedStatement object in the cache. If it does not find it, a new preparedStatement object is constructed and put it in the cache. Then, the SQL statement is sent to DB2 for z/OS. DB2 looks for the SQL statement in the thread storage. It is not found in thread storage, DB2 then looks for the SQL statement in the global statement cache. It is not found there as well, DB2 then does a “full prepare” which entails validating the SQL statement and coming up with an optimal access path. DB2 then puts this SQL statement in the global statement cache. It then also stores the artifacts in the DB2 thread storage. Then, control is returned to the application. The application then runs the statement and then issues a commit.

Notice that because keepDynamic is enabled, the information in thread storage is not destroyed, and the DB2 thread is still available for reuse. The same thread is used for this work in DB2. Now the application again issues a preparedStatement with the same SQL statement. WebSphere Application Server finds the java preparedStatement object in cache. It then sends the SQL statement to DB2 for prepare. Because “keepDynamic” is enabled, the SQL statement is found in thread storage and DB2 then returns control back to the application, which then runs SQL statement.

It is not a problem if the Java application issues the prepare again, the statement is “absorbed” by the driver and not routed DB2. This is different from other languages, like COBOL, where you cannot issue the prepare again after the commit. If you do, the local cached copy is destroyed and you do not get the benefit of keepDynamic.

Advantages of using keepDynamic:

- Reduction in CPU time as the short prepares are avoided

Disadvantages of using keepDynamic

- Because the statements are kept in thread storage, DB2 environments that have storage constraints should be careful in using this. Virtual storage should be monitored
- Sysplex workload balancing is not available if keepDynamic is used

The keepDymamic option is best for applications that have a limited number of SQL statements that are used heavily.

2.6.7 Trusted context support in WebSphere Application Server

In typical applications that run on WebSphere Application Server, the authentication is done at the WebSphere Application Server layer. These applications often access data in DB2 for z/OS. The connection to the DB2 from WebSphere Application Server uses a JAAS alias. The alias contains the user ID and password. Figure 2-33 depicts a typical scenario

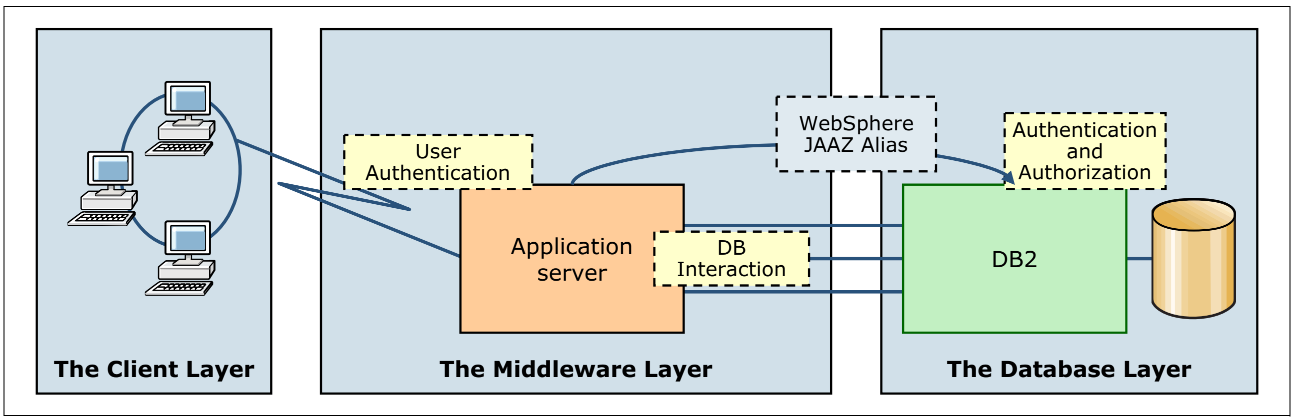


Figure 2-33 The standard security layers

The above setup presents the following challenges for customers:

- The user ID is never passed to DB2 for z/OS.
- The user ID defined in the JAAS alias often has significant privileges to access DB2 for z/OS data.
- Anyone who knows the user ID and password can use it to access the data in DB2 for z/OS and this compromises security.

The DB2 Trusted Context support in WebSphere Application Server provides an elegant solution for this problem. A trusted context is an object the database administrator defines that contains a system authorization ID and a set of trust attributes. The relationship between a database connection and a trusted context is established when the connection to the database server is first created, and that relationship remains for the life of the database connection. This feature allows WebSphere Application Server to use the trusted DB2 Connection under a different user without reauthenticating the new user at the database server (assuming the Trusted Context is created without authentication option being required).

There are two ways to set this up:

- There is nothing to be done from a WebSphere Application Server. In DB2, a Trusted Context is created with the system authid. This user ID is only granted access to connect to DB2. This user ID is used in the JAAS alias that the data source uses to connect to DB2. A ROLE is created in DB2, which has the required privileges to the application needs to access data in DB2. The user ID is then granted the role.

  Benefits of this approach

  - The user ID and password in the JAAS alias can be used only to access DB2 data from the WebSphere Application Server
  - Nothing needs to be configured in WebSphere Application Server

  Cons of this approach

  - The user ID is still not available in DB2

- Configure WebSphere Application Server to pass the user ID to DB2 for z/OS. In DB2, a Trusted Context is created with the system authid. This user ID is only granted access to connect to DB2. This user ID is used in the JAAS alias that the data source uses to connect to DB2. In the trusted context definition in DB2, user ID/groups should be added. Authentication requirements can be added to the trusted context definition. A ROLE is created in DB2, which has the required privileges to the application needs to access data in DB2. The added user ID/groups are then granted the role.

  The application should use resource references. In the resource definition panel, as part of the Modify Resource Authentication Method, Use Trusted Connection can be selected and a JAAS alias, which has the system user ID from the trusted context definition in DB2 specified.

  Benefits of this approach

  - The user ID and password in the JAAS alias can be used only to access DB2 data from the WebSphere Application Server
  - The user ID is available in DB2

  Cons of this approach

  - The user ID must be defined in the SAF environment

More information about this topic can be found at

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/csec_trustedconnections.html

2.6.8 Transaction Isolation Level support in WebSphere Application Server

During database access, transaction isolation determines the nature of locks to be acquired, which ultimately determines the transactional integrity. In addition, isolation level is a significant factor in determining whether two separate transactions can read or update the same data and how long the acquired locks prevent other transactions from performing specific tasks. The effects of transaction isolation can be described as the length of time the lock is held, known as lock duration, and the exclusiveness of the lock, which is known as the lock mode.

The default isolation level that is used in WebSphere Application Server 8.5 when accessing DB2 for z/OS is Read Stability (RS). To customize the default isolation level, you can use the webSphereDefaultIsolationLevel custom property for the data source.

Information about isolation level settings can be found at 6.8, “Locking” on page 331 and at:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/cdat_isolevel.html

2.6.9 Transactions in WebSphere Application Server

A transaction is a set of work for which either all individual work items or no items are performed. A failure of a subset requires that the entire work set is undone. This all-or-none attribute is called atomic. For example, assume an application attempts to update three tables within a transaction. A failure during the update of the third table would undo all updates to the first and second table within the transaction. This atomic attribute ensures that all dependent operations are completed in full. A transaction is also known as a Logical Unit of Work (LUW)

As an implementation of the Java Platform, Enterprise Edition specification, WebSphere Application Server supports both local and global transactions and can be either a transaction manager or a resource (manager) within a transaction. WebSphere Application Server is a transaction manager and supports the following types of transactions

- Resource Manager Local Transactions
- Local Transactions
- Global Transactions

Information on transaction support in WebSphere Application Server can be found at

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/cjta_trans.html

On distributed platforms, the transaction service in WebSphere Application Server handles both local and global transactions. If WebSphere Application Server on z/OS and the applications accesses DB2 by using JDBC type 2 connectivity, the transaction services in WebSphere Application Server defer to the z/OS Recoverable Resource Services (RRS) to handle local and global transactions. If JDBC type 4 access to DB2 is used, RRS is not used and the transaction service in WebSphere Application Server handles both global and local transaction

2.7 WebSphere Application Server - DB2 high availability configuration options

High availability is also known as resiliency. High availability is the ability of a system to tolerate a number of failures and remain operational. It is achieved by adding redundancy in the infrastructure to support failures. It is critical that your infrastructure continues to respond to client requests regardless of the circumstances and that you remove any single points of failure. Planning for a highly available system takes planning across all components of your infrastructure because the overall infrastructure is only available when all of the components are available. As part of the planning, you must define the level of high availability that is needed in the infrastructure.

2.7.1 WebSphere Application Server - DB2 for z/OS recommended high availability configuration when using JDBC type 4 connectivity

WebSphere Application Server Network Deployment configuration (on all platforms) is a recommended to be set up for high availability and scalability. We can have as many nodes as required in a single cell, to meet availability and scalability requirements.

Table 2-1 lists the implementation steps and provides a link to where the steps are described in this book.

Table 2-1 List of sections describing the JDBC type 4 implementation steps

JDBC type 2 - Implementation step number and definition	Cross-reference to where described
	WebSphere
1. Build at a minimum a WebSphere Application Server Network Deployment configuration spread across two nodes/LPARs following the preferred practice information.	5.1, "Configuring WebSphere Application Server Network Deployment on z/OS" on page 208
2. Build a DB2 for z/OS data sharing environment with at least two members that are spread across two LPARS on z/OS.	
3. Configure either a DB2 Universal JDBC Provider or DB2 Universal JDBC Provider (XA) depending on transaction requirements. Use DB2 Universal JDBC Provider (XA) if the application requires global transaction support using JDBC type 4 connection only.	5.2.1, "Defining a DB2 JDBC XA provider" on page 210
4. During the definition of the data source, make sure to provide the group DVIPA address for the server name property. This is important for high availability and scalability.	5.2.3, "Defining a JDBC type 4 XA data source" on page 218
5. Configure WebSphere Application Server data source connection pool properties depending on application requirements.	2.6.4, "WebSphere Application Server connection pooling" on page 51 5.8, "Configuring connection pool sizes on data sources in WebSphere Application Server" on page 273
6. Enable sysplex workload balancing.	2.6.4, "WebSphere Application Server connection pooling" on page 51 5.4, "Configuring WebSphere Application Server for sysplex workload balancing" on page 243
7. Use the high performance DBAT features available in DB2 10. Bind the JDBC packages to a different collection name with the RELEASE(DEALLOCATE) option. Configure the data source to use this collection. If the application uses SQLJ or pureQuery and uses static SQL, remember to bind those packages as well with the RELEASE(DEALLOCATE) and provide those collection names as well in the data source custom property.	5.11.2, "currentPackagePath" on page 292

JDBC type 2 - Implementation step number and definition	Cross-reference to where described
	WebSphere
8. Set client accounting information on the data source custom properties as shown at a minimum. This helps identify connection on which the SQL statements come into DB2.	5.5.1, "Setting client information on a data source" on page 247
9. Define trusted context and roles in DB2. Define only connection privilege to the user ID that is specified on the data source. Define the required privileges to the role in DB2.	2.6.7, "Trusted context support in WebSphere Application Server" on page 62 5.9, "Enabling trusted context for applications that are deployed in WebSphere Application Server" on page 276
10. Set up a profile table in DB2 (if using DB2 10) to monitor and control connections coming into DB2 from WebSphere Application Server.	Start at 4.3.17, "Using DB2 profiles" on page 180
11. Set the appropriate isolation level	5.11.1, "webSphereDefaultIsolationLevel" on page 288
12. Configure prepared statement cache in WebSphere Application Server	5.6, "Configuring the prepared statement cache in WebSphere Application Server" on page 268

For details, see Chapter 4, "DB2 infrastructure setup" on page 99.

By doing all the following steps above, one gets the following capabilities

- High availability
- Scalability
- Workload balancing
- Ability to track which SQL is coming from which application
- Ability to better classify individual application workload to WLM on z/OS
- Security, as the data sources user ID id cannot be misused

We set up the environment as recommended above and validated the following items:

- Trusted Context
- Sysplex Workload Balancing
- Client connection Strings
- High Availability

Figure 2-34 on page 67 represents the HA configuration that we built for JDBC type 4 connectivity following the recommendation above.

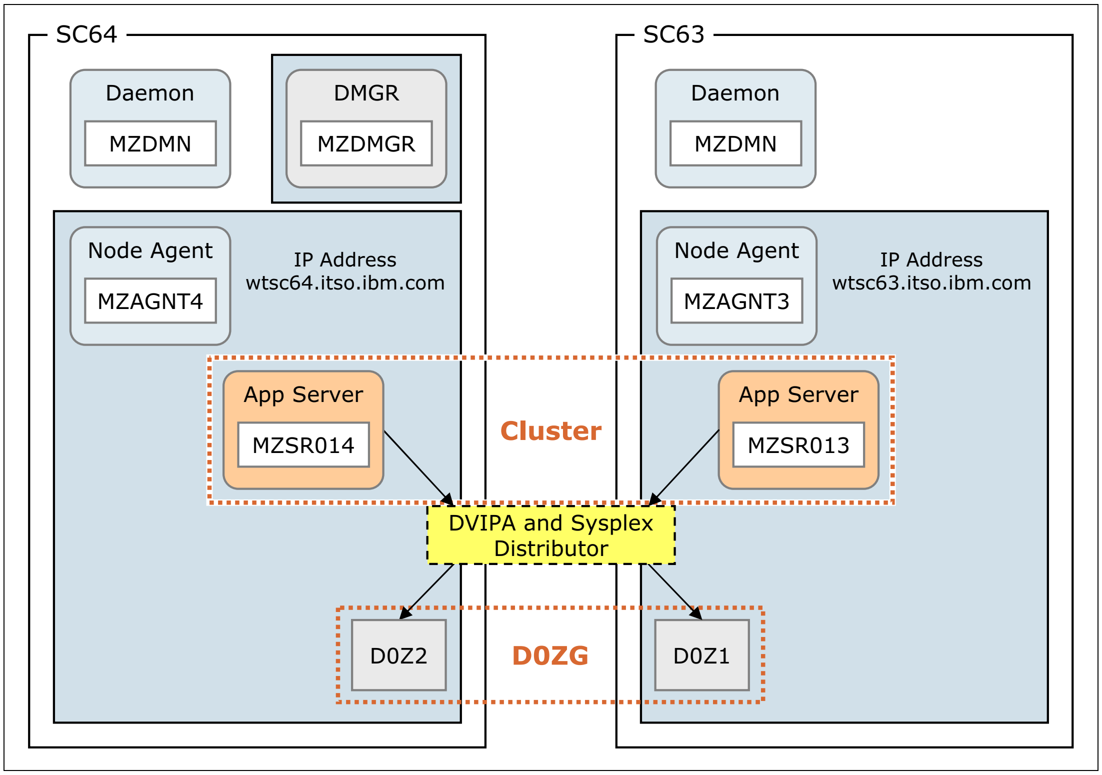


Figure 2-34 High availability configuration for JDBC type 4 connectivity

Validating high availability and sysplex workload balancing

We used the dayTradeEE6 application for this scenario. We configured the data source to be a JDBC type 4 data source. We set the client accounting information. We enabled sysplex workload balancing. We set the maximum connections in the connection pool properties in WebSphere Application Server for the JDBC type 4 data source to 50. We used workload simulator to start the workload and started 50 concurrent clients.

It is difficult to show that workload is truly balanced. We captured the output from the **DISPLAY DDF DETAIL** command in both the DB2 members. We saw the workload was distributed between both the data sharing members. See Example 2-1.

The following are our observations:

- The difference between ADBAT and DSCDBAT tells you how many threads are active currently in the DB2 subsystem. For DB0Z2, we see it is 20 - 9, which is 11.
- We see that the weights returned by WLM are the same for both the members.

Example 2-1 DISPLAY DDF DETAIL output

```
DSNL101I WT IPADDR            IPADDR
DSNL102I 32 ::9.12.4.142
DSNL102I 32 ::9.12.4.138

DSNL080I  -D0Z2 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION          LUNAME            GENERICLU
DSNL083I DB0Z              -NONE             -NONE
DSNL084I TCPPORT=39000 SECPORT=0     RESPORT=39003 IPNAME=IPDB0Z
DSNL085I IPADDR=::9.12.4.153
```

```
DSNL086I SQL    DOMAIN=d0zg.itso.ibm.com
DSNL086I RESYNC DOMAIN=d0z2.itso.ibm.com
DSNL089I MEMBER IPADDR=::9.12.4.142
DSNL090I DT=I  CONDBAT=  10000 MDBAT=  200
DSNL092I ADBAT=   20 QUEDBAT=      0 INADBAT=      0 CONQUED=      0
DSNL093I DSCDBAT=      9 INACONN=     39
DSNL100I LOCATION SERVER LIST:
DSNL101I WT IPADDR            IPADDR
DSNL102I 32 ::9.12.4.142
DSNL102I 32 ::9.12.4.138
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

- We had set the max connections in the data source connection pool property to be 50. We started a workload that had 50 clients. The JDBC type 4 driver opens 50 transports to each data sharing member. The **DISPLAY LOCATION** report in Example 2-2 shows you how many transports have been created. We can see that for the member D0Z2 we created 50 connections from a client at location 9.12.4.142. All 50 connections are workload balanced as shown by workload balancing. It also shows that all the 50 connections were coming from an XA driver.

Example 2-2 DISPLAY LOCATION report

```
DSNL200I  -D0Z2 DISPLAY LOCATION REPORT FOLLOWS-
LOCATION                                       PRDID    T ATT CONNS
::9.12.4.142                                   JCC03640 S        50
                                                          WLB    50
                                                          XA     50
::9.12.6.9                                     JCC03640 S         0
DISPLAY LOCATION REPORT COMPLETE
```

- We looked at the **DISPLAY DDF** output of Example 2-3 to validate the thread information.
  - The difference between ADBAT and DSCDBAT tells you how many threads are active currently in the DB2 subsystem. For DB0Z1, we see it is 29 -23, which is 6.
  - We see that the weights (WT) returned by WLM is almost the same for both the members.

Example 2-3 DISPLAY DDF report

```
DSNL101I WT IPADDR            IPADDR
DSNL102I 33 ::9.12.4.138
DSNL102I 31 ::9.12.4.142

DSNL080I  -D0Z1 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I DB0Z               -NONE             -NONE
DSNL084I TCPPORT=39000 SECPORT=0     RESPORT=39002 IPNAME=IPDB0Z
DSNL085I IPADDR=::9.12.4.153
DSNL086I SQL    DOMAIN=d0zg.itso.ibm.com
DSNL086I RESYNC DOMAIN=d0z1.itso.ibm.com
DSNL089I MEMBER IPADDR=::9.12.4.138
DSNL090I DT=I  CONDBAT=  10000 MDBAT=  200
DSNL092I ADBAT=   29 QUEDBAT=      0 INADBAT=      0 CONQUED=      0
DSNL093I DSCDBAT=     23 INACONN=     44
```

```
DSNL100I LOCATION SERVER LIST:
DSNL101I WT IPADDR           IPADDR
DSNL102I 33 ::9.12.4.138
DSNL102I 31 ::9.12.4.142
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

- We had set the max connections in the data source connection pool property to be 50. We started a workload that had 50 clients. The JDBC type 4 driver opens 50 transports to each data sharing member. The DISPLAY LOCATION report in Example 2-4 shows you how many transports have been created. We can see that for the member D0Z1 we created 50 connections from a client at location 9.12.4.142. Out of this 50 all 50 are workload balanced as shown by WLB. It also shows that all the 50 connections were coming from an XA driver.

Example 2-4 DISPLAY LOCATION report

```
DSNL200I  -D0Z1 DISPLAY LOCATION REPORT FOLLOWS-
LOCATION                                       PRDID    T ATT CONNS
::9.12.6.9                                     JCC03640 S         50
                                                          WLB     50
                                                          XA      50
::9.30.28.118                                  JCC04130 S          0
DISPLAY LOCATION REPORT COMPLETE
```

We then brought down D0Z2.

We saw the text in Example 2-5 in the WebSphere Application Server log. The JDBC driver automatic client reroute feature kicks in and it follows the behavior described earlier. We get an SQL code of -30108. This tells us that the current transaction failed and the application has the option to retry the logic (if the application was written to do so). Our DayTrader application was not written to handle the -30108 error code and hence some transactions failed, but the workload continued successfully to the other member.

Example 2-5 WebSphere Application Server log

```
Trace: 2012/08/10 20:52:57.176 02 t=7C0268 c=UNK key=P8 tag= (13007004)
   SourceId: com.ibm.ejs.j2c.ConnectionEventListener
   ExtendedMessage: J2CA0056I: The Connection Manager received a fatal connection
error from the Resource Adapter for resource jdbc/T radeDataSource.
The exception is: com.ibm.db2.jcc.am.ClientRerouteException:
[jcc][t4][20142][11212][3.64.97] A connection failed but has been re-established.
The host name or IP address is "d0z1.itso.ibm.com" and the service name or port
number is 39,000. ERRORCODE=-30108, SQLSTATE=08506
```

Validating trusted context

We used a simple web service app to validate trusted context. The application name is D0ZG_WASTestClientInfo. We secured it to use basic form authentication. This application issues uses a data source. We configured it to be a JDBC type 4 data source following the best practice configuration. We then installed the application on the server. We then enabled the application to use a trusted connection. We set the client application information to dwsClientinformationDS. In DB2 we created a trusted context. Then we ran the application. It prompted us for a user ID. We used “wastest”. We then captured the output from a **-DIS THREAD(*) SCOPE(GROUP)** command.

Example 2-6 shows the output from the command.

Example 2-6 -DIS THREAD(*) output

```
DSNV473I  -DOZ2 ACTIVE THREADS FOUND FOR MEMBER: DOZ1
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
SERVER   SW *    16 db2jcc_appli RAJESH   DISTSERV 00FB 11240
 V485-TRUSTED CONTEXT=CTXWASTESTT4,
     SYSTEM AUTHID=WASTEST,
     ROLE=WASTESTDEFAULTROLE
 V437-WORKSTATION=WTSC64, USERID=wastest,
      APPLICATION NAME=dwsClientinformationDS
 V429 CALLING FUNCTION=DB2R3.GRACFGRP,
      PROC=        , ASID=0000, WLM_ENV=DSNWLMDB0Z_GENERAL
```

For details about setting up trusted context, see Chapter 4, “DB2 infrastructure setup” on page 99.

2.7.2 WebSphere Application Server - DB2 z/OS recommended high availability configuration when using JDBC type 2 connectivity

There are three possible HA configurations when using WebSphere Application Server on z/OS and using JDBC type 2 connectivity to DB2 for z/OS:

- Configuring multiple members of the DB2 data sharing group on each LPAR
- Exploiting the Resource Adapter Failover feature
- Using new failure custom properties

Configuring multiple members of the DB2 data sharing group on each LPAR

You can have multiple members of the DB2 data sharing group on each LPAR. After you have multiple members of the DB2 data sharing group in the same LPAR, and the data source custom property ssid is configured with the group attach name, then the IBM Data Server Driver for JDBC and SQLJ automatically fails over to the second member if one member goes down.

It is important to note that it will not fail back after the original failed member comes back up again. It is also important to note that there is no workload balancing between two DB2 members on the same LPAR when we use JDBC type 2 connectivity. It is also important to note that the IBM Data Server Driver for JDBC and SQLJ picks one member randomly and does not have any special algorithm available.

Exploiting the Resource Adapter Failover feature

You can configure WebSphere Application Server to exploit the Resource Adapter Failover feature. This is new since WebSphere Application Server V8.

Figure 2-35 represents what is a common highly available clustered WebSphere Application Server and JDBC type 2 connectivity to DB2 for z/OS.

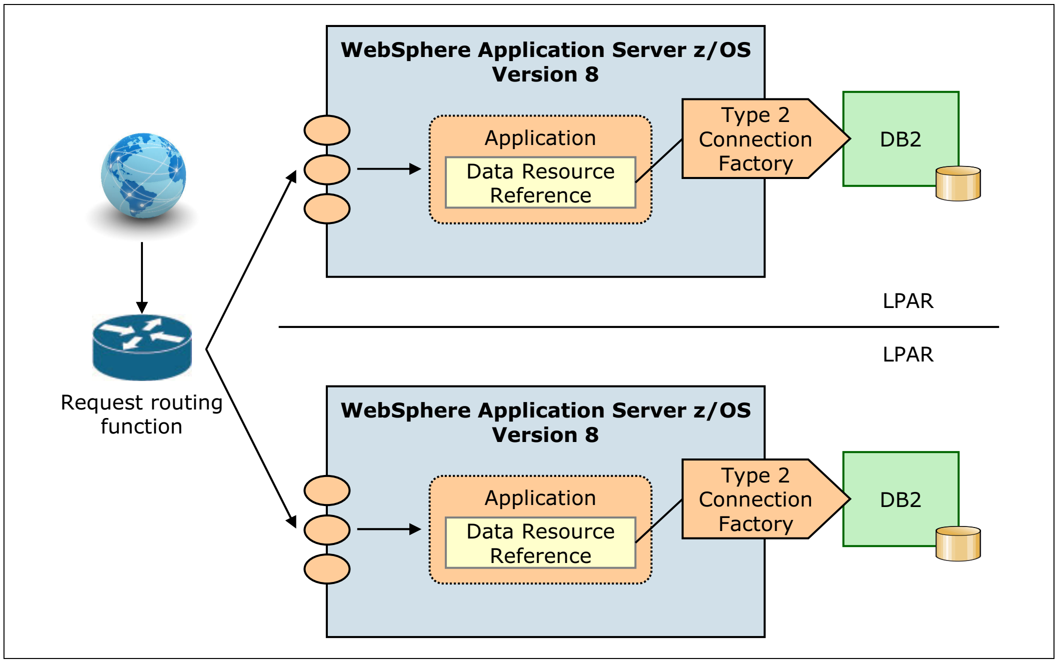


Figure 2-35 Highly available WebSphere Application Server with and JDBC type 2

When one of the DB2 member fails as shown in Figure 2-36, there is a potential outage as the front end router does not know that DB2 is down.

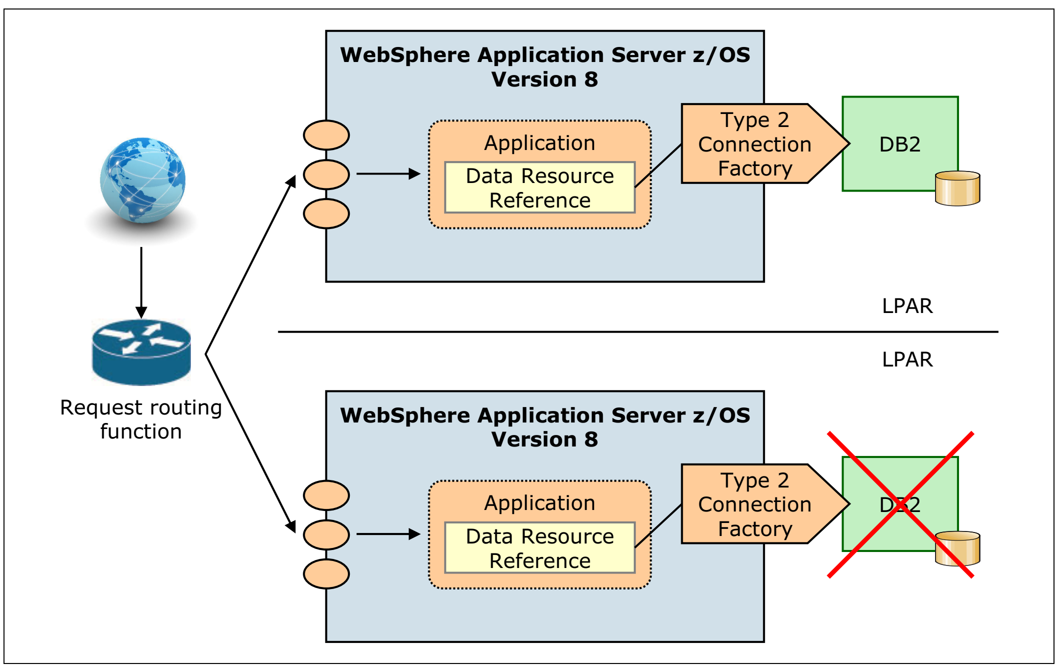


Figure 2-36 DB2 failure not notified to router

The solution is to configure what is called an alternate JDBC type 4 data source. WebSphere Application Server is smart enough to know that DB2 is down and will start to use the type 4 connection to the second DB2 member of the same data sharing group in the other LPAR

Figure 2-37 shows the alternate JDBC type 4 data source configured.

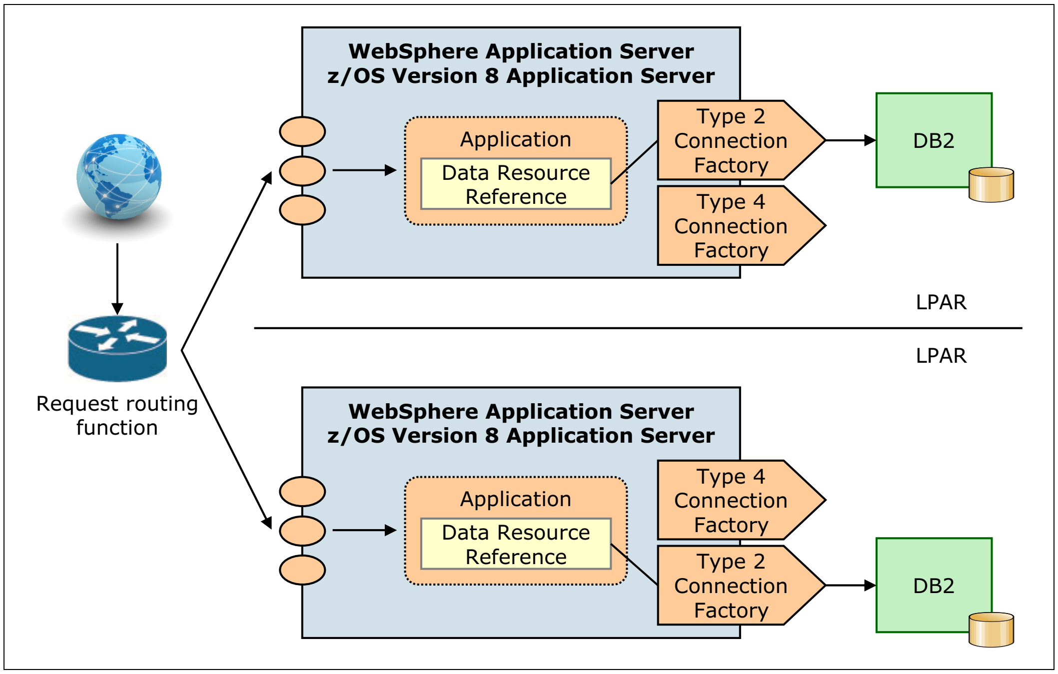


Figure 2-37 Alternate JDBC type 4 data source configuration

Then Figure 2-38 shows how, when the DB2 member goes down, existing connections and transactions fail but new connections use the alternate JDBC type 4 connection to available DB2 data sharing members and work is not affected.

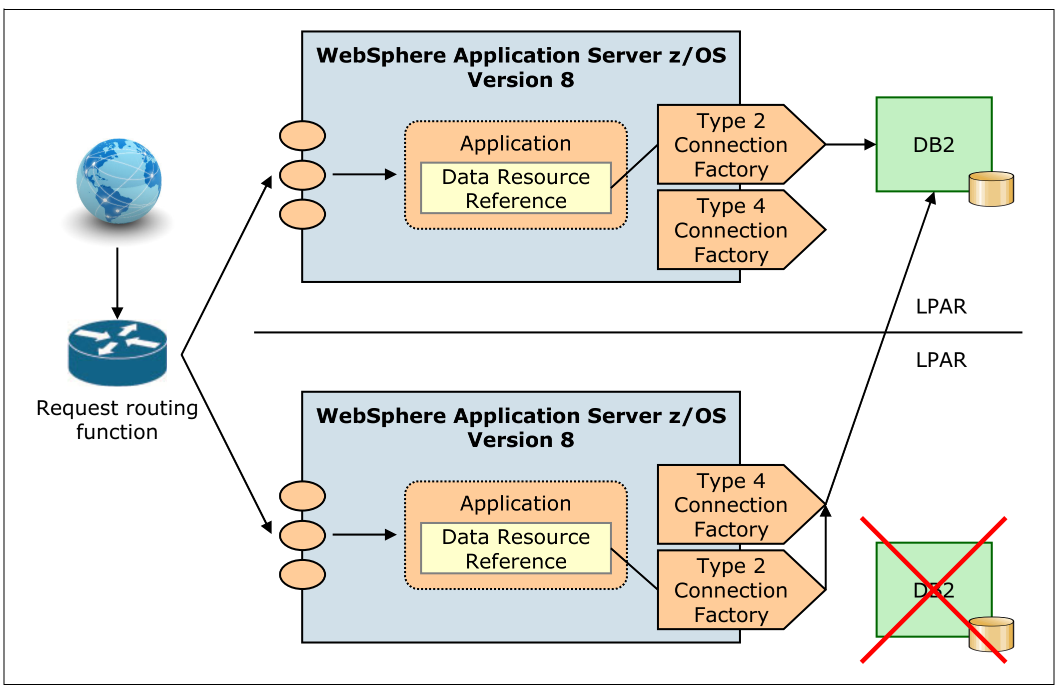


Figure 2-38 Alternate JDBC type 4 connection used to surviving DB2 member

Now when the failed DB2 member is brought up, WebSphere Application Server is smart enough to know that DB2 is back up, and starts using the JDBC type 2 connection. In this case, it is smart that it does not fail any existing transactions on the JDBC type 4 connection, instead it quiesces the current work and starts to using the JDBC type 2 connection for new work. There is a custom property resourceAvailabilityTestRetryInterval that can be configured to tell WebSphere Application Server how often to check if the failed DB2 member is up. See Figure 2-39.

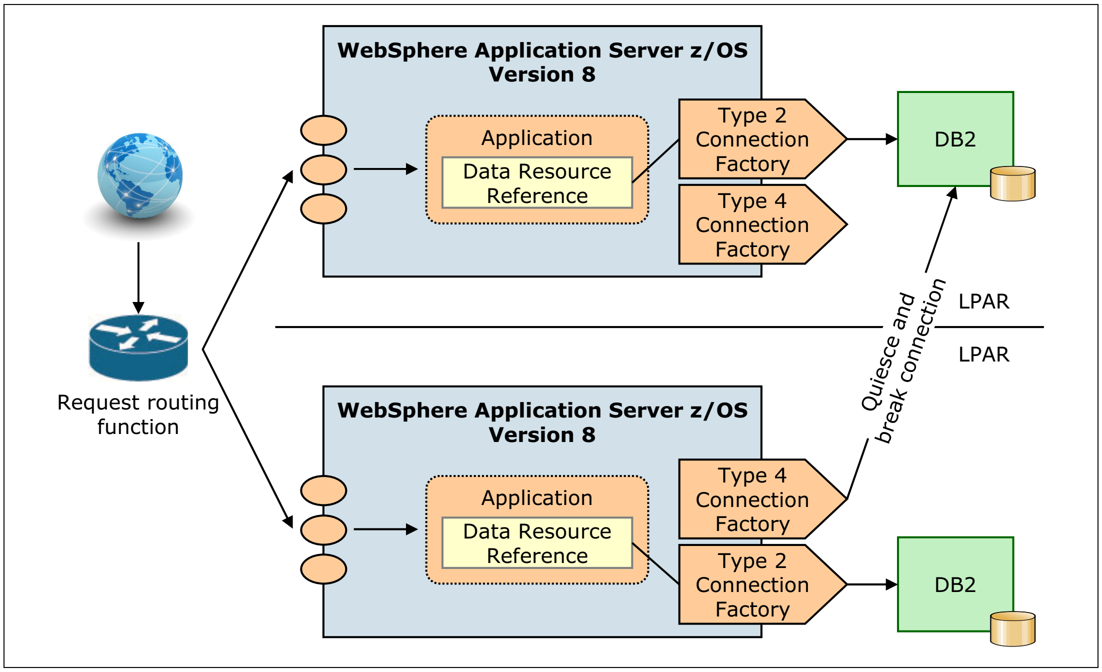


Figure 2-39 Reactivating the type 2 connection

Detailed step by step guidance can be found in the document at this url:

http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102033

Using new failure custom properties

Starting with WebSphere Application Server V8, a couple of custom properties were added to the data source connection pool custom properties. These are the failureNotificationActionCode and failureThreshold. The failure notification is option must be carefully evaluated. If the failureNotificationActionCode is set to 1, then a BBOJ0130I is issued to the operator log. Then using automation, appropriate recovery action can be taken.

Figure 2-40 shows the failure of a member causing the message notification when the failureNotificationActionCode is set to 1.

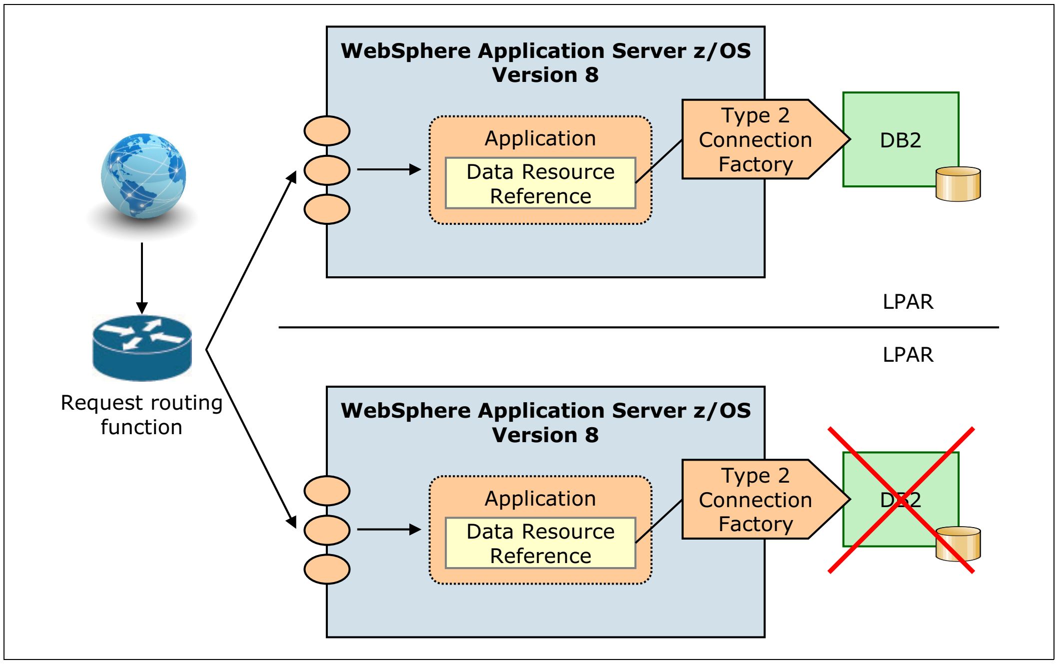


Figure 2-40 failureNotificationActionCode is set to 1

If the failureNotificationActionCode is set to 2, then an automatic pause listener command is issued. This command closes the ports on the server and the front end router is smart enough to know this and does not route work to this server. This means all the applications in this server are not available for work. This option is best if all the applications in that server require access to DB2. Figure 2-41 shows what happens when the failureNotificationActionCode is set to 2.

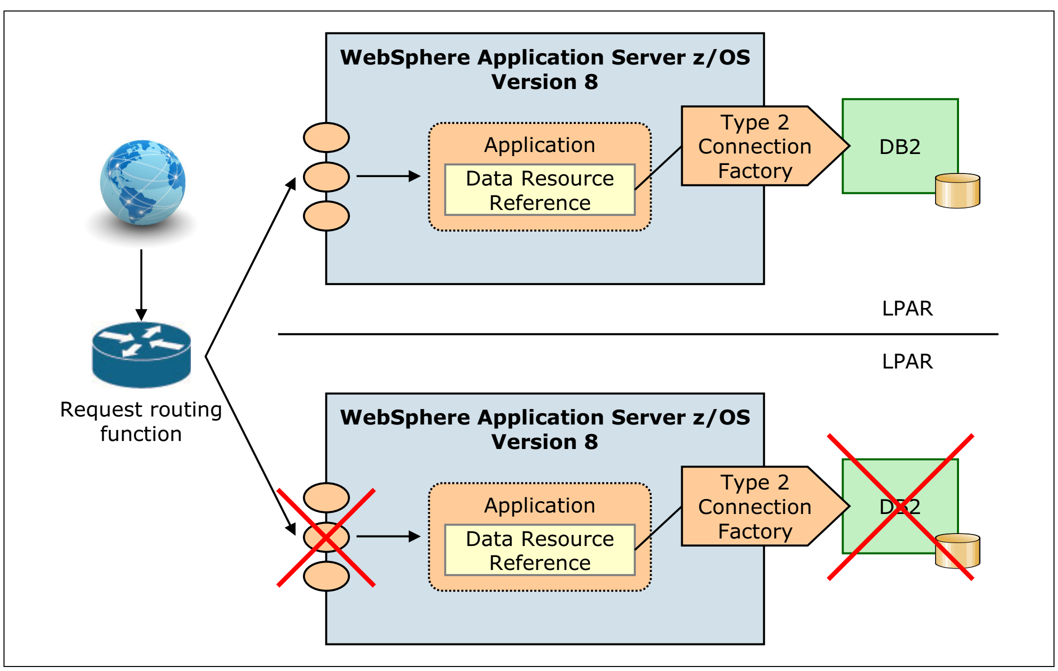


Figure 2-41 failureNotificationActionCode is set to 2

If the failureNotificationActionCode is set to 3, then WebSphere Application Server stops all applications that access that specific DB2 for z/OS. This means that all other applications which do not access DB2 for z/OS are available.

A highly intelligent front end router such as the On Demand Router (a WebSphere Application Server feature available in V8.5) is needed to recognize that the application is stopped and stop routing work to that server. Normal HTTP servers are not smart enough to know applications are stopped in the servers, they only know if a server is stopped. Figure 2-42 shows what happens when the failureNotificationActionCode is set to 3.

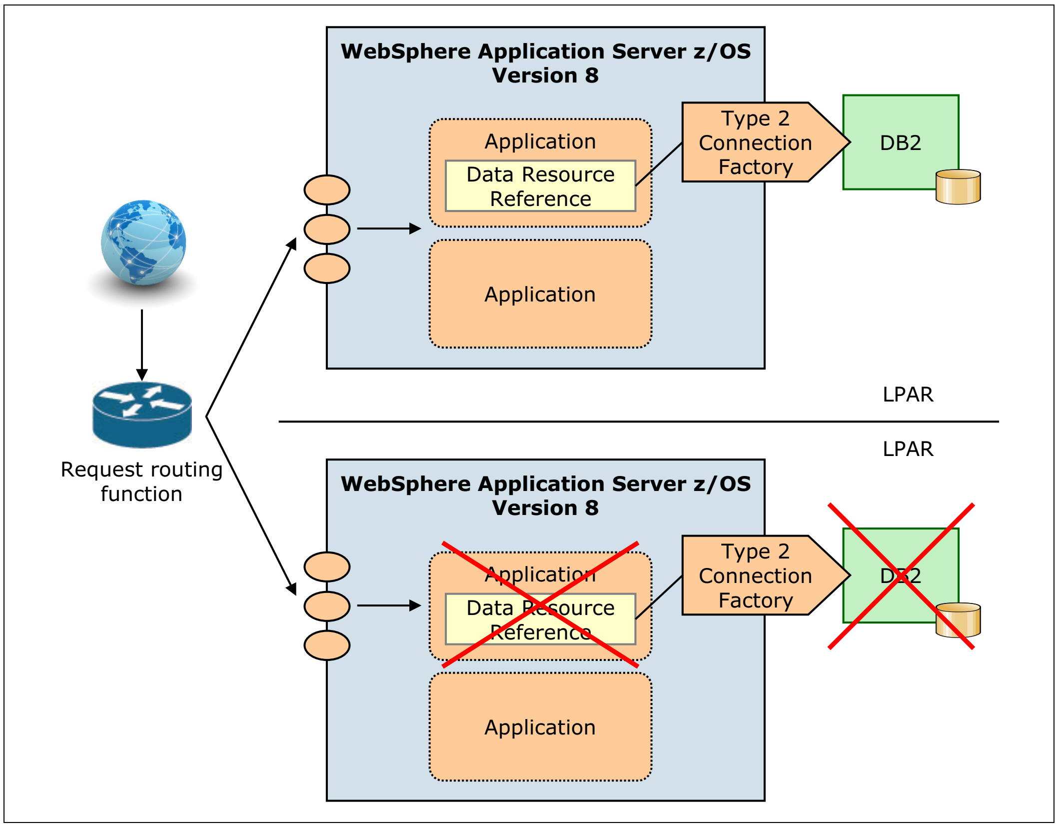


Figure 2-42 failureNotificationActionCode is set to 3

These factors must be taken into consideration before selecting this options.

Information about these properties can be found in the WebSphere Application Server information center at

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.multiplatform.doc%2Fae%2Frdat_conpoolcustprops.html

All three options are viable and depend on each customers environment and requirement. Hence it is difficult to really pick one as the recommendation.

Best practices to build a HA environment for JDBC type 2 connections

In general the following are best practices on how to build a HA environment.

Table 2-2 lists the implementation steps and provides a link to where the steps are described in this book.

Table 2-2 List of sections describing the JDBC type 2 implementation steps

JDBC type 2 - Implementation step number and definition	Cross reference to where described
	WebSphere
1. WebSphere Application Server Network Deployment configuration (on all platforms) is a recommended to be set up for high availability and scalability. We can have as many nodes as required in a single cell, to meet availability and scalability requirements. Build at a minimum a WebSphere Application Server Network Deployment configuration spread across 2 nodes/LPARs following the best practice information found in the following.	5.1, "Configuring WebSphere Application Server Network Deployment on z/OS" on page 208
2. Build a DB2 for z/OS data sharing environment with at least two members spread across 2 LPARS on z/OS.	
3. Configure a DB2 Universal JDBC Provider.	5.3.1, "Defining a DB2 JDBC provider" on page 223
4. Define a data source.	5.3.3, "Defining a JDBC type 2 data source" on page 233
5. Define the ssid custom property. Make sure to give the group attach name as the value.	5.3.4, "Configuring a subsystem ID on the data source" on page 238
6. Configure WebSphere Application Server data source connection pool properties depending on application requirements.	5.8, "Configuring connection pool sizes on data sources in WebSphere Application Server" on page 273
7. Configure WebSphere Application Server data source prepared statement cache size depending on application requirements.	5.6, "Configuring the prepared statement cache in WebSphere Application Server" on page 268
8. Set client accounting information on the data source custom properties as shown at a minimum. This will help identify connection on which the SQL statements come into DB2.	5.1, "Configuring WebSphere Application Server Network Deployment on z/OS" on page 208
9. Define trusted context and roles in DB2. Define only connection privilege to the user ID that is specified on the data source. Define the required privileges to the role in DB2.	5.9, "Enabling trusted context for applications that are deployed in WebSphere Application Server" on page 276
10. Set the appropriate isolation level	5.11.1, "websphereDefaultIsolationLevel" on page 288

By doing all the steps above, you get the following capabilities

- High availability
- Scalability
- Ability to track which SQL is coming from which application
- Ability to better classify individual application workload to WLM on z/OS
- Security, as the data source user ID id cannot be misused

We set up the environment as recommended above and validated the following:

- Validating fail over capability
- Validating trusted context

The following validation sections used the HA configuration we built for JDBC type 2 connectivity following the recommendation above. *Because we had only two data sharing members, we brought up both on the same LPAR to validate fail over.*

Validating fail over capability

We used the dayTradeEE6 application for this scenario. We configured the data source to be a JDBC type 2 data source. We set the ssid to use the group attach name as recommended. We set the client accounting information. We enabled sysplex workload balancing. We set the max connections in the connection pool properties in WebSphere Application Server for the JDBC type 2 data source to 50. Because we had only 2 data sharing members, we brought up both on the same LPAR to validate fail over. We used workload simulator to start the workload and started 50 concurrent clients.

We issued the **-DISPLAY THREAD(*) SCOPE(GROUP)** command when we started the workload. We noticed from the output listed in Example 2-7 that all JDBC type 2 connections went to a single member (D0Z2) as expected. D0Z1 did not have any connections from the DayTrader application.

Example 2-7 -DISPLAY THREAD(*) SCOPE(GROUP) output

```
DSNV401I  -D0Z1 DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -D0Z1 ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
RRSAF    T      115 D0Z1ADMT_DMN D0Z1ADMT ?RRSAF   00AD     2
 V437-WORKSTATION=RRSAF, USERID=D0Z1ADMT,
      APPLICATION NAME=D0Z1ADMT_DMN
RRSAF    T        5 D0Z1ADMT_II  D0Z1ADMT ?RRSAF   00AD     6
 V437-WORKSTATION=RRSAF, USERID=D0Z1ADMT,
      APPLICATION NAME=D0Z1ADMT_II
TSO      T  *     3 RAJESH       RAJESH            009D    38
 V437-WORKSTATION=TSO, USERID=RAJESH,
      APPLICATION NAME=RAJESH
TSO      T      970 DB2R3        DB2R3    ADB      00B0    17
 V437-WORKSTATION=TSO, USERID=DB2R3,
      APPLICATION NAME=DB2R3
DISPLAY ACTIVE REPORT COMPLETE
DSNV473I  -D0Z1 ACTIVE THREADS FOUND FOR MEMBER: D0Z2
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
DISCONN  DA *    28 NONE         NONE     DISTSERV 00A2    14
 V471-IPDB0Z.P85B.CA061AE373AA=14
RRSAF    TD   18005 MZSR014S     RAJESH   ?RRSAF   00B2    27
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
      APPLICATION NAME=TraderClientApplication
***
```

```
RRSAF    T  * 23674 MZSR014S     RAJESH   ?RRSAF   00B2    28
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
      APPLICATION NAME=TraderClientApplication
RRSAF    T  *  2659 MZSR014S     RAJESH   ?RRSAF   00B2    29
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
      APPLICATION NAME=TraderClientApplication
RRSAF    TD    3359 MZSR014S     RAJESH   ?RRSAF   00B2    30
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
      APPLICATION NAME=TraderClientApplication
RRSAF    TD    9842 MZSR014S     RAJESH   ?RRSAF   00B2    31
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
      APPLICATION NAME=TraderClientApplication
RRSAF    T  * 14650 MZSR014S     RAJESH   ?RRSAF   00B2    32
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
      APPLICATION NAME=TraderClientApplication
RRSAF    TD    5621 MZSR014S     RAJESH   ?RRSAF   00B2    33
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
      APPLICATION NAME=TraderClientApplication
RRSAF    TD   26928 MZSR014S     RAJESH   ?RRSAF   00B2    34
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
      APPLICATION NAME=TraderClientApplication
RRSAF    TD    2153 MZSR014S     RAJESH   ?RRSAF   00B2    35
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
***
```

We then brought down D0Z2. We then validated the fail over by issuing the **-DISPLAY THREAD(*) SCOPE(GROUP)** command again. In Example 2-8 we see that all the connections are now to member D0Z1.

Example 2-8 Validating the fail over with -DISPLAY THREAD(*) SCOPE(GROUP)

```
DSNV401I  -D0Z1 DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -D0Z1 ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
RRSAF    T  *  4888 MZSR014S     RAJESH   ?RRSAF   00B2    45
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
      APPLICATION NAME=TraderClientApplication
RRSAF    T  *  6539 MZSR014S     RAJESH   ?RRSAF   00B2    46
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
      APPLICATION NAME=TraderClientApplication
RRSAF    T  * 10223 MZSR014S     RAJESH   ?RRSAF   00B2    47
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
      APPLICATION NAME=TraderClientApplication
RRSAF    T  *  6274 MZSR014S     RAJESH   ?RRSAF   00B2    48
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
      APPLICATION NAME=TraderClientApplication
RRSAF    T  *  9043 MZSR014S     RAJESH   ?RRSAF   00B2    49
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
      APPLICATION NAME=TraderClientApplication
RRSAF    T  *  6176 MZSR014S     RAJESH   ?RRSAF   00B2    50
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
      APPLICATION NAME=TraderClientApplication
RRSAF    T  *  8188 MZSR014S     RAJESH   ?RRSAF   00B2    51
 V437-WORKSTATION=TraderClientWorkst, USERID=TraderClientUser,
***
```

Validating trusted context

We used the dayTraderEE6 application to validate trusted context. We configured it to be a JDBC type 2 data source following the best practice configuration. The JAAS alias we used had a user ID MZADMIN. This user ID had only connect privileges to DB2. It had a DB2 Role assigned to it that gave it the required privileges to access the application tables. We set the client application information to TraderClientApplication1.

In DB2 we created a trusted context. Then we ran the application. It prompted us for a user ID. We used "wastest". We then captured the output from a **`-DIS THREAD(*)`** command. Example 2-9 shows the output from the command.

Example 2-9 -DIS THREAD(*) output

```
RRSAF    TD   20335 MZSR014S     MZADMIN  ?RRSAF   00CF  6549
 V485-TRUSTED CONTEXT=CTXDTRADET2,
      SYSTEM AUTHID=MZADMIN,
      ROLE=DTRADEROLE
  V437-WORKSTATION=TraderClientWorkst, USERID=1TraderClientUse,
       APPLICATION NAME=TraderClientApplication1
```

3

DB2 configuration options for Java client applications

In this chapter we provide background information about the configuration options available in DB2 for z/OS when developing Java applications. We describe the connections available when accessing a data sharing group, the system parameters available to control the network, the main Java drivers option, and the suggested configuration options for high availability.

This chapter covers the following topics:

- The DB2 configuration
- IBM Data Server Drivers and Clients
- High availability configuration options

3.1 The DB2 configuration

A Sysplex is a set of z/OS systems that communicate and cooperate with each other through specialized hardware components and software services. A collection of one or more DB2 subsystems that share DB2 data is called a data sharing group. With data sharing, applications that run on more than one DB2 for z/OS subsystem can read from and write to the same set of data concurrently.

The data sharing group uses coupling facilities as hardware assist for efficient concurrency and coherency control. One or more coupling facilities provide high-speed caching and lock processing for the data sharing group. The Sysplex, together with the Workload Manager (WLM), dynamic virtual IP address (DVIPA), and the Sysplex Distributor, allow a client to access a DB2 for z/OS database over TCP/IP with network resilience, and distribute the work among the DB2 subsystems within the data sharing group.

Figure 3-1 shows the possible connections to a data sharing group.

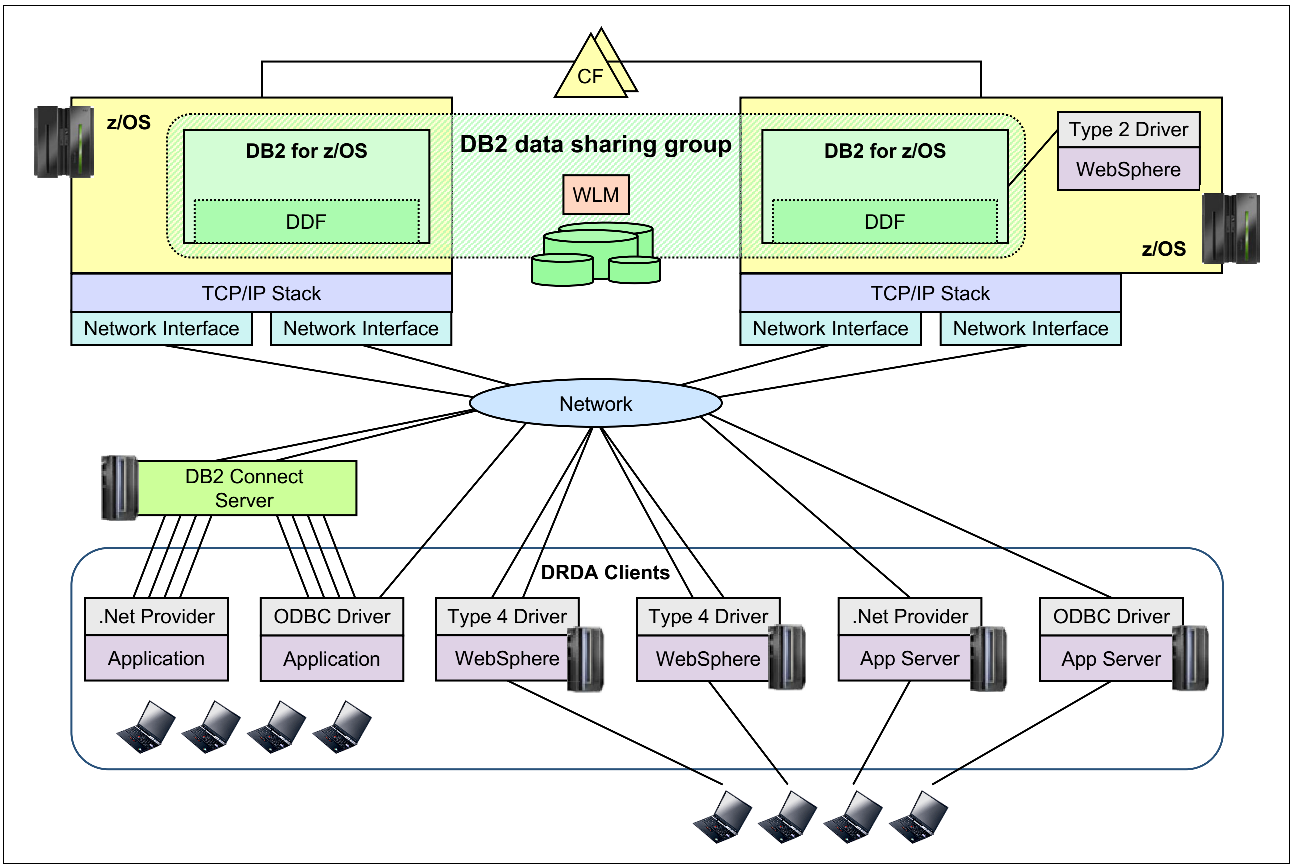


Figure 3-1 Connectivity and data sharing

This section provides recommendations for configuring the TCP/IP network and the DB2 subsystems.

3.1.1 Configuring the TCP/IP network

DB2 requires that all members of a data sharing group use the same port number to receive incoming SQL requests. The well-known DB2 registered port 446 is the recommended DRDA port using for SQL processing. Additionally, DB2 requires that each member of a data sharing group has a resynchronization port number that is unique within the Parallel Sysplex. The resync port is used by a requester in two situations. One is when the SQL connection fails leaving in-doubt threads, and the requester and server need to resynchronize after the error. The other one is for other connections used to interrupt SQL processing on a different application connection. Obviously, resynchronization needs to occur with the specific DB2 member with which the requester was in session, so this member must be reachable through a specific IP address (the member-specific DVIPA in this case).

In Figure 3-2 on page 84, there are three DB2 members DB2A, DB2B, and DB2C in the data sharing group with the group location named DB2LOC. These resync addresses are represented by ports 5001, 5002, and 5003, for the three DB2 members DB2A, DB2B, and DB2C, respectively.

Example 3-1 on page 84 first shows to register the well-known DRDA port 446 and a unique resynchronization port with TCP/IP on each member's z/OS system as shown in the TCP/IP PORT configuration profile statement. On each z/OS system where the DB2 member resides, replicate the TCP/IP PORT configuration profile statement.

Secondly, it shows the VIPADYNAMIC statement to define the group DVIPA for the DB2 data sharing group. The group DVIPA must be defined with the VIPADEFINE and VIPADISTRIBUTE statements on the TCP/IP stacks that are associated with the z/OS systems on which the Sysplex Distributor executes.

The group DVIPA must be defined with the VIPABACKUP statement on the TCP/IP stacks for DVIPA takeover. Note that the VIPABACKUP statements are coded with the MOVEABLE IMMEDIATE keywords, and that the VIPADISTRIBUTE statements are also specified on the backup TCP/IP stacks. This allows for the group DVIPA to be activated on one of the backup stacks if it is not active anywhere else in the Sysplex. For example, if z/OS-1 has not been started when z/OS-2 or z/OS-3 start, then group DVIPA is activated on one of the backup stacks. To allow for failover, the member-specific DVIPAs are defined with the VIPARANGE statement on all TCP/IP stacks.

Example 3-2 on page 85 shows how to use the DSNJU003 utility to define the group location name, the DRDA port, the resync port, the member-specific DVIPA, and the group DVIPA. These changes are applied to the bootstrap data sets (BSDSs).

Figure 3-2 shows three DB2 members and configured resync addresses with unique port numbers for location DB2LOC.

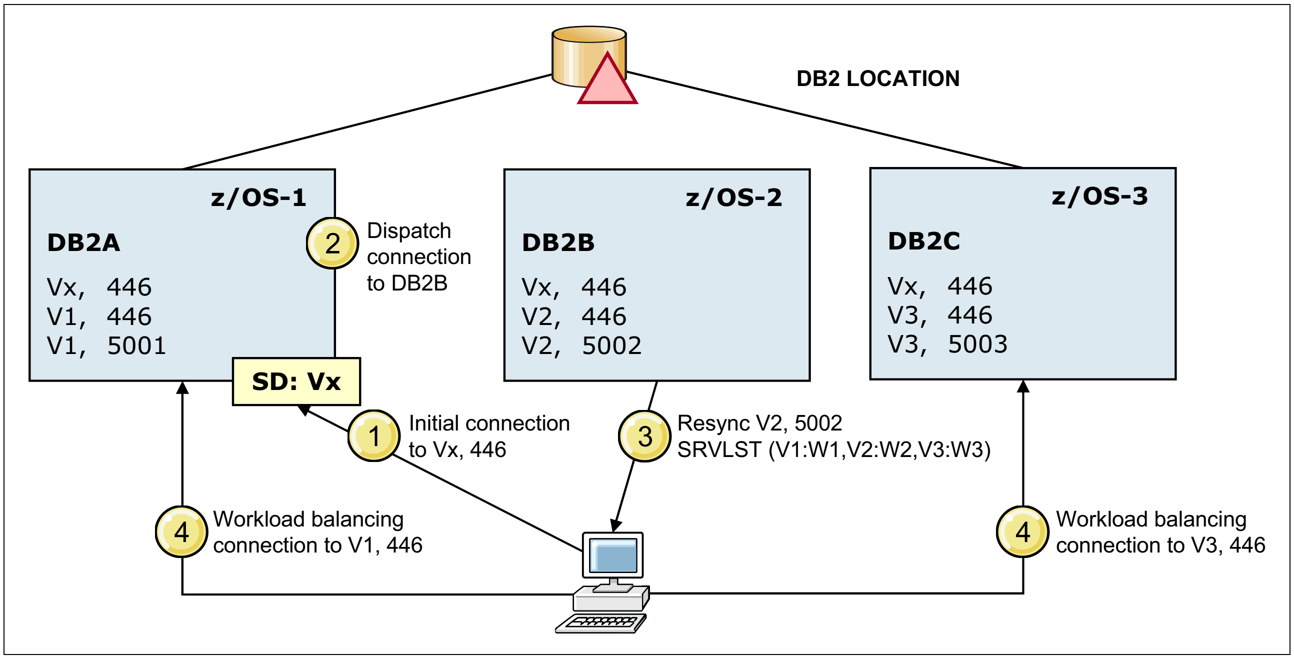


Figure 3-2 DB2 members and configured resync addresses with unique port numbers

This is the network flow as it occurs in Figure 3-2:

1. The initial connection uses the group DVIPA, Vx, on port 446 identified as the DRDA port in each members TCP/IP PORT statement.
2. The Syplex Distributor dispatches the initial connection request to the member with the lightest workload (DB2B in this case).
3. Resynchronization information (5001 for DB2A, 5002 for DB2B, and 5003 for DB2C) and a list of a unique IP address of each member and WLM weight information in the data sharing group are returned to the requester (V1:W1 for DB2A, V2:W2 for DB2B, and V3:W3 for DB2C).
4. Then the subsequent connection attempts are made to the DB2 group by using the member-specific DVIPAs that are returned. In this case the subsequent SQL requests are distributed to DB2A member and DB2C member according to the WLM weight information that DB2A and DB2C have the most apparent capacity at the time.

Example 3-1 Port and VIPA definitions for three DB2 members

```
z/OS-1 TCP/IP configuration setting
PORT
   446 TCP DB2ADIST SHAREPORT
   446 TCP DB2BDIST SHAREPORT
   446 TCP DB2CDIST SHAREPORT
  5001 TCP DB2ADIST
  5002 TCP DB2BDIST
  5003 TCP DB2CDIST
VIPADYNAMIC
```

```
  VIPARANGE 255.255.255.255 V1
  VIPARANGE 255.255.255.255 V2
  VIPARANGE 255.255.255.255 V3
 VIPADEFINE 255.255.255.255 Vx
 VIPADISTRIBUTE DEFINE Vx
               PORT 446
               DESTIP ALL
ENDVIPADYNAMIC

z/OS-2 TCP/IP configuration setting
PORT
   446 TCP DB2ADIST SHAREPORT
   446 TCP DB2BDIST SHAREPORT
   446 TCP DB2CDIST SHAREPORT
  5001 TCP DB2ADIST
  5002 TCP DB2BDIST
  5003 TCP DB2CDIST
VIPADYNAMIC
  VIPARANGE 255.255.255.255 V1
  VIPARANGE 255.255.255.255 V2
  VIPARANGE 255.255.255.255 V3
 VIPABACKUP 1 MOVE IMMED 255.255.255.255 Vx
 VIPADISTRIBUTE DEFINE Vx
               PORT 446
               DESTIP ALL
ENDVIPADYNAMIC

z/OS-3 TCP/IP configuration setting
PORT
   446 TCP DB2ADIST SHAREPORT
   446 TCP DB2BDIST SHAREPORT
   446 TCP DB2CDIST SHAREPORT
  5001 TCP DB2ADIST
  5002 TCP DB2BDIST
  5003 TCP DB2CDIST
VIPADYNAMIC
  VIPARANGE 255.255.255.255 V1
  VIPARANGE 255.255.255.255 V2
  VIPARANGE 255.255.255.255 V3
 VIPABACKUP 2 MOVE IMMED 255.255.255.255 Vx
 VIPADISTRIBUTE DEFINE Vx
               PORT 446
               DESTIP ALL
ENDVIPADYNAMIC
```

Example 3-2 BSDS definition for three DB2 members

```
DB2A
DDF LOCATION=DB2LOC,PORT=446,RESPORT=5001,IPV4=V1,GRPIPV4=Vx
DB2B
DDF LOCATION=DB2LOC,PORT=446,RESPORT=5002,IPV4=V2,GRPIPV4=Vx
DB2C
DDF LOCATION=DB2LOC,PORT=446,RESPORT=5003,IPV4=V3,GRPIPV4=Vx
```

DB2 10 for z/OS provides a feature that enables users to manage and define subsets of members in a data sharing group dynamically, without stopping and restarting DDF or DB2. This is so called dynamic location aliases and this is done by using the MODIFY DDF command. Before you can define the dynamic location aliases, DB2 must be started, but DDF may or may not be started. DB2 10 supports up to 40 dynamic location aliases. You can manage dynamic location aliases by issuing the MODIFY DDF command to stop or cancel the alias, modify its configuration, and restart it, all without stopping DDF or DB2.

3.1.2 Configuring the DB2 subsystems

The following are the general guidelines for setting DB2 installation parameters that impact the utilization of the connections and threads in DB2:

- DSN6FAC CMTSTAT

  Recommends to set to INACTIVE. CMTSTAT=INACTIVE setting enables threads to be pooled after threads successfully commit or roll back a transaction. It then allows the threads to be reused y other connections.

- DSN6SYSP MAXDBAT

  Maximum number of database access threads (DBATs) that can be active concurrently. This value should be set considerably. In many cases, the maximum value is determined by the available storage in DBM1 address space.

- DSN6SYSP CONDBAT

  Maximum number of concurrent inbound connections to DB2. This includes active and inactive connections. This value might be large and should accommodate the number of inactive connections concurrently that would connect to the subsystem at any point.

- DSN6FAC IDTHTOIN

  Maximum amount of time in seconds that an active server thread is allowed to remain idle.The DB2 default 2 minutes is recommended.

- DSN6FAC TCPKPALV (1 to 65534)

  Time to execute the longest SQL statement.

DB2 needs to enable threads to be pooled by setting the CMTSTAT to INACTIVE. An inactive connection uses less storage and frees up DB2 resources associated with the transaction when a thread commits a transaction. When connections are disassociated from the thread, the thread is allowed to be pooled and reused for other connections. This provides better resource utilization because there are typically a small number of threads that can be used to service a large number of connections. You can allow threads to be pooled to improve performance.

MAXDBAT constrains the total number of threads available to process remote SQL requests. If a request for a new connection to DB2 is received and MAXDBAT has been reached, the request is queued, waiting for a thread to become available to process the request. MAXDBAT generally should be set conservatively. It is usually constrained by the available DBM1 storage.

Specify the maximum number of concurrent remote connections by setting the CONDBAT installation parameter. This value must be greater than or equal to MAXDBAT. When a request to allocate a new connection to DB2 is received, and CONDBAT has been reached, the connection request is rejected. The value should be the largest number of pooled connections that would connect to the DB2 member at any point in time. Active threads that have not committed their work in a timely fashion are canceled after IDTHTOIN expires; locks and cursors are released. Inactive connections and in-doubt threads are not subject to time-out. Threads are checked every two minutes to see if they have exceeded the time-out value. If the timeout value is less than two minutes, the thread might not be canceled if it has been inactive for more than the time-out value but less than two minutes.

The quicker DB2 can detect the communication error and return the thread to the pool, the lower the chance to reach MAXDBAT. In cases where the z/OS TCP/IP KeepAlive value in the TCP/IP configuration is not appropriate for the DB2 subsystem, you can use the TCPKPALV as an override.

In addition to defining the IP addresses to TCP/IP, the member and group DVIPA corresponding host names are required to be defined prior to starting DDF. DDF recovery processing may require the use of these names during in-doubt resolution after a subsystem failure. You define the host names by configuring the hlq.HOSTS.LOCAL data set, the /etc/hosts file in the hierarchical file system (HFS), or the domain name server (DNS).

For a more general description of DB2 set up, see 4.3.1, “DB2 connectivity installation parameters” on page 138.

For adding more granular functions to the system level parameters, see 4.3.17, “Using DB2 profiles” on page 180.

3.2 IBM Data Server Drivers and Clients

The IBM strategy is to remove the reliance on the DB2 Connect modules and replace DB2 Connect with the IBM Data Server Drivers or Clients. Although DB2 Connect licenses (in the form of DB2 Connect license files) are still required, you can replace DB2 Connect modules with the IBM Data Server Drivers or Clients and receive equivalent or superior function. In addition, you can reduce complexity, improve performance, and deploy application solutions with smaller footprints for your business users.

With DB2 for LUW Version 9.5 Fix Pack 3 and above you can implement the DRDA requester functions for your distributed applications with varied degrees of granularity. Instead of the current function and large footprint of DB2 Connect, there are several types of IBM data server clients and drivers available. Each provides a particular type of support.

The IBM data server client and driver types are as follows:

- IBM Data Server Driver Package
- IBM Data Server Driver for JDBC and SQLJ
- IBM Data Server Driver for ODBC and CLI
- IBM Data Server Runtime Client
- IBM Data Server Client

Table 3-1 shows the details of what is contained in each offering.

Table 3-1 IBM Data Server Drivers and Clients comparison

Product	Smallest footprint	JDBC and SQLJ	ODBC and CLI	OLE DB and .NET	Open Source	CLP	DBA, Dev, GUI tools
IBM Data Server Driver for JDBC and SQLJ	X	X					
IBM Data Server Driver for ODBC and CLI	X		X				
IBM Data Server Driver Package		X	X	X	X		
IBM Data Server Runtime Client		X	X	X	X	X	
IBM Data Server Client		X	X	X	X	X	X

In this book we discuss IBM Data Server Driver for JDBC and SQLJ for Java applications. We describe the main connectivity options using IBM Data Server Driver for JDBC and SQLJ for WebSphere to connect to DB2 for z/OS system.

You can download the IBM Data Server Drivers and Clients from the IBM download site (`http://www.ibm.com/support/docview.wss?rs=4020&uid=swg21385217`) where you can see Table 3-2, which can help in identifying the package you need.

Table 3-2 IBM Data Server Client Packages: Latest downloads (DB2 10)

Driver package	Description
IBM Data Server Driver Package (DS Driver)	This package contains drivers and libraries for various programming language environments. It provides support for Java (JDBC and SQLJ), C/C++ (ODBC and CLI), .NET drivers and database drivers for open source languages like PHP and Ruby. It also includes an interactive client tool called CLPPlus that is capable of executing SQL statements and scripts, and can generate custom reports.
IBM Data Server Driver for JDBC and SQLJ (JCC Driver)	Provides support for JDBC and SQLJ for client applications developed in Java. Supports JDBC 3 and JDBC 4 standard. Also called as JCC driver.
IBM Data Server Driver for ODBC and CLI (CLI Driver)	This is the smallest of all the client packages and provides support for Open Database Connectivity (ODBC) and Call Level Interface (CLI) libraries for the C/C++ client applications.
IBM Data Server Runtime Client	This package is a superset of Data Server Driver package. It includes many DB2 specific utilities and libraries. It includes DB2 Command Line Processor (CLP) tool.
IBM Data Server Client	This is the all-in-one client package and includes all the client tools and libraries available. It includes DB2 Control Center, a graphical client tool that can be used to manage DB2 Servers. It also includes add-ins for Visual Studio.
IBM Database Add-Ins for Visual Studio	This package contains the add-ins for Visual Studio for .NET tooling support.

3.2.1 Connectivity options for IBM Data Server Driver for JDBC and SQLJ

IBM Data Server Driver for JDBC and SQLJ supports two types of connectivity: type 4 connectivity and type 2 connectivity.

For the DriverManager interface, you specify the type of connectivity through the URL in the `DriverManager.getConnection` method. For the DataSource interface, you specify the type of connectivity through the `driverType` property.

Connecting to DB2 using IBM Data Server Driver for JDBC and SQLJ type 4 connectivity

This configuration option is recommended for Java applications exist on a distributed platform that access the DB2 data remotely.

Type 4 driver is coded entirely in Java providing portability advantage and platform independence. It provides better performance for remote Java applications with type 4 connectivity. Also type 4 driver accesses DB2 system through TCP/IP and provides sysplex workload balancing support.

IBM ships two streams of the type 4 driver with the IBM Data Server Driver for JDBC and SQLJ product:

1. Version 3.5x is JDBC 3.0-compliant. It is packaged as db2jcc.jar and sqlj.zip and provides JDBC 3.0 and earlier support.
2. Version 4.x is JDBC 3.0-compliant and supports some JDBC 4.0 functions. It is packaged as db2jcc4.jar and sqlj4.zip.

The type 4 driver provides support for distributed transaction management. This support implements the Java 2 Platform, Enterprise Edition (J2EE), Java Transaction Service (JTS), and Java Transaction API (JTA) specifications, which conform to the X/Open standard for distributed transactions (Distributed Transaction Processing: The XA Specification).

Figure 3-3 shows types of type 4 connectivity with IBM Data Server Driver for JDBC and SQLJ.

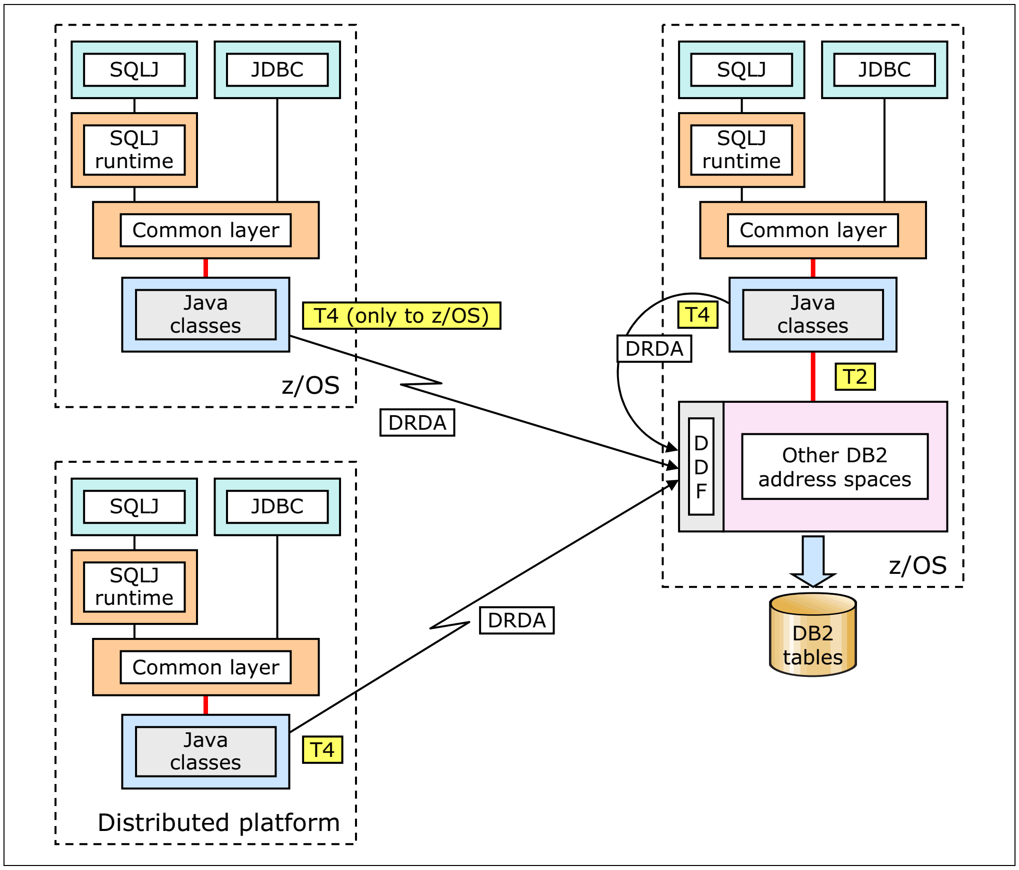


Figure 3-3 Various type 4 connectivity with IBM Data Server Driver for JDBC and SQLJ

Connecting to DB2 using IBM Data Server Driver for JDBC and SQLJ type 2 connectivity

This configuration option is suitable especially for Java applications that run on the same z/OS system or System z logical partition (LPAR) and access DB2 data locally.
Type 2 driver is needed for running Java stored procedures on DB2 for z/OS.
The DB2 JDBC type 2 Driver for LUW (DB2 JDBC type 2 Driver) is deprecated. Move your Java applications to use the IBM Data Server Driver for JDBC and SQLJ.

Figure 3-4 shows types of type 2 connectivity with IBM Data Server Driver for JDBC and SQLJ.

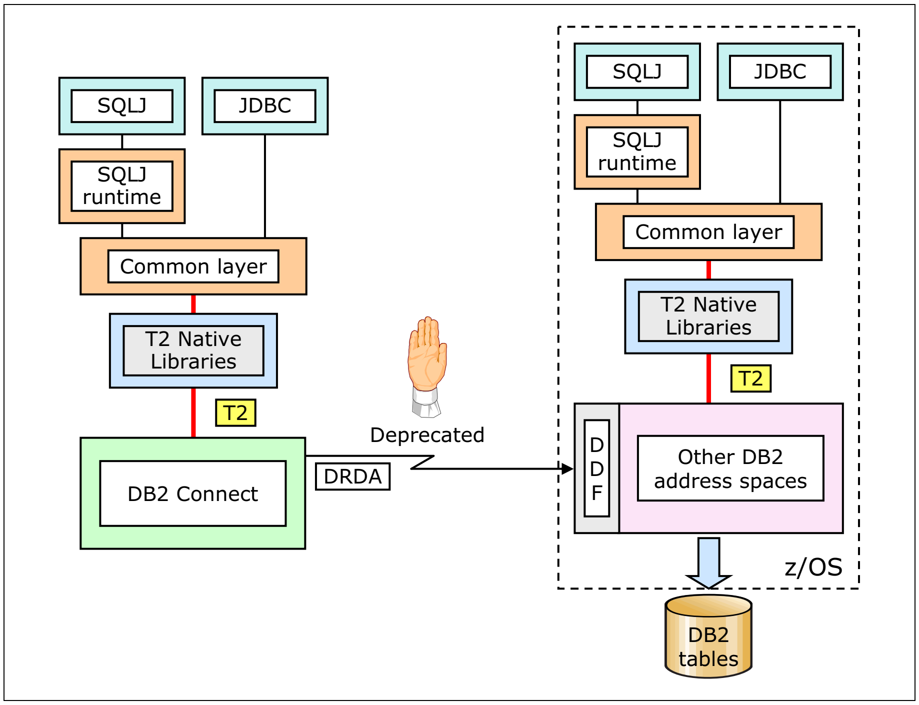


Figure 3-4 Type 2 connectivity with IBM Data Server Driver for JDBC and SQLJ

3.2.2 Limited block fetch extended to the JCC type 2 drivers

Over the past several DB2 versions, processing between the DB2 DDF and DBM1 address space was optimized and zIIP redirection has significantly reduced chargeable central processors consumption. Other improvements have included:

- Limited block fetch
- LOB progressive streaming
- Implicit CLOSE

These improvements were not available to *local* Java and ODBC applications that did not always perform faster compared to the same application called remotely. These improvement to remote Java applications were described in the*DB2 9 for z/OS Performance Topics*, SG24-74733 and the *DB2 Version 9.1 for z/OS Application Programming and SQL Guide*, SC18-9841. Refer to these documents for details about LOB progressive streaming and implicit CLOSE.

With DB2 10, many of these improvements are implemented for *local Java applications* using ODBC or JDBC. You can expect significant performance improvement for applications with the following queries:

- Queries that return more than 1 row
- Queries that return LOBs

Limited block fetch (LBF) support has been extended to the JCC type 2 drivers on z/OS. This technology, already available in the JCC type 4 and the distributed ODBC/CLI drivers, can provide dramatic improvements for applications involving large result set transfers; IBM observed more than 160% improvements in elapsed time and more than 170% improvements in CPU time in applications getting the advantages of this enhancement. This change leverages the drivers' functionalities and removes an inhibiting factor to the deployment of the Type 2 drivers for z based Java applications. The JCC type 2 driver gets installed or updated automatically when DB2 10 is installed.

This improvement is enabled by default, available in DB2 10 CM mode and there is no configuration required. It is not supported in JDBC/SQLJ stored procedures.

The number of rows returned per call depends on the buffer size (32767 to 262143 bytes with DB2 10), which is controlled by the queryDataSize property. queryDataSize specifies a hint that is used to control the amount of query data, in bytes, that is returned from the data source on each fetch operation. This value can be used to optimize the application by controlling the number of trips to the data source that are required to retrieve data.

Appropriate tuning of the DataSource property queryDataSize can improve performance by reducing the number of messages required between DB2 10 and a Java application when using JDBC type 2 driver and running on z/OS. This property also applies to the JDBC type 4 driver. Consider using a queryDataSize value bigger than 32 KB for large result sets if the utilization of a bigger buffer reduces the number of messages between DB2 and the application.

Regression is possible for simple OLTP transactions with single row result sets. In this case, LBF can be disabled through the configuration keyword: db2.jcc.override.enableT2zosLBF=2

3.3 High availability configuration options

The Sysplex built in features of the DB2 Connect clients provides the highest availability and fault tolerance possible with minimum configuration and application impact. This support is available for applications that use Java clients (JDBC, SQLJ, or pureQuery), or non-Java clients (ODBC, CLI, .NET, OLE DB, PHP, Ruby, or embedded SQL).

3.3.1 How to make your client application sysplex aware

A DB2 data sharing group is accessed using its group location name, a Sysplex wide dynamic virtual IP address, and the group port. The Sysplex IP address routes to all members in the group. It called the DB2 group IP address. This address is used to make the initial connection to the group. The group IP address should be distributed allowing connections to work as long a one member is started. This eliminates the initial connection point of failure. After it is connected to the group, the z/OS Workload Manger (WLM) provides a server list containing members used by the client in its routing decisions. This list is cached in the client. Some servers might not appear in the list due to WLM balancing decisions. The list returns automatically on connection boundaries and optionally on transaction boundaries.

A client configuration parameter can be set to ensure the list stays current. The default life span of the cached server list is 10 seconds. This list contains the member IP address and WLM weight for each data sharing group member. With this information, the client distributes transactions in a balanced manner, and seamlessly reroutes work even when there is a network failure, a member failure, a member slowdown, or when a member is quiesced for maintenance.

3.3.2 The difference between connections and transports

Sysplex workload balancing (WLB) feature supports transaction-level workload balancing for connections accessing a DB2 data sharing group. When a client is enabled with sysplex workload balancing, balancing decisions are performed at the start of each transaction. If just using the z/OS Sysplex Distributor, balancing decisions are performed at the start of each connection. Typically connection level balancing is not effective for most DB2 applications because connections have a long life.

After sysplex WLB is enabled in the data server driver client, application connections are no longer physical connections to DB2. Only when a connection is in use, a physical connection to DB2 is active. While a connection is not in use, the driver pools these connections. This pool of driver maintained connections are called transports. Transports are only associated with an application when a new transaction is started. A single transport can be used by many application connections. DB2 identifies unused transports as inactive connections. When a transaction is started, DB2 associates a thread with the inactive connection and associates to a thread called an active data base access thread (DBAT).

Figure 3-5 is an example of the Java driver but both clients have the same feature.

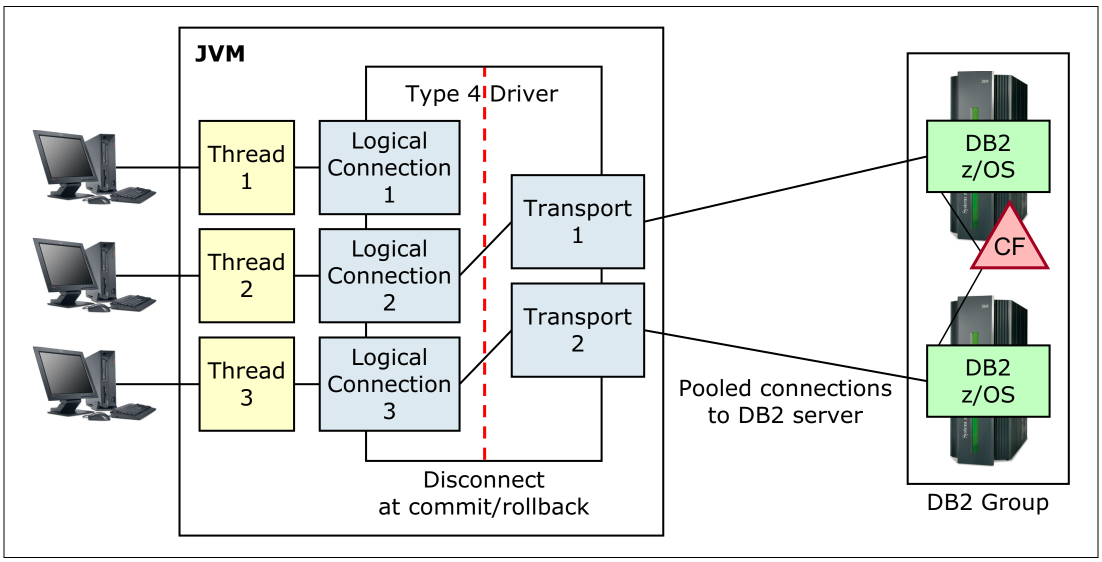


Figure 3-5 Java driver

At the start of each new transaction, the client reads the cached server list to identify a member that has unused capacity, and looks in the transport pool for an idle transport that is tied to the member. An idle transport is a transport that has no associated connection. If an idle transport is available, the client associates the connection with the transport. If after a user-configurable timeout period (db2.jcc.maxTransportObjectWaitTime for a Java client or maxTransportWaitTime for other clients), no idle transport is available in the transport pool and no new transport can be allocated because the transport pool reached its limit, an exception is returned to the application.

When the transaction runs, it accesses the server that is tied to the transport. When the transaction ends, the client verifies with the server that transport reuse is allowed for the connection. If the server identifies that the transport reuse is allowed, the server returns a list of SET statements for special registers that apply to the execution environment for the connection. The client caches these as SQL SET statements, which it replays to reconstruct the execution environment when the connection is associated with a new transport.

The server does not allow a transport to be reused if the connection has persistent resources opened such as an open held cursors or a global temporary tables that must maintained with the application until these resources are closed or dropped. To improve Sysplex workload balancing, it is important the application close any held cursors or drop any temporary tables when no longer needed. This allows database access threads to be effectively utilized by other applications.

3.3.3 What JCC client properties need to be changed

When enabling a JCC client to utilize Sysplex workload balancing and automatic client reroute, it is important to review and set the client properties that are associated with both functions. Starting in DB2 Connect V9.7 Fix Pack 6, JCC client starts to set the recommended defaults to the associated properties. Earlier JCC client driver levels, the client property defaults are not applicable to DB2 for z/OS and need to be reviewed and changed.

Generally, it is recommended to have the application always review and set the client properties. Proper setting of this information allows better isolation of problems, better classification of work which allows workload balancing to perform more efficiently. The DBA can quickly use client info to isolate issues to the specific client and even to the specific transaction.

Client information properties are managed by the application and need to be set prior to running the first SQL statement in each transaction if you want to use the client strings for WLM classification. The more granular you set and manage these properties, the more effective they are in managing the workload. On DB2, the client information can be used by WLM to classify work, is displayed in DB2 messages and is included in DB2 accounting data.

It is recommended to not use client affinities when accessing DB2 for z/OS Client affinities is not applicable to a DB2 data sharing environment, because all members of a data sharing group can access data concurrently.

How to enable Sysplex workload balancing and automatic client reroute for Java client

You should always configure Sysplex workload balancing and automatic client reroute together. When you configure a JCC client to use Sysplex WLB, automatic client reroute is also enabled by default. Therefore, you need to change JCC client properties related to automatic client reroute to control the reroute operation. Setting the enableSysplexWLB property to true for the JCC driver enables the Sysplex feature.

Table 3-3 shows the suggested property values for Java client enabled with Sysplex feature. For details, see *DB2 10 for z/OS Application Programming Guide and Reference for Java,* SC19-2970.

Table 3-3 Java client Sysplex property definitions

Property	Suggested value	Description
enableSysplexWLB	true	Enable Sysplex workload balancing and seamless automatic client reroute.
maxTransportObjects	Set to the number of concurrent transactions times the number of DB2 members	Maximum number of connections that the requester can make to the data sharing group.
maxTransportObjectIdleTime	30 seconds	Maximum elapsed time in seconds before an idle transport is dropped.

Property	Suggested value	Description
maxTransportObjectWaitTime	1 second	Time in seconds that the client waits for a transport to become available. When an application waits for longer than this value, the global transport object pool throws an SQLException.
maxRefreshInterval	30 seconds	Maximum amount of time in seconds between refreshes of the client copy of the server list.
memberConnectTimeout	1 second	Number of seconds that client application wait before routing to the next IP address in the server list.
maxRetriesForClientReroute	5 times	Number of times to retry after a connection failure before retrying the connection string
resultSetHoldability	CLOSE_CURSORS_AT_COMMIT(2)	Controls whether the cursor stays open across commit. This value overrides the default holdability for the connection.
queryCloseImplicit	QUERY_CLOSE_IMPLICIT_COMMIT(3)	Closes the cursor at the server after all the result sets are exhausted.
InterruptProcessingMode	INTERRUPT_PROCESSING_MODE_CLOSE_SOCKET(2)	This property is used to specify when an application executes the `Statement.cancel` method. Connection is dropped and the transaction is rolled back.
queryTimeoutInterruptProcessingMode	INTERRUPT_PROCESSING_MODE_CLOSE_SOCKET(2)	This property is used to specify when the query timeout interval for a `Statement` object expires. Connection is dropped and the transaction is rolled back.

How to collect DB2 group connection information

The DB2 **-DISPLAY DDF DETAIL** command issued on a member can be used to obtain the DB2 group and member information needed to construct the connection string for both the non-Java and Java drivers.

Example 3-3 shows the output of **-DISPLAY DDF DETAIL** command.

Example 3-3 DISPLAY DDF DETAIL output

```
14.10.29 STC00311 DSNL080I @ DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION LUNAME GENERICLU
DSNL083I STLEC1 USIBMSY.SYEC1DB2 USIBMSY.SYEC1GLU
DSNL084I TCPPORT=446 SECPORT=0 RESPORT=5001 IPNAME=NONE
DSNL085I IPADDR=::9.30.119.22
DSNL086I SQL DOMAIN=dvipa22.vmec.svl.ibm.com
DSNL086I RESYNC DOMAIN=dvipa23.vmec.svl.ibm.com
DSNL087I ALIAS PORT SECPORT STATUS
DSNL088I STLEC1ALIASSUB12 5052 0 STATIC
DSNL089I MEMBER IPADDR=::9.30.119.23
DSNL090I DT=I CONDBAT= 25000 MDBAT= 300
DSNL092I ADBAT= 0 QUEDBAT= 0 INADBAT= 0 CONQUED= 0
DSNL093I DSCDBAT= 0 INACONN= 0
DSNL100I LOCATION SERVER LIST:
DSNL101I WT IPADDR IPADDR
```

```
DSNL102I 21 ::9.30.119.23
DSNL102I 21 ::9.30.119.26
DSNL102I 21 ::9.30.119.28
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

The following is the explanation of various messages extracted from the **-DISPLAY DDF DETAIL** command above:

`DSNL083I STLEC1`	Group location
`DSNL085I IPADDR=::9.30.119.22`	Group distributed DVIPA address
`DSNL084I TCPPORT=446 RESPORT=5001`	Group port and the resync port
`DSNL088I STLEC1ALIASSUB12 5052 0 STATIC`	Defined DB2 location alias and port
`DSNL089I MEMBER IPADDR=::9.30.119.23`	Member IP address in the location alias

How to configure the Java client with high availability

There are two aspects that you need to consider:

- First, you need to set the global properties file. See also 5.10, "Configuring the JCC properties file in WebSphere Application Server" on page 282 for the WebSphere side.
- Second, you need to set the data source properties.

Setting the global properties file

The java Sysplex configuration properties are set in a global property file DB2Jccconfiguration.Properties. The file lets you set Sysplex property values that have driver-wide scope. Those settings apply across applications and DataSource instances. You can change the settings without having to change application source code or DataSource characteristics. The JDBC and SQLJan example of settings for the DB2JccConfiguration.properties file.

Example 3-4 Sample settings for the DB2JccConfiguration.properties

```
db2.jcc.maxRefreshInterval=30
db2.jcc.minTransportObjects=0
db2.jcc.maxTransportObjects=1000
db2.jcc.maxTransportObjectWaitTime=1
db2.jcc.maxTransportObjectIdleTime=30
```

See 5.10, "Configuring the JCC properties file in WebSphere Application Server" on page 282 for the WebSphere properties.

Setting the data source properties

Example 3-5 shows the recommended data source properties using a sample Java application.

Example 3-5 Data source properties using a sample Java application

```
public class SampleDS
{
   public static void main(String[] args) throws SQLException
   {
      DB2SimpleDataSource ds = new DB2SimpleDataSource();
      ds.setServerName("DB2IP.ibm.com");
```

```
ds.setPortNumber(12345);
ds.setDatabaseName("DB2ServerName");
ds.setUser("USERID");
ds.setPassword("PASSWORD");

// High availability properties
ds.setEnableSysplexWLB(true);
ds.setLoginTimeout(3);
ds.setMaxRetriesForClientReroute(5);
ds.setInterruptProcessingMode(DB2BaseDataSource.
INTERRUPT_PROCESSING_MODE_CLOSE_SOCKET);

// Performance and storage consumption properties
ds.setProgressiveStreaming(DB2BaseDataSource.YES);

// Thread Utilization properties
ds.setResultSetHoldability(DB2BaseDataSource.CLOSE_CURSORS_AT_COMMIT);
ds.setQueryCloseImplicit(DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_YES);

try
{
   DB2Connection con = (DB2Connection)ds.getConnection();
   // Thread Utilization properties
   con.setAutoCommit(false);

   // Problem determination correlation settings
   con.setDB2ClientApplicationInformation("This is a sample application");
   con.setDB2ClientUser("A Test End User1");
   con.setDB2ClientWorkstation("A Test End Wrkstn1");


   PreparedStatement ps;
   String insertsql = "INSERT INTO TABLE1 VALUES (?,?)";
   ps = con.prepareStatement(insertsql);
   for (int i =1;i<=200; i++){
      ps.setInt(1,i+1);
      ps.setString(2,i+"Test Sample : This is a Long Test String"+i);
      ps.addBatch();// Add batch processing
   }
   ps.executeBatch();// Execute batch processing

   String psSQL = "SELECT * FROM TABLE1";
   ps = con.prepareStatement (psSQL);


   // Performance impact properties on SQL level
   ps.setFetchSize(199);
   ps.setMaxRows(199);

   ResultSet rs = ps.executeQuery ();

   while (rs.next()){
      //fetch to the end;
   };
```

```
         rs.close();// Close cursor when done using it
         ps.close();
         con.commit();// Commit on regular basis
      }
      catch(Exception e)
      {
      System.out.println("main() Exception: " + e.getMessage());
      }
   }
}
```

For setting the data source properties in WebSphere Application Server, see 5.3.3, “Defining a JDBC type 2 data source” on page 233.

4

DB2 infrastructure setup

In this chapter we discuss the tasks you need to perform to provide a DB2 data sharing infrastructure that supports continuous availability in a WebSphere Application Server environment. With reference to continuous availability we first discuss the setup tasks to be performed regardless of the JDBC driver type being used and then highlight the tasks that are specific to each JDBC driver type used by WebSphere Application Server for connecting to DB2. For information about the configuration of the WebSphere Application Server environment refer to Chapter 5, “WebSphere Application Server infrastructure setup” on page 207.

This chapter covers the following topics:

- z/OS related setup
- Monitoring strategy
- DB2 for z/OS configuration
- Tivoli OMEGAMON XE for DB2 Performance Expert for z/OS
- DB2 database and application design considerations

4.1 z/OS related setup

Creating a DB2 data sharing group requires a certain infrastructure to be provided across all z/OS systems and subsystems in a Parallel Sysplex. The infrastructure to be provided involves configuration tasks to be performed within the following System z components:

- Parallel Sysplex
- Automatic Restart Manager policy
- WLM configuration
- Resource Recovery Services
- z/OS resource planning
- External storage configuration
- UNIX System Services file system configuration
- Monitoring infrastructure
- WebSphere Application Server and DB2 security

4.1.1 Parallel Sysplex

DB2 data sharing in its core provides high availability, scalability and optimum performance by proactively exploiting Parallel Sysplex functions and resources. To provide data sharing DB2 exploits Coupling Facility resources to allow multiple DB2 members running on any z/OS system in a Sysplex to share the same data. When using DB2 data sharing DB2 data remains available for as long as at least one member of the data sharing group is available. In case of a DB2 failure Parallel Sysplex functions are used to bring the failing DB2 member back online as soon as possible.

In this part of the documentation we outline the Parallel Sysplex resources that are required for DB2 data sharing, illustrate the Parallel Sysplex configuration used in our environment, and discuss important aspects we want you to consider.

For more information and suggested practices on how to set up and tune DB2 data sharing, refer to the following resources.

- Part 4 DB2 sysplex best practices of *System z Parallel Sysplex Best Practices*, SG24-78177.
- *DB2 10 for z/OS Data Sharing: Planning and Administration*. SC19-2973
- IBM developerWorks® DB2 for z/OS with best practices presentations available at http://www.ibm.com/developerworks/data/bestpractices/db2zos/

Coupling Facility resources

For our DB2 data sharing group we created the coupling facility (CF) structures shown in the IBM RMF™ III Coupling Facility Activity report in Figure 4-1.

```
 RMF V1R13  CF Activity      - SANDBOX       Line 1 of 28
Command ===>                                           Scroll ===> CSR

Samples: 120     Systems: 4    Date: 08/08/12  Time: 20.35.00  Range: 120   Sec

CF: CF1          Type  ST System   CF    --- Sync ---    --------- Async --------
                                   Util   Rate   Avg    Rate   Avg   Chng   Del
Structure Name                      %            Serv          Serv   %      %

DB0ZG_GBP0       CACHE AS *ALL      0.1    0.0      0    0.1   1031   0.0   0.0
DB0ZG_GBP2       CACHE AS *ALL      1.9    0.0      0   29.7    109   0.0   0.0
DB0ZG_GBP32K     CACHE AS *ALL      0.0    0.0      0    0.0      0   0.0   0.0
DB0ZG_GBP8K0     CACHE AS *ALL      0.0    0.0      0    0.1    544   0.0   0.0
DB0ZG_LOCK1      LOCK  A  *ALL      8.1  207.6     55   68.6    113   0.0   0.0
DB0ZG_SCA        LIST  A  *ALL      0.4    5.0     39    0.0      0   0.0   0.0
```

Figure 4-1 RMF Monitor III CFACT display

Our data sharing environment was configured for function testing rather than for performance and scalability testing for which we were happy to accept a minimum configuration in terms of CF structure sizes. On top of that, we implemented common best practice recommendations such as structure duplexing and structure failure isolation to support high availability.

Under normal circumstances you would need to plan your CF structure implementation to make sure they are appropriately sized and implemented to support your availability requirements.

Coupling Facility sizing

When you enable DB2 data sharing you can use the IBM CFSIZER application to size the DB2 CF structures. The CF sizer estimates the size of your DB2 CF structures based upon environmental parameters such as locking rate, number of systems, databases, table spaces, tables per database, and local buffer pool sizes. The DB2 CFSIZER application is available on http://www.ibm.com/systems/support/z/cfsizer/db2.

If for some reason you are not able to provide the input parameters required by the DB2 CFSIZER tool you can use the DB2 minimum structure sizing recommendations given in Chapter 8, "Best practices:", in *DB2 for z/OS: Data Sharing in a Nutshell*, SG24-7322.

Coupling Facility related recommendations

After the DB2 structures are in use by DB2 you need to perform regular monitoring and tuning to assure the DB2 CF structures have been configured to support your workload and availability requirements. You can use the recommendation provided in Part 4, DB2 sysplex best practices of *System z Parallel Sysplex Best Practices*, SG24-7817, to set up monitoring and tuning to make sure your CF structures are configured in that way. This book discusses topics such as:

- Avoid single point of failures
- Failure isolate the DB2 lock and SCA structure
- Why XCF auto alter is useful for your DB2 structures
- Importance of having your DB2 SCA structure generously sized
- Why to avoid GBP cross-invalidations

- Why the GBP checkpoint frequency might affect speed of recovery
- Why DB2 group buffer pools (GBP) duplexed is useful
- GBPCACHE recommendations
- Table space to buffer pool design considerations.

4.1.2 Automatic Restart Manager policy

If one of the members of the data-sharing group fails, it is almost certain that the DB2 is holding locks on DB2 objects when it fails. Until that DB2 is restarted and releases those locks, the related objects are not available to the other members of the sysplex.

There are two scenarios relating to a DB2 failure:

- DB2 failed
- The system that DB2 was running on failed

In either case, the most important thing is to get the failed DB2 back up and running as quickly as possible. The best way to achieve this is to use the IBM MVS™ Automatic Restart Manager (ARM). Many automation products provide support for ARM. This means that they manage DB2 for normal startup, shutdown, monitoring, and so on. However, if DB2 fails, they understand that they must allow ARM to take responsibility for restarting DB2.

If the failure was just in DB2, and the system it was running on is still available, restart DB2 in the same LPAR, with a normal start. DB2 automatically releases any retained locks as part of the restart.

If the system DB2 was running on is unavailable, start DB2 for another member of the sysplex as quickly as possible. The reason for this is that it results in DB2 coming up and cleaning up its locks far faster than it is able to do were you would have to wait for z/OS to be IPLed and brought back up.

Furthermore, if DB2 is started on another system in the Sysplex, you really only want it to release any locks that it was holding. More than likely, there is another member of the data-sharing group already running on that system. If you specify the LIGHT(YES) option on the START DB2 command, DB2 starts with the sole purpose of cleaning up its locks. In this mode, it only communicates with address spaces that it was connected to before the failure, and that have indoubt units of work outstanding. As soon as DB2 completes its cleanup, the address space automatically shuts itself down. Hopefully, the failed system is on its way back up at this time, and the DB2 can be brought up with a normal start in its normal location.

In addition to restarting DB2 using ARM and Restart Light, also define a restart group to ARM so that it also restarts any subsystems that were connected to DB2 prior to the failure. By restarting all the connected subsystems, any indoubt units of recovery can be cleaned up.

Note that when the Restart Light capability was introduced by DB2 V7, it did not handle cleanup for any INDOUBT units of work. However, in DB2 V8 the Restart Light capability was enhanced so that it cleans up any INDOUBT units of work, assuming that the associated address space is also restarted on the same system. If you do not want to have DB2 resolve the INDOUBT units of work, or if you do not plan to restart the connected address spaces on the other system, start DB2 with the NOINDOUBT option.

Suggestion: Use ARM to restart DB2 following a DB2 failure.

If only DB2 failed, ARM must do a normal restart for DB2 for the same z/OS system. Do a RESTART LIGHT of DB2 on a different system from the one on which it was running previously.

If the system failed, ARM should do a Restart Light for DB2 on another system in the sysplex. Also, define a restart group so that ARM can also restart any related subsystems together with the restarted DB2.

WebSphere Application Server considerations

WebSphere Application Server for z/OS uses the z/OS automatic restart management (ARM) to recover application servers. Each application server running on a z/OS system is automatically registered with ARM. The registration uses a special element type called SYSCB, which ARM treats as restart level 3, assuring that RRS and DB2 restarts before any application server, because RRS and DB2 are treated by ARM as restart level 2. The automatic ARM registration per se registers all WebSphere Application Server server instances in a default group to provide automatic restart in case of subsystem or system failure.

By way of derogation from the default ARM registration, we recommend to implement the following ARM policy changes to be in line with best practice recommendations:

- Set up your location service daemons for restart in place. If the location service daemon attempts to restart on an alternate system, it will fail.
- Set up you node agents for restart in place. If the node agent restarts on the alternate system, it will have no recovery work to do.

For more information about how to configure ARM with WebSphere Application Server for z/OS refer to IBM WebSphere Application Serve Information Center, Automatic restart management at `http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp`.

Implementing the DB2 ARM policy

Figure 4-2 illustrates the configuration of our WebSphere Application Server and DB2 infrastructure. The illustration shows the following infrastructure components:

- z/OS systems SC63 and SC64
- Each z/OS system hosts one DB2 data sharing member and its related application server instance. In ARM these components are also referred to as restart element. The DB2 members and application server instances are:
  - z/OS system SC63: DB2 member D0Z1, application server instance MZSR013
  - z/OS system SC64: DB2 member D0Z2, application server instance MZSR014
- In case of element failure (also referred to as subsystem failure) the failing element is to be restarted in the same system (restart in place)
- In case of system failure (z/OS failure) the DB2 member and its related application server is to be restarted in the surviving z/OS system which can be system SC63 or SC64.

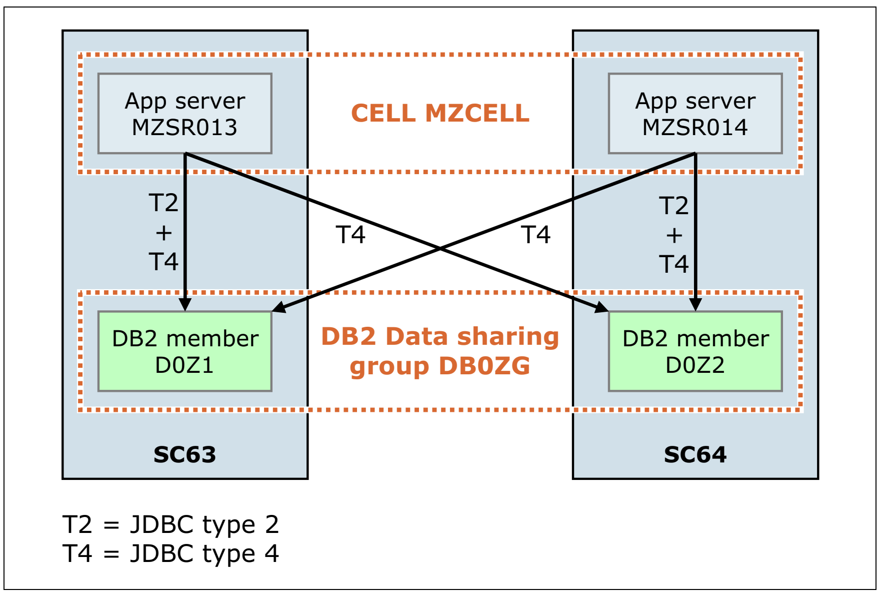


Figure 4-2 WebSphere Application Server for z/OS and DB2 for z/OS infrastructure

To provide the recommended level of availability for our application server environment we configured ARM to use the restart policy shown in Example 4-1. Through this policy the DB2 member and its related application server instance will be restarted in the same LPAR in case a subsystem failure occurs. In case of a system failure ARM restarts the DB2 member and its related application server instance either in system SC63 or SC64 depending on system availability.

Example 4-1 ARM policy DB2 WebSphere Application Server

```
RESTART_GROUP(DB2WAS)
  TARGET_SYSTEM(SC63,SC64)
  ELEMENT(DB0ZGD0Z1)
    RESTART_ATTEMPTS(3,)
    RESTART_METHOD(ELEMTERM,PERSIST)
    RESTART_METHOD(SYSTERM,STC,
       '-D0Z1 STA DB2,LIGHT(YES)')
  ELEMENT(DB0ZGD0Z2)
    RESTART_ATTEMPTS(3,)
```

```
    RESTART_METHOD(ELEMTERM,PERSIST)
    RESTART_METHOD(SYSTERM,STC,
       '-DOZ2 STA DB2,LIGHT(YES)')
  ELEMENT(MZCELLMZSR013)
    RESTART_ATTEMPTS(3,)
    RESTART_METHOD(ELEMTERM,PERSIST)
    RESTART_METHOD(SYSTERM,STC,
       'S MZACR3,'
       'JOBNAME=MZSR013,ENV=MZCELL.MZNODE3.MZSR013')
  ELEMENT(MZCELLMZSR014)
    RESTART_ATTEMPTS(3,)
    RESTART_METHOD(ELEMTERM,PERSIST)
    RESTART_METHOD(SYSTERM,STC,
       'S MZACR4,'
       'JOBNAME=MZSR014,ENV=MZCELL.MZNODE4.MZSR014')
```

After the policy shown in Example 4-1 on page 104 was activated we used the operating system command interface to confirm that our ARM policy was used for DB2 and its related application servers. The operating system command output is provided in Figure 4-3.

```
========= verify ARMSTATUS of application server MZSR014 ============
D  XCF,ARMSTATUS,JOBNAME=MZSR014,DETAIL
IXC392I  00.10.06  DISPLAY XCF 129
ARM RESTARTS ARE ENABLED
-------------- ELEMENT STATE SUMMARY --------------   -TOTAL-  -MAX-
STARTING  AVAILABLE  FAILED  RESTARTING  RECOVERING
       0          1       0           0           0        1    200
RESTART GROUP:DB2WAS           PACING  :    0    FREECSA:    0       0
ELEMENT NAME :MZCELLMZSR014    JOBNAME :MZSR014  STATE   :AVAILABLE
  CURR SYS :SC64               JOBTYPE :STC       ASID    :012A
  INIT SYS :SC64               JESGROUP:XCFJES2A TERMTYPE:ALLTERM
  EVENTEXIT:*NONE*             ELEMTYPE:SYSCB    LEVEL   :    3
  TOTAL RESTARTS :        0    INITIAL START:08/31/2012 23:45:33
  RESTART THRESH :   0 OF 3    FIRST RESTART:*NONE*
  RESTART TIMEOUT:      300    LAST  RESTART:*NONE*
======== verify ARMSTATUS of DB2 member DOZ2 ====================
D  XCF,ARMS,ELEMENT=DB0ZGD0Z2,DETAIL
IXC392I  00.15.31  DISPLAY XCF 133
ARM RESTARTS ARE ENABLED
-------------- ELEMENT STATE SUMMARY --------------   -TOTAL-  -MAX-
STARTING  AVAILABLE  FAILED  RESTARTING  RECOVERING
       0          1       0           0           0        1    200
RESTART GROUP:DB2WAS           PACING  :    0    FREECSA:    0       0
ELEMENT NAME :DB0ZGD0Z2        JOBNAME :D0Z2MSTR STATE   :AVAILABLE
  CURR SYS :SC64               JOBTYPE :STC       ASID    :00CE
  INIT SYS :SC64               JESGROUP:XCFJES2A TERMTYPE:ALLTERM
  EVENTEXIT:DSN7GARM           ELEMTYPE:SYSDB2   LEVEL   :    1
  TOTAL RESTARTS :        0    INITIAL START:08/16/2012 12:38:01
  RESTART THRESH :   0 OF 3    FIRST RESTART:*NONE*
  RESTART TIMEOUT:    21600    LAST  RESTART:*NONE*
```

Figure 4-3 DISPLAY XCF ARM command output

4.1.3 WLM configuration

You need to setup WLM to provide WLM application environments for DB2 external stored procedure processing and for the service classification of the DB2 system address spaces and for JDBC type 4 workloads.

DB2 stored procedure WLM application environments

The JDBC driver uses DB2 provided external stored procedures for metadata retrieval. These external stored procedures require WLM application environments (WLM APPLENV) to be defined and activated. DB2 install jobs DSNTIJRT or DSNTIJMS performs this configuration task during DB2 installation or migration. The WLM application environments used in our DB2 environment are illustrated in Figure 4-4.

```
 Application-Environment  Notes  Options  Help
-----------------------------------------------------------------------
                  Application Environment Selection List Row 164 to 178
Command ===> __________________________________________________________

Action Codes: 1=Create, 2=Copy, 3=Modify, 4=Browse, 5=Print, 6=Delete,
              /=Menu Bar

Action  Application Environment Name      Description
  __    DSNWLMDBOZ_DSNACICS               DB2-SUPPLIED WLM ENVIRONMENT
  __    DSNWLMDBOZ_GENERAL                DB2-SUPPLIED WLM ENVIRONMENT
  __    DSNWLMDBOZ_JAVA                   DB2-SUPPLIED WLM ENVIRONMENT
  __    DSNWLMDBOZ_MQSERIES               DB2-SUPPLIED WLM ENVIRONMENT
  __    DSNWLMDBOZ_NUMTCB1                DB2-SUPPLIED WLM ENVIRONMENT
  __    DSNWLMDBOZ_PGM_CONTROL            DB2-SUPPLIED WLM ENVIRONMENT
  __    DSNWLMDBOZ_REXX                   DB2-SUPPLIED WLM ENVIRONMENT
  __    DSNWLMDBOZ_UTILS                  DB2-SUPPLIED WLM ENVIRONMENT
  __    DSNWLMDBOZ_WEBSERVICES            DB2-SUPPLIED WLM ENVIRONMENT
  __    DSNWLMDBOZ_XML                    DB2-SUPPLIED WLM ENVIRONMENT
```

Figure 4-4 DB2 SPAS WLM application environments

After we had run one of our Java EE sample applications we verified the status of the procedures used by the JDBC driver for metadata retrieval. To perform this verification we executed a DISPLAY PROCEDURE command shown in Figure 4-5. For each procedure the command output confirms the procedure status and the WLM application environment name.

```
-dis proc
 DSNX940I  -DOZ1 DSNX9DIS DISPLAY PROCEDURE REPORT FOLLOWS -
 ------- SCHEMA=SYSIBM
 PROCEDURE      STATUS ACTIVE QUED MAXQ TIMEOUT FAIL WLM_ENV
 SQLCOLUMNS
                STARTED     0    0    1       0    0 DSNWLMDBOZ_GENERAL
 SQLCAMESSAGE
                STARTED     0    0    1       0    0 DSNWLMDBOZ_GENERAL
```

Figure 4-5 DISPLAY PROCEDURE output

Next we executed the DB2 command shown in Example 4-2 to verify the status and the JCL procedure name of the DSNWLMDB0Z_GENERAL WLM application environment. The command output confirms the application environment availability and the name of the JCL procedure used by WLM to start the WLM stored procedure address space (WLM SPAS).

Example 4-2 DISPLAY WLM APPLENV output

```
D WLM,APPLENV=DSNWLMDB0Z_GENERAL
 IWM029I  04.02.44  WLM DISPLAY 256
   APPLICATION ENVIRONMENT NAME     STATE     STATE DATA
   DSNWLMDB0Z_GENERAL               AVAILABLE
   ATTRIBUTES: PROC=D0ZGWLMG SUBSYSTEM TYPE: DB2
```

WLM service classification for DB2

You use z/OS Workload Manager (WLM) to assign performance goals and business importance to your batch, OLTP, and Java EE workloads. Performance goals and business importance in that sense controls how many resources, such as CPU and storage, should be given to the work to meet its goal. WLM controls the dispatching priority based on the goals you supply. WLM raises or lowers the priority as needed to meet the specified goal.

The three kinds of goals that you can use are:

- Response time

  Controls how quickly you want the work to be processed.

- Execution velocity

  Controls how fast the work should be run when ready, without being delayed for processor, storage, I/O access, and queue delay.

- Discretionary

  Defines a category for low priority work for which you define no performance goals.

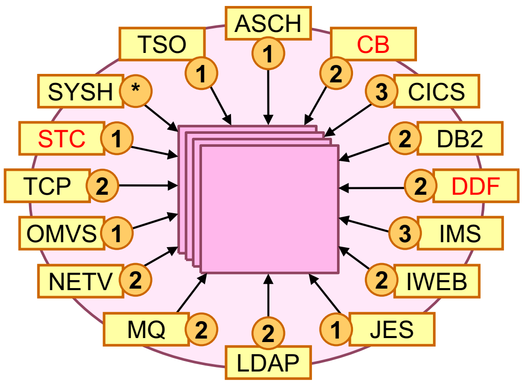
Response time goals are appropriate for user applications. User applications in the context of this book are WebSphere Application Server for z/OS Java applications connecting to DB2 for z/OS using JDBC type 2 and type 4 connections.

For the DB2 system address spaces, velocity goals are more appropriate. Only a small amount of the work done in DB2 is counted toward this velocity goal. Most of the work done in DB2 counts towards the user goal.

Your performance goals are implemented through WLM service classes. You create your WLM service classes using the attributes that are required to meet your service level agreement objective. WLM classes are categorized by subsystem types. WLM uses the subsystem type specific classification rules to assign service classes to incoming workloads. For simplification we use the term service classification to refer to the process of service class assignment by WLM.

WLM subsystem types

Figure 4-6 provides an overview over the subsystem types available in WLM.



- Subsystems follow one of three transaction type models.
- Need to understand how this affects the value of figures shown in the workload activity report.

  * SYSH is used for LPAR load balancing

Transaction Type	Allowable Goal Types	Allowable Number of Periods
Address space oriented (1)	Response Time Execution Velocity ← STC: DB2 Address spaces Discretionary	Multiple
Enclave (2)	Response Time Execution Velocity } DDF (and DB2) Discretionary	Multiple
CICS/IMS (3)	Response Time IMS Txn (class)	1

Figure 4-6 WLM classification subsystem types

The WLM subsystem types relevant to our WebSphere Application Server environment are

- STC (started task control)
  Subsystem type for the service classification of DB2 and WebSphere Application Server system address spaces. In this part of the book we only discuss the classification of the DB2 system address spaces.
- DDF
  Subsystem type for the service classification of transaction type enclave workloads that arrives in DB2 through the DB2 DIST address space through JDBC type 4 connections.
- CB
  Subsystem type for service classification of transaction type enclave Java workloads that run in WebSphere Application Server for z/OS regardless of the JDBC driver type being used.

Service classifications for subsystem type DB2 is only relevant for workloads related to DB2 Parallel Sysplex Query Parallelism which is not being used in our scenario.

DB2 system service classification

We used the WLM service classification illustrated in Figure 4-7 on page 109 for assigning velocity goals to our DB2 system service address spaces. The service classes we chose are in line with the recommendations given by *DB2 10 for z/OS Managing Performance*, SC19-2978. We additionally assigned unique report classes to each DB2 address space which can be useful for monitoring and problem determination purposes.

```
 Subsystem-Type  Xref  Notes  Options  Help
--------------------------------------------------------------------------
                 Modify Rules for the Subsystem Type     Row 8 to 34 of 35
Command ===> ___________________________________________ Scroll ===> CSR

Subsystem Type . : STC        Fold qualifier names?   N  (Y or N)
Description  . . . Started Tasks

Action codes:   A=After     C=Copy        M=Move     I=Insert rule
                B=Before    D=Delete row  R=Repeat   IS=Insert Sub-rule
                                                             More ===>
          --------Qualifier--------             -------Class--------
Action    Type       Name     Start                Service     Report
                                         DEFAULTS: STC         RSYSDFLT
 ____  1  TN         D0Z1ADMT ___                  STC         D0Z1ADMT
 ____  1  TN         D0Z1DBM1 ___                  STCHI       D0Z1DBM1
 ____  1  TN         D0Z1MSTR ___                  STCHI       D0Z1MSTR
 ____  1  TN         D0Z1DIST ___                  STCHI       D0Z1DIST
 ____  1  TN         D0Z1IRLM ___                  SYSSTC      D0Z1IRLM
 ____  1  TN         D0Z2ADMT ___                  STC         D0Z2ADMT
 ____  1  TN         D0Z2DBM1 ___                  STCHI       D0Z2DBM1
 ____  1  TN         D0Z2MSTR ___                  STCHI       D0Z2MSTR
 ____  1  TN         D0ZGWL*  ___                  STCHI       D0ZGWLM
 ____  1  TN         D0Z2DIST ___                  STCHI       D0Z2DIST
 ____  1  TN         D0Z2IRLM ___                  SYSSTC      D0Z2IRLM
```

Figure 4-7 WLM DB2 system service classification

After we had started DB2 we used the ISPF SDSF application to verify that z/OS used our WLM configuration for the DB2 started tasks. The SDSF display active output that we obtained for verification is shown in Figure 4-8.

```
 Display  Filter  View  Print  Options  Search  Help
-----------------------------------------------------
SDSF DA SC64     SC64     PAG  0  CPU/L/Z   2/  3/  1
COMMAND INPUT ===>
NP   JOBNAME  U% Workload SrvClass RptClass
     D0Z2MSTR  2 STCTASKS STCHI    D0Z2MSTR
     D0Z2IRLM  2 SYSTEM   SYSSTC   D0Z2IRLM
     D0Z2DBM1  2 STCTASKS STCHI    D0Z2DBM1
     D0Z2DIST  2 STCTASKS STCHI    D0Z2DIST
     D0Z2ADMT  2 STCTASKS STC      D0Z2ADMT
     D0ZGWLMG  2 STCTASKS STCHI    D0ZGWLM
     D0ZGWLM1  2 STCTASKS STCHI    D0ZGWLM
```

Figure 4-8 SDSF WLM DB2 system service classification

In our scenario the D0Z1 and D0Z2 DIST address spaces run in service class SCTHI which represents a performance goal that is as high as the goal for the DB2 database services address spaces. Classifying the DIST address spaces appropriately is important as the service class determines how quickly the DIST address space is able to perform operations associated with managing the distributed DB2 work load. Operations in that sense include adding new users or removing users that have terminated their JDBC type 4 connections.

JDBC type 4 service classification

The WLM classification of the DDF address space described in "DB2 system service classification" on page 108 does not govern the performance objective of data base access threads (DBATs) connecting to DB2 through WebSphere Application Server JDBC type 4 connections. As illustrated in Figure 4-9 the SQL workload submitted by DBATs, also referred to as DDF server threads, are scheduled by DDF to run in DB2 as enclave service request blocks (enclave SRB). The enclave SRB obtains its performance goal from the WLM service classification rule that is defined in the active WLM policy.

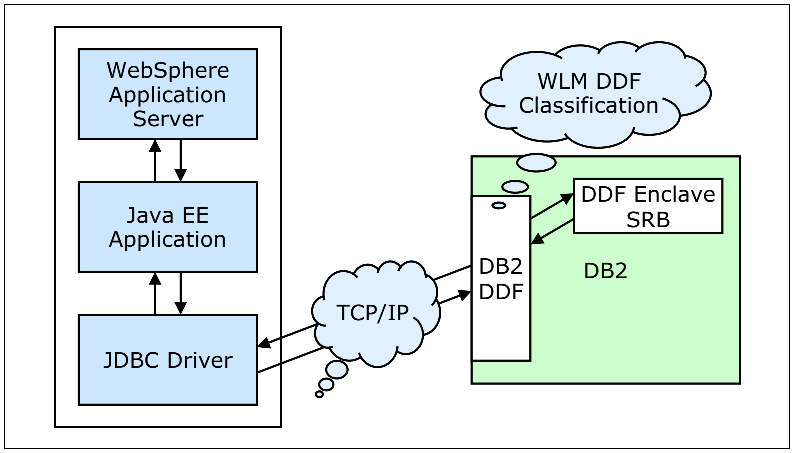


Figure 4-9 WebSphere Application Server JDBC type 4

The main characteristics of a DDF enclave are:

- An enclave is a single transaction, which starts when the enclave is created and ends when the enclave is deleted.
- DDF creates an enclave for an incoming request when it detects the first SQL statement and usually deletes the enclave at SQL COMMIT, thus a DDF enclave transaction consists of a single SQL COMMIT scope.
- You can use WLM to let individual DDF server threads (DDF enclaves) have their own z/OS performance objectives. For instance, you can assign a WLM service class with a short response time to service your business critical online DDF server threads.
- A DDF enclave SRB is scheduled to run in the target DB2 address space but executes work on behalf of the DDF enclave.
- Dispatching controls for enclave SRB processing are derived from the DDF enclave. As for our Java EE sample applications we use DB2 client information to control service classification in WLM.
- CPU time consumed by each SRB is accumulated back to the DDF enclave and is reported as enclave-related CPU service in the SMF type 30 records of the enclave owning DDF address space.
- RMF creates separate Type72 SMF records for independent enclaves.

In our environment we use two WebSphere Application Server applications to illustrate WLM service classification. For WLM classification each application provides the DB2 clientApplicationInformation data source custom property shown in Table 4-1 on page 111 when connecting to DB2.

Table 4-1 clientApplicationInformaton

Application	Context root	clientApplicationInformation
DayTrader-EE6	/daytrader	TraderClientApplication
D0ZG_WASTestClientInfo	/wastestClientInfo	dwsClientinformationDS

We used the DB2 clientApplicationInformation listed in Table 4-1 to define the service classification rules that are shown in Figure 4-10.

```
 Subsystem-Type  Xref  Notes  Options  Help
-----------------------------------------------------------------------
                 Modify Rules for the Subsystem Type    Row 19 to 41 of
Command ===> ___________________________________________ Scroll ===> CS

Subsystem Type . : DDF        Fold qualifier names?   N  (Y or N)
Description  . . . DDF Work Requests

Action codes:   A=After     C=Copy        M=Move     I=Insert rule
                B=Before    D=Delete row  R=Repeat   IS=Insert Sub-rule
                                                             More ===>
          --------Qualifier--------                --------Class--------
Action    Type       Name     Start                 Service     Report 3
                                          DEFAULTS: DDFBAT      ________
 ____   1  SSC  1    DB0ZG    ___                   DDFDEF      RD0ZGDEF
 ____   2    PC 2      Trade*  ___                  DDFONL      RTRADE0Z
 ____   2    PC        dwsClie* ___                 DDFONL      RDWS0Z
```

Figure 4-10 WLM DDF classification

1. Type SSC (subsystem collection) contains the data sharing group name which is not to be confused with the group attach name. You can determine group name by running the command shown in Figure 4-11.

```
-dis group
 DSN7100I  -D0Z2 DSN7GCMD
 *** BEGIN DISPLAY OF GROUP(DB0ZG   ) CATALOG LEVEL(101) MODE(NFM )
```

Figure 4-11 DB2 Display group output

2. Under data sharing group DB0ZG we use request type PC (process name) to assign WLM service and report classes based upon DB2 client application information that are used by our Java applications. The DB2 client information provided by our Java applications are:
   - clientApplicationInformation:TraderClientApplication
     - matches WLM process name Trade*
     - assigns WLM service class DDFONL
     - assigns WLM report class RTRADE0Z
   - clientApplicationInformation: dwsClientinformationDS
     - matches WLM process name dwsClie*
     - assigns WLM service class DDFONL
     - assigns WLM report class RDWS0Z

The WLM classification of our DB2 DBAT workload using the WLM configuration shown in Figure 4-10 on page 111 is illustrated in Figure 4-12.

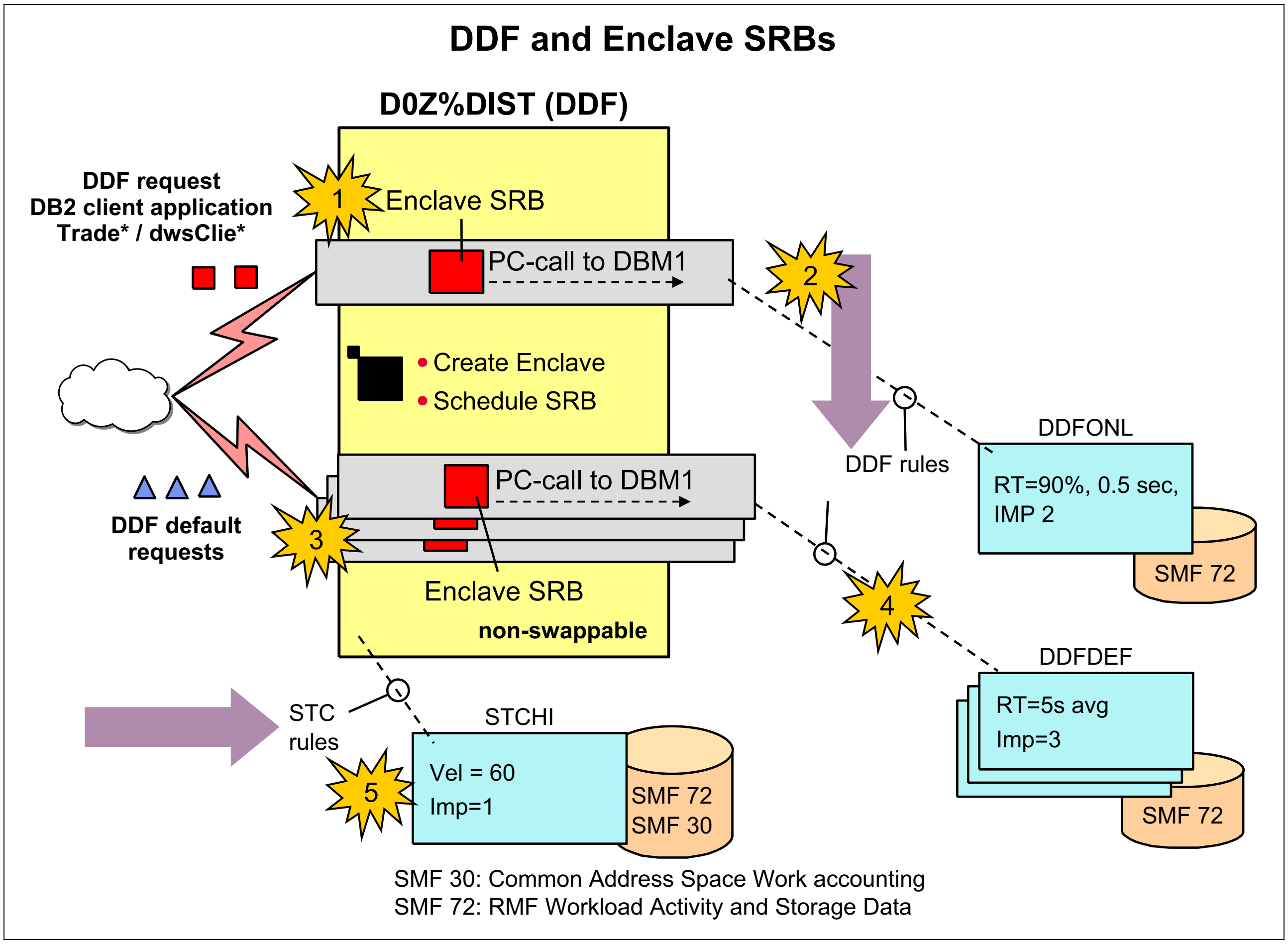


Figure 4-12 WLM DDF classification overview

1. When our WebSphere Application Server application connects to DB2 it provides its client application information referred to in Figure 4-10 on page 111. The DB2 distributed address space creates an enclave and schedules an enclave SRB. The enclave SRB uses a program call instruction to trigger processing in the DB2 database manager address space.
2. WLM considers the client application information to assign the performance goal referred to in service class DDFONL. WLM furthermore assigns the report class defined in the WLM policy shown in Figure 4-10 on page 111 which is useful when it comes to creating RMF workload activity reports.
3. All other DB2 DBATs will be classified using the data sharing group default service class and report class configuration referred to in the WLM policy shown in Figure 4-10 on page 111 under rule type SSC (subsystem collection).
4. Requests falling in rule type SSC will be classified using service class DDFDEF and report class RD0ZGDEF.
5. The DB2 system address spaces are classified using classification rules defined in subsystem type STC. For our environment these classification rules are discussed in “DB2 system service classification” on page 108.

When we tested the D0ZG_WASTestClientInfo data web service application we captured the SDSF enclave display output shown in Figure 4-13 to confirm that the WLM classification rule shown in Figure 4-10 on page 111 was correctly used by our runtime environment.

```
 Display  Filter  View  Print  Options  Search  Help
---------------------------------------------------------------------------
SDSF ENCLAVE DISPLAY  SC63     ALL                     LINE 1-13 (13)
COMMAND INPUT ===>                                            SCROLL ===>
NP   NAME             SSType Status   SrvClass Per PGN RptClass ResGroup
i    600000051A       DDF    ACTIVE   DDFONL     1     RDWSOZ
```

Figure 4-13   SDSF enclave display

From the SDSF panel provided in Figure 4-13 we issued the SDSF action character shown to obtain additional information about the enclave. This took us the panel shown in Figure 4-14.

```
 Enclave 600000051A on System SC63

 Subsystem type    DDF         Plan name         DISTSERV
 Subsystem name    D0Z1        Package name      SYSLH300
 Priority                      Connection type   SERVER
 Userid            WASCTX1     Collection name   JDBCAUTHTEST
 Transaction name              Correlation       db2jcc_appli
 Transaction class             Procedure name
 Netid                         Function name     DB2_DRDA
 Logical unit name             Performance group
 Subsys collection DB0ZG       Scheduling env
 Process name      dwsClientinformationDS
 Reset             NO
```

Figure 4-14   SDSF display enclave information

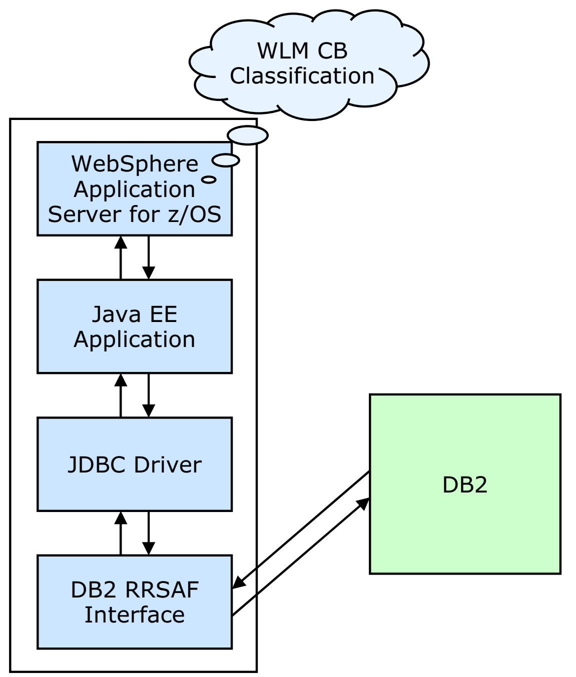
The information that we obtained through Figure 4-14 confirmed that the following runtime attributes were used because of our WLM service classification:

- Subsystem type DDF and subsystem name D0Z1 - the enclave was managed by the D0Z1DIST address space
- Subsystem collection DB0ZG - data sharing group name
- Process name dwsClientinformationDS - derived from DB2 clientApplicationInformation data source custom property setting.

Important: Always provide a classification rule in WLM. If you do not classify your DDF workload your DDF transactions will run unclassified in service class SYSOTHER, which has the lowest execution priority in your system. As a consequence transaction throughput of those applications will suffer from bad performance.

JDBC type 2 to connections

WebSphere Application Server applications connecting to DB2 using JDBC type 2 connections do not require WLM subsystem type DDF service classification rules as they communicate with DB2 through local DB2 RRSAF connections without going through the DDF address space. The SQL submitted over JDBC type 2 connections runs under the service class of the invoking Java application for which classification rules are provided in WLM subsystem type CB. The process of WLM classification of local JDBC type 2 connection is illustrated in Figure 4-15.



- DB2 SQL activity runs under dispatchable unit of invoker
  - WebSphere Application Server JDBC type 2
  - Inherited WLM service class of invoker
  - Priority and management of home unit
  - Service attributed back to invoker

Figure 4-15 WebSphere Application Server for z/OS JDBC type 2

For the WebSphere settings, see 5.3, “Configuring WebSphere Application Server for JDBC type 2 access” on page 222

DB2 WLM additional information

For more information about setting z/OS performance options for DB2 using z/OS Workload Manager refer to Chapter 4. z/OS performance options for DB2 of *DB2 10 for z/OS, Managing DB2 Performance*, SC19-2978.

4.1.4 Resource Recovery Services

Resource Recovery Services (RRS) is a z/OS subsystem running in its own address space. RRS is a critical resource that must be available to resource managers such as WebSphere MQ, CICS, IMS, WebSphere Application Server, DB2 and transactional VSAM to guarantee data integrity in a z/OS two-phase commit environment.

RRS is a prerequisite for DB2 for z/OS availability in a WebSphere Application Server environment. This is why you must not shut down RRS while DB2 and WebSphere Application Server are running, because if you do WebSphere Applications Server terminates and cannot be restarted until RRS has been restarted. As a result uncommitted units of recovery (UR) cannot be resolved in DB2 for as long as WebSphere Application Server is down. For this reason you should perform RRS shutdown only after resource managers such as DB2 and WebSphere Application Server have been quiesced. In case of RRS subsystem failure RRS must be restarted as quickly a possible which is usually assured through z/OS Automatic Restart Manager (ARM) or by other means of system automation.

For information about using and implementing RRS for high availability, see *z/OS Programming: Resource Recovery*, SA22-7616-11:available at http://www.ibm.com/systems/z/os/zos/bkserv/r13pdf.

Important: DB2 external stored procedures access DB2 through the RRSAF attachment interface which requires RRS to be available.

The DB2 JDBC driver (regardless whether you use JDBC type 2 or type 4 connections) transparently calls DB2 provided external stored procedures for metadata retrieval. Therefore, accessing DB2 for z/OS through JDBC already causes the requirement for the RRS subsystem to be available regardless of the JDBC connection type being used and regardless of the runtime environment your Java application is executing in.

DB2 RRS resource managers

DB2 for z/OS registers and deregisters its two resource managers with RRS during DB2 start and DB2 shut down. The DB2 resource managers our DB2 environment registers and deregisters with RRS are:

- RRS Attachment Facility (for instance, DSN.RRSATF.IBM.D0Z1 for DB2 member D0Z1)
- WLM controlled stored procedure address spaces (for instance, DSN.RRSPAS.IBM.D0Z1 for DB2 member D0Z1)

DB2 startup

External stored procedures and JDBC type 2 based applications communicate with RRS through the DB2 RRSAF attachment interface which ensures data integrity in case resource changes in other z/OS resource managers like IMS, CICS, WebSphere MQ are performed within the same unit of recovery and always for type 2 connections. To cater for this RRS requirement DB2 verifies the availability of RRS during startup and issues the messages shown in Figure 4-16.

```
DSN3029I  -D0Z1 DSN3RRRS RRS ATTACH PROCESSING IS AVAILABLE
DSNL512I  -D0Z1 DSNLILNR TCP/IP
```

Figure 4-16   DB2 RRS attach ok

If you are not sure whether the DB2 resource managers (RMs) are successfully registered with RRS you can verify their RRS state by running the z/OS command shown in Figure 4-17. To verify the RM state of all your DB2 members you need to issue the command once for each z/OS LPAR your DB2 members are running in.

```
D RRS,RM,SUM
ATR602I   18.01.34  RRS RM SUMMARY 351
RM NAME                         STATE             SYSTEM   GNAME
DSN.RRSATF.IBM.D0Z1             Run               SC63     SANDBOX
DSN.RRSPAS.IBM.D0Z1             Run               SC63     SANDBOX
```

Figure 4-17 DB2 start RRS RM state

DB2 shut down

During shut down DB2 deregisters its resource managers from RRS. When we shut down DB2 member D0Z1 we noticed the RRS deregistration messages shown in Figure 4-18.

```
-D0Z1 STOP DB2
DSNY002I  -D0Z1 SUBSYSTEM STOPPING
ATR169I RRS HAS UNSET EXITS FOR RESOURCE MANAGER
DSN.RRSATF.IBM.D0Z1 REASON: UNREGISTERED
ATR169I RRS HAS UNSET EXITS FOR RESOURCE MANAGER
DSN.RRSPAS.IBM.D0Z1 REASON: UNREGISTERED
```

Figure 4-18 DB2 RRS deregistration

We then verified the RRS state of the DB2 resource managers shown in Figure 4-18. As you can see in Figure 4-19 stopping DB2 member D0Z1 set the RRS resource manager state to a value of Reset.

```
D RRS,RM,SUM
ATR602I   18.29.44  RRS RM SUMMARY
RM NAME                         STATE             SYSTEM   GNAME
DSN.RRSATF.IBM.D0Z1             Reset             SC63     SANDBOX
DSN.RRSPAS.IBM.D0Z1             Reset             SC63     SANDBOX
```

Figure 4-19 RRS RM state upon DB2 shut down

Stopping RRS

If you stop RRS it deregisters from ARM and issues the system message shown in Figure 4-20.

```
ATR143I RRS HAS BEEN DEREGISTERED FROM ARM.
ASA2960I RRS SUBSYSTEM FUNCTIONS DISABLED. COMPONENT ID=SCRRS
```

Figure 4-20 Stop RRS

When a DB2 RRSAF application accesses DB2 while RRS is unavailable, the DB2 Resource Recovery Service Attachment Facility (RRSAF) interface returns error code and reason code information for the application to cater for that situation. The reason codes an application needs to take care of in case of RRS unavailability are shown in Table 4-2 on page 117.

Table 4-2 RRSAF reason codes

RRSAF reason code	Description
00C12219	The application program issued an SQL or IFI function request without completing CREATE THREAD processing. SQL or IFI requests cannot be issued until CREATE THREAD processing is complete. We experienced this reason code when using an implicit RRSAF connection to DB2. It indicates that the RRSAF CREATE THREAD processing failed due to RRS unavailability which subsequently caused a failure during SQL processing.
00F30091	The application program issued an RRSAF IDENTIFY function request, but RRS/MVS was not available.

WebSphere Application Server for z/OS considerations

As shown in Figure 4-21 uncontrolled RRS shutdown triggers termination of WebSphere Application Server for z/OS which causes additional complexity in terms of restart and recovery. To prevent others from causing such undesired situations we strongly recommend to restrict the use of RRS shut down to processes and procedures that cater for system management standards and data consistency. Such processes would not allow for RRS to be stopped while a WebSphere Application Server for z/OS instance or DB2 are running.

```
BBOT0024A RRS HAS BECOME UNAVAILABLE.  WEBSPHERE MUST BE RESTARTED.
BBOO0035W TERMINATING THE CURRENT PROCESS, REASON=C9C218F7.
BBOO0009E WEBSPHERE FOR Z/OS DAEMON SC63 ENDED ABNORMALLY, 310
REASON=C9C212C4.
BBOO0056E CONTEXT SERVICE 'CTXBEGC' FAILED WITH RETURN CODE=301.
```

Figure 4-21 WebSphere Application Server for z/OS and RRS termination

4.1.5 z/OS resource planning

When planning your DB2 environment, you need to have an idea of how much memory, processor capacity or disk storage your application is going to consume. To estimate this resource requirement you need to know the DB2 objects that are going to be created, how much data is going to be stored in DB2 tables and indexes, the SQL workload profile of your application, the amount of processor time an application invocation is going to consume, and the throughput your application is going to cause.

For Java applications or DB2 DRDA workloads you might want to consider adding zIIP or zAAP processor capacity to satisfy the additional processor requirement and to financially benefit from using these speciality engines. During pre-production stress testing you carefully monitor and tune your application aiming at reaching production like application throughput rates. As a result of pre-production stress testing and tuning you know which additional memory, processor and disk resources are required to run your application in your production environment. After these additional resources have been allocated to your production environment you are ready to promote your application to production level.

We created our WebSphere Application Server for z/OS and DB2 for z/OS runtime environment with system function testing in mind. The infrastructure we were provided with showed the following resource allocations:

- IBM z196 zEnterprise® 2817 Model 716
- 2 Parallel Sysplex z/OS 1.13 members, each with 8 GB real memory, 2 central processors (CP), 2 System z Application Assist Processors (zAAP), 2 System z Information Integration Processors (zIIP)
- 2 Coupling Facilities, each with 1607 MB internal storage and two shared processors
- 2 way DB2 10 for z/OS data sharing with one member running in each LPAR, configured for distributed workload balancing and with fault tolerant configuration
- WebSphere Application Server for z/OS cluster with two application server instances running, one in each LPAR
- DFSMS storage group set up with four 3390 model 9 volumes for DB2 user data and indexes

In the environment of our scenario we implemented DB2 and application objects of the WebSphere Daytrader application which we downloaded from
http://www14.software.ibm.com/webapp/download/preconfig.jsp?id=2011-06-08+10%3A34%3A22.216702R&cat=webservers&fam=&s=&S_TACT=&S_CMP=&st=52&sp=20

The Daytrader application consists of a WebSphere Applications Server application and a database model of 6 DB2 tables and 11 indexes. The scenario that we use causes just a few thousand table rows to be inserted per application execution. The tables are reset prior to application execution. Given these workload characteristics there was no need to perform more detailed capacity planning as our Parallel Sysplex infrastructure provided plenty of resources to efficiently deal with the anticipated data volume and workload throughput.

4.1.6 External storage configuration

DB2 user data requires DASD storage to be provided for DB2 table space and index space VSAM LDS allocation. Whenever you create a DB2 table space or index you want to be sure the underlying VSAM LDS data set is allocated in the volume pool that has been provided for your data. To ensure your user data volume pool is sufficiently sized and correctly used for data set allocations we recommend to provide for the following resources and configurations:

- Estimate space requirement for user table and index spaces
- DASD disk space and it DFSMS storage group setup
- Data set high level qualifier (HLQ) for VSAM LDS dat sets created for DB2 table spaces and index spaces.
- DB2 storage group referencing the HLQ for your DB2 table spaces and index spaces
- DFSMS automatic class selection (ACS) routines to assign the desired data set, management and storage attributes. The storage group ACS routine makes sure your DB2 table space or index space data set is going to be created in the volume pool that is dedicated to your DB2 data.

In the following sections we outline the major steps required for external storage configuration. For details. see *DB2 9 for z/OS and Storage Management*, SG24-7823.

Estimating space requirement

Before you create new tables and indexes in DB2 you should have an idea of the amount of disk space these new objects are going to use. For capacity planning purposes these space requirements should be discussed with your storage administrator so that your volume pool configuration can be changed to provide the additional disk space.

After the objects have been created and are operational you can use the tooling provided by DB2 to monitor space growth. The tools provided by DB2 supporting you in performing these tasks are:

- External stored procedure SYSPROC.ADMIN_DS_LIST
- DB2 real time statistics (RTS)
- External stored procedure SYSPROC.DSNACCOX

For a discussion of these tools refer to 4.3, "DB2 for z/OS configuration" on page 137

DFSMS storage group

The ISPF interactive storage management facility (ISMF) output shown in Figure 4-22 provides a list of system managed storage (SMS) storage groups defined for our DB0ZG data sharing group.

```
 Panel  List  Utilities  Scroll  Help
------------------------------------------------------------------------------
                               STORAGE GROUP LIST
Command ===>                                                  Scroll ===> HALF
                                                        Entries 1-8 of 8
                                                        Data Columns 3-6 of 48
CDS Name : ACTIVE

Enter Line Operators below:

     LINE       STORGRP  SG                VIO      VIO   AUTO
     OPERATOR   NAME     TYPE              MAXSIZE  UNIT  MIGRATE
    ---(1)----  --(2)--- -------(3)------  --(4)--  (5)-  --(6)---
                DB0ZARCH POOL              -------  ----  NO
                DB0ZCPB  COPY POOL BACKUP  -------  ----  --------
     LISTV      DB0ZDATA POOL              -------  ----  NO
                DB0ZIMAG POOL              -------  ----  NO
                DB0ZLOG1 POOL              -------  ----  NO
                DB0ZLOG2 POOL              -------  ----  NO
                DB0ZMISC POOL              -------  ----  NO
                DB0ZTARG POOL              -------  ----  NO
```

Figure 4-22 ISMF list storage group

We created SMS storage group DB0ZDATA to provide a volume pool for the disk space required to store the Daytrader DB2 tables and indexes. The other storage groups shown are for DB2 archive log data sets, image copy data sets, active log data sets as well as for other runtime data sets. We defined the DB0ZCPB COPY POOL BACKUP pool to provide the infrastructure for DB2 system backup and restore. We then used the ISMF LISTVOL command to obtain a list of volumes available in storage group DB0ZDATA.

The volume list is shown in Figure 4-23.

```
 Panel  List  Utilities  Scroll  Help
------------------------------------------------------------------------------
                                             VOLUME LIST
Command ===>                                                    Scroll ===> HALF
                                                          Entries 1-4 of 4
Enter Line Operators below:                               Data Columns 3-8 of 43

     LINE       VOLUME FREE       %     ALLOC      FRAG  LARGEST    FREE
     OPERATOR   SERIAL SPACE      FREE  SPACE      INDEX  EXTENT     EXTENTS
    ---(1)----  -(2)-- ---(3)---  (4)-  ---(5)---  -(6)-  ---(7)---  --(8)--
                0Z9B07  7913094K    95    401407K      0   7911102K       13
    lds         0Z9B86  8021885K    96    292616K      1   8009877K       14
                0Z9287  8080430K    97    234071K      0   8079600K        7
                0Z9309  7796114K    94    518387K      0   7788976K       14
```

Figure 4-23 ISMF list volumes

From the volume list shown in Figure 4-23 we then issued the user command **lds** (list data set) to display the data sets stored on volume 0Z9B86. **lds** is a user provided REXX program that uses ISPF services to display a volume related data set list which can be useful if you want to check out whether the volume data set placement works as planned. The REXX source is shown in Example 4-3.

Example 4-3 REXX program lds

```
/* REXX */
/* josef.klitsch@ch.ibm.com */
 TRACE OFF
 ADDRESS ISPEXEC
 "VGET (COBJ  ) ASIS"
 "LMDINIT LISTID(LISTEN)            VOLUME("COBJ")"
 "LMDDISP LISTID("LISTEN") VIEW(VOLUME) CONFIRM(YES)"
 "LMDFREE LISTID("LISTEN") "
RETURN
```

As a result we obtained the data set list shown in Figure 4-24.

```
 Menu  Options  View  Utilities  Compilers  Help
------------------------------------------------------------------------------
DSLIST - Data Sets on volume 0Z9B86                                Row 1 of 131
Command ===>                                                    Scroll ===> PAGE

Command - Enter "/" to select action                  Message          Volume
------------------------------------------------------------------------------
         DB0ZD.DSNDBD.DBTR8074.ORDERRAC.I0001.A001                     0Z9B86
```

Figure 4-24 List of data sets by volume

DB2 storage group and data set HLQ usage

Table space and index space creation triggers VSAM LDS data set creation in DB2. For data set creation DB2 obtains the data set HLQ from the DB2 storage group referenced in the create table space and create index DDL statement. In our environment we use DB2 storage group GR248074 through which DB2 uses a data set HLQ of DB0ZD for VSAM LDS data set creation. The DDL that we used to create our DB2 storage group is shown in Example 4-4.

Example 4-4 Create storage group and table space DDL

```
CREATE STOGROUP GR248074
  VOLUMES("*")
  VCAT DB0ZD ;

CREATE TABLESPACE TSACCEJB
  IN DBTR8074
  USING STOGROUP GR248074 ;
```

Our DFSMS configuration places data sets with an HLQ of DB0ZD on one of the volumes available in DFSMS storage group DB0ZDATA. If the volume chosen becomes full DB2 automatically adds a volume to the VSAM LDS data set definition allowing the data set to extend to the additional volume. If all volumes available in DFSMS storage group DB0ZDATA become full DFSMS configuration options can be used to overflow to another volume pool or to perform an online volume pool change to supply additional disk space to support high availability.

If you want to read more about DB2 and DFSMS storage, refer to *DB2 9 for z/OS and Storage Management*, SG24-7823 http://www.redbooks.ibm.com/abstracts/sg247823.html?Open.

Storage group ACS routine and data set placement

When you create a table or index space DB2 defines a VSAM LDS data set using the HLQ provided by the DB2 storage group that you use in your CREATE TABLESPACE or INDEX SQL DDL statement. Data set creation triggers DFSMS automatic class selection (ACS) routine processing. In case of the CREATE TABLESPACE DDL shown in Example 4-4 the storage group ACS routine assigns storage group DB0ZDATA to be used for VSAM LDS data set allocation. During data set allocation DFSMS places the data set on one of the volumes shown in the volume list in Figure 4-23 on page 120.

The ACS routine processing flow for DB2 objects that we created in our environment is illustrated in Figure 4-25.

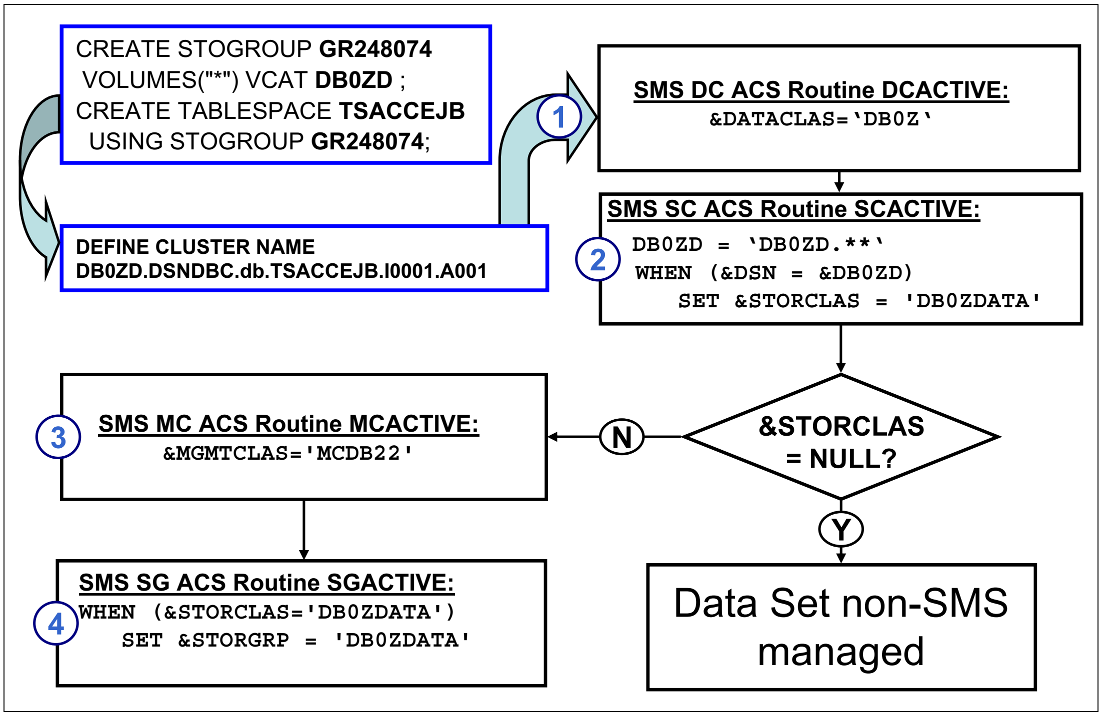


Figure 4-25 DFSMS ACS routine processing

1. The CREATE TABLESPACE DDL triggers the creation of a VSAM LDS cluster. The cluster name uses the data set HLQ provided by DB2 storage group GR248074 which causes the data class ACS routine to assign the DB0Z data class. The DB0Z data class provides data set attributes that are required to support VSAM extended format and extended addressability. These attributes are recommended to provide high availability and to support new features available with modern disk technology.
2. Next the storage class ACS routine receives control and assigns storage class DB0ZDATA. A DFSMS storage class controls data set level usage of storage performance attributes provided by DFSMS. For instance, in our environment the DB0ZDATA storage class assures the use of parallel access volumes (PAV) which is highly recommended to alleviate I/O queuing in case of high I/O currency on the same physical volume. After a non-null value has been assigned the data set to be created is going to be DFSMS managed.
3. Next the management class ACS routine receives control. The management class controls the actions that are to be taken by the DFHSM space management cycle. The management class used for our table and index spaces ensures that no DFHSM space management activity is taken that can have a negative impact on data availability and data integrity. For instance, the management class used ensures that our table and index space related VSAM LDS data sets are not migrated, deleted or backed up by DFHSM during space management cycle.
4. Finally the storage group ACS routine receives control. As mentioned before the DFSMS storage group provides a group of volumes the data set creation process can transparently choose from.

4.1.7 UNIX System Services file system configuration

If your production environment depends on the IBM Data Server Driver for JDBC to be available on z/OS you need to design your infrastructure to provide high availability for this infrastructure component.

SMP/E installs the DB2 command line processor (CLP) and IBM Data Server Driver for JDBC and SQLJ related UNIX System Services files into IBM eServer™ zSeries® File ZSystem (zFS) data sets. Copies of these SMP/E controlled zFS data sets are rolled out into target runtime environments to provide software upgrades or to participate in rolling maintenance processes.

Because you should not replace the IBM Data Server Driver for JDBC installation while it is being used by applications you need to design your UNIX System Services related DB2 software update strategy to support seamless software updates for installing and backing out JDBC driver changes. To address this problem we carried out the following activities:

- Provide one file system directory structure for each JDBC driver level we want to support
- Use UNIX System Services symbolic links to connect the appropriate JDBC driver level with a logical path name our application uses to load the JDBC driver

For an up to date list of driver levels currently supported, refer to the following information:

- DB2 for z/OS: http://www-01.ibm.com/support/docview.wss?uid=swg21428742
- DB2 LUW: http://www-01.ibm.com/support/docview.wss?uid=swg21363866

JDBC driver level related file system directories

Our runtime environment uses the UNIX System Services directories shown in Figure 4-26 to provide the different JDBC driver levels our application might have to use.

```
/pp/db2v10/
       +---d110809/    <---- zFS file: OMVS.DSNA10.BASE.D110809
       +---d120320/    <---- zFS file: OMVS.DSNA10.BASE.D120320
       +---d120719/    <---- zFS file: OMVS.DSNA10.BASE.D120719
```

Figure 4-26 UNIX System Services directories for JDBC driver level related rollout

Under directory /pp/db2v10 we created directories d110809, d120320, d120719 each of them representing a different software maintenance level. We then used these directories as mount point directories for mounting the corresponding zFS file data sets. The zFS files shown in Figure 4-26 are data sets that we previously copied from our SMP/E environment. In our runtime environment each of these mount point directories contains the directory structure shown in Figure 4-27.

```
/pp/db2v10/
       +---d120719/
             +-----base   <---- DB2 command line processor
             +-----jdbc   <---- IBM Data Server Driver for JDBC
             +-----mql    <---- MQ listener
```

Figure 4-27 DB2 product related directories

Our z/OS administrator configured UNIX System Services to automatically mount the zFS files at IPL time. To verify the mount status of our DB2 related zFS files we ran the z/OS system command shown in Example 4-5.

Example 4-5 Display OMVS DB2 file systems

```
D OMVS,F,NAME=OMVS.DSNA10*
BPXO045I 05.36.29 DISPLAY OMVS 059
OMVS     0010 ACTIVE             OMVS=(3A)
TYPENAME   DEVICE ----------STATUS----------- MODE  MOUNTED    LATCHES
ZFS           162 ACTIVE                      READ  10/23/2012  L=146
  NAME=OMVS.DSNA10.BASE.D120719                     21.26.08    Q=146
  PATH=/pp/db2v10/d120719
  OWNER=SC63     AUTOMOVE=Y CLIENT=N
ZFS           161 ACTIVE                      READ  10/23/2012  L=145
  NAME=OMVS.DSNA10.BASE.D120320                     21.26.08    Q=0
  PATH=/pp/db2v10/d120320
  OWNER=SC63     AUTOMOVE=Y CLIENT=N
ZFS           160 ACTIVE                      READ  10/23/2012  L=144
  NAME=OMVS.DSNA10.BASE.D110809                     21.26.08    Q=144
  PATH=/pp/db2v10/d110809
  OWNER=SC63     AUTOMOVE=Y CLIENT=N
```

For each mounted zFS file the command output shown in Example 4-5 confirms mount status, zFS file data set name, the mount point and the mount mode. In our environment we mounted each of the DB2 zFS files read only because this is recommended for performance reasons in case no write access is required.

Use of Symbolic links for loading the JDBC driver

JDBC applications should not be configured to load the JDBC driver directly from the installation directory, because if the install directory changes your application JDBC driver configuration needs to be changed.

To address this problem our applications use a data sharing group related path name to load the JDBC driver. We run the command shown in Example 4-6 to create an UNIX System Services symbolic link that connects the current JDBC driver installation directory with the data sharing group related logical path name.

Example 4-6 JDBC create UNIX System Services symbolic link

```
ln  -s /pp/db2v10/d120719 /usr/lpp/db2/d0zg
```

We than ran the command shown in Example 4-6 to verify which installation directory our data sharing group logical path name is connected with.

```
ls -l /usr/lpp/db2/d0zg
lrwxrwxrwx    /usr/lpp/db2/d0zg -> /pp/db2v10/d120719
```

Figure 4-28 Verify JDBC symbolic link

In case we need to fall back to the previous JDBC driver level we simply swap the symbolic link as shown in Example 4-7 on page 125. z/OS JDBC applications do not need to change their JDBC configuration because they use the data sharing group related path name for loading the JDBC driver.

Example 4-7   JDBC swap symbolic link

```
rm /usr/lpp/db2/d0zg
ln -s /pp/db2v10/d120320 /usr/lpp/db2/d0zg
```

WebSphere Application Server considerations

We used the symbolic link shown in Example 4-6 on page 124 in the WebSphere Application Server JDBC driver path and JDBC driver native path configuration.

For more information about performing this configuration, refer to 5.2.2, “Defining environment variables at the location of the IBM Data Server Driver for JDBC and SQLJ classes for JDBC type 4 connectivity” on page 213 and 5.3.2, “Defining environment variables to the location of the IBM Data Server Driver for JDBC and SQLJ classes for JDBC type 2 connectivity” on page 228.

If you run multiple instances of WebSphere Application Server for z/OS you might want to consider to use application server specific symbolic links for loading the JDBC driver. This provides the flexibility you might need in case an application server instance is bound to using a specific JDBC driver level due to failures that were introduced by a new JDBC driver installation.

4.1.8 Monitoring infrastructure

When you operate DB2 for z/OS you want be aware whether application and system resource usage is in line with the service level agreements (SLA) and within the estimated resource capacity. You furthermore want to be able to detect and analyze system and application growth to plan additional resource capacity. In case of performance problems you want to be able to track down its root cause to enable you to decide on the performance tuning that needs to be performed. Being able to perform these tasks requires a monitoring infrastructure to be in place.

In 8.1, “Performance monitoring” on page 362, we provide a more general overview of performance monitoring.

In Appendix A, “DB2 administrative task scheduler” on page 483. we describe the administrative task scheduler (ADMT) setup to trigger batch jobs, DB2 commands, and for autonomic statistics monitoring.

To support DB2 system and application monitoring we implemented the following monitoring infrastructure:

Software stack

For reporting and analysis we installed the following software:

- IBM OMEGAMON XE for DB2 PE on z/OS (OMPE). We describe this topic in 4.4, “Tivoli OMEGAMON XE for DB2 Performance Expert for z/OS” on page 201.
- IBM Resource Measurement Facility™ (RMF) on z/OS

SMF Browser for WebSphere Application Server for z/OS to report on SMF type 120 records for which you can download from
https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=zosos390.

For details, see Appendix E, “SMF 120 records subtypes 1, 3, 7, and 8” on page 545.

SMF records to be collected

We configured our environment to collect the following SMF records:

- RMF, DB2 and WebSphere Application Server information. We automatically archive the SMF data whenever a SMF data set switch occurs.
- DB2 statistics and RMF collection interval to be set to one minute to be able to line up the RMF data with the DB2 statistics data. Since DB2 10 for z/OS, the statistics traces are collected at one minute intervals
- DB2 to send the following statistics and accounting traces to the SMF trace destination:
  - Statistics trace classes 1, 3, 4, 5, 6, and 7
  - Accounting trace classes 1, 2, 3, 7, and 8
- DB2 audit authorization failure policy to be activated during DB2 startup. Collecting these information enables you to quickly identify the root cause of authorization failures. We ran the SQL statement shown in Example 4-8 to activate the policy.

Example 4-8 Define audit policy for authorization failures

```
INSERT INTO SYSIBM.SYSAUDITPOLICIES
   (AUDITPOLICYNAME, CHECKING,DB2START)
VALUES('AUTHFAIL','A','Y');
```

- DB2 trace IFCID 318 to be activated during DB2 member startup to enable the collection of global statement cache statistics. We configured the administrative scheduler to issue the DB2 command shown in Example 4-9 within DB2 start processing.

Example 4-9 Start IFCID 318

```
-START TRACE (P) CLASS(30) DEST(SMF)  IFCID(318)
```

4.1.9 WebSphere Application Server and DB2 security

Our WebSphere Application Server environment accesses DB2 through JDBC type 2 and JDBC type 4 connections and uses dynamic SQL to perform database changes. WebSphere Application Server authenticates to DB2 using its user ID and password which is provided in the data source definition. In the following discussion the application server user ID and password are also referred to as middle tier’s user ID and password.

We examine:

- Authentication in a three-tier architecture
- Authentication in a three-tier architecture using DB2 trusted context

Authentication in a three-tier architecture

WebSphere Application Server operates in a three-tier application model in which it represents the middleware layer. In that three-tier application model the application server authenticates users running client applications, authenticates to DB2 using its middle tier user ID and password and manages interactions with the database server. The privileges of the middle tier user ID are checked when accessing the database. This includes access that is performed on behalf of end-users.

Figure 4-29 on page 127 visualizes the process of client authentication in a three-tier WebSphere Application Server environment.

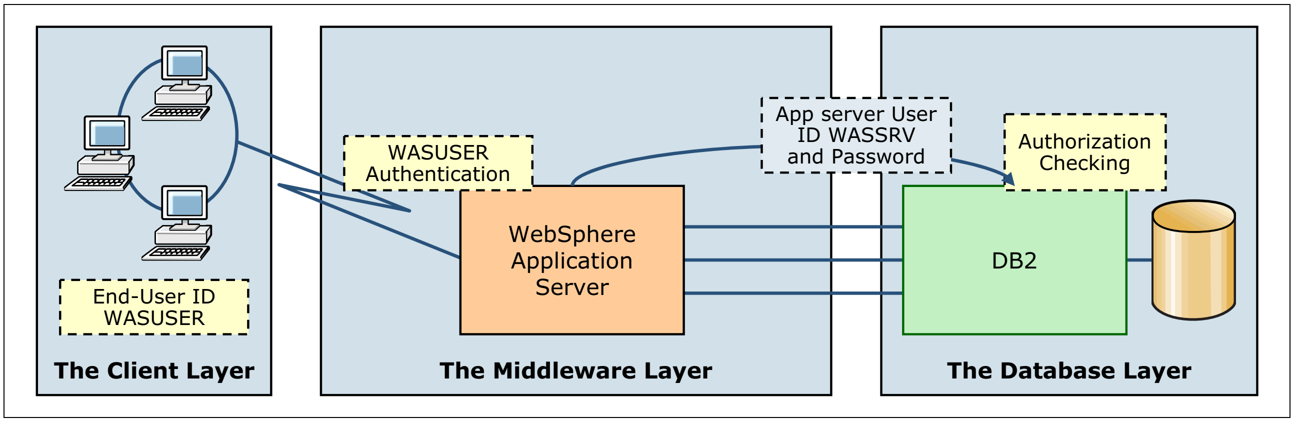


Figure 4-29 WebSphere Application Server three-tier environment

In the scenario illustrated in Figure 4-29, end-user WASUSER has been authenticated by the application server and is connected to DB2. The DB2 connection uses the application server's end-user credentials (user ID WASSRV provided in data source authentication related properties) for DB2 authentication and authorization checking. Therefore, SQL requests submitted by WASUSER are executed in DB2 using the application server's user credentials (user ID WASSRV).

Because all SQL access is performed under the middle tier's user ID, the three-tier application model causes the following issues:

- Loss of end-user identity in DB2.
- Loss of control over end-user access of the database.
- Diminished DB2 accountability.
- The middleware server's authorization ID (AUTHID WASSRV) needs the privileges to perform all requests from all end-users.
- If the middleware server's security is compromised, so is that of the database server.

Re-establishing a new connection every time the user ID changes does not provide a feasible solution due to the high performance overhead that would cause.

Let us review the illustration provided in Figure 4-30 to discuss the three-tier authentication scenario shown in Figure 4-29 on page 127. WebSphere Application Server handles the connection to DB2 using the application server's user credentials (user ID WASSRV). Because no trusted context is used WASSRV's database privileges are checked for SQL access on behalf of WASUSER.

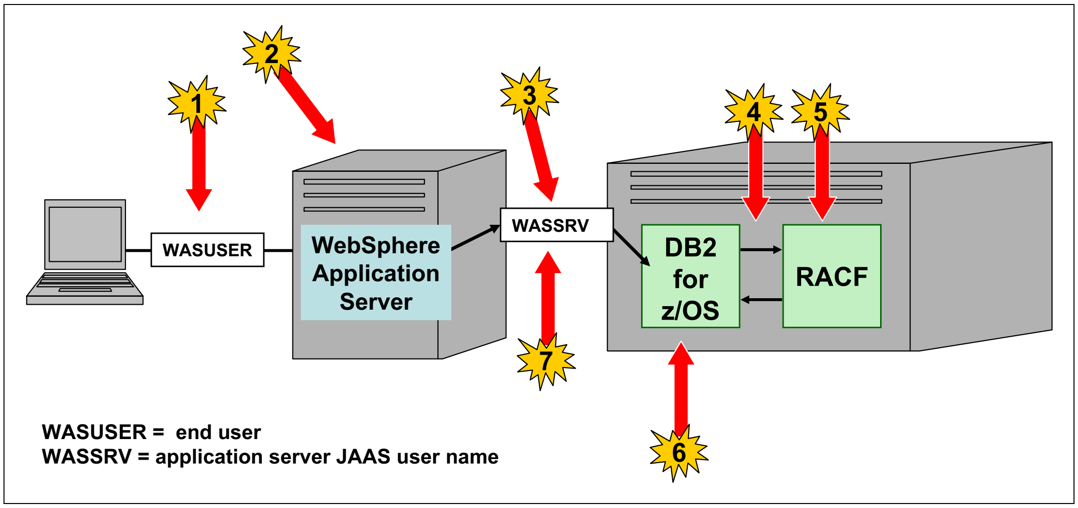


Figure 4-30 Three-tier authentication process

1. End-user logs on with user ID WASUSER and password
2. End-user WASUSER is authenticated by WebSphere Application Server
3. The application server requests a DB2 connection using user ID WASSRV and password
4. DB2 calls RACF to authenticate WASSRV
5. RACF verifies whether WASSRV is authorized to access DB2
6. Connection exit routine assigns WASSRV as the primary authorization id and as the CURRENT SQLID. Secondary authorization IDs may also be assigned. The connection has been established.
7. WASSRV's database privileges are checked for SQL access on behalf of WASUSER.

Authentication in a three-tier architecture using DB2 trusted context

To explain the benefits of using DB2 trusted contexts in a WebSphere Application Server environment we extend the scenario illustrated in Figure 4-30 to use the data source provided user ID of WASSRV for creating a DB2 connection and to use the application server authenticated user WASUSER for SQL authorization. In the scenario we configured the application server data source to support trusted connections (see 5.7, "Configuring the J2C authentication alias" on page 270.)

We then created the DB2 trusted context by running the SQL DDL shown in Figure 4-31 on page 129.

```
CREATE ROLE WASSRVROLE; 1
CREATE ROLE WASUSERROLE;
GRANT EXECUTE ON FUNCTION DB2R3.GRACFGRP TO ROLE WASUSERROLE; 2
CREATE TRUSTED CONTEXT CTXWASSRV
  BASED UPON CONNECTION USING SYSTEM AUTHID WASSRV 3
  DEFAULT ROLE WASSRVROLE
  WITHOUT ROLE AS OBJECT OWNER
  ENABLE
  NO DEFAULT SECURITY LABEL
  ATTRIBUTES (
     ENCRYPTION 'NONE',
     ADDRESS 'wtsc64.itso.ibm.com', 3
     ADDRESS 'd0z1.itso.ibm.com',
     ADDRESS 'wtsc63.itso.ibm.com',
     ADDRESS 'd0z2.itso.ibm.com'
  )
  WITH USE FOR WASUSER ROLE WASUSERROLE WITHOUT AUTHENTICATION; 4
```

Figure 4-31 Create trusted context

1. The roles used in the trusted context must exist prior to trusted context creation.
2. Role WASUSERROLE is granted to execute function DB2R3.GRACFGRP, because the UDF is invoked in our application scenario. User ID WASSRV does not require the execute or any other DB2 object privilege, because we use WASSRV just for DB2 connection creation. Any DB2 object privilege required within the trusted context needs to be granted to role WASUSERROLE. Role WASSRVROLE is not supposed to access any DB2 object and therefore holds no privilege in DB2.
3. The trusted context shown in Figure 4-31 is for WebSphere Application Server JDBC type 4 connections where the connection user ID (provided in the data source authentication property) matches the SYSTEM AUTHID of WASSRV and where the external entity runs on one of the IP hosts referred to by the domain names provided in the trusted context ADDRESS attributes.
4. Because the authenticated user matches authorization ID WASUSER DB2 assigns role WASEUSERROLE. In case the application server asks DB2 to perform an authorization ID switch for a user that does not match one of the user IDs specified in the trusted context WITH USE FOR clause the request fails with DB2 SQLCODE -20361. In that situation we observed the WebSphere Application Server error message shown in Figure 4-32.

```
+J2CA0056I: The Connection Manager received a fatal connection error
from the Resource Adapter for resource jdbc/Josef. The exception is:
 com.ibm.db2.jcc.am.DisconnectRecoverableException:
 Ýjcc¨Ýt4¨Ý2040¨Ý11215¨Ý3.64.82¨ An error occurred during a deferred
 connect reset and the connection has been terminated.  See chained
 exceptions for details. ERRORCODE=-4499,
 SQLSTATE=null:com.ibm.db2.jcc.am.SqlException:
 Ýjcc¨Ýt4¨Ý20130¨Ý12466¨Ý3.64.82¨ Trusted user switch failed.
 ERRORCODE=-4214,
 SQLSTATE=null:com.ibm.db2.jcc.am.SqlSyntaxErrorException:
```

Figure 4-32 Application server error message trusted user switch failed

Using the trusted context that we define in Figure 4-31 on page 129 our WebSphere Application Server scenario performs the following processing steps:

1. User logs on with user ID WASUSER and password (Figure 4-33).

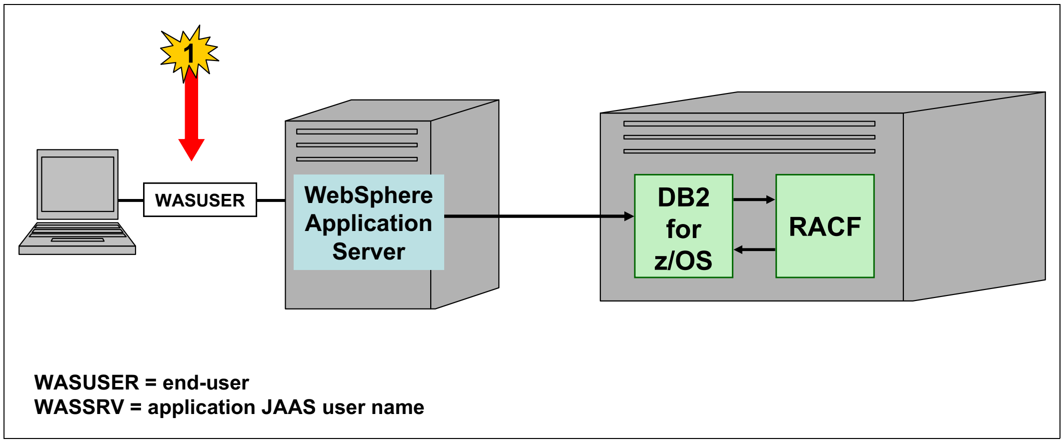


Figure 4-33 Step 1: Trusted context three tier authentication

2. User WASUSER is authenticated by WebSphere Application Server (Figure 4-34).

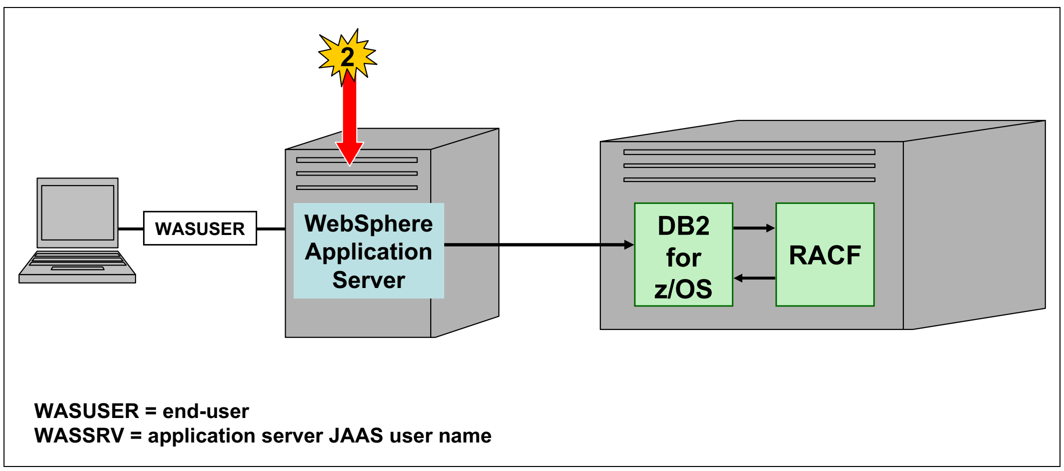


Figure 4-34 Step 2: Trusted context three tier authentication

3. The application server requests a DB2 connection using user ID WASSRV and its password (Figure 4-35 on page 131).

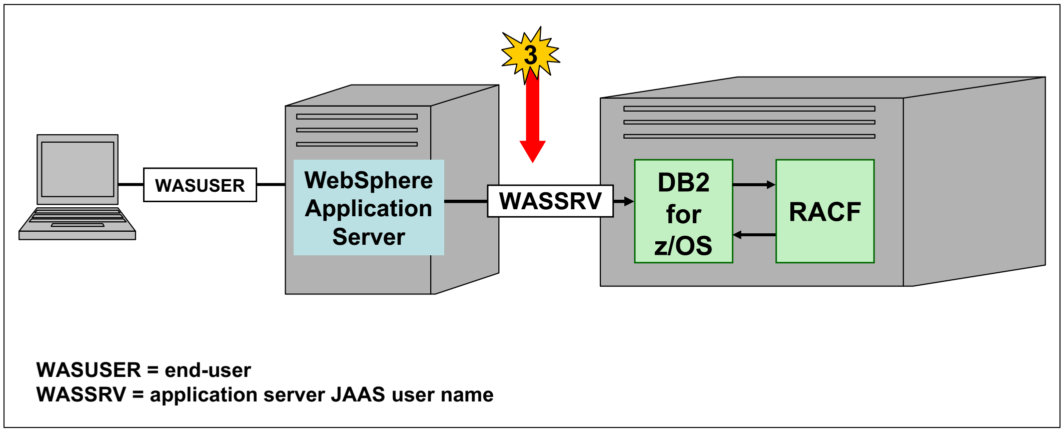


Figure 4-35 Step 3: Trusted context three tier authentication

4. DB2 calls RACF to authenticate WASSRV (Figure 4-36).

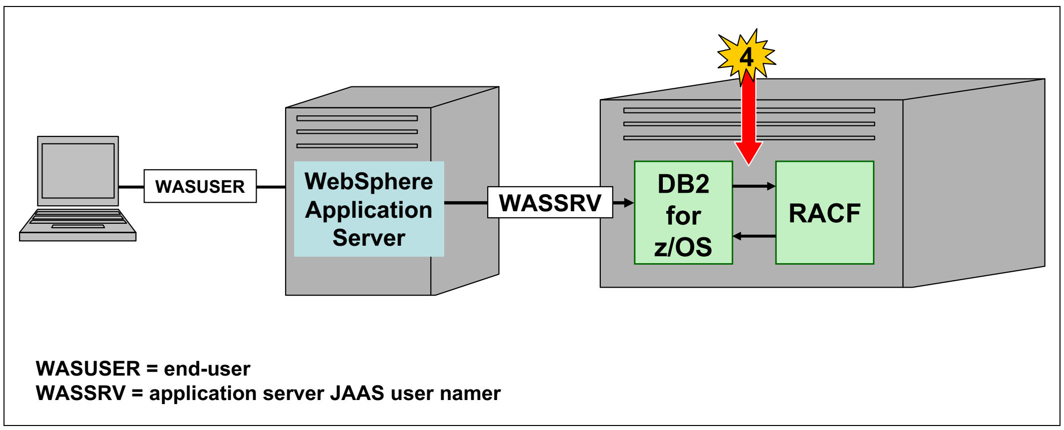


Figure 4-36 Step 4: Trusted context three tier authentication

5. RACF verifies whether WASSRV is authorized to access DB2 (Figure 4-37).

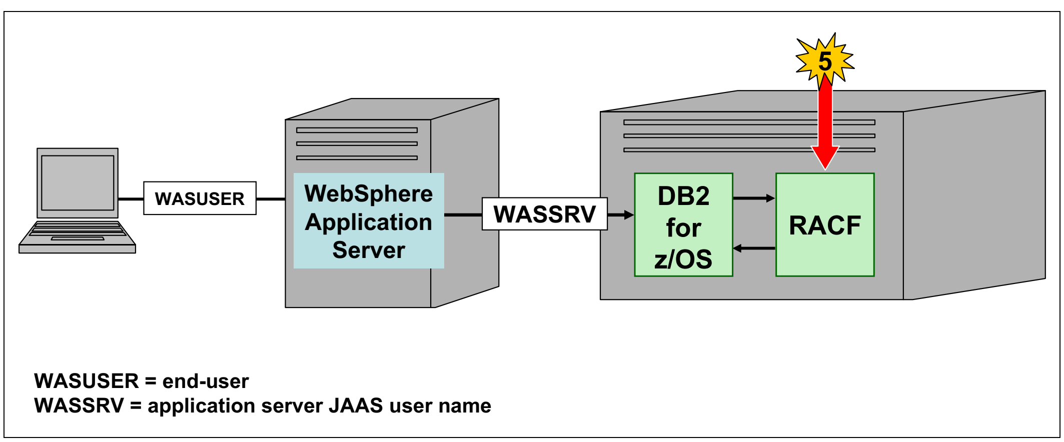


Figure 4-37 Step 5: Trusted context three tier authentication

6. Connection exit routine assigns WASSRV as the primary authorization id and as the CURRENT SQLID. Secondary authorization IDs may also be assigned (Figure 4-38).

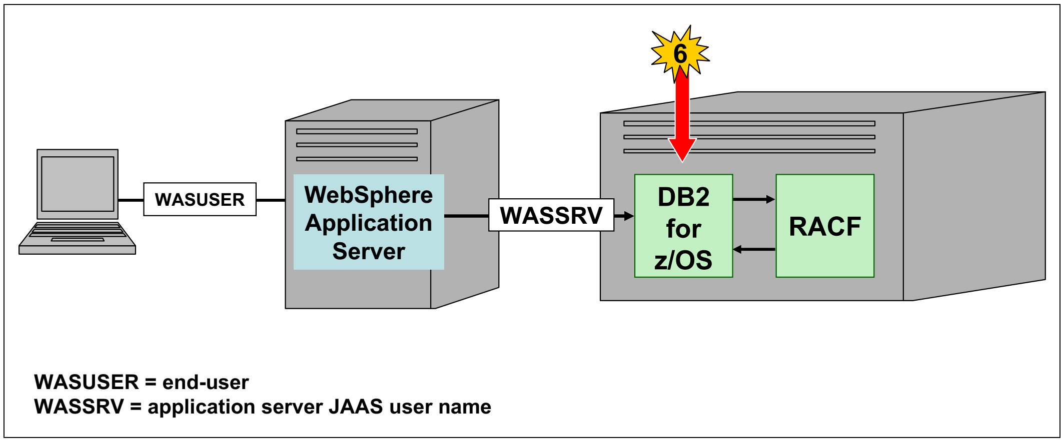


Figure 4-38 Step 6: Trusted context three tier authentication

7. DB2 looks for a trusted context with system authorization id WASSRV and validates the attributes of the context (for instance, SERVAUTH, ADDRESS, ENCRYPTION) (Figure 4-39). Depending on the trusted context DEFAULT ROLE attribute a role may also be assigned.

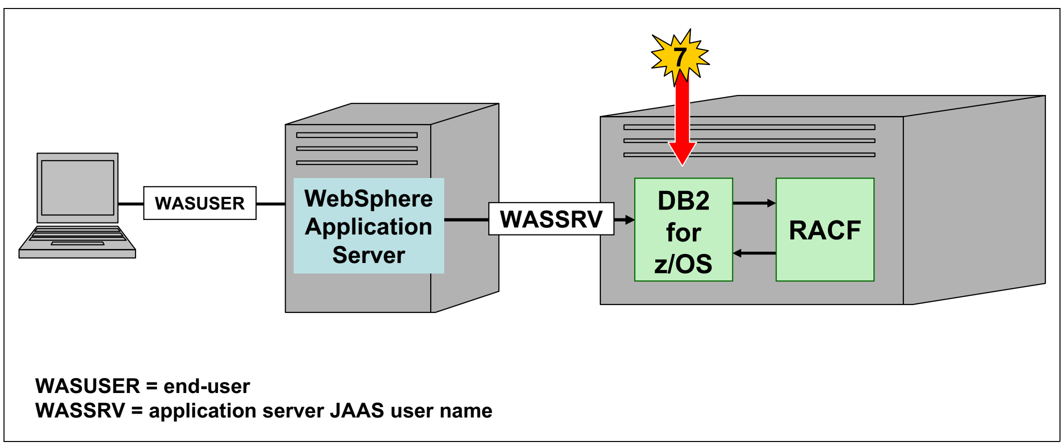


Figure 4-39 Step 7: Trusted context three tier authentication

8. The connection with user WASSRV as connection owner has been established (Figure 4-40 on page 133).

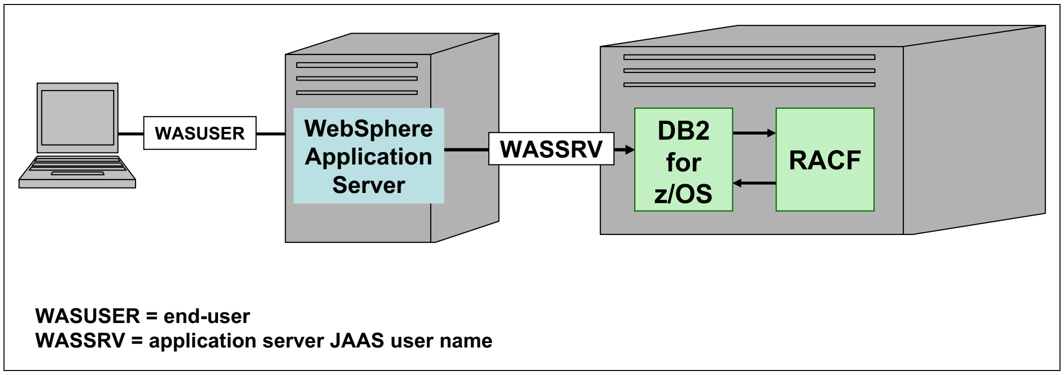


Figure 4-40 Step 8: Trusted context three tier authentication

9. WebSphere issues a switch user request using WASUSER (Figure 4-41). This requires no application code change. It is all implemented by the application server configuration.

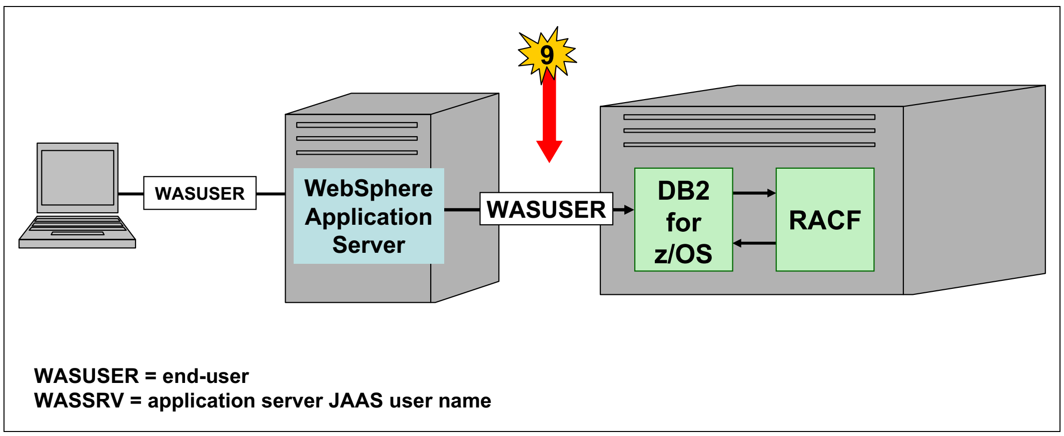


Figure 4-41 Step 9: Trusted context three tier authentication

10. DB2 determines whether WASUSER is allowed to switch to its authorization ID (Figure 4-42). In our scenario WASUSER is allowed to switch because the trusted context CTXWASSRV contains user ID WASUSER in its FOR AUTHID clause. If the authenticated user is not allowed to switch to its authorization ID DB2 aborts the request and returns SQLCODE -20361 to the application.

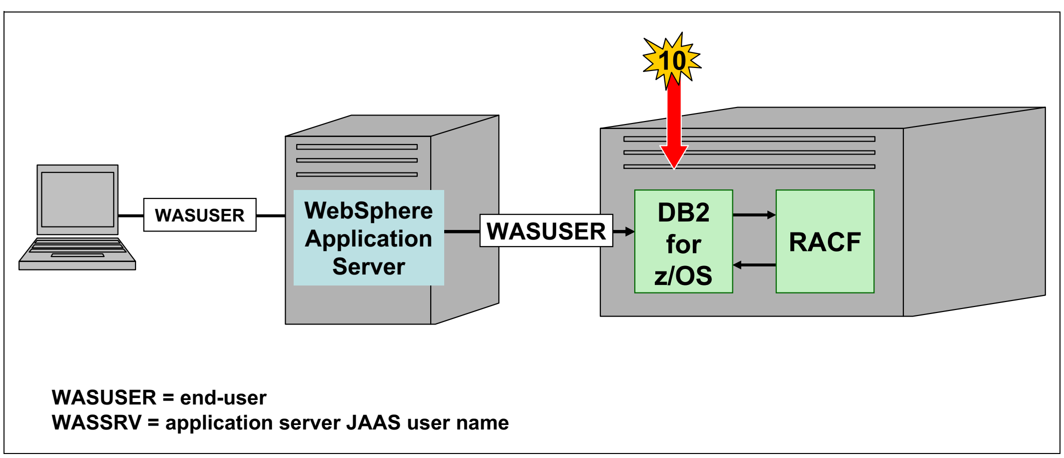


Figure 4-42 Step 10: Trusted context three tier authentication

11. Depending on the trusted context AUTHENTICATION attribute DB2 calls RACF to authenticate WASUSER (Figure 4-43). In the trusted context referred to in Figure 4-31 on page 129 we trigger RACF authentication for authorization ID WASUSER.

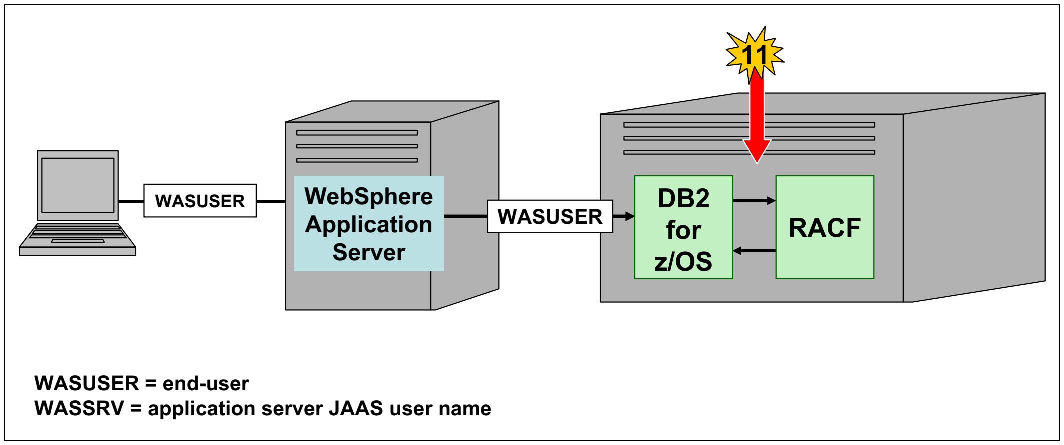


Figure 4-43 Step 11: Trusted context three tier authentication

12. The connection exit routine assigns WASUSER as the primary authorization ID and as the CURRENT SQLID (Figure 4-44). Secondary authorization IDs may also be assigned. DB2 assigns role WASUSERROLE which is going to be used for checking DB2 o
bject authorization.

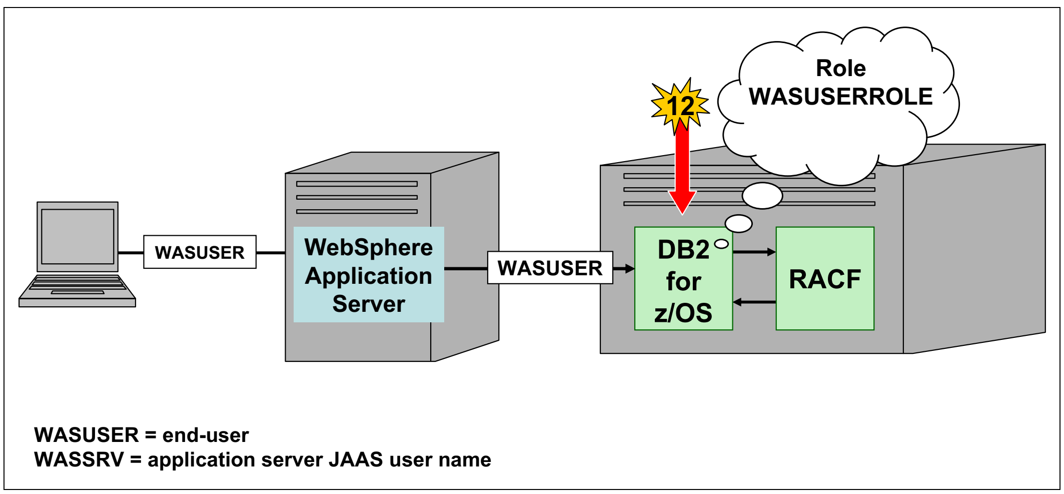


Figure 4-44 Step 12: Trusted context three tier authentication

13. The connection has been initialized using WASUSER as primary authorization ID and with role WASUERROLE assigned (Figure 4-45 on page 135). From now on DB2 uses role WASUSERROLE for checking SQL access authorization.

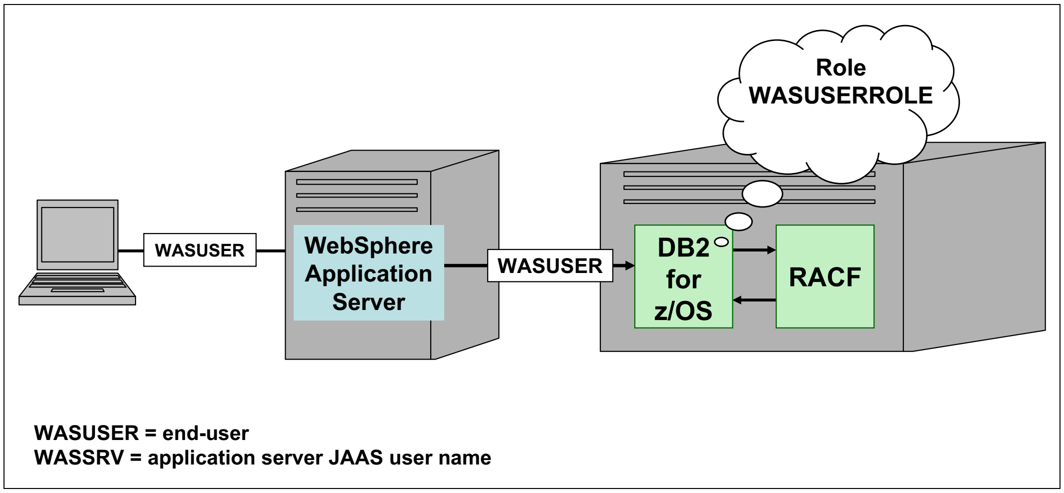


Figure 4-45 Step 13:Trusted context three tier authentication

14. While the application was running we collected the DB2 command output shown in Figure 4-46 to confirm that the CTXWASSRV trusted context was used by DB2 to establish a trusted connection and that DB2 performed an authorization ID switch for user ID WASUSER which resulted in the assignment of DB2 role WASUSERROLE. In that respect the command output shown in Figure 4-46 confirms the following trusted context related facts:

    a. The application server successfully establishes a trusted connection using the DB2 trusted context that we create in Figure 4-31 on page 129.

    b. The trusted context system authid matches with the application server JAAS provided username of WASSRV.

    c. Because WASUSER is identical with the user that has been authenticated by the application server, DB2 performs an authorization ID switch to WASUSER and assigns DB2 role WASUERROLE.

```
-DIS THREAD(SERVER) SCOPE(GROUP)
DSNV473I  -D0Z2 ACTIVE THREADS FOUND FOR MEMBER: D0Z1
 NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
 SERVER   SW *     7 db2jcc_appli WASUSER DISTSERV 0084   427
 V485-TRUSTED CONTEXT=CTXWASSRV, a
      SYSTEM AUTHID=WASSRV,b
      ROLE=WASUSERROLE c
  V437-WORKSTATION=WTSC64, USERID=WASSRV, b
       APPLICATION NAME=dwsClientinformationDS
  V441-ACCOUNTING=JCC03640WTSC64            dwsClientinformation
  V429 CALLING FUNCTION=DB2R3.GRACFGRP,
       PROC=        , ASID=0000, WLM_ENV=DSNWLMDB0Z_GENERAL
  V482-WLM-INFO=DDFONL:1:2:550
  V445-G90C0609.M72E.CA71F39F97F6=427 ACCESSING DATA FOR
   (  1)::9.12.6.9
  V447--INDEX SESSID           A ST TIME
  V448--(  1) 39000:26414      W S2 1231407175013
```

Figure 4-46 Step 14: Trusted context three tier authentication

To allow for the output in Figure 4-46 on page 135 to be collected while the application is running we issued the DB2 command shown in Example 4-10 to temporarily stop UDF DB2R3.GRACFGRP prior to application execution. While the STOP FUNCTION command is in effect, the attempt to execute the UDF is queued giving us sufficient time to issue the DISPLAY THREAD command to collect the information. Details of the UDF are provided in Appendix G, “External user-defined functions” on page 563.

Example 4-10 Step 14: Temporarily stop UDF DB2R3.GRACFGRP

```
-sto FUNCTION SPEC(DB2R3.GRACFGRP) scope(group) action(queue)
```

After we had collected the information we issued the command shown in Example 4-11 to start the UDF which allowed for the application to successfully complete.

Example 4-11 Step 14: Start UDF DB2R3.GRACFGRP

```
-sta FUNCTION SPEC(DB2R3.GRACFGRP) scope(group)
```

For more information about DB2 trusted contexts and the configuration we performed for running the DayTrader-EE6 workload refer to 4.3.13, “Trusted context” on page 173.

4.2 Monitoring strategy

We configured our monitoring infrastructure to support monitoring on DB2 system and application level. To enable these monitoring categories we performed the following infrastructure setup:

- Capture DB2 statistics traces for DB2 system monitoring, DB2 statistics traces are written to SMF and provide information about resource usage caused by the DB2 system. For the DSNZPARM setting we activate statistics traces at DB2 startup time refer to “DB2 statistics and accounting traces” on page 151.
- Capture DB2 accounting traces and z/OS Resource Measurement Facility (RMF) information for DB2 application monitoring. DB2 accounting traces provide information about application level resource usage in DB2. For the DSNZPARM setting we used to activate accounting traces at DB2 startup time refer to “DB2 statistics and accounting traces” on page 151.
- Capture DB2 audit trace class 1 to be able to quickly identify authorization failures. Refer to the information about how we configured DB2 to start the audit policy that we defined in Example 4-8 on page 126 at DB2 startup time.
- Capture DB2 audit trace class 10 to capture information about trusted context information. We capture this information to assure the correct trusted context is used by DB2. We use the DB2 administrative scheduler to start this trace at DB2 startup time. Refer to Example 4-9 on page 126 for information about how we configured DB2 to start audit trace class 10 at DB2 startup time.
- Use of DB2 administrative scheduler (ADMT) to automatically activate global dynamic statement cache statistics by starting a performance trace class 30, IFCID 318. Refer to Example 4-9 on page 126 for information about how you might configure DB2 to start IFCID 318 at DB2 startup time, if necessary.
- Regularly capture global statement cache information for subsequent analysis by using IBM Optim™ Query Tuner.

- Capture DB2 real time statistics (RTS) before and after DayTrader-EE6 workload stress testing. Among other things the RTS information captured are used to
  - Learn about the characteristics of the DayTrader-EE6 application,
  - Identify insert, update, delete hot spots,
  - Identify redundant indexes,
  - Learn about the REORG and RUNSTATS requirements of objects that are accessed by frequent insert, update, delete DML statements.
  - Estimate future data growth of DB2 tables and indexes and identify objects that are candidates for table partitioning.

  For more information about the implementation and usage examples of the RTS snapshot tables refer to 4.3.24, "DB2 real time statistics" on page 198.
- Configure RMF to capture SMF record type 70 to 79. For RMF and SMF monitoring, see Chapter 8, "Monitoring WebSphere Application Server applications" on page 361.
- WebSphere Application Server applications to provide unique DB2 client application information which we use for creating application level DB2 accounting and RMF workload activity reports for our WebSphere Application Server applications. The DB2 client application information used by our sample applications are shown in Table 4-1 on page 111.
- Configure DB2 to monitor the maximum number of concurrent active threads used by the DayTrader-EE6 application. The configuration steps that we took to implement DB2 profile monitoring for the DB2 threads used by the DayTrader-EE6 application are explained in 4.3.21, "Using profiles to disable idle thread timeout at application level" on page 194.
- WLM subsystem type DDF classification rules for DBATs to perform service classification based upon DB2 client application information. For information about how we setup WLM classification rules of DBATs refer to "JDBC type 4 service classification" on page 110.
- OMPE performance database processes to load DB2 statistics and accounting information into DB2 tables. We use these tables to run predefined SQL queries for application profiling and key performance indicator (KPI) monitoring. For information about implementing and using the performance database refer to 4.1.9, "WebSphere Application Server and DB2 security" on page 126.

4.3 DB2 for z/OS configuration

In this section we discuss the DB2 configuration options that we considered to support our DB2 for z/OS related WebSphere Application Server for z/OS application environment. For this we performed configuration tasks in the following areas:

- DB2 connectivity installation parameters
- Enabling DB2 dynamic statement cache
- Buffer pool configuration
- High Performance DBATs
- DB2 for z/OS Distributed Data Facility
- IBM Data Server Driver for JDBC and SQLJ
- JDBC type 2 DLL and the SDSNLOD2 library
- Bind JDBC packages
- UNIX System Services command line processor configuration
- Using the TestJDBC Java sample
- DB2 security considerations
- Trusted context

- Trusted context application scenarios
- DayTrader-EE6 application using JDBC connections
- Data Web Service servlet with trusted context AUTHID switch
- Using DB2 profiles
- Using profiles to optimize and monitor threads and connections
- Configure thread monitoring for the DayTrader-EE6 application
- Using profiles to keep track of DRDA client levels
- Using profiles to disable idle thread timeout at application level
- Using profiles for remote connection monitoring
- SYSPROC.ADMIN_DS_LIST stored procedure
- Using RTS to obtain COPY, REORG and RUNSTATS recommendations

4.3.1 DB2 connectivity installation parameters

In this section we cover some DB2 installation parameters (DSNZPARMs) that affect the Java applications connecting to DB2 in a WebSphere Application Server environment.

Adjusting the setting of DB2 threads and connections

A thread is a DB2 structure which describes a connection made by an application and traces its progress. There are two kinds of threads:

- Allied thread: A thread that is connected to DB2 from local subsystem, such as TSO, batch, IMS, CICS, CAF, or RRSAF. It is always active from allocation to termination. Requests from type 2 JDBC driver are allied threads.
- Database access thread (DBAT): A thread that is connected through a network with another system, Requests through type 4 JDBC driver make use of Database Access Threads (DBAT).

Because thread allocation can be a significant part of the cost in a short transaction, you need to set related parameters carefully according to your machine size, your work load, and other factors.

DDF THREADS field (CMTSTAT)

Database access threads differ from allied threads. They have two modes: ACTIVE MODE and INACTIVE MODE. The modes are controlled by this parameter.

- ACTIVE

  A database access thread is always active from initial creation to termination. It provides best performance for the thread but consumes more system resource.

- INACTIVE

  A database access thread can be active or inactive. When a database access thread is active, it is processing requests from client connections within units of work. When a database access thread is inactive, the connection is disassociated from the thread. The thread is pooled and reused for other connections, new or inactive, to start a new unit of work. So typically a small number of threads that can be used to service a large number of connections.

However, in some cases DB2 cannot pool database access threads. Table 4-3 summarizes whether a thread can be pooled or not. When the conditions are true, the thread can be pooled when a COMMIT is issued, otherwise, the thread remains active.

Table 4-3 Requirements for pooled threads

If the event is...	Thread can be pooled?
A hop to another location	Yes
A package bound with RELEASE(COMMIT)	Yes
A package bound with RELEASE(DEALLOCATE)	Yes
A declared temporary table that is active	No
An open and held cursor 1, a held LOB locator, or a package bound with KEEPDYNAMIC(YES), or RELEASE(DEALLOCATE) 2	No

1 A cursor can be closed with fast implicit close. For more information, see *DB2 10 for z/OS Managing Performance*, SC19-2978.
2 For more information of RELEASE(DEALLOCATE), see High Performance DBATs.

Use INACTIVE MODE threads instead of ACTIVE MODE threads whenever possible.

IDLE THREAD TIMEOUT field (IDTHDOIN)

This parameter controls the approximate time, in seconds, that an active server thread can remain idle before it is terminated. Threads are checked every two minutes. Specifying 0 disables idle thread time-out processing.

Inactive and indoubt threads are not subject to this time-out parameter. If CMTSTAT subsystem parameter is set to ACTIVE, your application must start its next unit of work within the specified time-out period, otherwise its thread is terminated.

POOL THREAD TIMEOUT field (POOLINAC)

In INACTIVE MODE, this parameter specifies the approximate time, in seconds, that a database access thread (DBAT) can remain idle in the pool before it is terminated. Threads are checked every three minutes. In addition, a database access thread is terminated after it has processed 200 units of work. The default value is 120. Increasing POOLINAC can potentially reduce the overhead for creating a new DBAT, but the disadvantage would be the virtual storage used by the pooled DBAT.

The default value is 120. Increasing POOLINAC can potentially reduce the overhead for creating a new DBAT, but the disadvantage would be the virtual storage used by the pooled DBAT.

Choosing a good number for maximum threads is important to keep applications from queuing and to provide good response time. Fewer threads than needed under utilize the processor and cause queuing for threads. More threads than needed do not improve the response time. They require more real storage for the additional threads and might cause more paging and, hence, performance degradation.

MAX USERS field (CTHREAD)

This parameter controls the maximum number of allied threads that are to be allocated concurrently. Requests from type 2 JDBC driver are allied threads.

MAX REMOTE ACTIVE field (MAXDBAT)

This parameter specifies the maximum number of database access threads (DBATs) that are allowed to be concurrently active. These threads are for connections coming into DB2 through DDF, such as requests through the type 4 JDBC driver.

When a request for a new connection to DB2 is received and MAX REMOTE ACTIVE has been reached, If DDF THREAD is ACTIVE mode, the allocation request is allowed but any further processing for the connection is queued waiting for an active database access thread to terminate. If DDF THREAD is INACTIVE mode, the allocation request is allowed and is processed when DB2 can assign an pooled idle database access thread to the connection. Pool idle thread counts as an active thread against MAX REMOTE ACTIVE.

MAX REMOTE CONNECTED field (CONDBAT)

This value must be greater than or equal to the value of MAX REMOTE ACTIVE. The MAX REMOTE ACTIVE is limitation of concurrent active database access threads, while this parameter sets the maximum number of concurrent DDF connections.

If a new connection request to DB2 is received, and MAX REMOTE CONNECTED has been reached or MAX REMOTE CONNECTED is zero, the connection request is rejected.

MAXCONQN in macro DSN6FAC

Specifies the maximum number of inactive or new connections that can be queued waiting for a DBAT to process the request.

OFF means that the depth of the connection queue is limited by the value of the CONDBAT subsystem parameter. ON means that the depth of the connection queue is limited by the value of the MAXDBAT subsystem parameter. A numeric value specifies the maximum number of connections that can be queued waiting for a DBAT to process a request.

When a request is added to the connection request queue and the thresholds specified by both the MAXDBAT and MAXCONQN subsystem parameters are both reached (unless MAXCONQN is set to OFF) then DDF closes the client connection longest waiting client connection in the queue. The closed connections give remote clients an opportunity to redirect the work to other members of the group that have more resources to process the work. The function is enabled only when DB2 subsystem is a member of a data sharing group.

The default value is OFF.

MAXCONQW in macro DSN6FAC

Specifies the maximum length of time that a client connection waits for a DBAT to process the next unit-of-work or new connection request.

ON means that connections wait as long as the value specified by the IDHTOIN subsystem parameter. OFF means that connections wait indefinitely for a DBAT to process requests. A numeric value specifies a time duration in seconds that a connection waits for a DBAT to process the request.

Each queued connection request entry is examined to see if its time waiting in the queue has exceeded the specified value. If the time is exceeded, the client connection is closed. After all entries in the queue have been processed or the last entry whose time in the queue exceeded the threshold has been processed, a DSNL049I message is issued to indicating how many client connections were closed because of the MAXCONQW value. The function is enabled only when DB2 subsystem is a member of a data sharing group.

The default value is OFF.

4.3.2 Enabling DB2 dynamic statement cache

The feature *dynamic statement caching* was introduced with DB2 Version 5. Whenever DB2 prepares an SQL statement, it creates a control structure that is used when the statement is executed. When dynamic statement caching is in effect, DB2 stores the control structure associated with a prepared dynamic SQL statement in a storage pool. If that same statement is executed again, DB2 can reuse the cached control structure, avoiding the expense of re-preparing the statement,

When using statement caching, four different types of prepare operations can take place:

- Full prepare

  A full prepare occurs when the skeleton copy of the prepared SQL statement does not exists in the global dynamic SQL cache (or the global cache is not active). It can be caused explicitly by a PREPARE or an EXECUTE IMMEDIATE statement or implicitly by an EXECUTE when using KEEPDYNAMIC(YES).

- Short prepare

  A short prepare occurs, if the skeleton copy of the prepared SQL statement in the global dynamic SQL cache can be copied into the local storage.

- Avoided prepare

  A prepare can only be avoided when using full caching. Because in this case, the full prepared statement is kept across commits, issuing a new EXECUTE statement (without a prepare after a commit) does not need to prepare anything. The full executable statement is still in the thread's local storage (assuming it was not removed from the local thread storage because MAXKEEPD was exceeded) and can be executed as such.

- Implicit prepare

  This is the case when an application, that uses KEEPDYNAMIC(YES), issues a new EXECUTE after a commit was performed and a prepare cannot be avoided (the previous case). DB2 will issue the prepare (implicitly) on behalf of the application. (The application must not explicitly code the prepare after a commit in this case.)
  Implicit prepares can result in a full or short prepare:

  - In full caching mode, when a statement has been removed from the local cache because MAXKEEPD was exceeded, but still exists in the global cache, the statement is copied from the global cache. This is a short prepare. (If MAXKEEPD has not been exceeded and the statement is still in the local cache the prepare is avoided.)
  - In full caching mode, and the statement is no longer in the global cache either, a full prepare is done.
  - In local caching only mode, a full prepare has to be done.

  Whether a full or short prepare is needed in full caching mode depends on the size of the global cache. The bigger the size, the more likely we can do a short prepare.

Comparing the relative cost of the different types of prepare:

- If a full prepare costs 100
- A short prepare costs 1
- And an avoided prepare costs nothing

When the prepared statements are cached in the EDM pool, DB2 will not regenerate the access path if the cached statement can be reused by a subsequent execution. It saves cost of SQL statement preparing. The following DB2 system parameters should be reviewed.

- CACHE DYNAMIC SQL field (CACHEDYN)

  DB2 global dynamic statement cache is enabled If you specify YES. You must also specify YES for the USE PROTECTION field on panel DSNTIPP. This cache pool is shared by different threads, plans and packages.

- EDM STATEMENT CACHE field (EDMSTMTC)

  This parameter determines the size (in KB) of the statement cache that is to be used by the EDM. It can be increased and decreased with the SET SYSPARM command, but it cannot be decreased below the value that is specified at DB2 startup. The calculated column of panel DSNTIPC is based on input from previous panels. If you want to set value in the override column, see more information in *DB2 10 for z/OS Installation and Migration Guide*, GC19-2974, Calculating EDM pool size.

- MAX KEPT DYN STMTS field (MAXKEEPD)

  BIND option KEEPDYNAMIC(YES) enables applications to keep prepared dynamic statement past commit points in local statement cache (thread based memory).

  This parameter specifies the maximum number of prepared statements kept in the local cache, thus it can help limit the amount of storage in DBM1 address space. If this limit is exceeded, DB2 honors the KEEPDYNAMIC(YES) behavior, but “implicit” prepares might be necessary to rebuild the executable version of SQL statements when they are executed after a commit.

Statements in plans or packages bound with REOPT(VARS) are not cached in the global cache. The bind options REOPT(VARS) and KEEPDYNAMIC(YES) are not compatible.

In a data sharing environment prepared statements cannot be shared among the members. As each member has its own EDM pool. A cached statement of one member is not available to an application that runs on another DB2 member.

There are different levels of statement caching, which are explained in the following sections:

- No caching
- Local dynamic SQL cache only
- Global dynamic statement cache only
- Full caching

No caching

Figure 4-47 on page 143 helps to visualize this behavior.

Program A prepares a dynamic SQL statement S, executes the prepared statement twice, and terminates.

Program B starts after program A has terminated, prepares exactly the same statement S as A did, executes the prepared statement, issues a commit, tries to execute S again, receives an error SQLCODE -514 or -518 (SQLSTATE 26501 or 07003), has to prepare the same statement S again, executes the prepared statement, and terminates.

Each time a prepare has been executed by the programs A and B, issuing the SQL PREPARE statement, DB2 prepared the statement from scratch. After the commit of program B, the prepared statement is invalidated, so program B had to repeat the prepare of statement S.

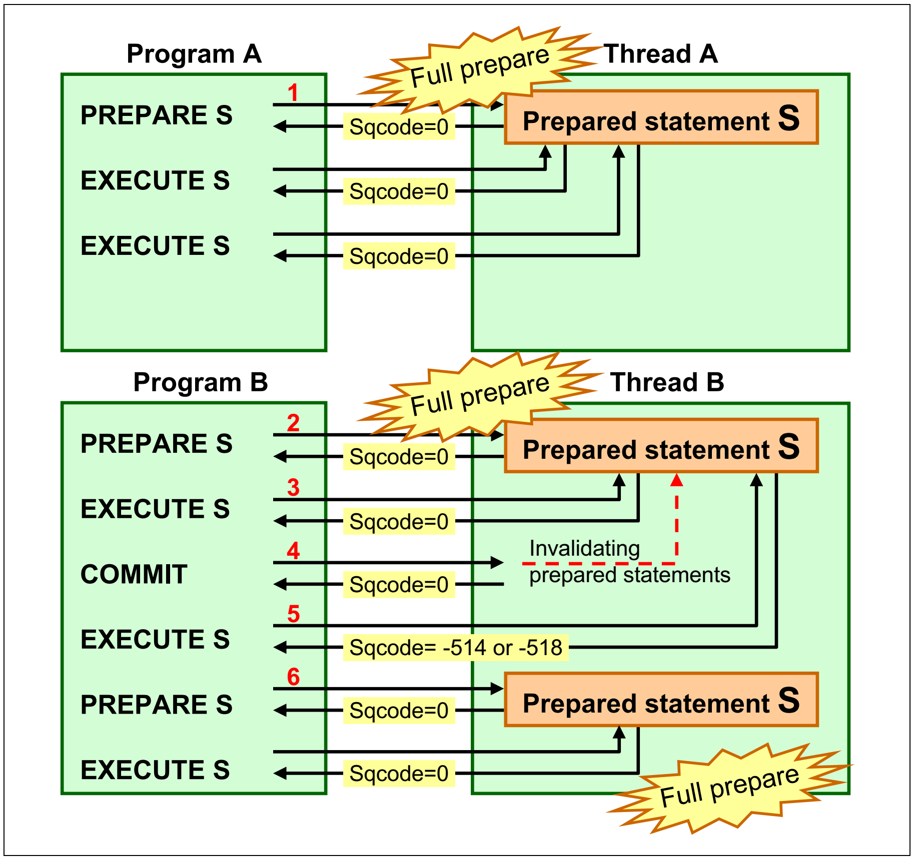


Figure 4-47 No caching, CACHEDYN = NO and KEEPDYNAMIC = NO

Local dynamic SQL cache only

A local dynamic statement cache is allocated in the storage of each thread in the DBM1 address space. You can control the usage of this cache by using the KEEPDYNAMIC bind option.

Bound with KEEPDYNAMIC(YES), an application can issue a PREPARE for a statement once and omit subsequent PREPAREs for this statement, even after a commit has been issued.

To understand how the KEEPDYNAMIC bind option works, it is important to differentiate between the executable form of a dynamic SQL statement (the *prepared statement*) and the character string form of the statement (the *statement text*).

Let us take a look at our two example programs, shown in Figure 4-48 on page 144.

Program A prepares a dynamic SQL statement S, executes the prepared statement twice, and terminates.

Program B starts after program A has terminated, prepares the same statement S as A did, executes the prepared statement, issues a commit, executes S again (causing an internal (implicit) prepare) and terminates.

Each time an SQL PREPARE is been issued by the programs (or DB2 for the implicit prepare), a complete prepare is executed. This process is a full prepare. After the COMMIT of program B, the prepared statement is invalidated (because the cursor was not open and not defined with hold), but the statement text has been preserved in the local statement cache of the thread (because it was bound with KEEPDYNAMIC(YES)). Therefore program B does not have to repeat the prepare of statement S explicitly; it can immediately issue the EXECUTE again. Under the covers, DB2 will execute a complete prepare operation, using the saved statement string. This operation is called an *implicit prepare*.

Be aware that application program B has to be able to handle the fact that the implicit prepare might fail and an error is returned. Any error that normally occur at prepare time can now be returned on the OPEN, EXECUTE, or DESCRIBE statement issued by the application.

The prepared statement and the statement text are held in the thread´s local storage within the DBM1 address space (outside the EDM pool). But only the statement text is kept across commits when you only use local caching.

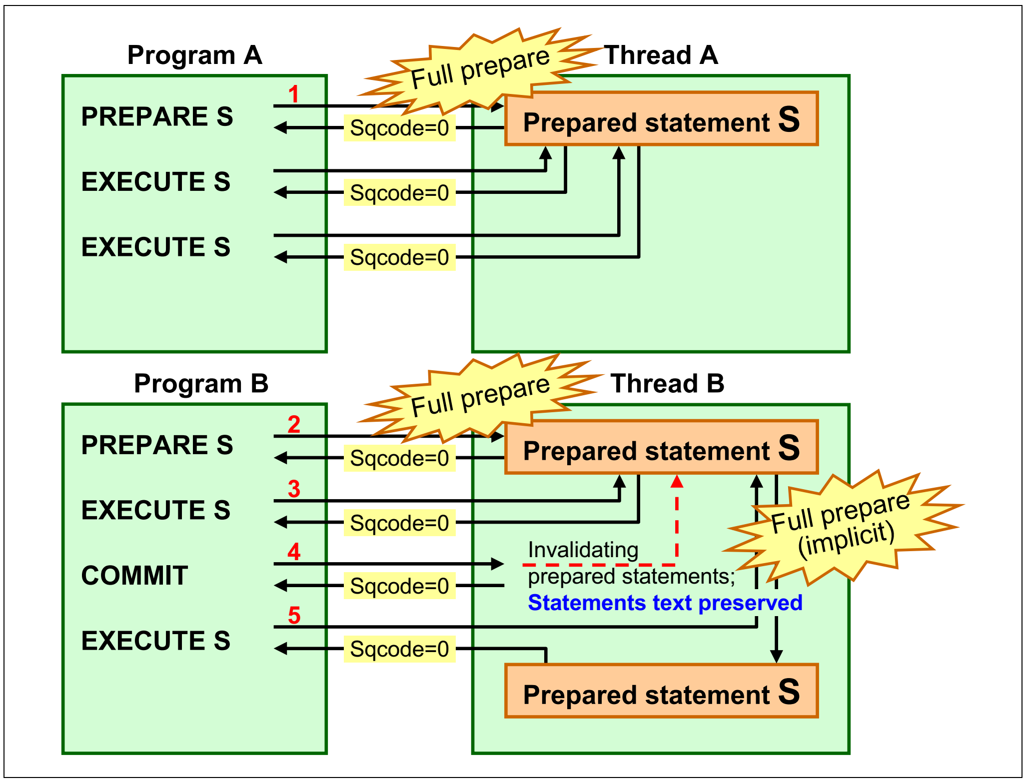


Figure 4-48 Local caching, CACHEDYN = NO and KEEPDYNAMIC = YES

The local instance of the prepared SQL statement (the prepared statement), is kept in DBM1 storage until one of the following occurs:

- The application process ends.
- The application commits and there is no open cursor defined WITH HOLD for the statement. (Because we are using only local caching, just the statement string is kept across commits.)
- A rollback operation occurs.
- The application issues an explicit PREPARE statement with the same statement name.

If the application issues a PREPARE for the same SQL statement name which is kept in the cache, the kept statement is discarded and DB2 prepares the new statement.

In a distributed environment, if the requester does not issue a PREPARE after a COMMIT, the package at the DB2 for z/OS server must be bound with KEEPDYNAMIC(YES). If both requester and server are DB2 for z/OS subsystems, the DB2 requester assumes that the KEEPDYNAMIC value for the package at the server is the same as the value for the plan at the requester.

The KEEPDYNAMIC option might have performance implications for DRDA clients that specify WITH HOLD on their cursors:

- If KEEPDYNAMIC(NO) is specified, a separate network message is required when the DRDA client issues the SQL CLOSE for the cursor.
- If KEEPDYNAMIC(YES) is specified, the DB2 for z/OS server automatically closes the cursor when SQLCODE +100 is detected, which means that the client does not have to send a separate message to close the held cursor. This reduces network traffic for DRDA applications that use held cursors. It also reduces the duration of locks that are associated with the held cursor.

When a distributed thread has touched any package which is bound with KEEPDYNAMIC(YES), the thread cannot become inactive.

This level of caching, used without other caching possibilities, is of minor value, because the performance improvement is limited. The only advantage is that you can avoid coding a PREPARE statement after a COMMIT because DB2 keeps the statement string around. This is of course most beneficial in a distributed environment where you can save a trip across the wire this way. On the other hand, by using the DEFER(PREPARE) bind option, you can obtain similar network message savings.

Global dynamic statement cache only

The global dynamic statement cache is normally allocated in the EDM pool within the DBM1 address space. You can activate this cache by setting CACHEDYN=YES in DSNZPARM.

When global dynamic statement caching is active, the skeleton copy of a prepared SQL statement (SKDS) is held in the global dynamic statement cache inside the EDM pool. Only one skeleton copy of the same statement (matching text) is held. The skeleton copy can be used by user threads to create user copies. An LRU algorithm is used for replacement.
If an application issues a PREPARE or an EXECUTE IMMEDIATE (and the statement has not been executed before in the same commit scope), and the skeleton copy of the statement is found in the global statement cache, it can be copied from the global cache into the thread´s storage. This is called a *short prepare*.

Note: Without local caching (KEEPDYNAMIC(YES)) active, the application cannot issue EXECUTEs directly after a commit. The statement returns an SQLCODE -514 or -518, SQLSTATE 26501 or 07003.

Let us take a look at our example. The global cache case is shown in Figure 4-49 on page 146.

Program A prepares a dynamic SQL statement S, executes the prepared statement twice, and terminates.

Program B starts after program A has terminated, prepares the same statement S as A did, executes the prepared statement and issues a COMMIT. Then program B tries to execute S again. The program receives an error SQLCODE -514 or -518 (SQLSTATE 26501 or 07003) and has to prepare the same statement S again. Then it executes the prepared statement and terminates.

The first time a prepare for statement S is issued by the program A, a complete prepare operation is performed. The SKDS of S is then stored in the global statement cache. When program B executes the prepare of S for the first time, the SKDS is found in the global statement cache and is copied to the local storage of B´s thread (short prepare). After the COMMIT of program B, the prepared statement is invalidated in B's local storage, but the SKDS is preserved in the global statement cache in the EDM pool. Because the statement string or the prepared statement is not kept after the commit, program B has to repeat the prepare of statement S explicitly. This causes another copy operation of the SKDS in the global cache to the local storage of the thread of application B (short prepare).

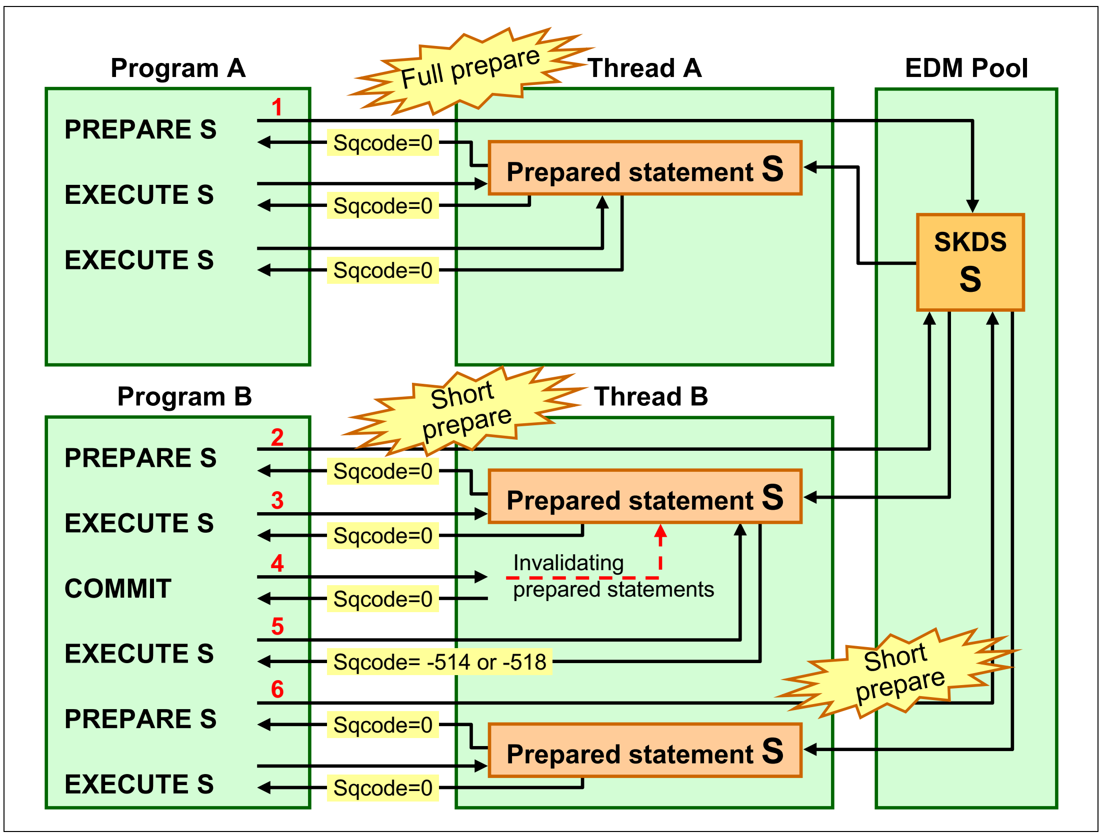


Figure 4-49 Global caching, CACHEDYN = YES and KEEPDYNAMIC = NO

This level of statement caching has important performance advantages.

Full caching

Full caching is a combination of local caching (KEEPDYNAMIC(YES)), a MAXKEEPD DSNZPARM value > 0, and global caching (CACHEDYN=YES). It is possible to avoid prepares, because a commit does not invalidate prepared statements in the local cache.

Let us look again at our example when full caching is active, shown in Figure 4-50 on page 147.

Program A prepares a dynamic SQL statement S, executes the prepared statement twice, and terminates.

Program B starts after program A has terminated, prepares the same statement S as A did, executes the prepared statement, issues a commit, executes S again, and terminates.

The first time a prepare for statement S is issued by the program A, a complete prepare is done (full prepare). The SKDS of S is stored in the global cache. When program B executes the prepare of S the first time, the SKDS is found in the global statement cache and is copied to the local statement cache of B´s thread (short prepare). The COMMIT of program B has no effect on the prepared statement. When full caching is active, both the statement string which is also kept for local caching only, and the prepared statement are kept in the thread's local storage after a commit. Therefore, program B does not have to repeat the prepare of statement S explicitly, and it was not necessary to do the prepare the statement implicitly because the full executable statement is now kept in the thread's local storage. This case is called *prepare avoidance*.

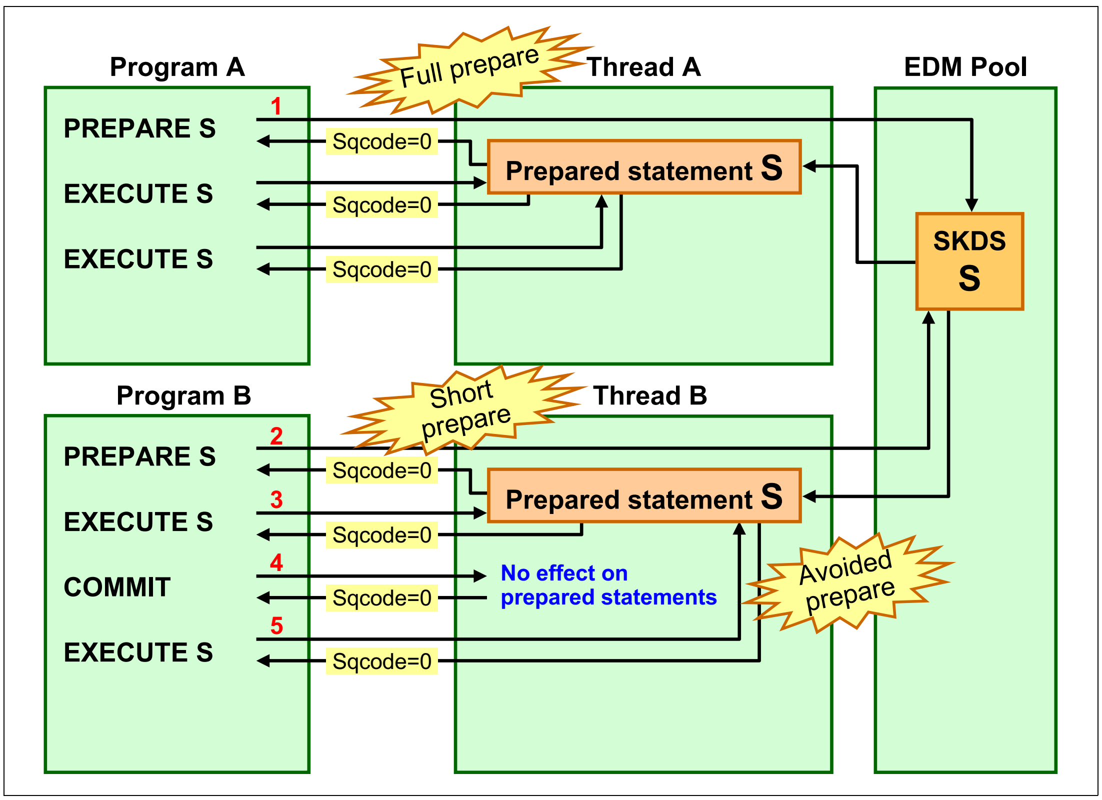


Figure 4-50 Full caching, CACHEDYN = NO, KEEPDYNAMIC = YES and MAXKEEPD > 0

Using full caching the maximum size of the local cache across all user threads is controlled by the MAXKEEPD DSNZPARM. A FIFO algorithm is used for replacement of statements in the local cache.

CACHEDYN should be turned on for dynamic SQL for WebSphere applications. For Local Dynamic Statement Cache because the statements are kept in thread storage, Sysplex workload balancing is not available if KEEPDYNAMIC is exploited. Use BIND option KEEPDYNAMIC(YES) for application with a limited number of SQL statements that are executed frequently.

To achieve a balance between performance and storage usage, you can adjust EDMSTMTC and MAXKEEPD according to the statistic report. Generally GLOBAL CACHE HIT RATIO should be higher than 90%-95%, and LOCAL CACHE HIT RATIO should be higher than 70%.

```
GLOBAL CACHE HIT RATIO = [Short Prepares] / [Short + Full Prepares]
LOCAL CACHE HIT RATIO = [Prepares Avoided]/[Prepares Avoided + Implicit Prepares]
```

```
DYNAMIC SQL STMT            QUANTITY  /SECOND  /THREAD  /COMMIT
--------------------------  --------  -------  -------  -------
PREPARE REQUESTS              124.5K     5.78     0.75     0.25
  FULL PREPARES             17446.00     0.81     0.10     0.04
  SHORT PREPARES              108.1K     5.02     0.65     0.22
GLOBAL CACHE HIT RATIO (%)     86.10      N/A      N/A      N/A

IMPLICIT PREPARES               0.00     0.00     0.00     0.00
PREPARES AVOIDED             5603.00     0.26     0.03     0.01
CACHE LIMIT EXCEEDED            0.00     0.00     0.00     0.00
PREP STMT PURGED                3.00     0.00     0.00     0.00
LOCAL CACHE HIT RATIO (%)     100.00      N/A      N/A      N/A
```

Figure 4-51 Information about the dynamic SQL statement in statistic report

WebSphere Application Server prepared statement cache and DB2 dynamic statement cache are different concepts. For how to make these two functions work together, refer to 2.6.6, “WebSphere Application Server prepared statement cache” on page 57.

For more information about cache matching criteria, see 6.7, “Coding practices for a good DB2 dynamic statement cache hit ratio” on page 329.

Note: DB2 10 for z/OS has largely reduced LC24 on the EDM pool by removing the areas dedicated to cursor tables and skeleton cursor tables.

4.3.3 Locking and accounting setup

In this section, we describe DB2 system parameters related to locking and lock performance of the DB2 subsystem and to accounting and administration:

- Application and system locking
- DB2 accounting accumulation setting
- DB2 statistics and accounting traces
- Miscellany

Application and system locking

We need to consider the following parameters related to application locking and lock performance:

RESOURCE TIMEOUT field (IRLMRWT)

This parameter specifies the number of seconds IRLM waits before detecting a time-out. IRLM checks for time-out on each deadlock detection cycle. So the actual wait time between the lock request and IRLM detecting the time-out will be:

- For non-data sharing:
  `IRLMRWT <= actual wait time <= IRLMRWT + DEADLOCK TIME`

- In a data sharing environment, because the deadlock detection process sends inter-system XCF messages, the actual wait time is longer:

```
IRLMRWT + DEADLOCK TIME <= actual wait time <= IRLMRWT + 4 * DEADLOCK TIME
```

If you can afford suspended process remaining inactive for 60 seconds, use the default. Sometimes TIMEOUT is caused by badly behaving application, you can simulate workload in testing environment and identify it:

1. Start with the default of 60 seconds.
2. Monitor the time-out.
3. Reduce the value a few seconds if none occur. Cycle back to 2.
4. If time-outs occur, identify the cause and correct the process if possible. Cycle back to 2.

You can change the TIMEOUT value using the IRLM modify command.

DEADLOCK TIME field and DEADLOCK CYCLE field

These two fields on panel DSNTIPJ, correspond to the IRLM start procedure DEADLOK parameter. DEADLOCK TIME controls the time for which local deadlock detection cycles are to run. DEADLOCK CYCLE specifies the number of local deadlock cycles that must expire before the IRLM does global deadlock detection, which is used only for DB2 data sharing. You can use (1,1) by default.

LOCKS PER TABLE (SPACE) field (NUMLKTS)

This parameter specifies the default maximum number of page, row, or LOB locks that an application can hold simultaneously in a table or table space. LOCKMAX clause of the CREATE TABLESPACE and ALTER TABLESPACE can overwrite this setting for a specific table space. If a single application exceeds the maximum number of locks in a single table or table space, lock escalation occurs. It obtains a table or table space lock, then releases all of the page or row locks.

This value is workload dependent. High setting or a value of 0 might result in excessive numbers of locks, which is storage and CPU time consuming. Whereas small value can trigger lock escalation frequently, which might lead to lock contention. Lock escalation is an expensive process as well.

LOCKS PER USER field (NUMLKUS)

This parameter specifies the maximum number of page, row, or LOB locks that a single application can hold concurrently for all table spaces. After that limit is reached, the program that accumulated these locks will terminate with SQLCODE -904. Do not specify 0 or a large value unless it is specifically required to run an application.

U LOCK FOR RR/RS field (RRULOCK)

This parameter specifies whether DB2 is to use U (UPDATE) locks or S (SHARE) locks when the isolation of the program is repeatable read (RR) or read stability (RS). If your programs with RR or RS make frequent updates, specify YES to get greater concurrency. For more information about LOCK mode and isolation level, see 6.8, “Locking” on page 331.

X LOCK FOR SEARCHED U/D field (XLKUPDLT)

This specifies locking method when performing a searched update or delete. The acceptable values are:

- NO (default), DB2 uses an S or U lock when scanning for qualifying rows. Before DB2 updates or deletes qualifying rows or pages, the lock is changed to an X lock.
- YES, DB2 uses an X lock on qualifying rows or pages based on stage 1 predicates.

- TARGET, it means a combination of YES and NO behavior. DB2 uses an X lock on qualifying rows or pages of the updating or deleting table, while it uses an S or U lock when scanning for rows or pages of other tables referenced by the query

A value of NO provides higher rates of concurrency.

EVALUATE UNCOMMITTED field (EVALUNC)

This parameter specifies whether predicate evaluation can occur on uncommitted data of other application processes. It applies only to stage 1 predicate processing that uses table access (table space scan, index-to-data access, and RID-list processing) for queries with isolation level RS or CS.

Default value is NO. Specify YES to improve concurrency if your applications can tolerate returned data that might falsely exclude any data that would be included as the result of undo processing. This parameter does not influence whether uncommitted data is returned to an application because queries with isolation level RS or CS return only committed data.

You can obtain similar results by using the SQL SKIPPED LOCKED DATA clause.

SKIP UNCOMM INSERTS field (SKIPUNCI)

This parameter specifies whether statements ignore a row that was inserted by another transaction but has not been committed or aborted. It applies only to statements with row-level locking and isolation level RS or CS

Default value is NO. If your applications do not need to wait for the inserts outcome of other transactions, specifying YES to get greater concurrency.

DB2 accounting accumulation setting

For DRDA threads and RRS attach threads, you can reduce the high volume DB2 accounting records by using accounting accumulation parameter, which consolidates multiple accounting records into one.

DDF/RRSAF ACCUM field (ACCUMACC)

The parameter controls whether DB2 accumulated accounting data by the user for DDF and RRSAF threads.

NO means DB2 writes an accounting record when a DDF thread becomes inactive or when sign-on occurs for an RRSAF thread.

A value n (between 2 and 65535) means DB2 writes an accounting record every n accounting intervals for a given user.

AGGREGATION FIELDS field (ACCUMUID)

This parameter controls the aggregation fields used for DDF and RRSAF accounting rollup. Each value (between 0 and 17) represents a rollup criteria. For more information, see DB2 10 for z/OS Installation and Migration Guide, GC19-2974, Tracing parameters panel: DSNTIPN.

> **Note:** For JDBC type 2 connections you might want to consider setting the account interval data source property to obtain DB2 accounting information written at DB2 commit.
>
> For more information, see 8.4.2, “Creating DB2 accounting records at a transaction boundary” on page 396.

DB2 statistics and accounting traces

We configured our DB2 members to collect DB2 statistics and accounting traces through SMF. For this we performed the following DB2 system parameter (DSNZPARM) settings through which DB2 accounting and statistics traces are started during DB2 member startup:

- SMFACCT=(1,2,3,7,8)
  The SMFACCT DSNZPARM controls the collection of DB2 accounting traces through SMF. Besides plan level accounting information (classes 1,2,3) we also collected package level accounting information (classes 7,8). For Java workloads you might want to consider not to collect package level information as you cannot use the JDBC package names to perform application level profiling and reporting.
- SMFSTAT=(1,3,4,5,6,7)
  The SMFSTAT DSNZPARM controls the collection of DB2 statistics traces through SMF. In our DB2 environment we collect statistics trace classes 1, 3, 4, 5, 6, and 7.

Miscellany

Below are other DB2 installation parameters you need to note for the Java applications running in a WebSphere Application Server environment.

DESCRIBE FOR STATIC field (DESCSTAT)

This parameter controls whether DB2 is to build a DESCRIBE SQL descriptor area (SQLDA) when binding static SQL statements. Use the default value YES especially you are using SQLJ. Packages size will slightly increase because the DESCRIBE SQLDA is stored with each bound SQL SELECT statement.

ADMTPROC - administrative task scheduler

ADMTPROC identifies a name for the JCL procedure that is used to start the DB2 administrative task scheduler that is associated with the DB2 subsystem. We set this parameter to the following values:

- D0Z1ADMT for subsystem D0Z1
- D0Z2ADMT for subsystem D0Z2

If you set this parameter to blanks, DB2 will not start the administrative task scheduler.

4.3.4 Buffer pool configuration

Our environment is configured with function testing in mind. For this purpose it is sufficient to provide separate buffer pools for the following object categories:

- DB2 catalog and directory
- User table spaces
- User index spaces
- Lob user data
- Workfile data base

To support separation of the object categories we created the buffer pools shown in Table 4-4 in both data sharing members:

Table 4-4 Buffer pool configuration

Buffer pool ID	Object category	Size in no of pages
BP0	Catalog Directory	20000
BP8K0	Catalog Directory 8K pages	20000

Buffer pool ID	Object category	Size in no of pages
BP16K0	Catalog Directory 16K pages	20000
BP32K	Catalog Directory 32K pages	20000
BP1	User table spaces	20000
BP2	User index spaces	20000
BP3	Lob user data	20000
BP8K1	User table spaces 8 KB pages	10000
BP16K1	User table space 16 KB pages	10000
BP16K3	XML table spaces	10000
BP32K1	User table spaces 32 KB pages	10000
BP7	Workfile 4 KB pages	20000
BP8	Workfile index buffer pool	20000
BP32K7	Workfile 32 KB pages	20000

Buffer pool related DSNZPARM configuration

To support the buffer pool separation shown in Table 4-4 on page 151 on DB2 subsystem level we used the DSNZPARM settings on both data sharing members to keep user table and index spaces away from the buffer pools used by catalog and directory and the workfile database:

- IDXBPOOL=BP2
- TBSBPOOL=BP1
- TBSBP8K=BP8K1
- TBSBP16K=BP16K1
- TBSBP32K=BP32K1
- TBSBPLOB=BP3,
- TBSBPXML=BP16K3

Database related buffer pool configuration

To support the buffer pool separation shown in Table 4-4 on page 151 on database level we configured the DayTrader-EE6 database to use the default buffer pools settings shown in Example 4-12 for table space and index space creation:

Example 4-12 Create database default buffer pool settings

```
CREATE DATABASE DBTR8074
  BUFFERPOOL BP1
  INDEXBP    BP2
  CCSID      EBCDIC
  STOGROUP   GR248074;
```

Buffer pool tuning

With the buffer pool configuration shown in Table 4-4 on page 151 we hardly ran into performance problem caused by undersized buffer pools. However, if you need to tune your buffer pools, refer to *DB2 9 for z/OS: Buffer Pool Monitoring and Tuning*, REDP-4604.

Simulate production like buffer pool sizes and catalog statistics

To tune your DB2 application or queries under production like conditions is a pre-production environment is an important requirement which enables you to discover problems with applications and SQL queries in time prior to application or SQL production deployment. For this it is recommended to have your tables to reflect production like data volumes or, if this is not an option, to configure your DB2 catalog to reflect production like statistics for the tables against which you are going to run application workloads or you need to perform query tuning.

For more information about cloning catalog statistics, refer to *DB2 10 for z/OS Managing Performance*, SC19-2978, “Modeling your production system statistics in a test subsystem”.

When preparing an SQL statement the optimizer takes important hardware configurations such as buffer pool sizes, CPU speed, and the number of processor into account to make the most suitable cost based access path decision.

In cases in which a DB2 test system is constraint on CPU and real storage resources the optimizer cannot provide you with an access path decision it would have made if the same access path decision was taken in a DB2 production environment with more and faster CPUs and with more real storage and bigger buffer pools. To provide help in such situations you can use DB2 profiles to model your DB2 test environment based on the configuration of your production environment. Without having to have the corresponding hardware resources installed and available to your DB2 test system, you can use profiles to provide the following parameters to emulate your production hardware and DB2 pool configuration for DB2 access path selection:

- Processor speed
- Number of processors
- Maximum number of RID blocks
- Sort pool size
- Buffer pool size

For more information about this topic refer to “Modeling a production environment on a test subsystem”, *DB2 10 for z/OS, Managing Performance*, SC19-2978.

4.3.5 DB2 for z/OS Distributed Data Facility

Accessing DB2 for z/OS from Java applications require the JDBC packages to be bound in DB2 for z/OS. The only way to bind the JDBC packages is through a JDBC type 4 connection using the DB2Binder utility or the DB2 command line processor. This is why the DB2 base setup for JDBC includes setting up Distributed Data Facility (DDF) even if you do not plan to use DDF is you only want to use local JDBC type 2 connections.

DB2 boot strap data set configuration

DDF setup requires you to change the BSDS distributed data facility communication record to provide location name, port numbers, and optionally the IP addresses to be used by the member and the group.

Configuration with IP address in the BSDS

We used the DSNJU003 utility control statements shown in Example 4-13 to set up DDF in DB2 for z/OS once for each DB2 data sharing member involved.

Example 4-13 DSNJU003 DDF configuration with IP address

```
//BSDSCH3 EXEC PGM=DSNJU003
//STEPLIB   DD DISP=SHR,DSN=DB0ZT.SDSNEXIT
//          DD DISP=SHR,DSN=DB0ZT.SDSNLOAD
//SYSUT1   DD  DISP=OLD,DSN=DB0ZB.D0Z1.BSDS01
//SYSUT2   DD  DISP=OLD,DSN=DB0ZB.D0Z1.BSDS02
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
* DDF intial record setting
 DDF    LOCATION=DB0Z,RESPORT=39002,PORT=39000,SECPORT=0
* IPV4=<member IP addr>,GRPIPV4=<group IP addr>
 DDF    IPV4=9.12.4.138,GRPIPV4=9.12.4.153
* we are not using a VTAM LUNAME
 DDF    NOLUNAME
* DB2 to initialize the TCP/IP interface only
 DDF    IPNAME=IPDB0Z
```

We configured the member and group DVIPA in the BSDS using the IPV4 and GRPIPV4 parameters. With this BSDS setting the TCP/IP port statements shown in "Port definition without IP address binding" on page 157 must not have any BIND IP address configuration. When DB2 starts it automatically binds to the IP addresses given in the BSDS. DB2 accepts connections not only on the IP address specified in the BSDS, but on any IP address that is active on the TCP/IP stack. Additionally, connections are accepted on both secure and non-secure SQL ports. In contrary, using bind specific TCP/IP port statements as discussed in "Port definition with IP address binding" on page 156 do not support secure DB2 SQL ports.

> **Important:** With IP addresses in the BSDS a client can connect to DB2 using IP addresses other than the group or member IP address provided these are active on the current TCP/IP stack. This can be useful as it provides the flexibility to choose between IP addresses available on the current IP stack. However, DB2 clients connecting to DB2 for z/OS using an IP address other than the DB2 group or member specific IP address might break if a DB2 member has been moved to run in a different LPAR.

Configuration without IP address in the BSDS

With IP address bindings defined on the port statement as shown in "Port definition with IP address binding" on page 156 the BSDS distributed data facility communication record must not have the member and group IP addresses provided through the IPV4 and GRPIPV4 BSDS parameters. The DSNJU003 utility control statements required to perform that configuration is shown in Example 4-14.

Example 4-14 DSNJU003 DDF configuration without IP address

```
//BSDSCH3 EXEC PGM=DSNJU003
//STEPLIB   DD DISP=SHR,DSN=DB0ZT.SDSNEXIT
//          DD DISP=SHR,DSN=DB0ZT.SDSNLOAD
//SYSUT1   DD  DISP=OLD,DSN=DB0ZB.D0Z1.BSDS01
//SYSUT2   DD  DISP=OLD,DSN=DB0ZB.D0Z1.BSDS02
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
```

```
* DDF intial record setting
 DDF    LOCATION=DB0Z,RESPORT=39002,PORT=39000,SECPORT=0
* we are not using a VTAM LUNAME
 DDF    NOLUNAME
* DB2 to initialize the TCP/IP interface only
 DDF    IPNAME=IPDB0Z
```

When we initially setup our DDF configuration we setup the SSLPORT to support SSL encryption. During startup DB2 issued the error message shown in Figure 4-52 indicating that the TCP/IP IP address bindings on the PORT statement were not supported with DB2 secure port configurations. As a consequence we corrected the TCP/IP port configuration to remove the IP address bindings and defined the IP addresses in the BSDS.

```
DSNL512I  -D0Z1 DSNLILNR TCP/IP BINDSPECIFIC NOT
SUPPORTED WITH SECURE PORT FAILED WITH
           RETURN CODE=0 AND REASON CODE=00000000
```

Figure 4-52 DB2 secure port and BINDSPECIFIC

Important: If you use IP address bindings on the TCP/IP port configuration in your DDF configuration you will not be able to configure an SSL port in DB2. If you do DB2 issues the error message DSNL512I and the DDF initialization fails.

Member specific location alias

In some situations it is useful to be able to connect to an individual member of a data sharing group by using the DB2 group IP address. To cater for that requirement, we defined one member specific location alias for each DB2 member. We decided to define those aliases through the DB2 modify command interface (see Example 4-15) to take advantage of the online change capabilities provided by DB2 MODIFY DDF ALIAS command interface. You cannot use the DB2 MODIFY DDF command to change alias names that have been statically defined in the BSDS.

Example 4-15 Dynamically define location alias

```
-D0Z1 MODIFY DDF ALIAS(D0Z1) ADD
-D0Z2 MODIFY DDF ALIAS(D0Z2) ADD
-D0Z1 MODIFY DDF ALIAS(D0Z1) START
-D0Z2 MODIFY DDF ALIAS(D0Z2) START
```

Upon successful command completion we displayed the status of DDF on both DB2 members and obtained the command output shown in Figure 4-53.

```
-D0Z1 DIS DDF
DSNL080I  -D0Z1 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL087I ALIAS             PORT  SECPORT STATUS
DSNL088I D0Z1 0     0       STARTD
-D0Z2 DIS DDF
DSNL080I  -D0Z2 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL087I ALIAS             PORT  SECPORT STATUS
DSNL088I D0Z2              0     0       STARTD
```

Figure 4-53 Display DDF alias

Activate high performance DBAT

After we successfully activated DDF we activated high performance DBAT in both DB2 members by running a DB2 command as shown in Example 4-21 on page 161. For a discussion on high performance DBAT refer to “Configuration for high performance DBATs” on page 160

DB2 related TCP/IP configuration

For the DB2 data sharing group to be able to provide TCP/IP based DB2 connections, ideally based upon DB2 Sysplex workload balancing in conjunction with TCP/IP Sysplex Distributor, you need to cater for the following configuration:

- DB2 member specific and group Dynamic Virtual IP address (DVIPA) with automatic VIPA takeover
- DB2 member specific resynchronization port
- DB2 group well-known SQL port

DB2 member and group DVIPA

We configured TCP/IP to use the IP addresses shown in Example 4-16 to define the DVIPA resources for our data sharing group and its DB2 members.

Example 4-16 TCP/IP DVIPA configuration

```
VIPADYNAMIC
; BEGIN DB2 D0ZG
 VIPARANGE DEFINE 255.255.255.255 9.12.4.138 ; D0Z1
 VIPARANGE DEFINE 255.255.255.255 9.12.4.142 ; D0Z2
 VIPADEFINE 255.255.255.255 9.12.4.153 ; D0ZG
 VIPADISTRIBUTE DEFINE 9.12.4.153 PORT 39000 DESTIP ALL
; END   DB2 D0ZG
ENDVIPADYNAMIC
```

Port definition with IP address binding

We used the port configuration shown in Example 4-17 to define the TCP/IP ports required to support the BSDS configuration shown in “Configuration without IP address in the BSDS” on page 154.

Example 4-17 TCP/IP Port configuration with IP address binding

```
PORT
 39000 TCP D0Z1DIST(1) SHAREPORT(3) BIND(2) 9.12.4.153
 39000 TCP D0Z2DIST(1) SHAREPORT(3) BIND(2) 9.12.4.153
 39002 TCP D0Z1DIST(1) BIND(2) 9.12.4.138
 39003 TCP D0Z2DIST(1) BIND(2) 9.12.4.142
```

1. By specifying the DDF address space names in the port statements we restrict port usage to the address space given in the port statement. This prevents others from accidentally using this port number.
2. The BIND parameter causes the specified address space to bind to the IP address given in the same port statement.
3. SHAREPORT allows for D0Z1DIST and D0Z2DIST to share port 39000 which represents the well-known SQL port of the data sharing group.

Port definition without IP address binding

We used the port configuration shown in Example 4-18 to define the TCP/IP ports required to support the BSDS configuration shown in "Configuration with IP address in the BSDS" on page 154.

Example 4-18 TCP/IP port configuration without IP address binding

```
PORT
 39000 TCP D0Z1DIST SHAREPORT
 39000 TCP D0Z2DIST SHAREPORT
 39002 TCP D0Z1DIST
 39003 TCP D0Z2DIST
```

DB2 startup messages

Upon successful DDF configuration we started each data sharing member and verified the DDF configuration by reviewing the DDF start messages shown in Figure 4-54.

```
DSNL003I  -D0Z1 DDF IS STARTING
DSNL523I  -D0Z1 DSNLILNR TCP/IP SERVICES AVAILABLE
           FOR IP ADDRESS ::9.12.4.138 AND PORT 39000 1
DSNL523I  -D0Z1 DSNLILNR TCP/IP SERVICES AVAILABLE
           FOR IP ADDRESS ::9.12.4.153 AND PORT 39000 2
DSNL004I  -D0Z1 DDF START COMPLETE
           LOCATION  DB0Z
           LU        -NONE 3
           GENERICLU -NONE
           DOMAIN d0zg.itso.ibm.com
           TCPPORT   39000
           SECPORT   0 4
           RESPORT   39002
           IPNAME    IPDB0Z 5
           OPTIONS:
            PKGREL = BNDOPT
DSN9022I  -D0Z1 DSNYASCP 'STA DB2' NORMAL COMPLETION
DSNL523I  -D0Z1 DSNLIRSY TCP/IP SERVICES AVAILABLE
           FOR IP ADDRESS ::9.12.4.138 AND PORT 39002 6
```

Figure 4-54 DB2 DDF startup messages

The DDF part of the D0Z1MSTR startup messages confirmed our customization:

1. The DB2 member is ready to accept connections on SQL port 39000 and the member specific IP address
2. The DB2 member is ready to accept connections on SQL port 39000 and the data sharing group IP address
3. An IBM VTAM® LU name is not required by DRDA workloads. Most DDF connections use TCP/IP. Configuring DB2 without a VTAM LU name saves resources required for initializing and maintaining the DB2 VTAM interface.
4. SECPORT was set to 0 to disable DDF SSL processing. We intentionally used that configuration option as the DB2 DDF address space was placed in a secure network, front ended by WebSphere Application Server. SSL encryption was therefore not required.
5. We set up DDF to use IPNAME to make sure the DB2 VTAM interface is not initialized during DB2 startup.

6. Member D0Z1 is ready to accept requests on its resynchronization port which is required for all resynchronizations.

As shown in Figure 4-55 on page 159 you can alternatively issue the "DISPLAY DDF" command to review the DDF configuration of an active DB2 data sharing member. The command output below additionally shows the following DB2 system parameter settings and DDF thread management related information that are important for system monitoring and tuning:

- DT=I, DSNZPARM CMTSTAT=INACTIVE
- CONDBAT=10000, DSNZPARM CONNDBAT=10000
- MDBAT=200, DSNZP
- MAXDBAT=200
- ADBAT=0, Current number of database access threads
- QUEDBAT=0, cumulative counter that is always incremented when the DSNL090I MDBAT[1] limit has been reached
- INADBAT=0, Current number of inactive DBATs. This value only applies if the dt value specified in the DSNL090I message indicates that DDF. INACTIVE support is enabled. Any database access threads reflected here can also be observed in the DISPLAY THREAD TYPE(INACTIVE) command report.
- CONQUED=0, Current number of connection requests that have been queued and are waiting to be serviced. This value only applies if the dt value that is specified in the DSNL090I message indicates that DDF INACTIVE support is enabled.
- DSCDBAT=0, Current number of disconnected database access threads. This value only applies if the dt value specified in the DSNL090I message indicates that DDF. INACTIVE support is enabled.
- INACONN=0, current number of inactive connections. This value only applies if the dt value specified in the DSNL090I message indicates that DDF. INACTIVE support is enabled.

[1] Maximum number of database access threads as determined by the "MAX REMOTE ACTIVE" value in the DSNTIPE installation panel.

```
-D0Z1 DIS DDF DETAIL
DSNL080I  -D0Z1 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I DB0Z               -NONE             -NONE
DSNL084I TCPPORT=39000 SECPORT=0     RESPORT=39002 IPNAME=IPDB0Z
DSNL085I IPADDR=::9.12.4.153
DSNL086I SQL    DOMAIN=d0zg.itso.ibm.com
DSNL086I RESYNC DOMAIN=d0z1.itso.ibm.com
DSNL089I MEMBER IPADDR=::9.12.4.138
DSNL090I DT=I  CONDBAT=  10000 MDBAT=  200
DSNL092I ADBAT=    0 QUEDBAT=      0 INADBAT=      0 CONQUED=      0
DSNL093I DSCDBAT=      0 INACONN=      0
DSNL100I LOCATION SERVER LIST:
DSNL101I WT IPADDR            IPADDR
DSNL102I 30 ::9.12.4.142
DSNL102I 12 ::9.12.4.138
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = BNDOPT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

Figure 4-55 DB2 display DDF command output

To enable remote DB2 clients to connect to DB2 using a group domain name we then managed to have the group and member DVIPA addresses registered in our domain name server (DNS). To test the set up, we configured the DB2 for LUW database directory as shown in Example 4-19.

Example 4-19 DB2 for LUW db directory setup

```
catalog db db0z at node db0z authentication dcs;
catalog tcpip node db0z remote "d0zg.itso.ibm.com" server 39000 ostype mvs;
catalog dcs db db0z as db0z;
```

As shown in Example 4-20 we were then able to use the DB2 command line processor (CLP) to connect to the database that we cataloged in Example 4-19.

Example 4-20 DB2 DDF verification

```
db2 => connect to db0z user db2r3
Enter current password for db2r3:

   Database Connection Information

 Database server        = DB2 z/OS 10.1.5
 SQL authorization ID   = DB2R3
 Local database alias   = DB0Z

db2 => ping db0z 5

Elapsed time: 82264 microseconds
Elapsed time: 81563 microseconds
Elapsed time: 82586 microseconds
Elapsed time: 82217 microseconds
Elapsed time: 81808 microseconds
```

```
db2 =>
```

In the example shown in Example 4-20 on page 159 we issued a DB2 PING command to measure DB2 for z/OS server turnaround elapsed time. The average network turnaround time shown is higher than 0.08 seconds indicating high network latency. Depending on your throughput requirement you should expect to see turnaround times well below 0.001 seconds.

For more information about setting up DB2 for z/OS for a distributed load balancing and fault tolerant configuration refer to 3.3, “High availability configuration options” on page 92 and *DB2 9 for z/OS Data Sharing: Distributed Load Balancing and Fault Tolerant Configuration*, REDP-4449.

4.3.6 High Performance DBATs

Before DB2 10, all packages that were accessed at server through DRDA behave as RELEASE(COMMIT) even they were bound with RELEASE(DEALLOCATE). DB2 10 for z/OS, if configured for High Performance DBAT honors the RELEASE(DEALLOCATE) package bind parameter for database access threads, which reduces CPU cost for package allocation and deallocation processing. Performance results can vary and the benefits are more pronounced for short transactions.

Feature of High Performance DBATs

If a package that is associated with a distributed application is bound with RELEASE (DEALLOCATE), it is allocated to the DBAT up until the DBAT is terminated. Although CMTSTAT is set to INACTIVE, DDF does not pool the DBAT and disassociates it from its connection after the unit-of-work is ended. Thus the DBATs hold package allocation locks even while they are not being used for client unit-of-work processing.

The High Performance DBAT will be terminated after 200 (not user changeable) units-of-work are processed by it. On the next request to start an unit-of-work by the connection, a new DBAT is created or a pooled DBAT is assigned to process the unit-of-work. Normal idle thread time-out detection is applied to these DBATs. IDTHDOIN will not apply if the DBAT is waiting for the next client unit-of-work.

Configuration for high performance DBATs

High performance DBATs are available only under the following conditions:

- KEEPDYNAMIC YES is not enabled.
- CURSOR WITH HOLD is not enabled.
- CMTSTAT is set to INACTIVE.

These are the steps when dealing with High Performance DBAT

1. BIND or REBIND packages with RELEASE(DEALLOCATE)

   We recommend to bind the JDBC packages that you want to use with High Performance DBAT into their own package collection. For information about the procedure we used to bind the JDBC packages into their own collections refer to 4.3.9, “Bind JDBC packages” on page 165.

2. Use **`-MODIFY DDF PKGREL(COMMIT)`**

   When you want to increase resource concurrency and the likelihood for your SQL DDL, BIND operations, and utilities to be successfully executed while the application workload is running, you can deactivate High Performance DBAT by issuing the command **`-MODIFY DDF PKGREL(COMMIT)`**.

3. Use **`-MODIFY DDF PKGREL(BNDOPT)`** command

   This command enables the RELEASE bind option (COMMIT or RELEASE) that was previously used for remote client processing for any package that is used for remote client processing.

   Example 4-21 shows the results of the **`MODIFY DDF PKGREL`** command.

   Example 4-21 MODIFY DDF PKGREL(BNDOPT) output

```
-D0Z1 MODIFY DDF PKGREL(BNDOPT)
DSNL300I  -D0Z1 DSNLTMDF MODIFY DDF REPORT FOLLOWS:
DSNL302I PKGREL IS SET TO BNDOPT
DSNL301I DSNLTMDF MODIFY DDF REPORT COMPLETE
```

   Example 4-22 shows the results of the **`-DIS DDF`** command. You can check setting of PKGREL through message DSNL106I.

   Example 4-22 -DIS DDF command reporting the PKGREL option

```
-D0Z1 DIS DDF
DSNL080I  -D0Z1 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I DB0Z               -NONE             -NONE
DSNL084I TCPPORT=39000 SECPORT=0     RESPORT=39002 IPNAME=IPDB0Z
DSNL085I IPADDR=::9.12.4.153
DSNL086I SQL    DOMAIN=d0zg.itso.ibm.com
DSNL086I RESYNC DOMAIN=d0z1.itso.ibm.com
DSNL089I MEMBER IPADDR=::9.12.4.138
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = BNDOPT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

Because activating High Performance DBAT for distributed applications avoids pooling of DBATs, you might have to increase subsystem parameter MAXDBAT to avoid queuing of distributed requests.

BNDOPT is the default value of the **`MODIFY DDF PKGREL`** command.

By using these commands, you do not need to perform REBIND to activate or deactivate High Performance DBAT.

JDBC bind recommendation

High Performance DBATs and bind option KEEPDYNAMIC(YES) are mutually exclusive. You need to choose between using High Performance DBAT and bind option KEEPDYNAMIC(YES) depending on the characteristics of your applications.

- Bind option KEEPDYNAMIC(YES) is recommended for applications with a limited amount of SQL statements that are frequently executed.
- High Performance DBAT fits best for a light transaction environment.

To allow for High Performance DBAT to be chosen on application level we recommend to create dedicated package collections which you explicitly bind with RELEASE(DEALLOCATE) and which you explicitly choose to be used by your application by setting the setCurrentPackagePath data source custom property to the name of the package collection ID. For details, see 5.11.2, “currentPackagePath” on page 292.

In our environment we created to following JDBC package collections to support this intention:

- Collection JDBCHDBAT bound with RELEASE(DEALLOCATE)
- Collection JDBCNOHDBAT bound with RELEASE(COMMIT)

For more information about how we created these package collections refer to 4.3.9, “Bind JDBC packages” on page 165.

4.3.7 IBM Data Server Driver for JDBC and SQLJ

Our workload scenario accesses DB2 for z/OS using JDBC type 2 and JDBC type 4 connections. You use a JDBC type 2 connection to establish a local connection to a DB2 system or a data sharing member running in the same z/OS system as your JDBC application. You use a JDBC type 4 connection to establish a remote connection to a DB2 subsystem or data sharing group. In case of JDBC type 4 the connection requires a TCP/IP network to be available between the DB2 client and the DB2 server. An overview over DB2 clients connecting to DB2 for z/OS using JDBC type 2 and type 4 connections is illustrated in Figure 4-56. See 3.2.1, “Connectivity options for IBM Data Server Driver for JDBC and SQLJ” on page 89 for more information.

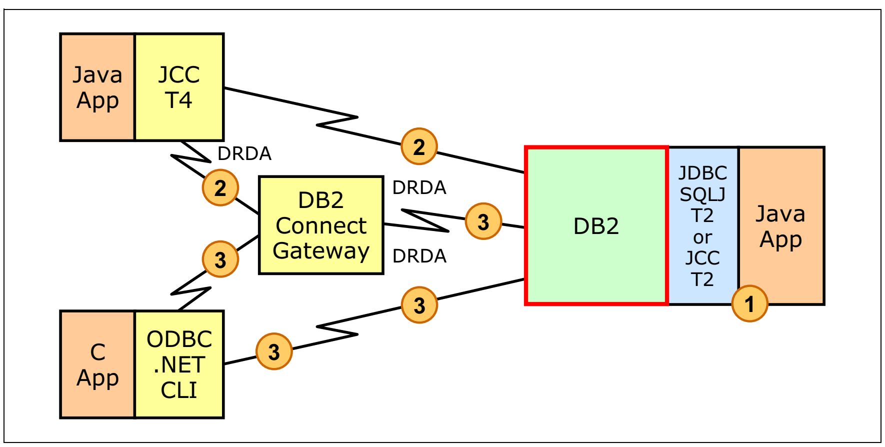


Figure 4-56 Overview applications using JDBC type 2 and type 4

1. A Java application running on z/OS uses JDBC type 2 to connect to DB2 for z/OS
2. A Java application uses JDBC type 4 to directly or indirectly connect to DB2 for z/OS. The Java application can run on z/OS or non-z/OS platforms.
3. A multiplatform ODBC, .NET or DB2 call level interface (CLI) client directly or indirectly connects to DB2 for z/OS using the IBM Data Server Driver for ODBC and CLI.

Driver configuration

As explained in 4.1.7, “UNIX System Services file system configuration” on page 123 the JDBC driver related files have been installed by SMP/E and made available to our runtime environment by using symbolic links that we defined to point to the appropriate zFS file system data sets.

As a prerequisite for binding the JDBC packages using the DB2Binder utility under UNIX System Services we need to complete the UNIX System Services JDBC configuration to support High Performance DBAT and the DB2 command line processor.

Based on the JDBC install base we carry out the following configuration tasks:

- Set DB2 subsystem parameter DESCSTAT to YES as already discussed in "DESCRIBE FOR STATIC field (DESCSTAT)" on page 151
- STEPLIB libraries
  The following load libraries need to be available through STEPLIB data set allocation in case the application used JDBC type 2 connections to access DB2.
  - DB0ZT.SDSNEXIT
  - DB0ZT.SDSNLOAD
  - DB0ZT.SDSNLOD2

  The SDSNLOD2 library contains the JDBC type 2 DLL load modules which are referred to by UNIX System Services through external link definitions (see 4.3.8, "JDBC type 2 DLL and the SDSNLOD2 library" on page 164 for details).

  Our WebSphere Application Server environment defines these data sets in application server STEPLIB concatenation to cater for the JDBC type 2 requirement.
- Modify the global UNIX System Service profile (/etc/profile) to customize the environment variable settings to reflect the JDBC libraries, paths, and files that the IBM Data Server Driver for JDBC and SQLJ uses. We used the export commands shown in Figure 4-57 to perform these changes.

```
export PATH=/usr/lpp/db2/d0zg/jdbc/bin:$PATH
export LIBPATH=/usr/lpp/db2/d0zg/jdbc/lib:$LIBPATH
export CLASSPATH=/usr/lpp/db2/d0zg/jdbc/classes/db2jcc.jar: \
/usr/lpp/db2/d0zg/jdbc/classes/db2jcc_javax.jar: \
/usr/lpp/db2/d0zg/jdbc/classes/sqlj.zip: \
/usr/lpp/db2/d0zg/jdbc/classes/db2jcc_license_cisuz.jar: \
$CLASSPATH
```

Figure 4-57 /JDBC etc/profile changes

- Enable the DB2-supplied stored procedures
  The following IBM Data Server Driver for JDBC and SQLJ required stored procedures have been implemented in our environment during DB2 installation.
  - SQLCOLPRIVILEGES
  - SQLCOLUMNS
  - SQLFOREIGNKEYS
  - SQLFUNCTIONCOLS
  - SQLFUNCTIONS
  - SQLGETTYPEINFO
  - SQLPRIMARYKEYS
  - SQLPROCEDURECOLS
  - SQLPROCEDURES
  - SQLPSEUDOCOLUMNS
  - SQLSPECIALCOLUMNS
  - SQLSTATISTICS
  - SQLTABLEPRIVILEGES
  - SQLTABLES

– SQLUDTS
– SQLCAMESSAGE

You can run installation job DSNTIJRV to confirm that these procedure have been appropriately implemented.

For more information about installing and setting up the IBM Data Server Driver for JDBC refer to Chapter 8. Installing the IBM Data Server Driver for JDBC and SQLJ of *DB2 10 for z/OS, Application Programming Guide and Reference for Java*, SC19-2970.

4.3.8 JDBC type 2 DLL and the SDSNLOD2 library

The JDBC type 2 DLL[2] are loaded through the JDBC DLL directory that we specified in the export LIBPATH command shown in Figure 4-57 on page 163. When we ran the UNIX System Services command shown in Figure 4-58 we noticed that the executables in the JDBC type 2 DLL directory consist of external links that point to DLL load modules that reside outside the unix file system in the SDSNLOD2 load library.

```
ls -l /usr/lpp/db2/d0zg/jdbc/lib
 1                                                         2
 erwxrwxrwx   8 Nov 16  2010 libdb2jcct2zos.so     -> DSNAQJL2
 erwxrwxrwx   8 Nov 16  2010 libdb2jcct2zos4.so    -> DSNAJ3L2
 erwxrwxrwx   8 Nov 16  2010 libdb2jcct2zos4_64.so -> DSNAJ6L2
 erwxrwxrwx   8 Nov 16  2010 libdb2jcct2zos_64.so  -> DSNAQ6L2
```

Figure 4-58 JDBC type 2 DLL external links

1. The first character of the command output (the e character) identifies the file as an external link.
2. Following the right arrow the output shows the name of the external load module the external link points to.

When the runtime environment loads a DLL that refers to an external load module it uses the following search order when locating the DLL:

1. STEPLIB
2. Link Pack Area (LPA)
3. z/OS Linklist

To be able to use the JDBC type 2 driver we included the SDSNLOD2 library in the WebSphere Application Server STEPLIB library concatenation. When we listed the members of the SDSNLOD2 library as shown in Figure 4-59 we located the external load module names referred to in Figure 4-58.

```
BROWSE            DB0ZT.SDSNLOD2
Command ===>
           Name       Size      TTR     AC   AM   RM
_________ DSNAJ3L2   00064F68   000010   00    31  ANY
_________ DSNAJ6L2   00082FB8   00000E   00    64  ANY
_________ DSNAQJL2   00064E40   00000F   00    31  ANY
_________ DSNAQ6L2   00082E48   00000D   00    64  ANY
```

Figure 4-59 JDBC type 2 DLL in SDSNLOD2

[2] dynamic link library

During DB2 installation SMP/E executes the UNIX System Services commands shown in Figure 4-60 to associate the SDSNLOD2 load modules shown in Figure 4-59 on page 164 with the UNIX System Services path names shown in Figure 4-58 on page 164.

```
ln -e DSNAQJL2 /usr/lpp/db2a10/jdbc/lib/libdb2jcct2zos.so
ln -e DSNAJ3L2 /usr/lpp/db2a10/jdbc/lib/libdb2jcct2zos4.so
ln -e DSNAJ6L2 /usr/lpp/db2a10/jdbc/lib/libdb2jcct2zos4_64.so
ln -e DSNAQ6L2 /usr/lpp/db2a10/jdbc/lib/libdb2jcct2zos_64.so
```

Figure 4-60 UNIX System Services SDSNLOD2 external link definition

4.3.9 Bind JDBC packages

Upon successful DDF implementation and after we completed the JDBC driver configuration we use the DB2Binder utility to bind the JDBC packages in our DB2 for z/OS system into the following two package collection IDs:

- NULLID - default collection ID used if no collection ID is set in the setCurrentPackagePath data source custom property.
- JDBCHDBAT - collection ID for applications wanting to take advantage of DB2 High Performance DBAT. Packages in this collection ID are bound with bind parameter RELEASE(DEALLOCATE).
- JDBCNOHDBAT - collection ID for applications not wanting to take advantage of DB2 High Performance DBAT. Packages in this collection ID are bound with bind parameter RELEASE(COMMIT).

By using dedicated JDBC collections we deliberately do not change the NULLID collection ID which is commonly used by the majority of DB2 remote applications. Globally rebinding packages belonging to the NULLID collection with RELEASE(DEALLOCATE) is not suitable, because some of your workload better qualifies for using bind options KEEPDYNAMIC(YES) and RELEASE(COMMIT). See 4.3.6, “High Performance DBATs” on page 160 where we discuss these bind options.

The DB2Binder invocation examples shown in Example 4-23, Example 4-24 on page 167, and in Example 4-25 on page 167 use a JDBC type 4 connection to bind the packages shown in Figure 4-61.

```
Binder performing action "add" to
"jdbc:db2://d0zg.itso.ibm.com:39000/DB0Z"
under collection "JDBCHDBAT":
Package "SYSSTAT": Bind succeeded.
Package "SYSLH100": Bind succeeded.
Package "SYSLH200": Bind succeeded.
Package "SYSLH300": Bind succeeded.
Package "SYSLH400": Bind succeeded.
Package "SYSLN100": Bind succeeded.
Package "SYSLN200": Bind succeeded.
Package "SYSLN300": Bind succeeded.
Package "SYSLN400": Bind succeeded.
Package "SYSLH101": Bind succeeded.
Package "SYSLH201": Bind succeeded.
Package "SYSLH301": Bind succeeded.
Package "SYSLH401": Bind succeeded.
Package "SYSLN101": Bind succeeded.
Package "SYSLN201": Bind succeeded.
Package "SYSLN301": Bind succeeded.
Package "SYSLN401": Bind succeeded.
Package "SYSLH102": Bind succeeded.
Package "SYSLH202": Bind succeeded.
Package "SYSLH302": Bind succeeded.
Package "SYSLH402": Bind succeeded.
Package "SYSLN102": Bind succeeded.
Package "SYSLN202": Bind succeeded.
Package "SYSLN302": Bind succeeded.
Package "SYSLN402": Bind succeeded.
DB2Binder finished.
```

Figure 4-61 JDBC packages bound by DB2Binder utility

Important: The DB2Binder utility requires a JDBC type 4 connection. Binding the JDBC packages therefore require the DB2 Distributed Data Facility (DDF) address space to be operating even if you only plan to use JDBC type 2 connections which do not require DDF.

Package collection NULLID

To bind the JDBC packages into collection NULLID we executed the DB2Bind command shown in Example 4-23 under UNIX System Services.

Example 4-23 Bind NULLID package collection

```
java com.ibm.db2.jcc.DB2Binder -url \
 jdbc:db2://d0zg.itso.ibm.com:39000/DB0Z \
 -user DB2R3  -password <password> -collection NULLID    \
 -action replace
```

Package collection JDBCHDBAT

As recommended in "JDBC bind recommendation" on page 161 we create JDBC package collections to provide support for High Performance DBAT. JDBC applications potentially enable themselves for High Performance DBAT processing by including the JDBCHDBAT collection ID in their setCurrentPackagePath data source custom property setting.

To bind the JDBC packages into collection JDBCHDBAT we executed the DB2Bind command shown in Example 4-24 under UNIX System Services.

Example 4-24 Bind High Performance DBAT eligible package collection

```
java com.ibm.db2.jcc.DB2Binder -url \
 jdbc:db2://d0zg.itso.ibm.com:39000/DB0Z \
 -user DB2R3  -password <password> -collection JDBCHDBAT \
 -release deallocate
```

For information about the execute privileges that we granted on these packages refer to "Grant execute privileges on JDBC packages" on page 171

Package collection JDBCNOHDBAT

Our JDBC applications exclude themselves from High Performance DBAT processing by including the JDBCNOHDBAT collection ID in their setCurrentPackagePath data source custom property setting.

In z/OS UNIX System Services we ran the DB2Bind command shown in Example 4-25 to bind the JDBC packages into collection JDBCNOHDBAT

Example 4-25 Bind High Performance DBAT ineligible package collection

```
java com.ibm.db2.jcc.DB2Binder -url \
 jdbc:db2://d0zg.itso.ibm.com:39000/DB0Z \
 -user DB2R3  -password <password> -collection JDBCNOHDBAT \
 -release commit
```

For information about the execute privileges that we granted on these packages refer to "Grant execute privileges on JDBC packages" on page 171.

4.3.10 UNIX System Services command line processor configuration

DB2 for z/OS provides a Java based DB2 command line processor that runs under UNIX System Services. Command line processor (CLP) uses the IBM Data Server Driver for JDBC to connect to DB2 using a JDBC type 4 connection. Besides the SQL capabilities that you have with SPUFI. you can use CLP to bind packages using DBRMs that are stored in a UNIX System Services file system directory, invoke stored procedures, register and remove XML schemas, issue describe for tables and SQL call statements.

As CLP is a Java application that connects to DB2 using a JDBC type 4 connection it provides an excellent tool to check out your local JDBC configuration. You can invoke CLP from a UNIX System Services shell and as such it can be invoked from telnet, secure shell, under TSO from OMVS, from BPXBATCH or from the JZOS batch launcher. Because CLP connects to DB2 through a JDBC type 4 connection it furthermore triggers zIIP offload for local database connections.

For the CLP implementation we performed the following configuration tasks:

- Change global profile in /etc/profile to include the clp.jar file in the CLASSPATH configuration and configure the CLPPROPERTIES variable
  - export CLPHOME=/usr/lpp/db2/d0zg/base
  - export CLASSPATH=$CLPHOME/lib/clp.jar
  - export CLPPROPERTIES=$HOME/clp.roperties
- Copy file /usr/lpp/db2/d0zg/base/samples/clp.properties into the home directory
- Customize local clp.properties file
- Define the following alias in the global profile (/etc/profile)
  - alias db2="java com.ibm.db2.clp.db2"
- Invoke CLP from an UNIX System Services shell
  - db2
    Within CLP we then ran the commands shown in Figure 4-62 to check out our local JDBC configuration.

```
db2 => connect to 9.12.4.153:39000/DBOZ user DB2R3 using <password>;
connect to 9.12.4.153:39000/DBOZ user DB2R3 using <password>
com.ibm.net.SocketKeepAliveParameters

     Database Connection Information
 Database server        =DB2 DSN10015
 SQL authorization ID   =DB2R3
 JDBC Driver            =IBM Data Server Driver for JDBC and SQLJ
4.13.136

DSNC101I : The "CONNECT" command completed successfully.

db2 => select current server from sysibm.sysdummy1;
select current server from sysibm.sysdummy1
1
DBOZ
  1 record(s) selected


db2 => terminate;
```

Figure 4-62 DB2 CLP to check out JDBC configuration

For more information about implementing and using the DB2 UNIX System Services CLP refer to:

- GC19-2974-07, DB2 10 for z/OS, Installation and Migration Guide, Configuring the DB2 command line processor
- SC19-2972-04, DB2 10 for z/OS, Command Reference, Chapter 9. Command line processor

4.3.11 Using the TestJDBC Java sample

DB2 for z/OS provides the TestJDBC Java sample program in the JDBC samples directory illustrated in Figure 4-63 on page 169.

```
DB2R3 @ SC64:/u/db2r3>ls -l /usr/lpp/db2/d0zg/jdbc/samples
total 66
drwxr-xr-x   2 HARJANS  TTY          320 Nov 16  2010 IBM
-rw-r--r--   2 HARJANS  TTY        13783 Jun 12 15:06 TestJDBC.java
-rw-r--r--   2 HARJANS  TTY        11752 Jun 12 15:06 TestSQLJ.sqlj
```

Figure 4-63 TestJDBC samples directory

The TestJDBC application exercises basic JDBC functionality (by default through a Type-2 z/OS Connection) using the DB2 JDBC Driver. TestJDBC receives its parameters (JDBC connect url either in type 2 or type 4 format) as input argument. in Figure 4-64 we use the TestJDBC application to confirm appropriate driver installation. For more information about the TestJDBC application and the input parameters it supports, see the inline documentation of the TestJDBC Java program.

```
javac TestJDBC.java
java TestJDBC jdbc:db2://d0zg.itso.ibm.com:39000/DB0Z db2r3 <password>

Loading DB2 JDBC Driver: com.ibm.db2.jcc.DB2Driver
com.ibm.net.SocketKeepAliveParameters
  successful driver load, version 4.13.136

Establishing Connection to URL: jdbc:db2://d0zg.itso.ibm.com:39000/DB0Z
  successful connect

Acquiring DatabaseMetaData
  successful... product version: DSN10015

Creating Statement
  successful creation of Statement

About to execute SELECT
  successful execution of SELECT

About to fetch from ResultSet, maxRows=10
    CREATOR: <ATE> NAME: <DEPT>
...
```

Figure 4-64 Invoke TestJDBC application

4.3.12 DB2 security considerations

The WebSphere Application Server instances illustrated in Figure 4-2 on page 104 access our data sharing members from within the Parallel Sysplex and within a secure TCP/IP network. We therefore had no need to use SSL encryption in DB2 for z/OS. Beyond this we performed the following security configuration:

- Permit users to create DB2 connections using RRSAF and DDF
- Grant access to JDBC packages
- Grant access to the DayTrader-EE6 tables

We did not use DB2 SSL encryption because our DB2 data sharing members.

Allow users to connect to DB2 using RRSAF and DDF

In a WebSphere Application Server environment the following users need to be authorized to connect to DB2:

- WebSphere Application Server Deployment Manager address space user for performing data source connection testing
- WebSphere Application Server JAAS alias user names referred to in data source definitions
- User IDs referred to in DB2 trusted context WITH USE FOR clauses

To allow these users to connect to DB2 through JDBC type 2 or JDBC type 4 we executed the RACF commands shown in Example 4-26.

Example 4-26 RACF DSNR class D0Z.DIST, D0Z*.RRSAF*

```
/* secure RRSAF and DIST connection creation */
RDEFINE DSNR (D0Z*.DIST)   UACC(NONE) OWNER(DB2)
RDEFINE DSNR (D0Z*.RRSAF)  UACC(NONE) OWNER(DB2)
/* PERMIT group D0ZGDIST to access D0Z*.RRS  */
AG D0ZGRRS  OWNER(DB2) SUPGROUP(DB2)  /* RRSAF RACF GROUP*/
CO WASTEST  GROUP(D0ZGRRS)
CO WASUSER  GROUP(D0ZGRRS)
CO MZACRU   GROUP(D0ZGRRS)
CO MZADMIN  GROUP(D0ZGRRS)
CO MZASRU   GROUP(D0ZGRRS)
CO WASCTX1  GROUP(D0ZGRRS)
CO WASCTX2  GROUP(D0ZGRRS)
CO WASCTX3  GROUP(D0ZGRRS)
PERMIT  D0Z*.RRSAF CLASS(DSNR) ID(D0ZGRRS)  ACCESS(READ)
/* PERMIT group D0ZGDIST to access D0Z*.DIST */
AG D0ZGDIST OWNER(DB2) SUPGROUP(DB2)
CO WASTEST  GROUP(D0ZGDIST)
CO WASUSER  GROUP(D0ZGDIST)
CO MZACRU   GROUP(D0ZGDIST)
CO MZADMIN  GROUP(D0ZGDIST)
CO MZASRU   GROUP(D0ZGDIST)
CO WASCTX1  GROUP(D0ZGDIST)
CO WASCTX2  GROUP(D0ZGDIST)
CO WASCTX3  GROUP(D0ZGDIST)
PERMIT  D0Z*.DIST CLASS(DSNR) ID(D0ZGDIST) ACCESS(READ)
SETROPTS RACLIST(DSNR) REFRESH
```

Plan authorization considerations

In our workload scenario we drive the DayTrader-EE6 workload using JDBC type 2 and JDBC type 4 connections.

JDBC type 4 is not using user bound application plans, it uses DB2 packages.

With JDBC type 2 you can bind your own application plan with a package list referring to the JDBC package collection ID that you intend to use. If you intend to use an application plan for your JDBC type 2 connections you will have to take care of plan authorization. In our workload scenario we do not use an application plan for JDBC type 2 connections. Instead we use JDBC packages which we authorized as described in "Grant execute privileges on JDBC packages" on page 171. Plan authorization for our DayTrader-EE6 JDBC type 2 workload scenario was therefore not required.

Grant execute privileges on JDBC packages

We bound the JDBC packages into the package collections JDBCHDBAT and JDBCNOHDBAT. For these collections we ran the SQL data control language (DCL) statements shown in Example 4-27 to revoke the execute privilege from PUBLIC and to grant execute authorization to the packages of these collection IDs. The GRANT TO PUBLIC was implicitly performed by the DB2Binder utility that we explained in 4.3.9, “Bind JDBC packages” on page 165.

Example 4-27 Grant execute authorization on packages

```
revoke execute on package  jdbchdbat.*    from public;
revoke execute on package  jdbcnohdbat.*  from public;
grant  execute on package  NULLID.SYSSTAT to MZASRU;
grant  execute on package  jdbchdbat.*    to role  WASTESTROLE;
grant  execute on package  jdbchdbat.*    to role  DTRADEROLE;
grant  execute on package  jdbcnohdbat.*  TO ROLE  WASTESTROLE;
grant  execute on package  jdbcnohdbat.*  TO ROLE  DTRADEROLE;
```

WebSphere Application Server Deployment Manager package authorization

When we used the WebSphere Application Server Integrated Solutions Console (ISC) to perform a data source connection test as described in 6.6.1, “Data source connection tests on z/OS” on page 328, we obtained the error message shown in Figure 4-65 indicating a lack of package execute authorization on package NULLID.SYSSTAT.

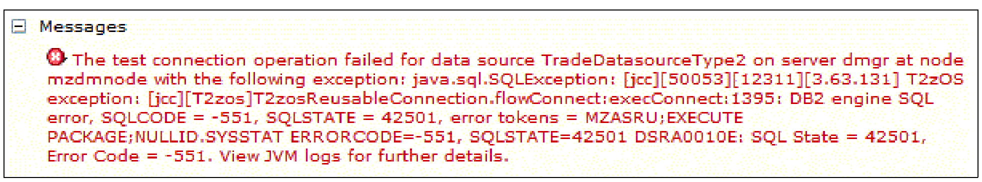


Figure 4-65 WebSphere deployment manager data source test error message

DB2 performed authorization checking on package NULLID.SYSSTAT because we configured the JDBC type 2 data source to use a DB2 package list for locating DB2 packages required for SQL execution. As a consequence WebSphere Application Server Deployment Manager created a DB2 RRSAF thread with a RRSAF default plan name of ?RRSAF using the NULLID collection ID for package allocation. DB2 package NULLID.SYSSTAT caused the error shown in message Figure 4-65 because it was the first package the connection test tried to use.

The error message shown in Figure 4-65 refers to mzdmnode with user MZASRU not being authorized to execute package NULLID.SYSSTAT which we did not expect as we had configured the data source to use package collection JDBCHBAT and to use the JAAS alias user name for creating the connection to DB2. Instead, the data source connection request was performed by the WebSphere Application Server deployment manager address space user (MZASRU) trying to probe the DB2 connection using package NULLID.SYSSTAT.

Because we had activated the DB2 audit trace to keep track of authorization failures we could confirm this behavior through the audit report shown in Figure 4-66.

```
 LOCATION: DBOZ                  OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1M1)
      GROUP: DBOZG                      AUDIT REPORT - DETAIL
     MEMBER: D0Z2
  SUBSYSTEM: D0Z2                     ORDER: PRIMAUTH-PLANNAME
DB2 VERSION: V10                            SCOPE: MEMBER
PRIMAUTH CORRNAME CONNYPE
ORIGAUTH CORRNMBR INSTNCE
PLANNAME CONNECT              TIMESTAMP   TYPE                             DETAIL
-------- -------- ----------- ----------- -------- ---------------------------------------------------
MZASRU   MZDMGRS  RRS        04:31:38.90 AUTHFAIL AUTHID CHECKED: MZASRU   PRIVILEGE: EXECUTE
MZASRU   'BLANK'  CA82947981B                     OBJECT TYPE   : PACKAGE  REASON:       0 RC: -    1
?RRSAF   RRSAF                                    SOURCE OBJECT : SYSSTAT  SOURCE OWNER:   NULLID
                                                  TARGET OBJECT : N/A      TARGET OWNER:   N/A
                                                  MLS    RID    : N/P      SECLABEL:       N/P
                                                  TEXT:  N/P
```

Figure 4-66 WebSphere deployment manager AUTHFAIL audit report

After we had granted the deployment manager address space user the privilege to execute the NULLID.SYSSTAT package as shown in Example 4-28 we successfully completed the data source connection test using the ISC application.

Example 4-28 Grant NULLID.SYSSTAT privilege to deployment manager

```
grant  execute on package  NULLID.SYSSTAT to MZASRU ;
```

Upon data source connection test completion we received the ISC message box shown in Figure 4-67.

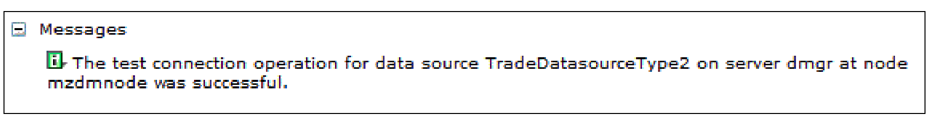


Figure 4-67 WebSphere deployment manager data source test successful

Grant access to the DayTrader-EE6 tables

We ran the SQL grant statements shown in Example 4-29 to grant the required table privileges to role DTRADEROLE. The role s assigned through trusted context when running the DayTrader-EE6 application.

Example 4-29 Grant DayTrader-EE6 table privileges

```
GRANT ALL ON TABLE SG248074.HOLDINGEJB          TO ROLE DTRADEROLE;
GRANT ALL ON TABLE SG248074.ACCOUNTPROFILEEJB   TO ROLE DTRADEROLE;
GRANT ALL ON TABLE SG248074.QUOTEEJB            TO ROLE DTRADEROLE;
GRANT ALL ON TABLE SG248074.KEYGENEJB           TO ROLE DTRADEROLE;
GRANT ALL ON TABLE SG248074.ACCOUNTEJB          TO ROLE DTRADEROLE;
GRANT ALL ON TABLE SG248074.ORDEREJB            TO ROLE DTRADEROLE;
```

4.3.13 Trusted context

A trusted context object is entirely defined in DB2 and is used to establish a trusted relationship between DB2 and an external entity. An external entity includes the following types of DB2 for z/OS clients:

- DB2 allied address space locally connect to DB2 through RRSAF, TSO or the CAF attachment facility interface. WebSphere Application Server connecting to DB2 through JDBC type 2 use RRSAF DB2 attachment interface.

  Note: APAR PM69429 adds support for Trusted Context calls for a CAF application.

- DRDA application requestors connected to DB2 through database access threads (DBAT). WebSphere Application Server connecting to DB2 through JDBC type 4 use the DB2 Distributed Data Facility (DDF) interface.

During connection processing DB2 evaluates a set of trust attributes to determine if a specific context is to be trusted. The trust attributes specify a set of characteristics about a specific connection. These attributes include the IP address, domain name, or SERVAUTH security zone name for remote DRDA clients and the job or task name for local clients.

In case the trusted context applies DB2 performs all authorization checking using the authorization ID or database role that assigned by the trusted context

4.3.14 Trusted context application scenarios

Based on the information we provided in “Authentication in a three-tier architecture using DB2 trusted context” on page 128 we explain the use of a trusted context in a WebSphere Application Server environment for the following scenarios:

- DayTrader-EE6 application workload using JDBC type 2 connections
- DayTrader-EE6 application workload using JDBC type 4 connections
- IBM Data Web Service servlet application D0ZG_QueriesWASTestTC1_war using a JDBC type 4 connection

4.3.15 DayTrader-EE6 application using JDBC connections

The DayTrader-EE6 scenario does not require any further configuration in WebSphere Application Server, because the trusted context definitions we use for that application do not perform an authorization ID switch.

A trusted context is based on the system authid (in WebSphere Application Server often referred to as the technical data source user) and a set of trust attributes. We describe the trust attributes that we used for running the DayTrader-EE6 application in “DayTrader-EE6 JDBC type 2 related trusted context attributes” on page 174, “DayTrader-EE6 JDBC type 4 related trusted context attributes” on page 174.

DayTrader-EE6 JDBC type 2 related trusted context attributes

For JDBC type 2 connections the trusted context provides the job names (address space names) from which local data base connections are established. A star can be specified for the last character of the job name. For WebSphere Application Server on z/OS this are the address space names of the WebSphere Application Server servant region establishing the JDBC type 2 connections. Example 4-30 shows a trusted context covering the attributes that we specified for our DayTrader-EE6 application server JCBC type 2 environment.

Example 4-30 JDBC type 2 trusted context with system authid and job name

```
CREATE TRUSTED CONTEXT CTXDTRADET2
        BASED UPON CONNECTION USING SYSTEM AUTHID MZADMIN 1
        ATTRIBUTES (JOBNAME 'MZSR01*') 2
        DEFAULT ROLE DTRADEROLE 3
        WITHOUT ROLE AS OBJECT OWNER
        ENABLE
```

1. Data source JAAS alias user name. This user name is often referred to by the data source technical user.
2. Address space names of our WebSphere Application Server servant regions. Our STC names start with the characters MZSR01
3. Optional: DB2 role to be assigned when the trusted context is applied

We use the trusted context shown in Example 4-30 to run the JDBC type 2 DayTrader-EE6 workload. Because database privileges are exercised by role DTRADEROLE the data source user MZADMIN does not need to hold any privileges in DB2. This solves an important audit concern as MZADMIN can no longer be used to access data in DB2 within or outside the trusted context. Granting privileges to a role increases data security further as a role is unusable outside a trusted context.

DayTrader-EE6 JDBC type 4 related trusted context attributes

For JDBC type 4 connections the trusted context contains the IP addresses or domain names from which DRDA connections are established. Generic names are not supported. For WebSphere Application Server instances the ADDRESS attribute includes IP addresses or domain names of the IP host the application server instances run in. We recommend the use of domain names to avoid problems in case a server dynamically obtains its IP address using domain name service (DNS). Example 4-31 shows a trusted context covering the attributes that we specified for the DayTrader-EE6 application server JCBC type 4 environment.

Example 4-31 JDBC type 4 trusted context with system authid and address

```
CREATE TRUSTED CONTEXT CTXDTRADET4
  BASED UPON CONNECTION USING SYSTEM AUTHID MZADMIN 1
  DEFAULT ROLE DTRADEROLE 3
  WITHOUT ROLE AS OBJECT OWNER
  ENABLE
  NO DEFAULT SECURITY LABEL
  ATTRIBUTES (
  ENCRYPTION 'NONE',
  ADDRESS 'wtsc64.itso.ibm.com', 2
  ADDRESS 'd0z1.itso.ibm.com',
  ADDRESS 'wtsc63.itso.ibm.com',
  ADDRESS 'd0z2.itso.ibm.com'
  ) ;
```

1. Data source JAAS alias user name
2. Domain names the application server instance runs on
3. Optional: DB2 role to be assigned if the trusted context is to be applied

We use the trusted context shown in Example 4-31 on page 174 to run the JDBC type 4 DayTrader-EE6 workload. Because database privileges are exercised by role DTRADEROLE the data source user MZADMIN does not need to hold any privileges in DB2. This solves an important audit concern as MZADMIN can no longer be used to access data in DB2 within or outside the trusted context. Granting privileges to a role increases data security further as a role is unusable outside the trusted context.

4.3.16 Data Web Service servlet with trusted context AUTHID switch

The IBM Data Web Service servlet application that we use requires the application server data source configuration steps described in 5.9, "Enabling trusted context for applications that are deployed in WebSphere Application Server" on page 276, because this application has been configured to use HTTP base authentication to enable the application server to pass the ID of the authenticated user for trusted context AUTHID ID switch to DB2.

Building the Data Web Service application

We use the IBM Data Studio full client Web Services feature to generate an IBM Data Web Services servlet application that provides Web Service operations for the SQL statements shown in Example 4-32 and in Example 4-33. For further reference, we use the name D0ZG_QueriesWASTestTC1_war when we refer to that application.

Example 4-32 Data Web Service query to select DB2 special registers

```
SELECT
 CURRENT CLIENT_ACCTNG      AS   CLIENT_ACCTNG
,CURRENT CLIENT_APPLNAME    AS   CLIENT_APPLNAME
,CURRENT CLIENT_USERID      AS   CLIENT_USERID
,CURRENT CLIENT_WRKSTNNAME  AS   CLIENT_WRKSTNNAME
,CURRENT PATH               AS   PATH
,CURRENT SCHEMA             AS   SCHEMA
,CURRENT TIMESTAMP          AS   TIMESTAMP
,CURRENT TIMEZONE           AS   TIMEZONE
,CURRENT SERVER             AS   LOCATION
,GETVARIABLE('SYSIBM.DATA_SHARING_GROUP_NAME') AS GROUPNAME
,GETVARIABLE('SYSIBM.SSID') AS SSID
,GETVARIABLE('SYSIBM.SYSTEM_NAME') AS ATTACH
,GETVARIABLE('SYSIBM.VERSION') AS DB2VERSION
,GETVARIABLE('SYSIBM.PLAN_NAME') AS PLAN
,GETVARIABLE('SYSIBM.PACKAGE_NAME') AS PACKAGE
,GETVARIABLE('SYSIBM.PACKAGE_SCHEMA') AS COLLID
FROM SYSIBM.SYSDUMMY1;
```

Example 4-33 Data Web Service query to invoke the GRACFGRP external scalar UDF

```
SELECT T.* FROM XMLTABLE
 ('$d/GROUPS/GROUP'
   PASSING XMLPARSE (DOCUMENT GRACFGRP()) AS "d"
 COLUMNS
  "RACF User"    VARCHAR(08) PATH '../USER/text()',
```

```
  "RACF Group"   VARCHAR(08) PATH './text()'
 ) AS T
```

The external UDF GRACFGRP is an assembler program that extracts the RACF groups the current UDF caller is connected to from the RACF ACEE control block which has been created by DB2 because of the SECURITY USER UDF attribute. GRACFGRP then returns an XML document containing the RACF group names as a VARCHAR scalar value. Listing of DDL and ASM is provided in

For information about how to convert SQL statements into IBM Data Web Services, refer to *IBM Data Studio V2.1: Getting Started with Web Services on DB2 for z/OS*, REDP-4510.

Creating the trusted context

The D0ZG_QueriesWASTestTC1_war WebSphere Application Server servlet application has been configured to use a JDBC type 4 connection to connect to DB2. It invokes an IBM Data Web Service operation to obtain either the DB2 special registers referred to in Example 4-32 on page 175 or to invoke the external scalar UDF referred to in Example 4-33 on page 175. When you create the trusted context you can use one of the following options to control the user names that are to be enabled for trusted context switching:

- Trusted context users hard coded in the DDL
- Trusted context users controlled by RACF profile

Trusted context users hard coded in the DDL

We created the trusted context using the SQL DDL shown in Figure 4-68 on page 177. In that sample we hard code the list of trusted context users (WASCTX1 through WASCTX9) in the DDL. As a consequence you need to alter the trusted context if you want to change the list of trusted context users.

```
CREATE  ROLE WASTESTDEFAULTROLE;
CREATE  ROLE WASTESTROLE;

GRANT EXECUTE ON SPECIFIC FUNCTION DB2R3.GRACFGRP
   TO ROLE WASTESTROLE;

CREATE TRUSTED CONTEXT  CTXWASTESTT4 1
        BASED UPON CONNECTION USING SYSTEM AUTHID WASTEST 2
        ATTRIBUTES (ADDRESS 'wtsc63.itso.ibm.com', 3
                    ADDRESS 'wtsc64.itso.ibm.com',
                    ADDRESS 'd0z1.itso.ibm.com',
                    ADDRESS 'd0z2.itso.ibm.com')
        DEFAULT ROLE WASTESTDEFAULTROLE 4
        WITHOUT ROLE AS OBJECT OWNER WITH USE FOR
        WASCTX1 ROLE WASTESTROLE 5,
        WASCTX2 ROLE WASTESTROLE  ,
        WASCTX3 ROLE WASTESTROLE  ,
        WASCTX4 ROLE WASTESTROLE  ,
        WASCTX5 ROLE WASTESTROLE  ,
        WASCTX6 ROLE WASTESTROLE  ,
        WASCTX7 ROLE WASTESTROLE  ,
        WASCTX8 ROLE WASTESTROLE  ,
        WASCTX9 ROLE WASTESTROLE
        WITHOUT AUTHENTICATION
        ENABLE 6
```

Figure 4-68 JDBC type 4 trusted context

1. Trusted context name
2. SYSTEM AUTHID - the system user ID provided by the application server. This ID corresponds to the user ID the data source JAAS alias user name.
3. ADDRESS - IP addresses or domain names
   We included the list of domain names that we observed during workload testing. After initial workload testing we used DB2 accounting traces to determine the IP addresses that had to be taken into account for trusted context creation. For convenience we loaded the DB2 accounting information into OMPE data warehouse tables and ran the query shown in Example 4-34 to determine these IP addresses that we had to consider for our JDBC type 4 trusted context definition.

   Example 4-34 Determine trusted context IP addresses and domain names

```
---------+---------+---------+---------+---------+--------
 SELECT COUNT(*) , REQ_LOCATION
   FROM  "DB2R3"."DB2PMFACCT_GENERAL"
   GROUP BY REQ_LOCATION
 ---------+---------+---------+---------+---------+--------
              REQ_LOCATION
 ---------+---------+---------+---------+---------+--------
      231416  ::9.12.4.142
      134789  ::9.12.4.138
      664221  ::9.12.6.9
```

```
     127981  ::9.12.6.70
 DSNE610I NUMBER OF ROWS DISPLAYED IS 4
```

For each IP address shown in Example 4-34 on page 177 we ran the UNIX System Services command shown in Example 4-35 to determine the domain names that we had to consider in our trusted context definition.

Example 4-35 Determine domain names by IP address

```
host 9.12.4.142
EZZ8321I d0z2.itso.ibm.com has addresses 9.12.4.142
host 9.12.4.138
EZZ8321I d0z1.itso.ibm.com has addresses 9.12.4.138
host 9.12.6.9
EZZ8321I wtsc64.itso.ibm.com has addresses 9.12.6.9
host 9.12.6.70
EZZ8321I wtsc63.itso.ibm.com has addresses 9.12.6.70
```

4. DEFAULT ROLE - database role to be used if no role assignment is performed by the trusted context
5. Optionally user IDs and roles for authorization ID (AUTHID) switching

Trusted context users controlled by RACF profile

DB2 supports an option that allows you to control the list of trusted context users through a RACF profile.

To use these options we performed the following implementation tasks:

- Create a RACF profile in the DSNR class as shown in Example 4-36. You might notice that we permitted read access to the DSNR profile to RACF group WASCTX into which we had connected the trusted context users WASCTX1 through WASCTX9. In case you want to remove or add trusted context users there is no need to alter the trusted context in DB2. All you need to do is to add or remove users from RACF group WASCTX.

Example 4-36 Create RACF DSNR trusted context profile

```
RDEFINE DSNR (DOZG.TRUSTEDCTX.DBZGWAS) UACC(NONE)
PERMIT DOZG.TRUSTEDCTX.DBZGWAS  CLASS(DSNR) ACCESS(READ) -
       ID( WASCTX)
SETROPTS RACLIST(DSNR) REFRESH
```

- Create a trusted context that refers to the RACF profile created in Example 4-36 in its WITH USE FOR clause as shown in Example 4-37.

Example 4-37 Create trusted context using RACF DSNR trusted context profile

```
CREATE TRUSTED CONTEXT CTXWASTESTT5
   BASED UPON CONNECTION USING SYSTEM AUTHID WASSRV
   DEFAULT ROLE WASTESTDEFAULTROLE
   WITHOUT ROLE AS OBJECT OWNER
   ENABLE
   NO DEFAULT SECURITY LABEL
   ATTRIBUTES (
   ENCRYPTION 'NONE',
   ADDRESS 'wtsc64.itso.ibm.com',
   ADDRESS 'd0z1.itso.ibm.com',
   ADDRESS 'wtsc63.itso.ibm.com',
   ADDRESS 'd0z2.itso.ibm.com'
```

```
)
WITH USE FOR
     EXTERNAL SECURITY PROFILE  "D0ZG.TRUSTEDCTX.D0ZGWAS"
     ROLE WASTESTROLE  1
WITHOUT AUTHENTICATION
```

1. The trusted context DDL shown in Example 4-37 on page 178 uses the same attributes as the trusted context DDL shown in Figure 4-68 on page 177 except for the RACF DSNR profile D0ZG.TRUSTEDCTX.D0ZGWAS profile which we created in Example 4-36 on page 178.

Testing the trusted connection

After the trusted connection has been established the trusted context allows the external entity to use a database connection under a different end-user ID without the database server having to authenticate that ID. This process, which is also known as authorization ID switching, calls RACF to check the authorization ID and, if provided by the trusted context, assigns a role that is to be used for authorization checking in DB2. A role disassociates DB2 privileges from the end-user. DB2 privileges granted to a role can only be acquired through a trusted context and thus unavailable outside of it.

During Data Web Service testing we collected the DB2 command output shown in Figure 4-69 which confirms trusted context usage with exactly the attributes we defined in Figure 4-68 on page 177.

```
DSNV473I  -D0Z2 ACTIVE THREADS FOUND FOR MEMBER: D0Z1
 NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
 SERVER   SW *     7 db2jcc_appli WASCTX1  DISTSERV 0084   427
 V485-TRUSTED CONTEXT=CTXWASTESTT4,
      SYSTEM AUTHID=WASTEST,
      ROLE=WASTESTROLE
  V437-WORKSTATION=WTSC64, USERID=wastest,
       APPLICATION NAME=dwsClientinformationDS
  V441-ACCOUNTING=JCC03640WTSC64            dwsClientinformation
        '
  V429 CALLING FUNCTION=DB2R3.GRACFGRP,
       PROC=        , ASID=0000, WLM_ENV=DSNWLMDB0Z_GENERAL
  V482-WLM-INFO=DDFONL:1:2:550
  V445-G90C0609.M72E.CA71F39F97F6=427 ACCESSING DATA FOR
   (  1)::9.12.6.9
```

Figure 4-69 DWS trusted context display thread output

Authorization failure during authid switch

The trusted context definition shown in Figure 4-68 on page 177 allows for an authorization switch to be performed to one of the users specified in the trusted context WITH USE FOR clause. We ran the Data Web Service application under user WASUSER which is not defined in the WITH USE FOR clause of the trusted context definition. WebSphere Application Server successfully authenticated WASUSER. The application server then tried to reuse the existing database connection and asked DB2 to perform an authorization ID switch on that connection to user WASUSER. Because WASUSER cannot be used by the trusted connection DB2 returned SQLCODE -20361 to the application.

The SQLCODE was confirmed by the IFCID 269 (audit trace class 10) record trace shown in Figure 4-70.

```
!------------------------------------------------------------------------
!CONNECTION TYPE: REUSED          STATUS: FAILED    SQLCODE:     -20361
!SECURITY LABEL : N/P
!
!TRUSTED CONTEXT NAME: CTXWASTESTT4
!SYSTEM AUTHID USED  : WASTEST
!REUSE AUTHID        : WASUSER
!------------------------------------------------------------------------
```

Figure 4-70 Trusted context IFCID 269 record trace with SQLCODE -20361

The application server log provided the corresponding runtime message shown in Figure 4-71 indicating the auth ID switch failure.

```
J2CA0056I:The Connection Manager received a fatal connection error
from the Resource Adapter for resource jdbc/Josef. The exception is:
 com.ibm.db2.jcc.am.DisconnectRecoverableException:
 Ÿjcc¨Ÿt4¨Ÿ2040¨Ÿ11215¨Ÿ3.64.82¨ An error occurred during a deferred
 connect reset and the connection has been terminated.  See chained
 exceptions for details. ERRORCODE=-4499,
 SQLSTATE=null:com.ibm.db2.jcc.am.SqlException:
 Ÿjcc¨Ÿt4¨Ÿ20130¨Ÿ12466¨Ÿ3.64.82¨ Trusted user switch failed.
ERRORCODE=-4214,SQLSTATE=null:com.ibm.db2.jcc.am.SqlSyntaxErrorException:
 DB2R3;CTXWASTESTT4
```

Figure 4-71 Failure of trusted user switch

4.3.17 Using DB2 profiles

DB2 for z/OS provides a profile table facility that you can use to:

- Optimize subsystem parameters for SQL statements by setting or disabling DB2 subsystem parameters (DSNZPARM) for particular SQL statements. The DSNZPARMs you can control include:
  - NPGTHRSH
  - OPTIOWGT
  - STARJOIN
  - SJTABLES
- Maintain copies of access paths by overriding the PLANMGMT and PLANMGMTSCOPE bind options and subsystem wide parameters settings for particular collections and packages.
- Create a test subsystem modelled on production environment CPU, memory and DB2 pool settings. Refer to "Simulate production like buffer pool sizes and catalog statistics" on page 153 for a discussion on this topic.
- Set thresholds for query acceleration
- Monitor database access threads and connections

In the workload scenario used in this book we focus on using profiles to monitor database access threads and connections. Other use cases for using profiles are not discussed. If you need further information about using these additional DB2 profile use cases refer to "Using profiles to monitor and optimize performance". *DB2 10 for z/OS, Managing Performance*, SC19-2978.

4.3.18 Using profiles to optimize and monitor threads and connections

DB2 10 for z/OS provides a profile table monitoring facility to support the filtering and threshold monitoring for system related activities, such as the number of connections, the number of threads, and the period of time that a thread can stay idle.

This enhancement allows you to enforce the thresholds (limits) that were previously available only at the system level using DSNZPARM, such as CONDBAT, MAXDBAT, and IDTHTOIN, at a more granular level. Setting these limits allows you to control connections using the following categories:

- IP Address (LOCATION)
- Product Identifier (PRDID)
- Role and Authorization Identifier (ROLE, AUTHID)
- Collection ID and Package Name (COLLID, PKGNAME)
- DB2 client information (CLIENT_APPLNAME, CLIENT_USERID, CLIENT_WORKSTNNAME)

This enhancement also provides the option to define the type of action to take after these thresholds are reached. You can display a warning message or an exception message when the connection, thread, and idle thread timeout thresholds are exceeded. If you choose to display a warning message, a DSNT771I or DSNT772I message is issued, depending on DIAGLEVEL and processing continues. In the case of exception processing, a message is displayed to the console and the action taken (that is queuing, suspension, or rejection).

DB2 profile tables

Profile monitoring requires the following tables to be created:

- SYSIBM.DSN_PROFILE_TABLE
- SYSIBM.DSN_PROFILE_HISTORY
- SYSIBM.DSN_PROFILE_ATTRIBUTES
- SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY

These tables are created by installation job DSNTIJSG. The profile history and attributes history tables have the same columns as their corresponding profile and profile attributes tables, except for the STATUS column which is added to keep track of profile status information and except for the REMARKS column that does not exist in the profile history and attributes history tables. The STATUS column indicates whether a profile was accepted or why it was rejected during START PROFILE command execution.

Storing a profile in the DSN_PROFILE_TABLE

DSN_PROFILE_TABLE stores one row per monitoring or execution profile. Rows are inserted by authorized users using SQL. A row can apply either to statement monitoring or to system level activity monitoring, but not both. Monitoring can be performed based on options such as IP address, product ID, authid, role, collection ID, package name, and DB2 client information.

To monitor connections or threads, you need to insert a row into DSN_PROFILE_TABLE with the appropriate criteria. Valid filtering criteria for monitoring system activities can be organized into categories as shown in Table 4-5.

Table 4-5 Profile table filter criteria

Filter category	Columns to specify
IP address or domain name	Specify only the LOCATION column
Client product identifier	Specify only the PRDID column
Role and / or authorization ID	Specify one or all of the following columns ► ROLE ► AUTHID
Collection ID and / or package name	Specify one or all of the following columns ► COLLID ► PKGNAME
DB2 client information	Specify on of the following columns ► CLIENT_APPNAME ► CLIENT_USERID ► CLIENT_WORKSTNNAME

For connection monitoring you can only filter on IP address or domain name for which you provide the filter value by populating the profile table LOCATION column.

You create a profile by inserting a row into SYSIBM.DSN_PROFILE_TABLE providing the column values that are required to implement one of the filter criteria referred to in Table 4-5. For illustration, see the list of profile table columns in Figure 4-72.

```
AUTHID                  1 VARCHAR     128
PLANNAME                2 VARCHAR      24
COLLID                  3 VARCHAR     128
PKGNAME                 4 VARCHAR     128
LOCATION                5 VARCHAR     254
PROFILEID               6 INTEGER       4
PROFILE_TIMESTAMP       7 TIMESTMP     10
PROFILE_ENABLED         8 CHAR          1
GROUP_MEMBER            9 VARCHAR      24
REMARKS                10 VARCHAR     762
ROLE                   11 VARCHAR     128
PRDID                  12 CHAR          8
CLIENT_APPLNAME        13 VARCHAR     255
CLIENT_USERID          14 VARCHAR     255
CLIENT_WRKSTNNAME      15 VARCHAR     255
```

Figure 4-72 DSN_PROFILE_TABLE

Besides the PROFILEID, which also is the profile table primary key, there are further columns that you use to provide information about the monitoring filter criteria identifying the thread, connection, or SQL statement you want monitoring to be performed for.

For MAXDBAT and IDTHTOIN monitoring you can enter the filter criteria using any of the combinations shown in Table 4-5.

For CONDBAT monitoring you can only specify an IP address or a domain name in the LOCATION column. Other combinations of criteria are not accepted for CONDBAT monitoring function.

Storing profile attributes in the DSN_PROFILE_ATTRIBUTES table

After you have created your profile by inserting a profile table row into the DSN_PROFILE table you need to provide profile attributes to provide monitoring thresholds and actions that are to be performed in case the threshold is exceeded.

To provide these information you insert a row into the profile attributes table (SYSIBM.DSN_PROFILE_ATTRIBUTES) to store the required threshold and action related information. For illustration purpose we provide a list of the profile attributes table columns in Figure 4-73. The table contains a PROFILEID column which corresponds to a profile table row with the same PROFILEID column value.

```
PROFILEID                1 INTEGER      4
KEYWORDS                 2 VARCHAR    128
ATTRIBUTE1               3 VARCHAR   1024
ATTRIBUTE2               4 INTEGER      4
ATTRIBUTE3               5 FLOAT        8
ATTRIBUTE_TIMESTAM       6 TIMESTMP    10
REMARKS                  7 VARCHAR    762
```

Figure 4-73 DSN_PROFILE_ATTRIBUTES table

For DBAT or remote connection monitoring you can enter one of the attribute values shown in Table 4-6 to provide monitoring threshold and actions depending on the kind of thread, connection or IDLE thread monitoring you want to perform. Profile attribute column ATTRIBUTE3 is not used for thread and connection monitoring.

Table 4-6 Profile attributes

Keywords	Attribute1	Attribute2
MONITOR IDLE THREADS (IDTHTOIN)	► WARNING ► WARNING_DIAGLEVEL1 ► WARNING_DIAGLEVEL2 ► EXCEPTION ► EXCEPTION_DIAGLEVEL1 ► EXCEPTION DIAGLEVEL2	Maximum number of seconds that active server threads are allowed to remain idle A value of 0 disables IDTHTOIN for this profile
MONITOR THREADS (MAXDBAT)	► WARNING ► WARNING_DIAGLEVEL1 ► WARNING_DIAGLEVEL2 ► EXCEPTION ► EXCEPTION_DIAGLEVEL1 ► EXCEPTION DIAGLEVEL2	Insert a value to indicate the threshold for the maximum allowed number of server threads that meet the profile criteria. The value that you specify must be less than or equal to the value of the MAXDBAT subsystem parameter.
MONITOR CONNECTIONS (CONDBAT)	► WARNING ► WARNING_DIAGLEVEL1 ► WARNING_DIAGLEVEL2 ► EXCEPTION ► EXCEPTION_DIAGLEVEL1 ► EXCEPTION DIAGLEVEL2	Insert a value to indicate the threshold for the maximum allowed number of remote connections that meet the profile criteria. The value that you specify must be less than or equal to the value of the CONDBAT subsystem parameter

Starting profiles

You start DB2 profiles by issuing the DB2 START PROFILE command:

```
-START PROFILE
```

Triggered by the START PROFILE command DB2 starts profile rows with the value Y in the PROFILE_ENABLED profile table column (SYSIBM.DSN_PROFILE_TABLE column PROFILE_ENABLED = Y).

In data sharing the START and STOP PROFILE commands have member scope and affect only the data sharing member they have been issued for. You therefore need to issue these commands for each data sharing member you want to have profile monitoring started or stopped.

In our environment we use the administrative task scheduler to issue the START PROFILE command at DB2 startup time. In Appendix A, "DB2 administrative task scheduler" on page 483. we describe the administrative task scheduler (ADMT) setup to trigger batch jobs, DB2 commands, and for autonomic statistics monitoring.

Stopping profiles

You stop profiles by issuing the STOP PROFILE command:

```
-STOP PROFILE
```

Monitoring for individual profiles can be stopped by updating the PROFILE_ENABLED column in the SYSIBM.DSN_PROFILE_TABLE to N and issuing a START PROFILE command again.

Profile history tables

During START PROFILE command execution DB2 considers DSN_PROBILE_TABLE rows with the ENABLE column set to Y and their corresponding DSN_PROFILE_ATTRIBUTES rows for profile activation. Before DB2 starts an individual profile it uses the profile information found in the profile and the profile attributes tables to perform profile validation and externalizes the profile and profile attributes information together with profile status information into the following corresponding profile history and profile attributes history tables.

- SYSIBM.DSN_PROFILE_HISTORY
- SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY

DSN_PROFILE_HISTORY table

During profile activation DB2 validates each profile to be started and documents its activation status by inserting one row into table SYSIBM.DSN_PROFILE_HISTORY. As shown in Figure 4-74 on page 185 the DSN_PROFILE_HISTORY table consists of column information of the DSN_PROFILE_TABLE (except for the REMARKS column) plus a STATUS column to provide information about the profile activation status.

```
AUTHID                   1 VARCHAR     128
PLANNAME                 2 VARCHAR      24
COLLID                   3 VARCHAR     128
PKGNAME                  4 VARCHAR     128
LOCATION                 5 VARCHAR     254
PROFILEID                6 INTEGER       4
PROFILE_TIMESTAMP        7 TIMESTMP     10
PROFILE_ENABLED          8 CHAR          1
GROUP_MEMBER             9 VARCHAR      24
STATUS                  10 VARCHAR     254
ROLE                    11 VARCHAR     128
PRDID                   12 CHAR          8
CLIENT_APPLNAME         13 VARCHAR     255
CLIENT_USERID           14 VARCHAR     255
CLIENT_WRKSTNNAME       15 VARCHAR     255
```

Figure 4-74 DSN_PROFILE_HISTORY table

The STATUS column provides one of the following information:

- REJECTED - DUPLICATED SCOPE SPECIFIED
- REJECTED - INVALID LOCATION SPECIFIED
- REJECTED - INVALID SCOPE SPECIFIED
- REJECTED - NO VALID RECORD FOUND IN ATTRIBUTE TABLE
- REJECTED - INVALID SCOPE SPECIFIED. SYSTEM LEVEL MONITORIN SCOPE CAN BE SPECIFIED ONLY ON NFM
- REJECTED - INVALID SCOPE SPECIFIED. FOR SYSTEM LEVEL MONITORING, ONLY IP ADDR, PRDID, ROLE AND/OR AUTHID,COLLECTION ID AND/OR PACKAGE NAME CAN BE SPECIFIED
- ACCEPTED - DOMAIN NAME IS RESOLVED INTO IP ADDRESS
- ACCEPTED

DSN_PROFILE_ATTRIBUTES_HISTORY table

Profile activation that we describe in “DSN_PROFILE_HISTORY table” on page 184 furthermore triggers profile attribute validation.

During START PROFILE execution DB2 externalizes the attribute status of each profile attribute involved by inserting corresponding rows into the profile attributes history table (SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY).

As shown in Figure 4-75 the DSN_PROFILE_ATTRIBUTES_HISTORY table consists of column information of the DSN_PROFILE_TABLE_ATTRIBUTES table (except for the REMARKS column) plus a STATUS column to provide information about the profile attribute activation status.

```
PROFILEID               1 INTEGER      4
KEYWORDS                2 VARCHAR    128
ATTRIBUTE1              3 VARCHAR   1024
ATTRIBUTE2              4 INTEGER      4
ATTRIBUTE3              5 FLOAT        8
ATTRIBUTE_TIMESTAM      6 TIMESTMP    10
STATUS                  7 VARCHAR    254
```

Figure 4-75 DSN_PROFILE_ATTRIBUTES_HISTORY table

The STATUS column indicates whether the profile was accepted, and when a profile was rejected contains information about the reason for the rejection.

Verify profile activation status

Each time DB2 attempts to start a profile a row is inserted into the DSN_PROFILE_HISTORY table. After we issued the START PROFILE command we ran the query shown in Example 4-38 to verify the status of the profile that we created for active thread monitoring.

Example 4-38 Verify DSN_PROFILE_TABLE status

```
SELECT PROFILEID,PROFILE_TIMESTAMP,STATUS
   FROM  "SYSIBM"."DSN_PROFILE_HISTORY"
   WHERE PROFILEID = 1
   ORDER BY PROFILE_TIMESTAMP DESC
   FETCH FIRST ROW ONLY
---------+---------+---------+---------+---------+---------+-
  PROFILEID  PROFILE_TIMESTAMP          STATUS
---------+---------+---------+---------+---------+---------+-
          1  2012-10-25-14.18.04.615794  ACCEPTED BY D0Z2
```

To verify the profile activation status of the attributes that we defined for the profile we ran the query shown in Example 4-39.

Example 4-39 Verify DSN_PROFILE_ATTRIBUTES status

```
---------+---------+---------+---------+---------+---------+--------
SELECT
    SUBSTR(KEYWORDS,1,14) AS KEYWORDS
   ,SUBSTR(ATTRIBUTE1,1,20) AS ATTRIBUTE1
   ,ATTRIBUTE2
   ,STATUS
FROM  "SYSIBM"."DSN_PROFILE_ATTRIBUTES_HISTORY"
WHERE PROFILEID = 1
 ORDER BY ATTRIBUTE_TIMESTAMP DESC
 FETCH FIRST ROW ONLY
---------+---------+---------+---------+---------+---------+--------
KEYWORDS        ATTRIBUTE1            ATTRIBUTE2  STATUS
---------+---------+---------+---------+---------+---------+--------
MONITOR THREAD  WARNING_DIAGLEVEL2             7  ACCEPTED BY D0Z2
```

The status returned by the queries shown in Example 4-38 on page 186 and in Example 4-39 on page 186 confirms that the profile with PROFILEID = 1 was successfully activated on member D0Z2.

4.3.19 Configure thread monitoring for the DayTrader-EE6 application

In our application scenario we configure active thread monitoring for the DayTrader-EE6 application to determine the number of active threads the application consumes in DB2.

Creating the DayTrader-EE6 thread monitoring profile

We ran the SQL insert statements shown in Example 4-40 to populate table DSN_PROBILE_TABLE with the information required for thread monitoring. DayTrader-EE6 provides the clientApplicationInformaton data source custom property value of TraderClientApplication when connecting to DB2. The PROFILE_ENABLED column is set to Y to have the profile activated when a START PROFILE command is issued.

Example 4-40 DayTrader-EE6 DSN_PROFILE_TABLE row

```
INSERT INTO SYSIBM.DSN_PROFILE_TABLE  ( "AUTHID" , "PLANNAME" ,
  "COLLID" ,
  "PKGNAME" , "LOCATION" , "PROFILEID" , "PROFILE_TIMESTAMP" ,
  "PROFILE_ENABLED" , "GROUP_MEMBER" , "REMARKS" , "ROLE" , "PRDID" ,
  "CLIENT_APPLNAME" , "CLIENT_USERID" , "CLIENT_WRKSTNNAME" )
 VALUES (
  NULL                       -- AUTHID
 ,NULL                       -- PLANNAME
 ,NULL                       -- COLLID
 ,NULL                       -- PKGNAME
 ,NULL                       -- LOCATION
 , 1                         -- PROFILEID
 ,CURRENT TIMESTAMP          -- PROFILE_TIMESTAMP
 ,'Y'                        -- PROFILE_ENABLED
 ,''                         -- GROUP_MEMBER
 ,'DayTrader profile'        -- REMARKS
 ,NULL                       -- ROLE
 ,NULL                       -- PRDID
 ,'TraderClientApplication'  -- CLIENT_APPLNAME
 ,NULL                       -- CLIENT_USERID
 ,NULL                       -- CLIENT_WRKSTNNAM
 )
 ;
```

We then ran the SQL statement shown in Example 4-41 to insert a corresponding row into DSN_PROFILE_ATTRIBUTES table.

Example 4-41 DayTrader-EE6 DSN_PROFILE_ATTRIBUTES row

```
INSERT INTO SYSIBM.DSN_PROFILE_ATTRIBUTES
  ( "PROFILEID" , "KEYWORDS" , "ATTRIBUTE1" , "ATTRIBUTE2" ,
  "ATTRIBUTE3" , "ATTRIBUTE_TIMESTAMP" , "REMARKS" )
 VALUES (
  1                     -- PROFILEID
 ,'MONITOR THREADS'     -- monitors number of concurrent active threads
 ,'WARNING_DIAGLEVEL2'  -- DB2 issues DSNT772I when threshold exceeded
 , 7                    -- number of active threads allowed
```

```
, NULL                  -- ATTRIBUTE3
, CURRENT TIMESTAMP     -- ATTRIBUTE_TIMESTAMP
,'DayTrader'            -- REMARKS
);
```

The attributes shown in Example 4-41 on page 187 define active thread monitoring for PROFILEID 1, allowing for a maximum of seven active threads, causing DB2 to issue warning message DSNT772I in case this number of active threads is exceeded. Processing continues with no thread queuing or suspension.

Activating thread monitoring

We configured the administrative task scheduler (ADMT) to issue START PROFILE and DISPLAY PROFILE commands within DB2 subsystem startup processing. In Appendix A, “DB2 administrative task scheduler” on page 483. we describe the administrative task scheduler (ADMT) setup to trigger batch jobs, DB2 commands, and for autonomic statistics monitoring.

The output of the ADMT initiated DB2 command processing is shown in Figure 4-76.

```
 -START PROFILE
DSNT741I  -DOZ1 DSNT1SDV START PROFILE IS COMPLETED.
DSN9022I  -DOZ1 DSNT1STR 'START PROFILE' NORMAL COMPLETION
DSN
 -DIS   PROFILE
DSNT753I  -DOZ2 DSNT1DSP DISPLAY PROFILE REPORT FOLLOWS:
              STATUS = ON
              TIMESTAMP = 2012-10-25-14.18.04.615794
              PUSHOUTS = 0 OUT OF 10000
              DISPLAY PROFILE REPORT COMPLETE.
DSN9022I  -DOZ2 DSNT1DSP 'DISPLAY PROFILE' NORMAL COMPLETION
```

Figure 4-76   START PROFILE command

Verifying thread monitoring status

For each monitoring profile that is to be started (SYSIBM.DSN_PROFILE_TABLE, column PROFILE_ENABLED = Y) DB2 externalizes profile status information to the corresponding profile history tables. We ran the query shown in Example 4-42 to verify the status of the monitoring profile referred to in Example 4-40 on page 187 and Example 4-41 on page 187.

Example 4-42   verify DSN_PROFILE_TABLE status

```
SELECT PROFILEID,PROFILE_TIMESTAMP,STATUS,CLIENT_APPLNAME
   FROM  "SYSIBM"."DSN_PROFILE_HISTORY"
   WHERE PROFILEID = 1
   ORDER BY PROFILE_TIMESTAMP DESC
   FETCH FIRST ROW ONLY
---------+---------+---------+---------+---------+---------+-
  PROFILEID  PROFILE_TIMESTAMP           STATUS
---------+---------+---------+---------+---------+---------+-
          1  2012-10-25-14.18.04.615794  ACCEPTED BY DOZ2
```

We then ran the query shown in Example 4-43 to verify the status of the monitoring attributes.

Example 4-43 Verify DSN_PROFILE_ATTRIBUTES status

```
---------+---------+---------+---------+---------+---------+--------
SELECT
    SUBSTR(KEYWORDS,1,14) AS KEYWORDS
   ,SUBSTR(ATTRIBUTE1,1,20) AS ATTRIBUTE1
   ,ATTRIBUTE2
   ,STATUS
FROM  "SYSIBM"."DSN_PROFILE_ATTRIBUTES_HISTORY"
WHERE PROFILEID = 1
 ORDER BY ATTRIBUTE_TIMESTAMP DESC
 FETCH FIRST ROW ONLY
---------+---------+---------+---------+---------+---------+--------
KEYWORDS        ATTRIBUTE1             ATTRIBUTE2  STATUS
---------+---------+---------+---------+---------+---------+--------
MONITOR THREAD  WARNING_DIAGLEVEL2              7  ACCEPTED BY D0Z2
```

The status returned by the queries shown in Example 4-42 on page 188 and Example 4-43 confirms that our thread monitoring profile was successfully activated.

DayTrader-EE6 active thread monitoring messages

When we run the DayTrader-EE6 workload we observed the DB2 messages shown in Figure 4-77 issued by the DB2 master address space.

```
DSNT772I  -D0Z1 DSNLQDIS A MONITOR PROFILE WARNING
CONDITION OCCURRED
1 TIME(S)
IN PROFILE ID=1
WITH PROFILE FILTERING SCOPE=CLIENT_APPLNAME
WITH REASON=00E30505
```

Figure 4-77 DSNT772I active thread monitoring warning message

4.3.20 Using profiles to keep track of DRDA client levels

In this scenario we show how to monitor the DB2 clients that use certain levels of the DB2 client software. To perform this kind of monitoring we use profiles to monitor client threads that connect to DB2 for z/OS using a certain client level.

This monitoring function can assist you in identifying outdated levels of DB2 client software used in your environment. After you have identified the clients and remote locations you can use profiles to issue warnings in case such back level clients are being used and finally disable the use of such client levels after a planned grace period has expired.

Use of DB2 Connect

Keeping track of DRDA client levels becomes especially important when your clients go through a DB2 Connect gateway to connect to DB2 for z/OS, because upgrading to a new version of DB2 for z/OS might force you migrate your DB2 Connect gateways to the level that is supported by the new version of DB2. Using a new level of DB2 Connect in turn might trigger DB2 client migrations, as DB2 Connect itself only supports certain back-levels of clients.

With DB2 clients directly connecting to DB2 for z/OS servers this back-level consideration no longer is an issue, because during the DRDA hand shake DB2 for z/OS and the DB2 client agree on the DRDA level to be used which happens to be the lowest DRDA level either of the client or the server. Supporting the lowest DRDA level for application processing alleviates the requirement of having to upgrade your DB2 clients to the most recent level. However, you are reminded that upgrading your clients is recommended, especially if you want to take advantage of new functions provided by the DB2 for z/OS server.

DB2 for z/OS and DB2 Connect

Up to DB2 9 for z/OS the use of DB2 Connect was required in some situations due to the capacity limit of that version of DB2. For instance, due to virtual storage constraints a DB2 9 for z/OS server was only able to support a limited number of database access threads (DBATs).

In DB2 10 for z/OS this and many other constraints are relieved which enables DB2 for z/OS to support a generous number of database access threads that is sufficient enough to replace existing DB2 Connect functionality by DB2 clients that directly connect to the DB2 for z/OS server. An illustration of that architecture is shown in Figure 4-78.

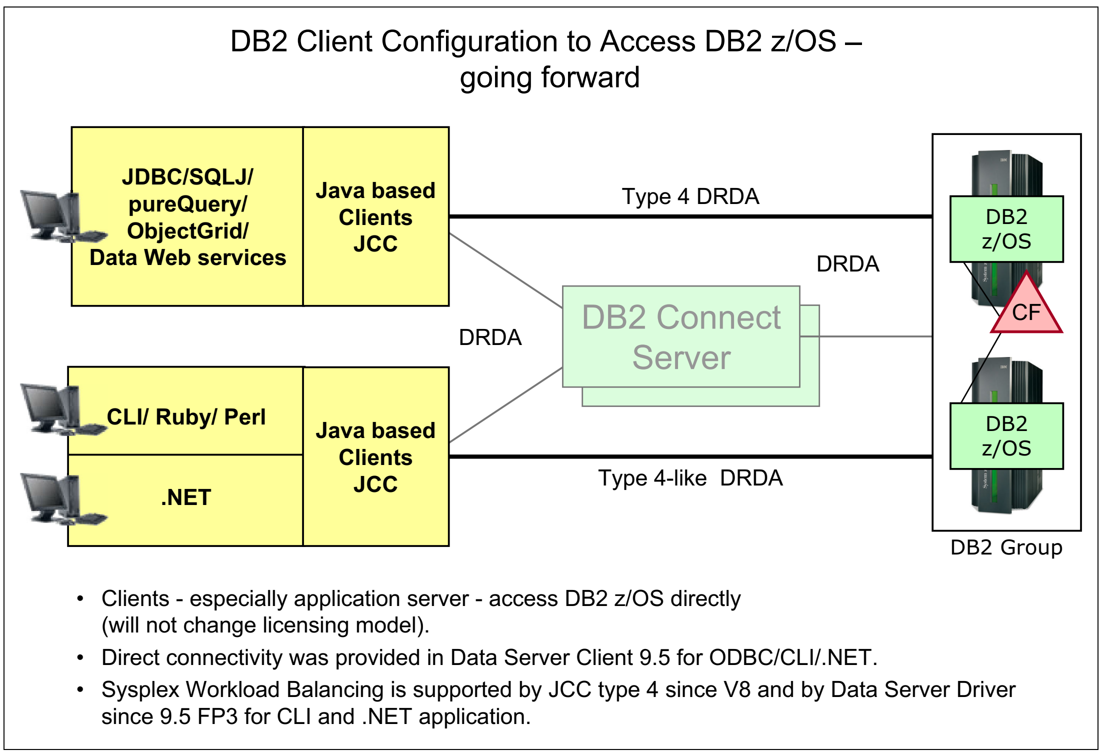


Figure 4-78 DB2 Client configuration to directly access DB2 for z/OS

Figure 4-78 shows Java clients directly connecting to DB2 for z/OS using JDBC type 4 connections while the DB2 Connect infrastructure still is in place. This approach allows for a staged migration of DB2 clients in which DB2 client access is redirected from using DB2 Connect to DB2 direct access by updating the DB2 client configuration as illustrated in Figure 4-79 on page 191.

Changing the DB2 client configuration in that situation enables you to make use of new DB2 10 for z/OS configuration options. For instance you can perform online changes to dynamically activate DB2 location aliases allowing you to direct workloads to the data sharing group, to a subset of data sharing members, or to a single data sharing member.

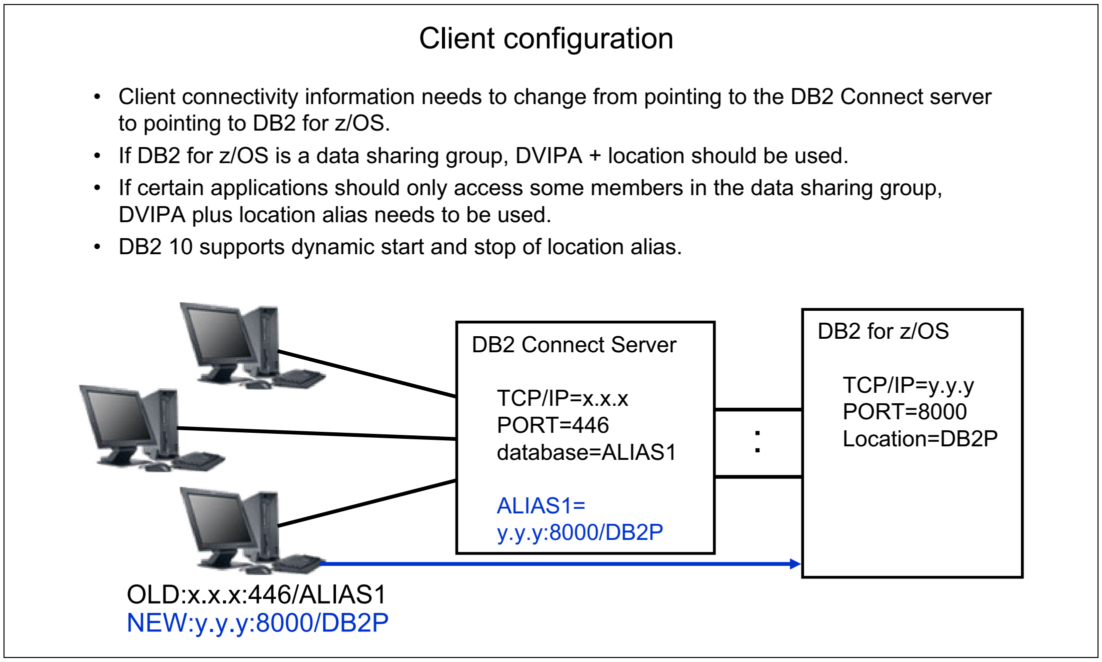


Figure 4-79 DB2 client configuration for DB2 direct access

Controlling database access threads

DB2 10 for z/OS is now able to serve huge numbers of DB2 clients directly connected to the DB2 server; new monitoring functionality has been introduced to estimate resource usage of DRDA applications and to avoid situations in which DRDA clients monopolize DB2 server resources.

Identify DRDA client levels

The product identifier is the access relational database product specific ID (PRDID) representing the product ID of the DB2 client (also referred to by the application requestor).

The DB2 client product ID has the format PPPVVRRM where

- PPP is the product identifier. Possible values are
  - DSN - DB2 for z/OS
  - ARI - DB2 for VSE and VM
  - SQL - DB2 for Linux, UNIX, and Windows
  - JCC - IBM Data Server Driver for JDBC and SQLJ
  - QSQ - DB2 for IBM eServer iSeries®
- VV is the version number
- RR is the release number
- M is the modification level

Identifying DRDA PRDIDs used in your system

You can use one of the following options to keep track of the DRDA PRDIDs used by your remote DB2 clients.

- DB2 command **DISPLAY LOCATION**

  DB2 command **DISPLAY LOCATION** returns the PRDID of DRDA clients connected to your system. The command output provided in Figure 4-80 shows to remote locations currently connected to member D0Z1. One of them of PRDID JCC04130 (JDBC version 4 release 13 modification level 0) and the other of SQL10010 (DB2 LUW version 10 release 01 modification level 0).

```
-dis location
 DSNL200I  -D0Z1 DISPLAY LOCATION REPORT FOLLOWS-
 LOCATION                                        PRDID    T ATT CONNS
 ::9.12.6.9                                      JCC04130 S         1
 ::9.145.139.205                                 SQL10010 S         1
 DISPLAY LOCATION REPORT COMPLETE
```

Figure 4-80   DISPLAY LOCATION with PRDID information

- Field QLSTPRID of the statistics trace record (IFCID 0001)
- Field QLACPRID of the accounting trace record (IFCID 0003)
  You can use DB2 accounting reports to determine the product IDs used by your distributed clients. For convenience we run a query against the PDB that we discuss in 4.4, "Tivoli OMEGAMON XE for DB2 Performance Expert for z/OS" on page 201 to obtain the different distributed product IDs used in our environment. The query result is shown in Figure 4-81.

```
SELECT
   COUNT(*) AS NO
 , SUBSTR(REQ_LOCATION        ,01,15) AS REQ_LOCATION
 , SUBSTR(CLIENT_TRANSACTION  ,01,15) AS CLIENT_TRANSACTION
 , REMOTE_PRODUCT_ID
FROM DB2PMSACCT_DDF
 GROUP BY
   REQ_LOCATION
 , CLIENT_TRANSACTION
 , REMOTE_PRODUCT_ID
---------+---------+---------+---------+---------+---------+---------+-
         NO  REQ_LOCATION     CLIENT_TRANSACTION  REMOTE_PRODUCT_ID
---------+---------+---------+---------+---------+---------+---------+-
         11  ::9.12.4.142     TraderClientApp     JCC03640
          2  ::9.12.6.9       db2jcc_applicat     JCC03630
          2  ::9.12.6.9       db2jcc_applicat     JCC03640
         15  ::9.12.6.9       TraderClientApp     JCC03640
         11  ::9.12.6.9       TraderClientApp     JCC03630
          1  ::9.30.28.118    db2jcc_applicat     JCC04130
DSNE610I NUMBER OF ROWS DISPLAYED IS 6
```

Figure 4-81   Use PDB to query PRDIDs

Activate PRDID based thread monitoring

In our scenario we illustrate how to use profiles to monitor DB2 clients using a certain JDBC driver level. The profile tables changes we performed for this kind of monitoring are shown in Example 4-44.

Example 4-44 Profile table changes for PRDID monitoring

```
-- -----------------------------------------------------------------
-- DSN_PROFILE_TABLE
-- -----------------------------------------------------------------
INSERT INTO SYSIBM.DSN_PROFILE_TABLE  ( "AUTHID" , "PLANNAME" ,
  "COLLID" ,
  "PKGNAME" , "LOCATION" , "PROFILEID" , "PROFILE_TIMESTAMP" ,
  "PROFILE_ENABLED" , "GROUP_MEMBER" , "REMARKS" , "ROLE" , "PRDID" ,
  "CLIENT_APPLNAME" , "CLIENT_USERID" , "CLIENT_WRKSTNNAME" )
 VALUES (
  NULL                       -- AUTHID
 ,NULL                       -- PLANNAME
 ,NULL                       -- COLLID
 ,NULL                       -- PKGNAME
 ,NULL                       -- LOCATION
 , 4                         -- UNIQUE PROFILEID
 ,CURRENT TIMESTAMP          -- PROFILE_TIMESTAMP
 ,'Y'                        -- PROFILE_ENABLED
 ,''                         -- GROUP_MEMBER
 ,'THREAD MONITORING PRDID' -- REMARKS
 ,NULL                       -- ROLE
 ,'SQL10010'                 -- PRDID
 ,NULL                        -- CLIENT_APPLNAME
 ,NULL                       -- CLIENT_USERID
 ,NULL                       -- CLIENT_WRKSTNNAM
 )
 ;
-- -----------------------------------------------------------------
-- DSN_PROFILE_ATTRIBUTES
-- -----------------------------------------------------------------
INSERT INTO SYSIBM.DSN_PROFILE_ATTRIBUTES
  ( "PROFILEID" , "KEYWORDS" , "ATTRIBUTE1" , "ATTRIBUTE2" ,
  "ATTRIBUTE3" , "ATTRIBUTE_TIMESTAMP" , "REMARKS" )
 VALUES (
  4                     -- PROFILEID
 ,'MONITOR THREADS'     -- monitors number of concurrent active threads
 ,'WARNING_DIAGLEVEL2'  -- DB2 issues DSNT772I when threshold exceeded
 , 1                    -- NUMBER OF ACTIVE THREADS ALLOWED
 , NULL                 -- ATTRIBUTE3
 , CURRENT TIMESTAMP    -- ATTRIBUTE_TIMESTAMP
 ,'PRDID'               -- REMARKS
 );
```

In Example 4-44 we configure DB2 profile monitoring to issue warning message DSNT772I when the number of threads using the DB2 client level indicated by product ID SQL10010 (max 1 in our example used for illustration) is exceeded. The application itself continues processing as we configured the profile attribute to issue a warning in case the threshold is exceeded. If we wanted the application to receive a negative SQLCODE we would have set profile attribute ATTRIBUTE1 to the value of EXCEPTION.

We then created multiple DB2 connections using DB2 clients of product ID SQL10010 to cause DB2 to issue message DSNT772I. The message that we received is shown in Figure 4-82.

DB2 reason code 00E30505 indicates that a warning occurred because the number of concurrent active threads exceeded the warning setting for the MONITOR THREADS keyword in a monitor profile for one of the PRDID filtering scope.

```
DSNT772I  -D0Z1 DSNLTACC A MONITOR PROFILE WARNING 421
CONDITION OCCURRED
1 TIME(S)
IN PROFILE ID=4
WITH PROFILE FILTERING SCOPE=PRDID
WITH REASON=00E30505
```

Figure 4-82 DSNT772I PRDID monitoring

4.3.21 Using profiles to disable idle thread timeout at application level

We explain the subsystem wide setting for idle thread timeout "IDLE THREAD TIMEOUT field (IDTHDOIN)" on page 139. IDTHTOIN controls idle thread timeout interval at subsystem level which affects all database access threads (DBAT) served by the subsystem or data sharing member. In case an application misbehaves (for instance, held locks due to missing commit processing or declared temporary tables not explicitly dropped at the end of the application) IDLTHDOIN might need to be set to 0 to keep your production up and running. Setting the parameter to 0 disables the idle thread timeout processing for the entire subsystem or data sharing member affecting not only the misbehaving application.

You can use profiles to control IDTHTOIN processing at application level which gives you the option to disable idle thread timeout processing just for the application you have to disable timeout processing for. The subsystem wide setting for IDTHTOIN still applies to all the DBATs not qualifying for idle thread timeout profile processing.

For instance, to disable idle thread timeout processing for the client application name NonCommittingProgram you would have run the SQL insert statements shown in Example 4-45. and subsequently issue the command shown in "Stopping profiles" on page 184 to activate the profile. In this example the misbehaving application set its clientApplicationInformation to the value of NonComittingProgram.

Example 4-45 Profile sample disable IDTHTOIN

```
-- SYSIBM.DSN_PROBILE_TABLE
INSERT INTO SYSIBM.DSN_PROFILE_TABLE  ( "AUTHID" , "PLANNAME" ,
  "COLLID" ,
  "PKGNAME" , "LOCATION" , "PROFILEID" , "PROFILE_TIMESTAMP" ,
  "PROFILE_ENABLED" , "GROUP_MEMBER" , "REMARKS" , "ROLE" , "PRDID" ,
  "CLIENT_APPLNAME" , "CLIENT_USERID" , "CLIENT_WRKSTNNAME" )
 VALUES (
  NULL                      -- AUTHID
 ,NULL                      -- PLANNAME
 ,NULL                      -- COLLID
 ,NULL                      -- PKGNAME
 ,NULL                      -- LOCATION
 , 1                        -- unique PROFILEID
 ,CURRENT TIMESTAMP         -- PROFILE_TIMESTAMP
 ,'Y'                       -- PROFILE_ENABLED
```

```
 ,''                         -- GROUP_MEMBER
 ,'Disable IDTHTOIN timout' -- REMARKS
 ,NULL                       -- ROLE
 ,NULL                       -- PRDID
 ,'NonCommittingProgram'  -- CLIENT_APPLNAME
 ,NULL                       -- CLIENT_USERID
 ,NULL                       -- CLIENT_WRKSTNNAM
 )
 ;
-- ------------------------------------------------------------------------
-- SYSIBM.DSN_PROBILE_ATTRIBUTES table
-- ------------------------------------------------------------------------
INSERT INTO SYSIBM.DSN_PROFILE_ATTRIBUTES
  ( "PROFILEID" , "KEYWORDS" , "ATTRIBUTE1" , "ATTRIBUTE2" ,
  "ATTRIBUTE3" , "ATTRIBUTE_TIMESTAMP" , "REMARKS" )
 VALUES (
  1                    -- unique PROFILEID
 ,'MONITOR IDLE THREADS' -- IDHTDOIN monitoring
 ,'WARNING_DIAGLEVEL2'  -- DB2 issues DSNT772I when threshold exceeded
 , 0                   -- IDTHTOIN = 0
 , NULL                -- ATTRIBUTE3
 , CURRENT TIMESTAMP    -- ATTRIBUTE_TIMESTAMP
 ,'disable IDTHTOIN'    -- REMARKS
 );
```

4.3.22 Using profiles for remote connection monitoring

You can use profiles to monitor the number of concurrent inbound DDF connections at the requesting location level. This monitoring function helps you to keep track of the number of remote connections used by a particular remote location. To use this monitoring function you need to provide the requestor's IP address or domain name as filter criteria in the profile table LOCATION column. To activate connection monitoring for IP address 9.146.231.122 we created a profile using the SQL statements shown in Example 4-46 and issued a START PROFILE command.

Example 4-46 Sample of profile for remote connection monitoring

```
-- ----------------------------------------------------------------
-- DSN_PROFILE_TABLE
-- ----------------------------------------------------------------
INSERT INTO SYSIBM.DSN_PROFILE_TABLE  ( "AUTHID" , "PLANNAME" ,
  "COLLID" ,
  "PKGNAME" , "LOCATION" , "PROFILEID" , "PROFILE_TIMESTAMP" ,
  "PROFILE_ENABLED" , "GROUP_MEMBER" , "REMARKS" , "ROLE" , "PRDID" ,
  "CLIENT_APPLNAME" , "CLIENT_USERID" , "CLIENT_WRKSTNNAME" )
 VALUES (
  NULL                       -- AUTHID
 ,NULL                       -- PLANNAME
 ,NULL                       -- COLLID
 ,NULL                       -- PKGNAME
 ,'9.146.231.122'            -- LOCATION
 , 5                         -- UNIQUE PROFILEID
 ,CURRENT TIMESTAMP          -- PROFILE_TIMESTAMP
 ,'Y'                        -- PROFILE_ENABLED
 ,''                         -- GROUP_MEMBER
```

```
 ,'Connection Monitoring  ' -- REMARKS
 ,NULL                       -- ROLE
 ,NULL                       -- PRDID
 ,NULL                        -- CLIENT_APPLNAME
 ,NULL                       -- CLIENT_USERID
 ,NULL                       -- CLIENT_WRKSTNNAM
 )
 ;
-- -------------------------------------------------------------
-- DSN_PROFILE_ATTRIBUTES
-- -------------------------------------------------------------
INSERT INTO SYSIBM.DSN_PROFILE_ATTRIBUTES
  ( "PROFILEID" , "KEYWORDS" , "ATTRIBUTE1" , "ATTRIBUTE2" ,
  "ATTRIBUTE3" , "ATTRIBUTE_TIMESTAMP" , "REMARKS" )
 VALUES (
  5                     -- PROFILEID
 ,'MONITOR CONNECTIONS' -- monitors number of connections
 ,'WARNING_DIAGLEVEL2'  -- DB2 issues DSNT772I when threshold exceeded
 , 1                    -- NUMBER OF ACTIVE THREADS ALLOWED
 , NULL                 -- ATTRIBUTE3
 , CURRENT TIMESTAMP    -- ATTRIBUTE_TIMESTAMP
 ,'PRDID'               -- REMARKS
 );
```

From the requesting location (in our test scenario this was a DB2 LUW client machine) we used multiple instances of the DB2 command line processor to create the desired number of DB2 connections. After the profile threshold entered in Example 4-46 on page 195 was exceeded we observed the DB2 message shown in Figure 4-83.

DB2 reason code 00E30503 indicates that a warning occurred because the number of connections exceeded the warning setting for the MONITOR CONNECTIONS keyword in a monitor profile for the LOCATION filtering scope.

```
DSNT772I  -D0Z1 DSNLILNR A MONITOR PROFILE WARNING
CONDITION OCCURRED
1 TIME(S)
IN PROFILE ID=5
WITH PROFILE FILTERING SCOPE=IPADDR
WITH REASON=00E30503
```

Figure 4-83   Message DSNT772I for threshold exceeded

Additional information

For information about managing and implementing DB2 profile monitoring, refer to Chapter 45. Using profiles to monitor and optimize performance of *DB2 10 for z/OS, Managing Performance*, SC19-2978.

4.3.23 SYSPROC.ADMIN_DS_LIST stored procedure

The SYSPROC.ADMIN_DS_LIST stored procedure invokes the z/OS Catalog Search Interface (CSI) to obtain information about data sets contained in integrated catalog facility (ICF) catalogs. Data set entries are selected using a generic data set filter. The data set filter can be a fully-qualified name, in which case one entry is returned, or a generic filter key containing wild cards so that multiple entries can be returned on a single invocation. The syntax for providing a generic filter keys is similar to providing the dsname level information in the ISPF data set list utility.

You can use the SYSPROC.ADMIN_DS_LIST stored procedure to perform regular monitoring on data set extends, DASD usage, VSAM high allocated and high used RBA (relative byte address). SYSPROC.ADMIN_DS_LIST returns the data set information through a result set cursor that it opens on the temporary table SYSIBM.DSLIST. A list of columns returned by the result set cursor is shown in Example 4-47.

Example 4-47 SYSPROC.ADMIN_DS_LIST result set

```
DSNAME                    1 VARCHAR      44
CREATE_YEAR               2 INTEGER       4
CREATE_DAY                3 INTEGER       4
TYPE                      4 INTEGER       4
VOLUME                    5 CHAR          6
PRIMARY_EXTENT            6 INTEGER       4
SECONDARY_EXTENT          7 INTEGER       4
MEASUREMENT_UNIT          8 CHAR          9
EXTENTS_IN_USE            9 INTEGER       4
DASD_USAGE               10 CHAR          8
HARBA                    11 CHAR          6
HURBA                    12 CHAR          6
ERRMSG                   13 VARCHAR     256
```

A sample on how to invoke the stored procedure to retrieve data set related information for table and index space related VSAM LDS data sets for database `DBTR8074` is provided in Example 4-48.

Example 4-48 SYSPROC.ADMIN_DS_LIST stored procedure invocation

```
CALL SYSPROC.ADMIN_DS_LIST('DB0ZD.DSNDBD.DBTR8074.*.I%%%%.A%%%', 'N', 'N',
99999,'N',?,?)
```

The information about DASD usage (DASD_USAGE), high used (HURBA) and high allocated RBA (HARBA) are returned as binary character string which is not useful when it comes to performing computations using these information. For instance, you might want to subtract the high used RBA from the high allocated RBA to calculate the real DASD usage in bytes or to determine table or index space over or under allocation. To cast the binary character string information to a big integer value we use the DB2 UNIX System Services command line processor to run the SQL shown in Example 4-49.

Example 4-49 SYSPROC.ADMIN_DS_LIST cast to BIGINT

```
UPDATE COMMAND OPTIONS using c OFF ;    1
CONNECT TO localhost:39000/DB0Z USER DB2R3 USING <password>; 2
--
CALL SYSPROC.ADMIN_DS_LIST( 3
'DB0ZD.DSNDBD.DBTR8074.*.I%%%%.A%%%', 'N',
'N', 99999,'N',?,?);
```

```
--
SELECT 4
  DSNAME
, db2r3.bigint(DASD_USAGE) AS DASD_USAGE 5
, db2r3.bigint(HARBA)  AS HARBA5
- db2r3.bigint(HURBA)  AS HURBA 5
, CREATE_YEAR , CREATE_DAY , TYPE , VOLUME , PRIMARY_EXTENT
, SECONDARY_EXTENT , MEASUREMENT_UNIT , EXTENTS_IN_USE
FROM SYSIBM.DSLIST           ;
--
terminate;
```

The SQL shown in Example 4-49 on page 197 performs the following processing steps:

1. SYSPROC.ADMIN_DS_LIST stores its result in the temporary tale SYSIBM.DSLIST. The temporary table is dropped at commit. The update command in Example 4-49 on page 197 deactivates auto commit to make the temporary table available for processing across the current commit scope.
2. Next we connect to DB2 using the data sharing group IP address, the SQL port and the DB2 location name.
3. We then call the SYSPROC.ADMIN_DS_LIST stored procedure. We ignore the procedure result because we
4. subsequently query the SYSIBM.DSLIST temporary table that was created and populated by the stored procedure.

   In the SQL select list we use the BIGINT user defined scalar function (scalar UDF) to cast the binary character value to BIGINT which enables us to use SQL to calculate the difference between high allocated RBA and high used RBA. This calculation determines the amount of table or index space over or under allocation.

   We provided the program source and DDL for implementing and defining the DB2R3.BIGINT scalar UDF in Appendix G, “External user-defined functions” on page 563.

4.3.24 DB2 real time statistics

DB2 RTS provides another powerful tool that you can use in your daily DB2 object maintenance strategy. DB2 provides the following RTS tables that you can query:

- SYSIBM.SYSTABLESPACESTATS - RTS for table spaces and partitions
- SYSIBM.SYSINDEXSPACESTATS - RTS for index spaces and partitions

You can query DB2 RTS to obtain the following information about table space, index space, and on partition level.

- SQL DELETE, INSERT and UPDATE frequency since the last LOAD, RUNSTATS or COPY utility. You can use this information to determine how frequently table spaces and indexes are accessed for DELETE, INSERT and UPDATE DML operations.
- Number of active pages
- Number of allocated pages
- Number of data set extents
- Whether you should run the REORG, RUNSTATS or COPY utility.
- Total number of rows stored in the table space
- Total number of index entries in the index space

- Size of data occupied by rows. You can compare this information with the number of active pages to review page usage efficiency.
- Type of the disk (HDD or SSD) the table or index space VSAM data set resides on
- High performance list prefetch facility capability indicator of the disk the VSAM data set resides on
- Number of index levels in the index tree
- Number of pages containing pseudo deleted index entries
- The date when the index was last used for SELECT, FETCH, searched UPDATE, searched DELETE, or used to enforce referential integrity constraints. This information can be useful to determine unused indexes.
- The number of times the index was used for SELECT, FETCH, searched UPDATE, searched DELETE, or used to enforce referential integrity constraints, or since the object was created.

RTS snapshot tables

In our workload scenario we take RTS snapshots to determine the number of DELETE, INSERT, and UPDATE statements. For index spaces we use RTS to identify unused indexes. We use the SQL statements shown in Example 4-50 to create our RTS shadow table used to store the RTS snapshots taken before and after workload execution.

Example 4-50 Create RTS snapshot table

```
CREATE TABLE TABLESPACESTATS LIKE SYSIBM.SYSTABLESPACESTATS;
COMMIT;
ALTER  TABLE TABLESPACESTATS ADD COLUMN SNAPSHOTTS TIMESTAMP;
COMMIT;
CREATE TABLE INDEXSPACESTATS LIKE SYSIBM.SYSINDEXSPACESTATS;
COMMIT;
ALTER  TABLE INDEXSPACESTATS ADD COLUMN SNAPSHOTTS TIMESTAMP;
COMMIT;
```

Populating the RTS snapshot tables

Before and after workload execution we stopped and started the DayTrader database table spaces to trigger the externalization of the RTS information an ran the SQL statements shown in Example 4-51 to take the RTS snapshot information.

Example 4-51 Take RTS snapshot information

```
-- ------------------------------
-- snapshot indexspace RTS
-- ------------------------------
INSERT INTO INDEXSPACESTATS
SELECT A.* , CURRENT TIMESTAMP
FROM SYSIBM.SYSINDEXSPACESTATS  A
WHERE DBNAME = 'DBTR8074';
-- ------------------------------
-- snapshot tablespace RTS
-- ------------------------------
INSERT INTO TABLESPACESTATS
SELECT A.* , CURRENT TIMESTAMP
FROM SYSIBM.SYSTABLESPACESTATS A
WHERE DBNAME = 'DBTR8074';
```

Querying the RTS snapshot tables

You can use SQL queries on the RTS snapshot tables to determine the number of table or index changes that occurred during workload testing. All you need to do is to take the actions described in "Populating the RTS snapshot tables" on page 199 and run SQL queries to determine the difference between the SQL DML counters that you stored in your RTS snapshot tables before and after workload execution.

In the SQL sample query shown in Example 4-52 we query the RTS table space snapshot table to determine the number of inserts, updates and deletes performed on the DayTrader database during the workload execution that we performed between `2012-08-17-22.57.57.673670` and `2012-08-17-22.57.57.673670`.

Example 4-52 Query RTS snapshot table

```
WITH
Q1 AS
(  SELECT
   DBNAME,NAME,PARTITION,
   NACTIVE,NPAGES,REORGINSERTS,REORGDELETES,REORGUPDATES,
   REORGMASSDELETE,TOTALROWS,SNAPSHOTTS
   FROM TABLESPACESTATS
   WHERE SNAPSHOTTS = '2012-08-17-22.57.57.673670'
     AND DBNAME = 'DBTR8074'
   ORDER BY DBNAME, NAME, SNAPSHOTTS),
Q2 AS
(  SELECT
   DBNAME,NAME,PARTITION,
   NACTIVE,NPAGES,REORGINSERTS,REORGDELETES,REORGUPDATES,
   REORGMASSDELETE,TOTALROWS,SNAPSHOTTS
   FROM TABLESPACESTATS
   WHERE SNAPSHOTTS = '2012-08-17-23.08.49.191718'
     AND DBNAME = 'DBTR8074'
   ORDER BY DBNAME, NAME, SNAPSHOTTS)
SELECT
SUBSTR(Q1.DBNAME,1,8)                       AS DBNAME,
SUBSTR(Q1.NAME  ,1,8)                       AS NAME,
Q1.PARTITION,
Q2.TOTALROWS          - Q1.TOTALROWS        AS #ROWS,
Q2.REORGINSERTS       - Q1.REORGINSERTS     AS INSERTS  ,
Q2.REORGDELETES       - Q1.REORGDELETES     AS DELETES  ,
Q2.REORGUPDATES       - Q1.REORGUPDATES     AS UPDATES  ,
Q2.REORGMASSDELETE    - Q1.REORGMASSDELETE  AS MASSDELETE
FROM Q1,Q2
WHERE
(Q1.DBNAME,Q1.NAME,Q1.PARTITION) = (Q2.DBNAME,Q2.NAME,Q2.PARTITION)
---------+---------+---------+---------+---------+---------+---------+---
DBNAME    NAME      PARTITION   #ROWS INSERTS DELETES  UPDATES MASSDELETE
---------+---------+---------+---------+---------+---------+---------+---
DBTR8074  TSACCEJB          0   11489   11489       0   133129          1
DBTR8074  TSACPREJ          0   11489   11489       0    21603          1
DBTR8074  TSHLDEJB          0    2476   24037   21561    21561          1
DBTR8074  TSKEYGEN          0       3       3       0       83          0
DBTR8074  TSORDEJB          0   45598   45598       0   158305          1
DBTR8074  TSQUOEJB          0    1000    1000       0    45580          1
DSNE610I NUMBER OF ROWS DISPLAYED IS 6
```

4.3.25 Using RTS to obtain COPY, REORG and RUNSTATS recommendations

Rather than querying the RTS tables yourself we recommend to use the SYSPROC.DSNACCOX stored procedure to obtain COPY, REORG and RUNSTATS utility recommendations. DSNACCOX intelligently combines your input parameters and filters with built in intelligence, data from the DB2 catalog and RTS to determine whether table space or index space reorganizations, Runstats or Copy utilities are due for execution. For instance, DSNACCOX with DB2 10 for z/OS implements specific code to reduce the reorg requirements for table spaces residing on SSD volumes.

For information about DB2 REORG and SSD disks refer to deverloperWorks article Solid-state drives: Changing the data world at
http://www.ibm.com/developerworks/data/library/dmmag/DMMag_2011_Issue3/Storage/index.html.

As for RUNSTATS recommendations we use the administrative scheduler to make use of the autonomic statistics maintenance feature. Autonomic statistics maintenance internally calls the DSNACCOX procedure to obtain its RUNSTATS recommendations. See Appendix A, “DB2 administrative task scheduler” on page 483.

Additional information

For additional information about using the DSNACCOX stored procedure refer to Chapter 34. Setting up your system for real-time statistics of *DB2 10 for z/OS, Managing Performance*, SC19-2978.

4.4 Tivoli OMEGAMON XE for DB2 Performance Expert for z/OS

We use the OMEGAMON performance database (PDB) tables to store historical DB2 accounting and statistics information in DB2 tables. For information about how we create and load the PDB tables refer to Appendix D, “IBM OMEGAMON XE for DB2 performance database” on page 527.

4.4.1 Extract, transform, and load DB2 accounting FILE and statistics information

The processing flow shown in Figure 4-84 illustrates the major processing steps that are required to extract and load non-aggregated accounting and statistics information into OMPE PDB tables.

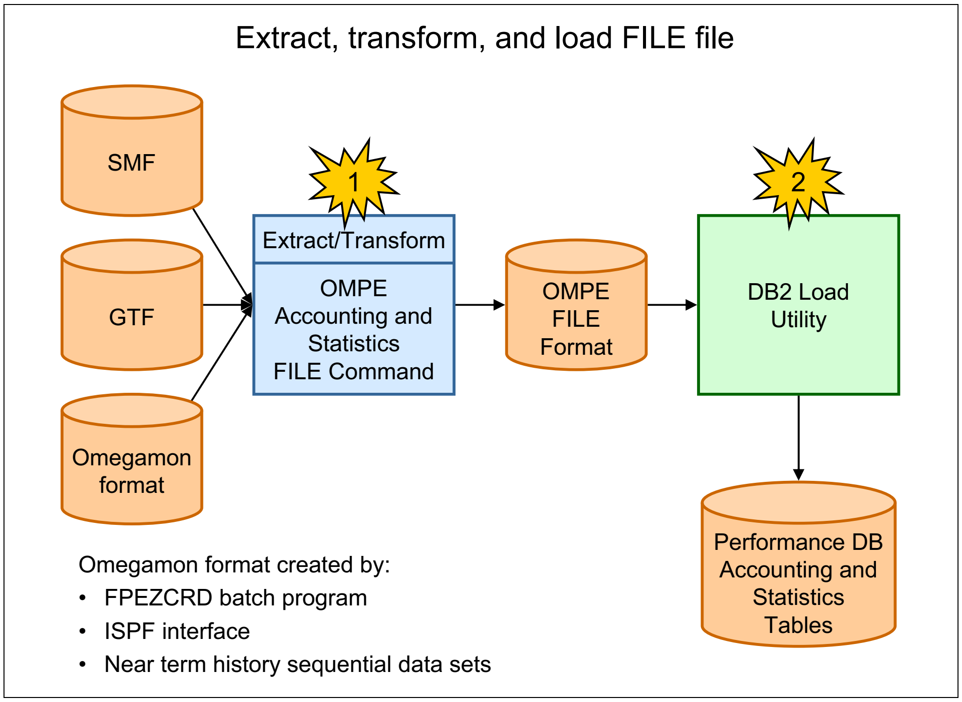


Figure 4-84 ETL accounting FILE and statistics data

1. The OMEGAMON XE Performance Expert batch utility executes ACCOUNTING and STATISTICS FILE commands to convert the information provided by SMF, GTF or OMEGAMON formatted DB2 trace data into the OMEGAMON XE Performance Expert FILE format output.
2. The DB2 load utility loads the OMEGAMON XE Performance Expert formatted FILE data set into the PDB accounting and statistics tables.

4.4.2 Extract, transform and load DB2 accounting SAVE information

The processing flow shown in Figure 4-85 on page 203 illustrates the major processing steps that are required to extract and load aggregated accounting information into the OMPE PDB accounting SAVE tables.

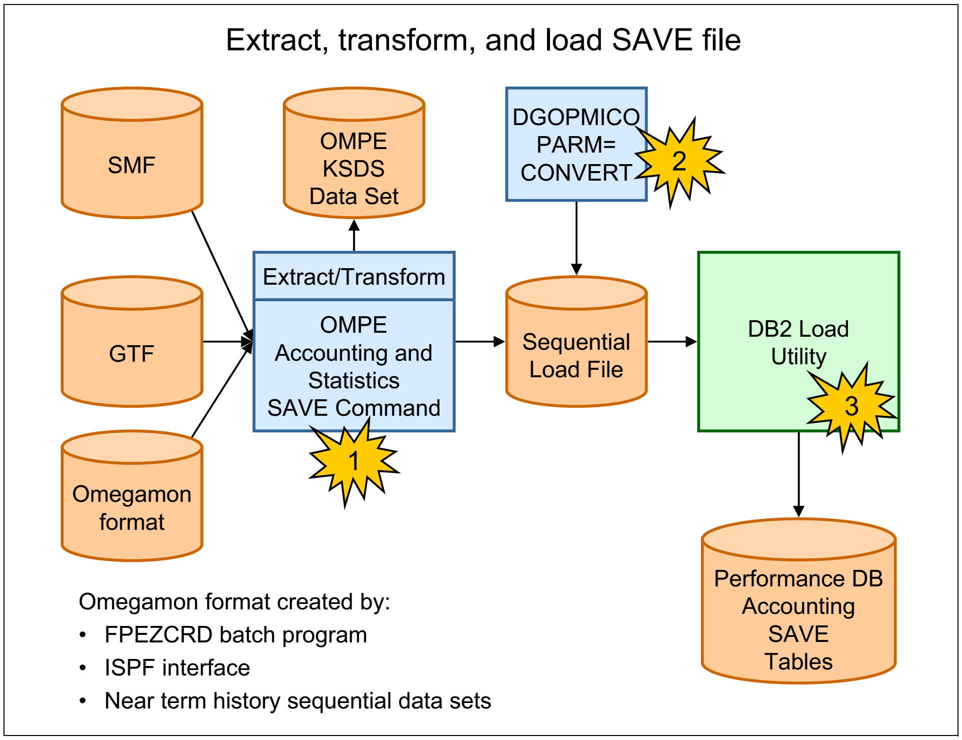


Figure 4-85 ETL accounting SAVE data

1. The OMEGAMON XE Performance Expert DB2PM batch utility executes an ACCOUNTING SAVE command to provide the requested one minute interval aggregated accounting data. The aggregated data is written to a VSAM KSDS data set.
2. OMPE utility DGOPMICO converts the information provided by the VSAM KSDS into a loadable sequential data set.
3. The DB2 load utility loads the OMEGAMON XE Performance Expert formatted accounting SAVE data set its corresponding the PDB accounting tables.

4.4.3 Querying the performance database tables

After the PDB accounting and statistics tables are in place and regularly populated you can create and run your own queries to profile and monitor your applications. In our scenario we use the DB2 clientApplicationInformation provided by the DayTrader application for application profiling and monitoring. To encapsulate query complexity we created an SQL table UDF to allow others to reuse the PDB query just by referencing the UDF as shown in Example 4-53.

Example 4-53 Using the PDB table UDF

```
---------+---------+---------+---------+---------+---------+---------+--
select * from table(accounting_profile('TraderClientApplication')) a;
---------+---------+---------+---------+---------+---------+---------+--
DATETIME          CLIENT_TRANSACTION                                ELAPSED
---------+---------+---------+---------+---------+---------+---------+--
2012-08-14-22.48  TraderClientApplication                               250
2012-08-14-22.47  TraderClientApplication                              3242
```

See Appendix D.4, “Sample query for application profiling” on page 540.

4.4.4 Additional information

For additional information about implementing and using the OMPE performance database you might want to refer to the following manuals:

- *A Deep Blue View of DB2 Performance: IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS*, SG24-72244
- Chapter 5. Advanced reporting concepts, The Performance Database and the Performance Warehouse of I*BM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS*, SH12-6927

4.5 DB2 database and application design considerations

There are design and implementation standards that are to be considered for application and database resilience in a DB2 for z/OS environment. Such standards provide recommendations to help you with database design, to support you in implementing your backup, recovery and reorg strategy and to provide application design and coding guidelines.

Design and implementation best practice recommendations are extensively discussed in the DB2 for z/OS documentation and in the DB2 for z/OS Best Practices web site. For further information, refer to the following documentation:

- *Achieving the Highest Levels of Parallel Sysplex Availability in a DB2 Environment*, IBM REDP-3960.
- *DB2 10 for z/OS, Managing Performance*, SC19-2978.
  - Part 4, Improving concurrency
  - Part 6, Programming applications for performance
  - Part 7, Maintaining data organization and statistics
  - Part 8, Managing query access paths
- IBM developerWorks DB2 for z/OS Best Practices papers available at
  https://www.ibm.com/developerworks/mydeveloperworks/groups/service/html/communityview?communityUuid=f8b4b297-1cd7-49b6-8e7a-8bfdcc4901e7

Database migration projects do not always apply best practice recommendations. This leads to SLA violations because of elongated application response times which often has a negative impact on application availability and scalability. To bring the most commonly observed issues to your attention, we provide the following list of database and application design pitfalls that can cause such undesired application behavior:

- There is a tendency to accept default configuration properties for WebSphere Application Server data source properties, which can be extremely painful. Always review the data source custom properties to make sure, the recommended settings in 5.11, “Configuring data source properties (webSphereDefaultIsolationLevel, currentPackagePath, pkList, and keepDynamic)” on page 288 are being used.
  - AutoCommit
    The default setting switches autocommit to ON. For read-only SQL, this can cause high CPU on the DB2 server because of connection and DBAT management. This can happen especially when the application designer believes no unit of work is necessary. The attitude is “I don't care about the unit of work - all I want is the data. Why does the database impose a unit of work on me by asking me to choose a commit point?”

- CursorHold
  The default, again, is to turn this on. If the application fails to close a cursor, then the connection cannot go inactive. This can inflate the number of threads required. Prior to DB2 10 the major concern is virtual storage, DB2 10 onwards it is real storage.
- Default isolation level
  TRANSACTION_REPEATABLE_READ (i.e. RS) with obvious consequences for concurrency - locking conflicts, time-outs and deadlocks.

- Where AutoCommit has been turned off, there can be a problem where read-only applications fail to commit. This can cause an increase in the number of threads, and can also make if difficult for utilities to execute concurrently with application workloads.
- Some update transactions commit too infrequently. This usually happens where data volumes exceed the application design expectations (or lack of) and can have detrimental effects, especially in data sharing, as it affects the global CLSN making the lock avoidance mechanism ineffective. This also occurs where the Java object is represented in a hierarchy of tables. meaning a large number of locks might have to be taken.
- As well as update transactions, there is the impact of Java Batch, where the mechanism for calculating commit frequency either is not present or means commits are too infrequent. The biggest challenge is those Java Batch applications which contain no restart logic and those where the batch process is an all-or-nothing process. These latter often occur where the data volumes exacerbate the duration of the batch window, such processes can linger on into the online day and cause severe problems.
- Unrestricted use of KEEPDYNAMIC(YES) can prevent threads from going inactive.
- Numerous tables are often stored in the same table space. In DB2 for z/OS each table should be stored in its own table space, because important tasks such as DB2 utilities, I/O tuning and table space tuning can only be performed at table space level. For instance you cannot backup or restore individual tables within the same table space. Instead, you can perform the copy or recover utility at table space level which copies or recovers all tables in the table space. The same applies to the other utilities and to table space tuning parameters. Creating one table per table space enables you to perform such tasks at table level.
- DB2 large object (LOB) auxiliary table spaces are often defined with LOG NO. This setting, while saving LOG space and improving performance for real large LOBs, might compromise data integrity in case of rollback or point in time recovery processing.
- The number of indexes can run out of control. For instance, an application might depend on DB2 for Linux, UNIX, and Windows to detect and eliminate duplicate index specifications. The DDL, as a result, has a significant number of duplicate index specifications which are not eliminated by DB2 for z/OS. As well as impacting INSERT and UPDATE performance, this also increases PREPARE time, and can in some cases make access path selection less effective because of the number of choices available.
- In some case, the installation DDL allows no customization of buffer pool assignment. This means when installing into an environment which supports multiple applications, that applications can impact each other. The DBA has to find out about this by experience and then has to perform post-installation customization, which is likely to be undone when a fix pack is applied.
- There is also a tendency to almost random buffer pool assignment, meaning indexes, data pages and LOBs are all staged in the same buffer pool, with inevitable consequences. As well as separating these out, the application designers should have some understanding of random versus sequential objects and assign them to separate buffer pools where appropriate.

- Careless use of page sizes. Some Java application tables can have large row sizes, meaning the page size can be significant in terms of space usage and performance. This is perhaps most true of indexes which are susceptible to leaf page splits because of INSERT patterns, though the most frequent problem is with LOBs which are often assigned less-than-optimal page sizes. Another point about index page sizes is the sensitivity of the Optimizer to the number of index levels.
- Blanket use of row-level locking, even where not needed. This can have a significant impact in data sharing because of page P-lock propagation.
- No provision of appropriate RUNSTATS advice. In many cases, it is assumed the customer will collect the correct statistics. DBA standards vary, of course, which is the first drawback, but the main problem is that a lot of these applications depend on optimal access path selection to achieve maximum concurrency. Without the correct statistics, of course, this is difficult, and there are often key tables which require specific statistics to be collected. The most frequent complaint is that this information is missing and that educated guesses have to be made post-installation.

  Most difficult to manage are the work-flow tables, which can grow and shrink rapidly and frequently. These often require statistics to be collected at the correct time, and then the DBA has to ensure these statistics are not overwritten. Having to solve this problem post migration by experience requires a lot of DBA experience and application knowledge which makes it extremely difficult to identify and solve this problem as one team on its own often lacks the required knowledge, skills or experience.
- Applications not been designed with high transaction volumes in mind, and as such tend to perform poorly, and are likely to have contention problems.
- Some application design causes the DB2 thread never to become inactive which in turn causes the idle thread timeout to be triggered. In some cases this causes the application to fail. Until DB2 9 for z/OS the idle thread timeout (IDTHTOIN) DSNZPARM had to be disabled at subsystem level to avoid such application failures. In DB2 10 for z/OS profiles can be used to disable IDTHTOIN at application level. For a discussion on this topic refer to 4.3.21, "Using profiles to disable idle thread timeout at application level" on page 194.

5

WebSphere Application Server infrastructure setup

Enterprises today typically use Java as the language of choice when developing enterprise class applications. These applications are typically hosted in an application server environment. WebSphere Application Server is usually the server of choice to host these applications. These applications require access to data. The data typically is on an enterprise class relational database, such as DB2 for z/OS.

Customers have many questions about how best to configure a WebSphere Application Server environment to access DB2 for z/OS. Here are some typical challenges and questions:

- What do I need to consider when I configure WebSphere Application Server, which accesses DB2 for z/OS?
- How do I configure JDBC type 2 driver access to DB2 for z/OS by using WebSphere Application Server on z/OS?
- I am a DBA. I do not know which application a SQL statement is coming from. What can I configure in my WebSphere Application Server to help me track this statement without an application change?
- What are the preferred practices for JDBC type 4 access to DB2 for z/OS to best use sysplex workload balancing?
- Why is there an XA provider for JDBC type 4 access and nothing like that for JDBC type 2 access to DB2 for z/OS?
- I do not want to grant a user ID that is used in my data source DBADM or has access to DB2 tables. I am worried that the user ID might be compromised. How can I avoid this situation in a WebSphere Application Server environment that is accessing DB2 for z/OS?
- There are many levels of JDBC driver properties; which should I use when?
- What are the preferred practices for WebSphere Application Server connection pool and prepared statement cache settings?
- What do I need to do in WebSphere Application Server to help me classify JDBC type 4 access to DB2 for z/OS in WLM?

- How do I enable JDBC driver tracing in my WebSphere Application Server?
- I am a DBA; how do I make sure that WebSphere Application Server applications are using the correct isolation level?
- How does failover work with a JDBC type 2 connection from WebSphere Application Server on z/OS to DB2 for z/OS?

In this chapter, we build an example environment that we use to provide the answers to these questions.

This chapter covers the following topics:

- Configuring WebSphere Application Server Network Deployment on z/OS
- Configuring WebSphere Application Server for JDBC type 4 XA access
- Configuring WebSphere Application Server for JDBC type 2 access
- Configuring WebSphere Application Server for sysplex workload balancing
- Configuring client information in WebSphere Application Server
- Configuring the prepared statement cache in WebSphere Application Server
- Configuring the J2C authentication alias
- Configuring connection pool sizes on data sources in WebSphere Application Server
- Enabling trusted context for applications that are deployed in WebSphere Application Server
- Configuring the JCC properties file in WebSphere Application Server
- Configuring data source properties (webSphereDefaultIsolationLevel, currentPackagePath, pkList, and keepDynamic)

5.1 Configuring WebSphere Application Server Network Deployment on z/OS

A WebSphere Application Server Network Deployment configuration (on all platforms) should be set up for high availability and scalability. It is the gold standard of deployments. High availability, also known as *resiliency*, is the ability of a system to tolerate a number of failures and remain operational. It is achieved by adding redundancy to the infrastructure to manage failures. It is critical that your infrastructure continues to respond to client requests regardless of the circumstances and that you remove all single points of failure. Planning for a highly available system takes planning across all components of your infrastructure because the overall infrastructure is available only when all of the components are available. As part of the planning, you must define the level of high availability that is needed in the infrastructure.

We chose to use WebSphere Application Server on z/OS for our example. Here are the main reasons that we chose WebSphere Application Server on z/OS:

1. WebSphere Application Server on System z has the same features and functions of WebSphere Application Server on other platforms.
2. We want to show the features of the JDBC type 2, which is the only driver that is normally used with WebSphere Application Server on z/OS to access the local DB2 for z/OS.

We used WebSphere Application Server V8.5. We built the WebSphere Application Server Network Deployment topology spread across two LPARS, as shown in Figure 5-1.

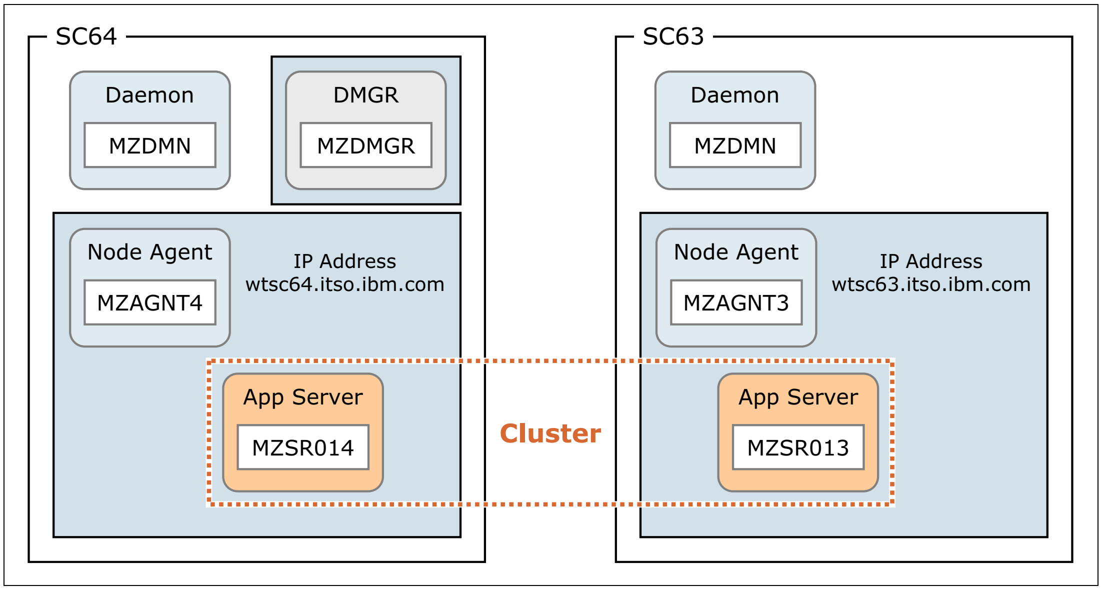


Figure 5-1 WebSphere Application Server Network Deployment configuration

The two node cell was built by following the preferred practices recommendations. These recommendations are found in the WebSphere Application Server Information Center and various documents, such as IBM Redbooks publications and techdocs. Here is the link to the Information Center for WebSphere Application Server V8.5:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp

We built the application server cluster MZSR014, which is spread across two LPARS: SC63 and SC64. The Deployment Manager MZDMGR was built to run on SC64.

The application that we used is the Apache DayTrader Sample application. Information regarding this application can be found at the following website:

https://cwiki.apache.org/GMOxDOC20/daytrader.html

This application was installed on the MZSR014 cluster.

For more information about our configuration, see Appendix B, “Configuration and workload” on page 511.

5.2 Configuring WebSphere Application Server for JDBC type 4 XA access

This section looks at the following items:

- Defining a DB2 JDBC XA provider
- Defining environment variables at the location of the IBM Data Server Driver for JDBC and SQLJ classes for JDBC type 4 connectivity
- Defining a JDBC type 4 XA data source

5.2.1 Defining a DB2 JDBC XA provider

To define a DB2 JDBC XA provider, complete the following steps:

1. In the navigation window of the administration console of WebSphere Application Server, expand **Resources**. Under resources, expand **JDBC** and you see the window that is shown in Figure 5-2.

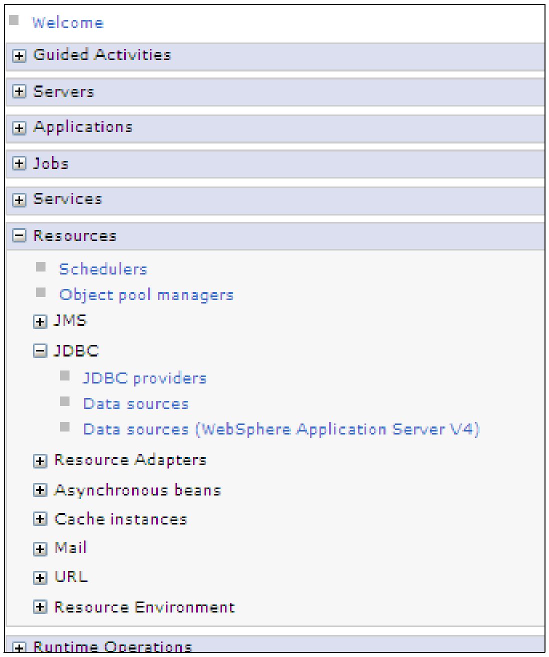


Figure 5-2 WebSphere navigation window

2. Double-click **JDBC providers** and you see the window that is shown in Figure 5-3. This window shows a list of existing JDBC providers that are defined on your server.

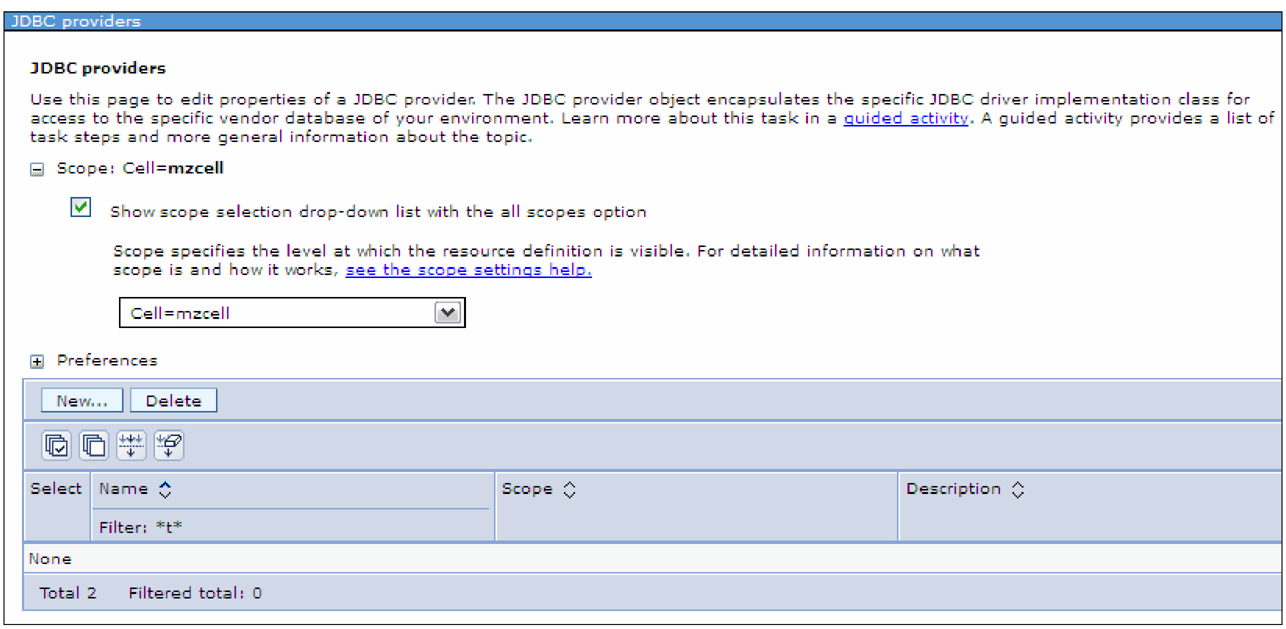


Figure 5-3 Existing JDBC providers

Resources such as Java Database Connectivity (JDBC) providers, namespace bindings, or shared libraries can be defined at multiple scopes. Resources that are defined at more specific scopes override duplicate resources that are defined at more general scopes:

– The application scope has precedence over all the other scopes.

– For WebSphere Application Server Network Deployment, the server scope has precedence over the node, cell, and cluster scopes.

– For WebSphere Application Server Network Deployment, the cluster scope has precedence over the node and cell scopes.

– The node scope has precedence over the cell scope.

In this example, select a cell scope. Click **New**. The window that is shown in Figure 5-4 opens.

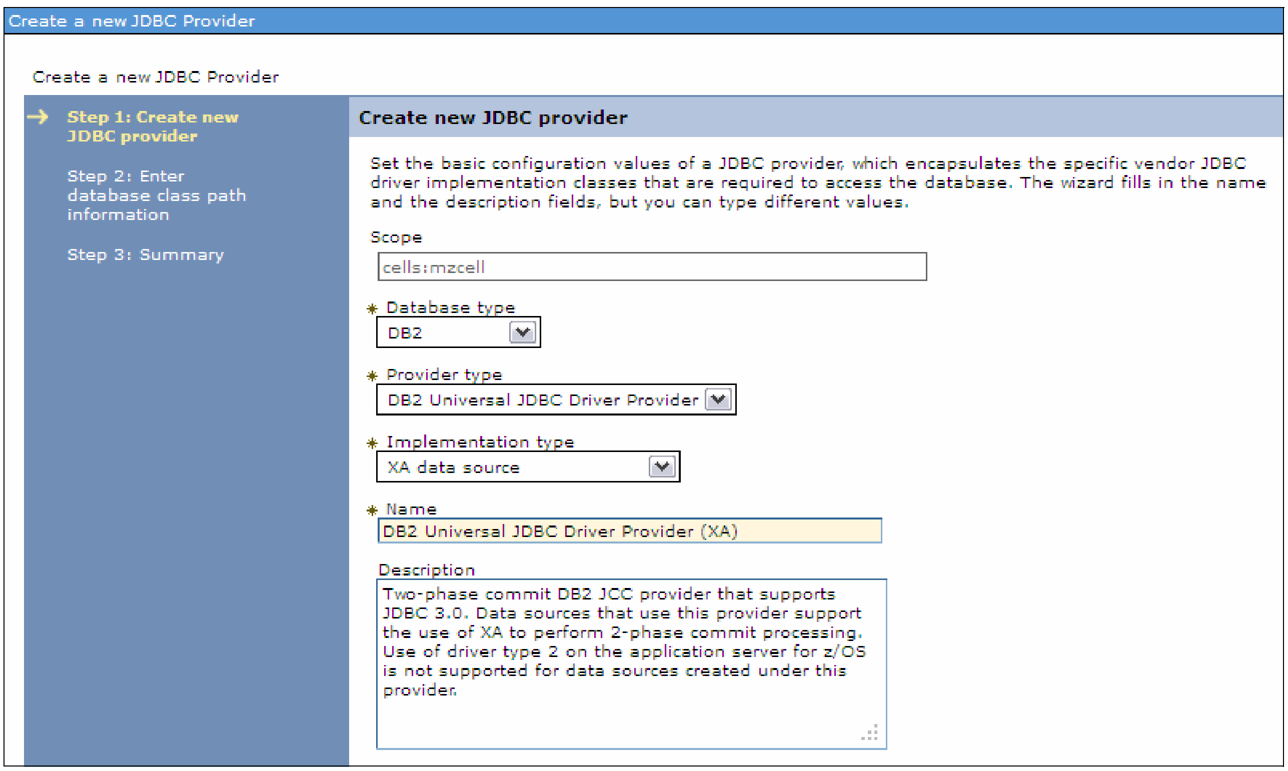


Figure 5-4 New JDBC provider definition

3. In this window, complete the following steps:

   a. Select **DB2** as the Database type from the drop-down menu.

   b. Select **DB2 Universal JDBC Driver Provider** as the provider type from the drop-down menu.

   c. Select the **XA** data source from the drop-down menu for the implementation type. The IBM Data Server Driver for JDBC and SQLJ provides a separate implementation class that supports XA transactions. This is true only for JDBC type 4 connections. If the application does not need XA capability, then select the **Connection pool** data source, which supports normal 1-phase commit transactions.

   d. Enter a provider name. In this example, use **DB2 Universal JDBC Driver Provider (XA)**.

Click **Next**. The window that is shown in Figure 5-5 opens.

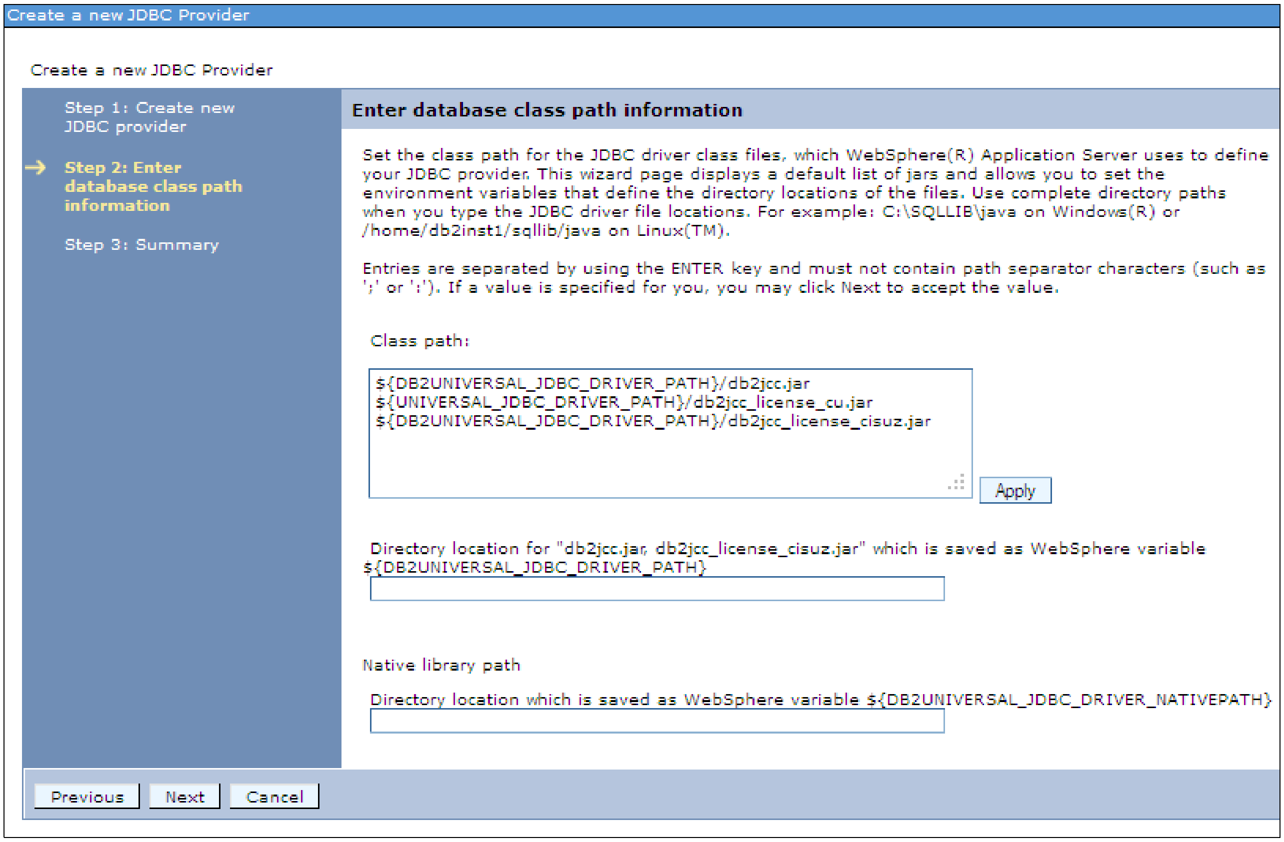


Figure 5-5 Class path definition

4. The purpose of this window is to define the location of the IBM Data Server Driver for JDBC and SQLJ classes. This is done by using variables. The usage of variables provides flexibility so that you can define the location at a single point and use that point for many JDBC providers that can be defined in a WebSphere Application Server. Write down the following variables from the window that is shown in Figure 5-5:

   – DB2UNIVERSAL_JDBC_DRIVER_PATH
   – UNIVERSAL_JDBC_DRIVER_PATH

   We show how to define these variables and their values later in this book.

   Click **Next**. The summary window that is shown in Figure 5-6 on page 213 opens.

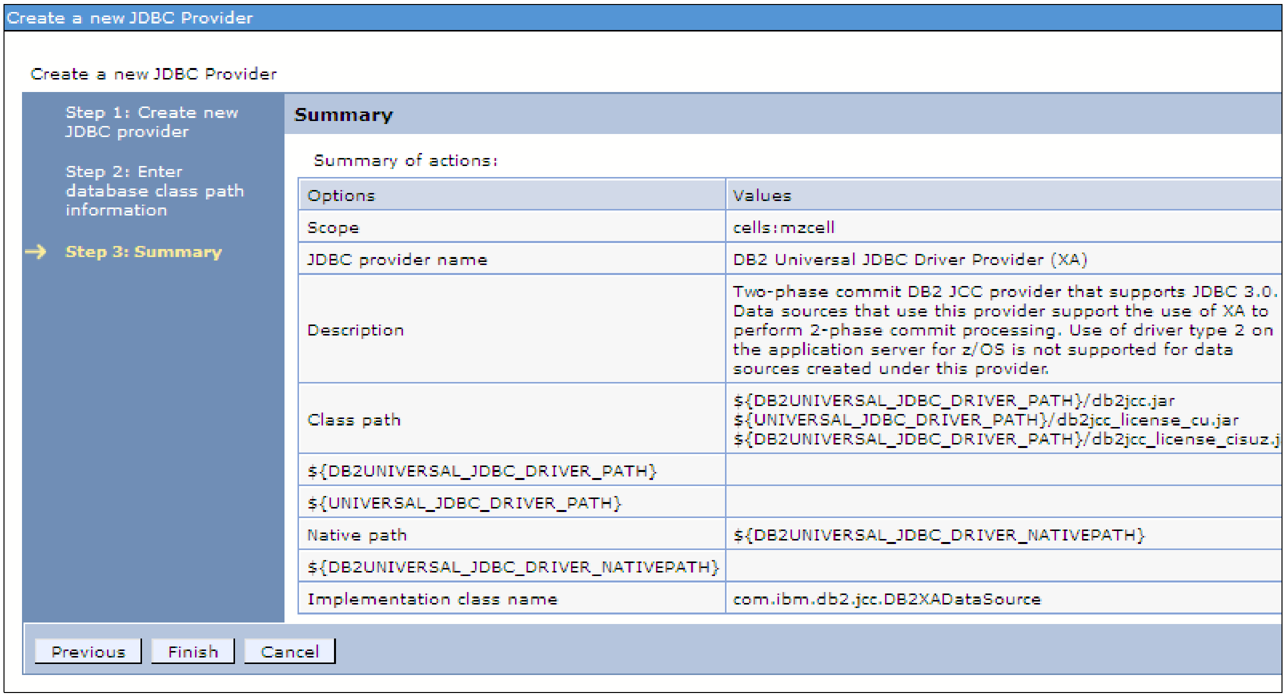


Figure 5-6 Summary window for JDBC provider

5. Click **Finish** and then save the changes.

You have defined a JDBC type 4 XA provider successfully.

5.2.2 Defining environment variables at the location of the IBM Data Server Driver for JDBC and SQLJ classes for JDBC type 4 connectivity

To define environment variables at the location of the IBM Data Server Driver for JDBC and SQLJ classes for JDBC type 4 connectivity, complete the following steps:

1. In the navigation window of the administrative console of WebSphere Application Server, expand **Environment**, as shown in Figure 5-7.

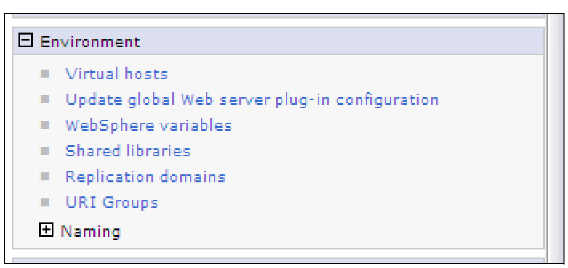


Figure 5-7 Environment window

2. Click **WebSphere variables**. The window that is shown in Figure 5-8 opens.

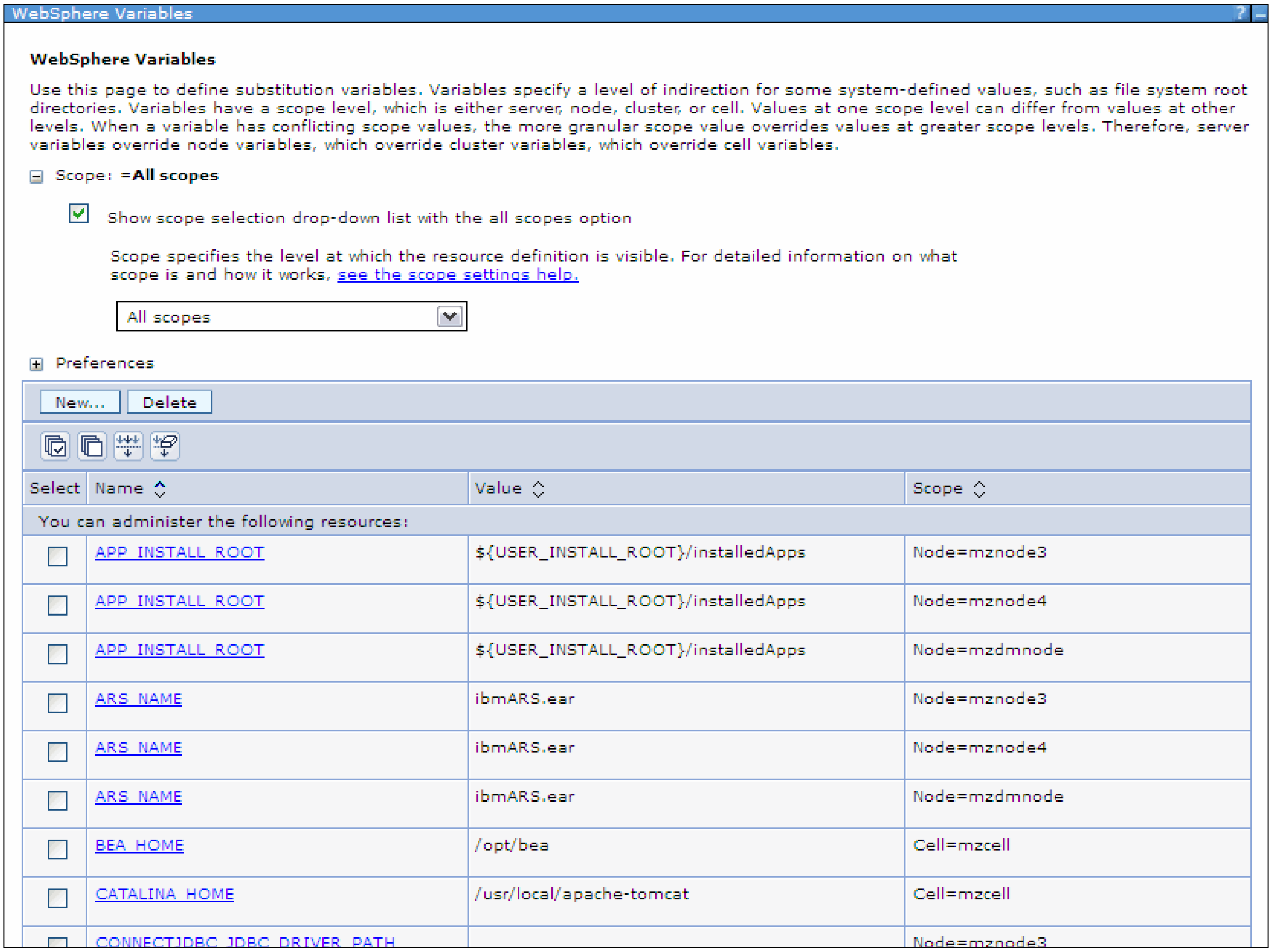


Figure 5-8 List of WebSphere variables

3. By default, the variables are defined to WebSphere Application Server at *all scopes*. The variables do not have specific values defined by default. To see the variables, click the filter icon, as shown in Figure 5-9.

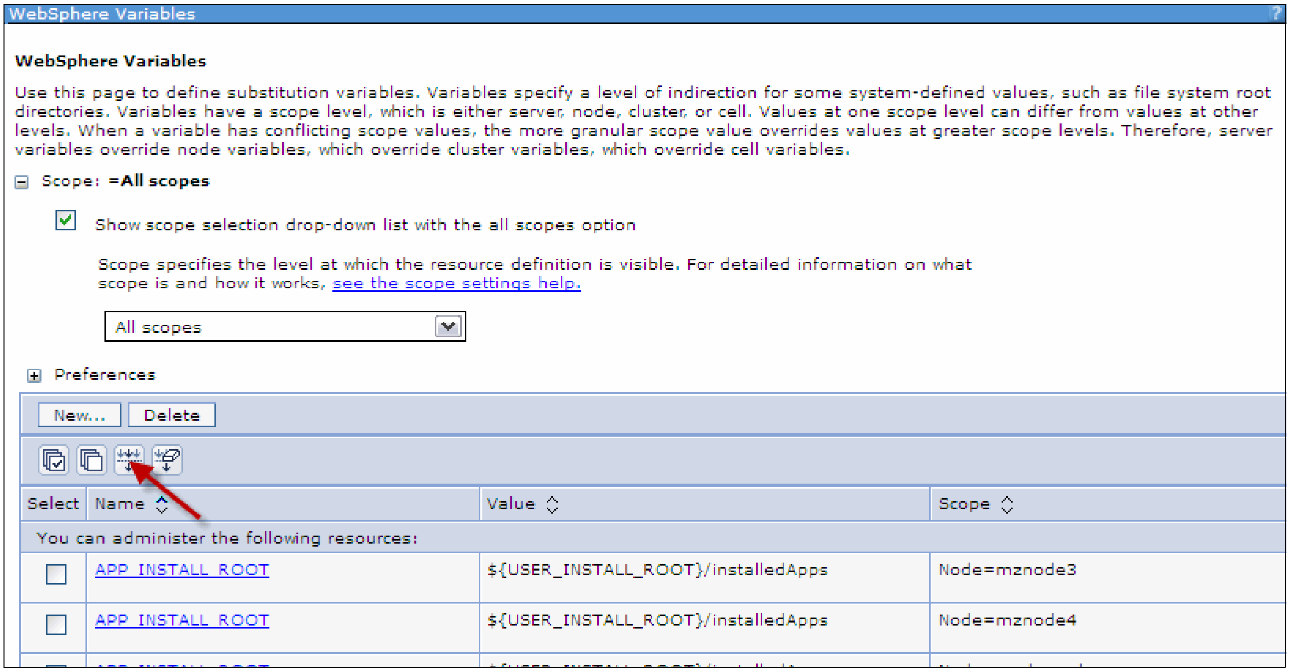


Figure 5-9 Filtering variables

4. Enter DB2 in to the search terms and click **Go**. A window with the default list of variables opens, as shown in Figure 5-10.

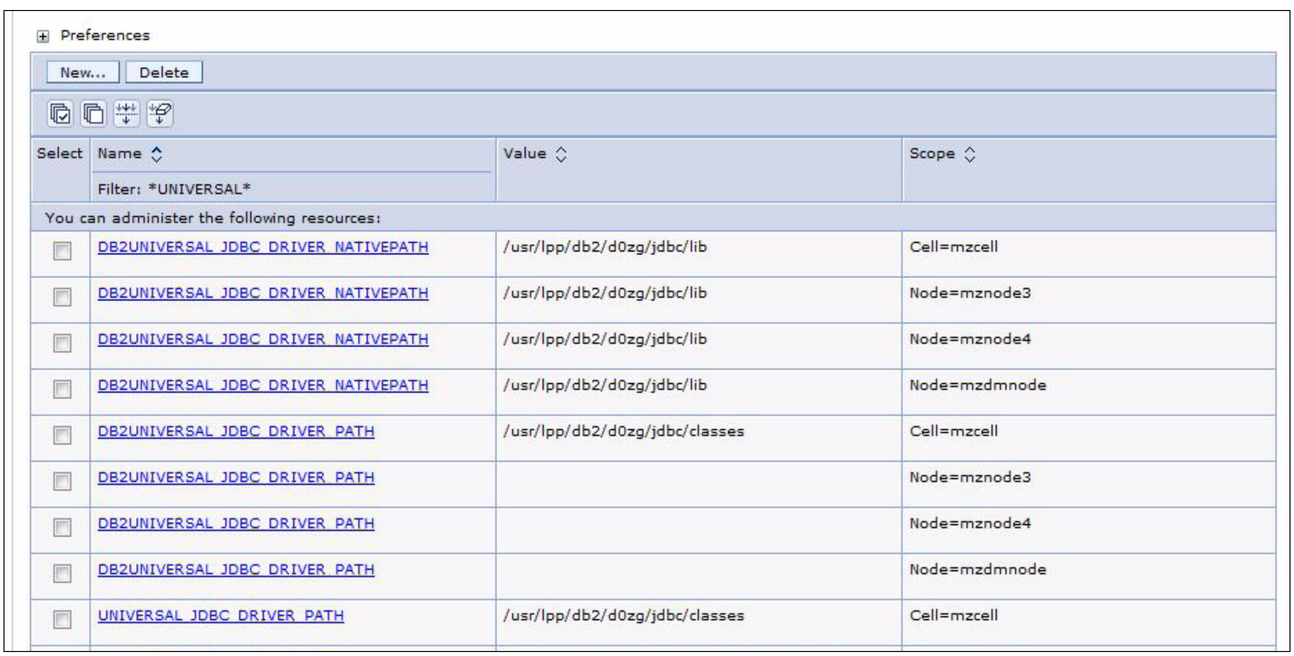


Figure 5-10 List of DB2 related variables

5. The variables are defined at all possible scopes in the cell. Pick the appropriate scope. In this example, pick the DB2UNIVERSAL_JDBC_DRIVER_PATH variable at the cell scope.

The window that is shown in Figure 5-11 opens.

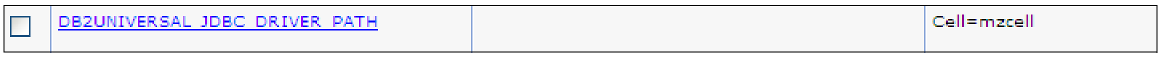


Figure 5-11 Variable and scope

6. Double-click the variable name. The window that is shown in Figure 5-12 opens.

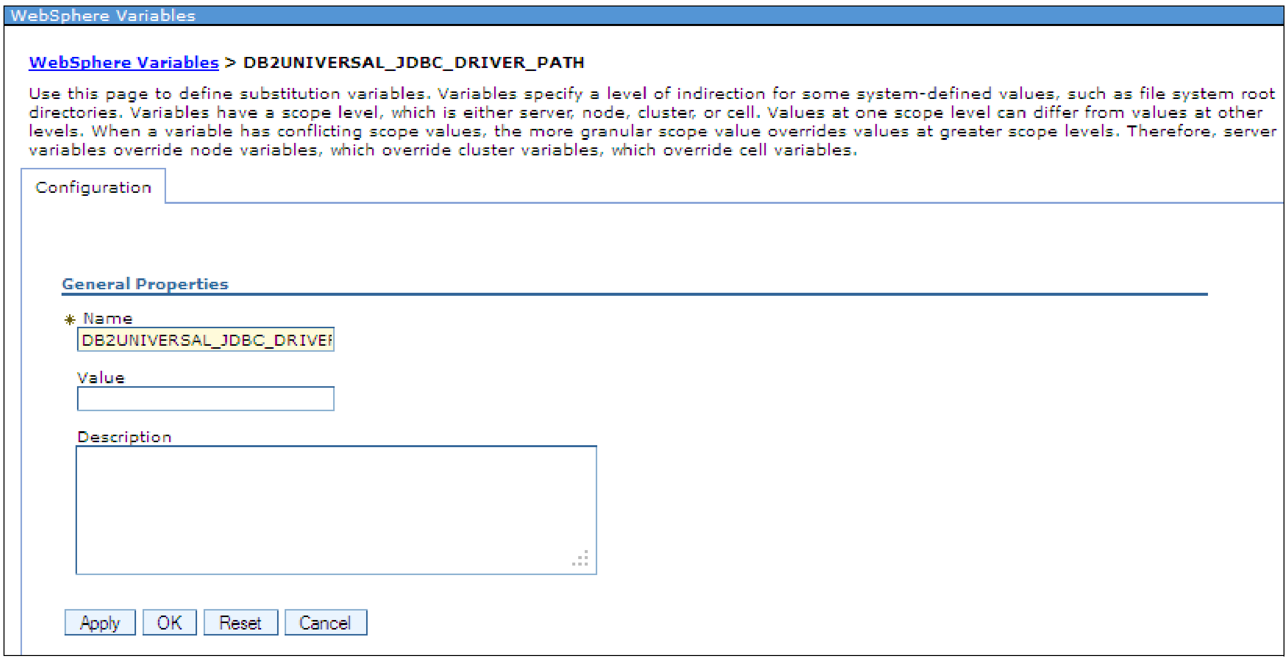


Figure 5-12 Variable for DB2UNIVERSAL_JDBC_DRIVER_PATH

7. Enter the location of the IBM Data Server Driver for JDBC and SQLJ classes in the value text box. In this example, enter /usr/lpp/db2/d0zg/jdbc/classes, as shown in Figure 5-13.

Figure 5-13 Location of the IBM Data Server Driver for JDBC and SQLJ classes

8. Click **Apply** and then save your configuration. Repeat the same steps for the UNIVERSAL_JDBC_DRIVER_PATH variable.

You have defined the variables at the cell scope successfully.

5.2.3 Defining a JDBC type 4 XA data source

To define a JDBC type 4 XA data source, complete the following steps:

1. In the navigation window of the administrative console of the WebSphere Application Server, expand **Resources** and click **Data Sources**, as shown in Figure 5-14.

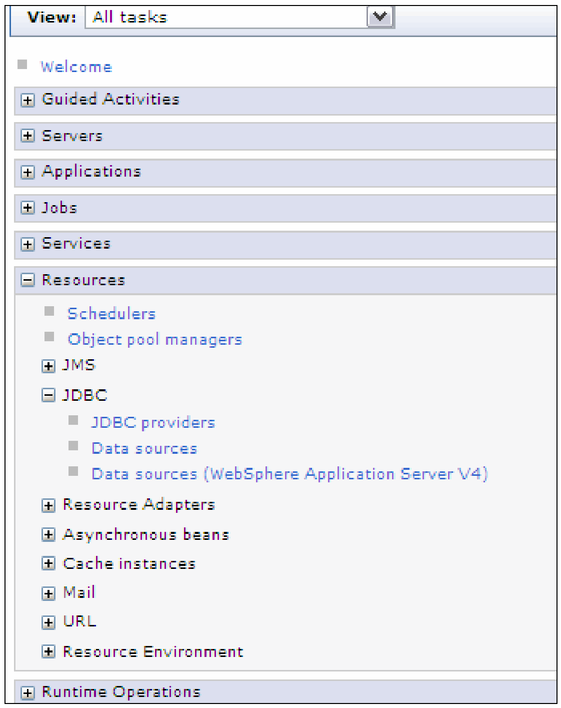


Figure 5-14 WebSphere navigation window

The window that is shown in Figure 5-15 opens. This window shows a list of existing JDBC data sources that are defined in your environment.

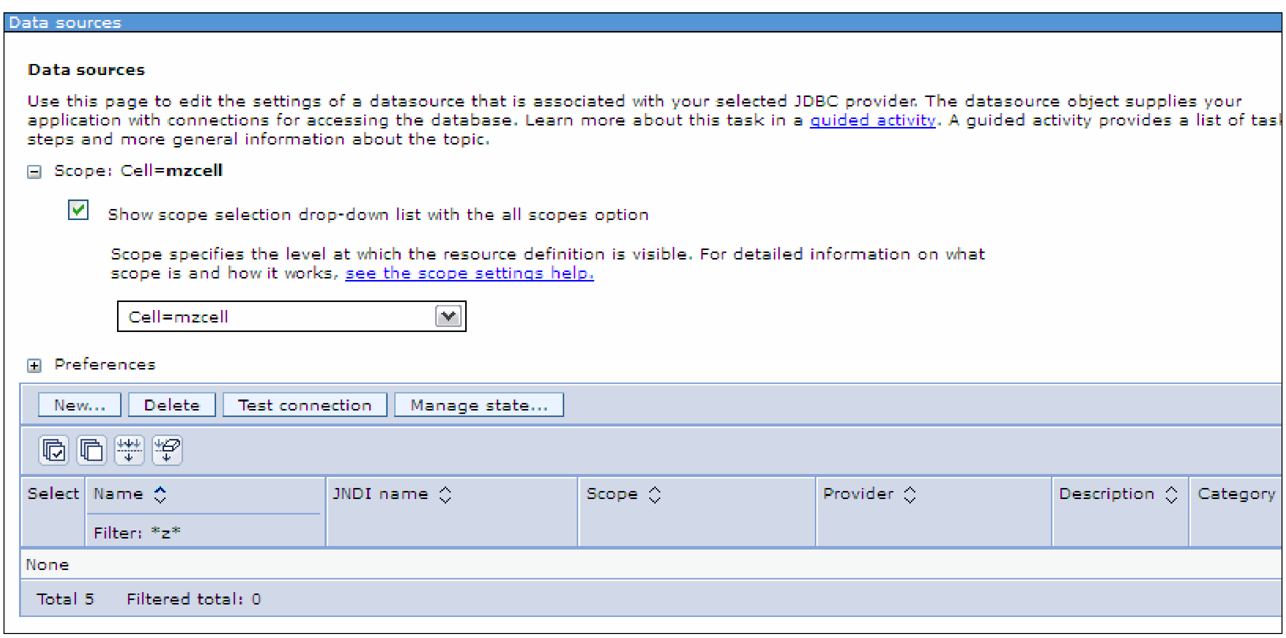


Figure 5-15 JDBC data sources

2. In this example, select the cell scope. Click **New**. The window that is shown in Figure 5-16 opens.

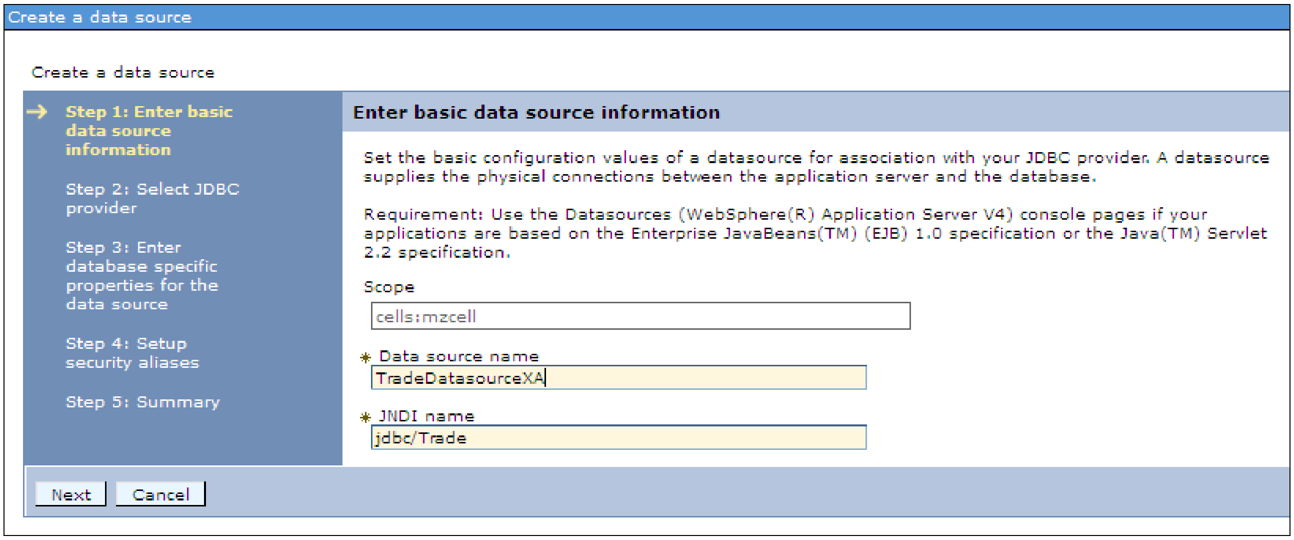


Figure 5-16 Data source definition

3. In this window, enter a name for the data source and the JNDI name. For this example, enter TradeDataSourceXA for the data source name and jdbc/Trade for the JNDI name.

   Click **Next**. The window that is shown in Figure 5-17 opens.

4. In this window, you need a JDBC type 4 XA connection, so select the DB2 Universal JDBC Driver Provider (XA) that was created earlier.

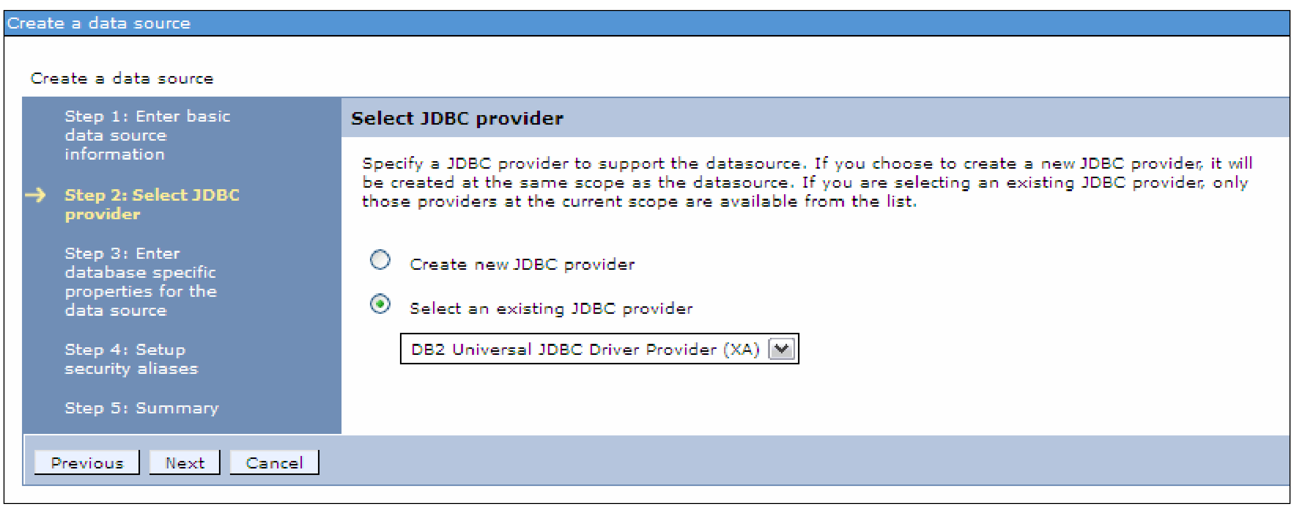


Figure 5-17 Selecting the JDBC provider

Click **Next**. The window that is shown in Figure 5-18 opens.

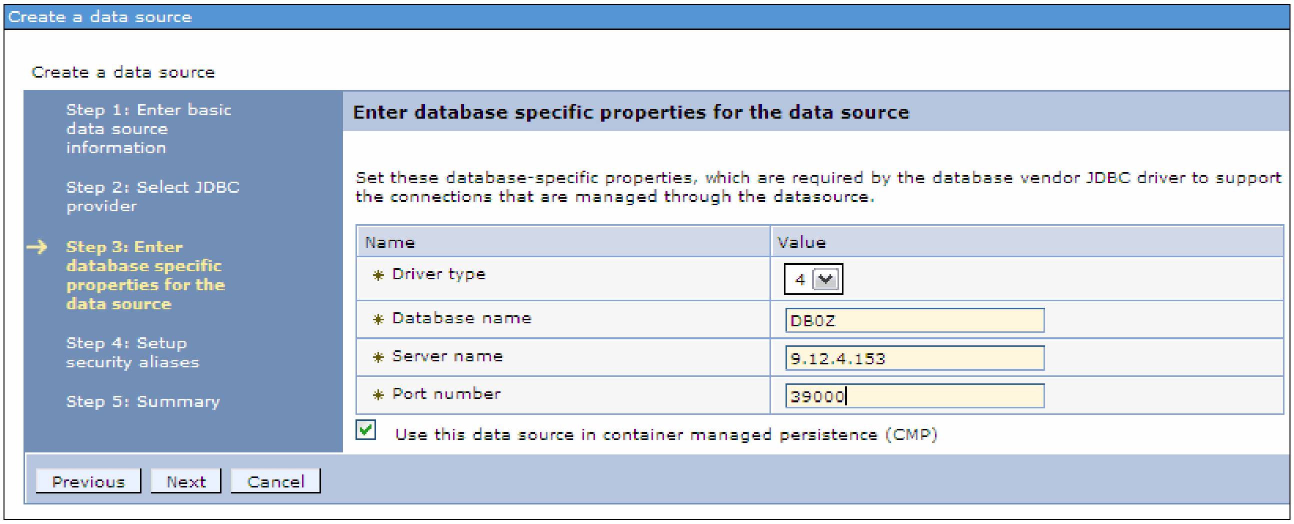


Figure 5-18 Database properties

5. In this window, enter the following values
   - For driver type, select **4** from the drop-down menu, which directs WebSphere Application Server to use a JDBC type 4 connection to the database.
   - Enter the DB2 for z/OS location name for the name of the database. The location name is *very* specific to DB2 for z/OS. In this example, enter DB0Z.
   - The server name is the IP address at which DB2 for z/OS is located. It can be an IP address or a DNS name. In this example, enter 9.12.4.153, which is the group DVIPA address of our DB2 for z/OS data sharing group. In an ideal setup, the server name should be the group DVIPA address of a DB2 for z/OS data sharing group. You must use this value if you want to use the benefits of sysplex workload balancing. You must not use the member-specific VIPA or IP addresses. If you do not have a data sharing group, then use the IP address or DNS name of the DB2 for z/OS instance. These three values are specific to a JDBC type 4 connection. The values that are entered for a JDBC type 2 connection are described later.
   - Enter the port number on which DB2 for z/OS is listening, In this example environment, DB2 use port number 39000.

   You can find the values in Figure 5-18 by running **`DISPLAY DDF`**. Example 5-1 shows the command and the values in the example z/OS system.

Example 5-1 DISPLAY DDF command for ports

```
DSNL080I  -D0Z2 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I DB0Z               -NONE             -NONE
DSNL084I TCPPORT=39000 SECPORT=0     RESPORT=39003 IPNAME=IPDB0Z
DSNL085I IPADDR=::9.12.4.153
DSNL086I SQL    DOMAIN=d0zg.itso.ibm.com
DSNL086I RESYNC DOMAIN=d0z2.itso.ibm.com
DSNL087I ALIAS              PORT  SECPORT STATUS
DSNL088I D0Z2               0     0       STARTD
DSNL089I MEMBER IPADDR=::9.12.4.142
DSNL105I CURRENT DDF OPTIONS ARE:
```

```
DSNL106I PKGREL = COMMIT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

Click **Next**. The window that is shown in Figure 5-19 opens.

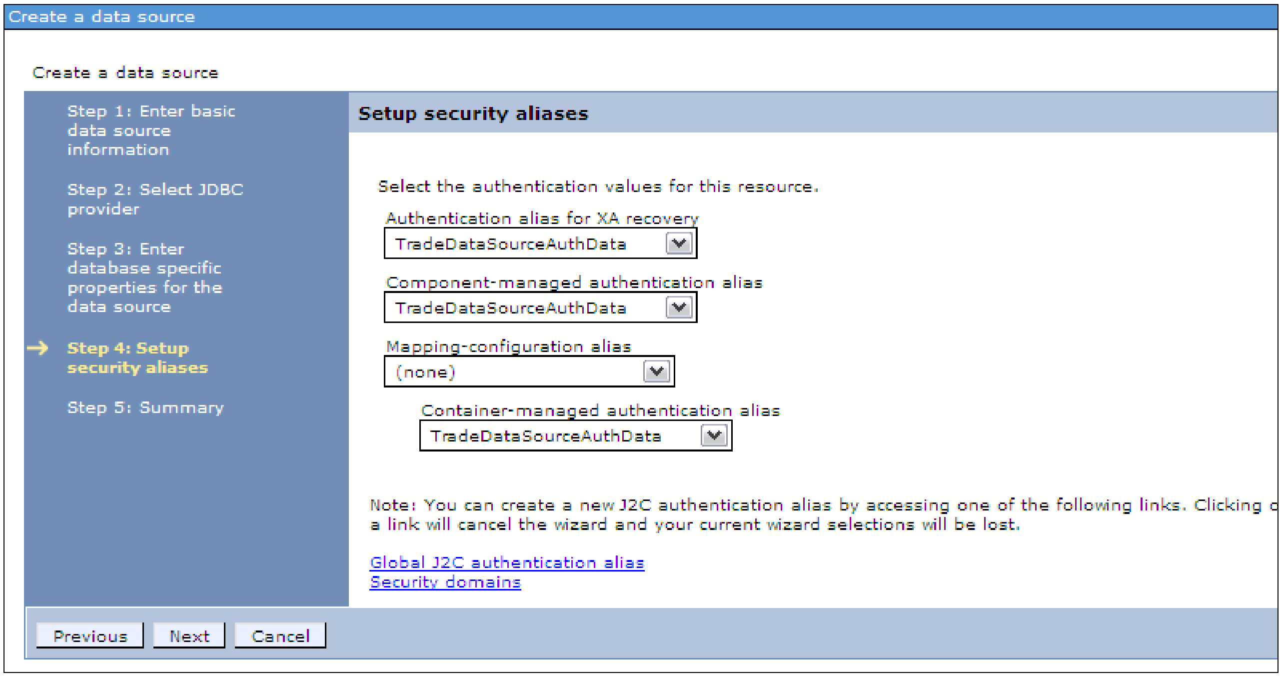


Figure 5-19 Security alias setup

6. The information in this window directs WebSphere Application Server about what user ID to use when you connect to DB2 for z/OS. Here is a brief description of what each ID means:

   Authentication alias for XA recovery: This alias is used by WebSphere Application Server when it tries to resolve any in-doubt transactions as part of XA recovery.

   Component-managed Authentication Alias: This is the user ID/password that is used to access DB2 with component managed security. The alias must be defined beforehand.

   Container-managed Authentication Alias: This is the user ID/password that is used to access DB2 with container managed security. The alias must be defined beforehand.

   These aliases are called J2C aliases. They can be defined by using the administration console.

Click **Next**. The window that is shown in Figure 5-20 opens.

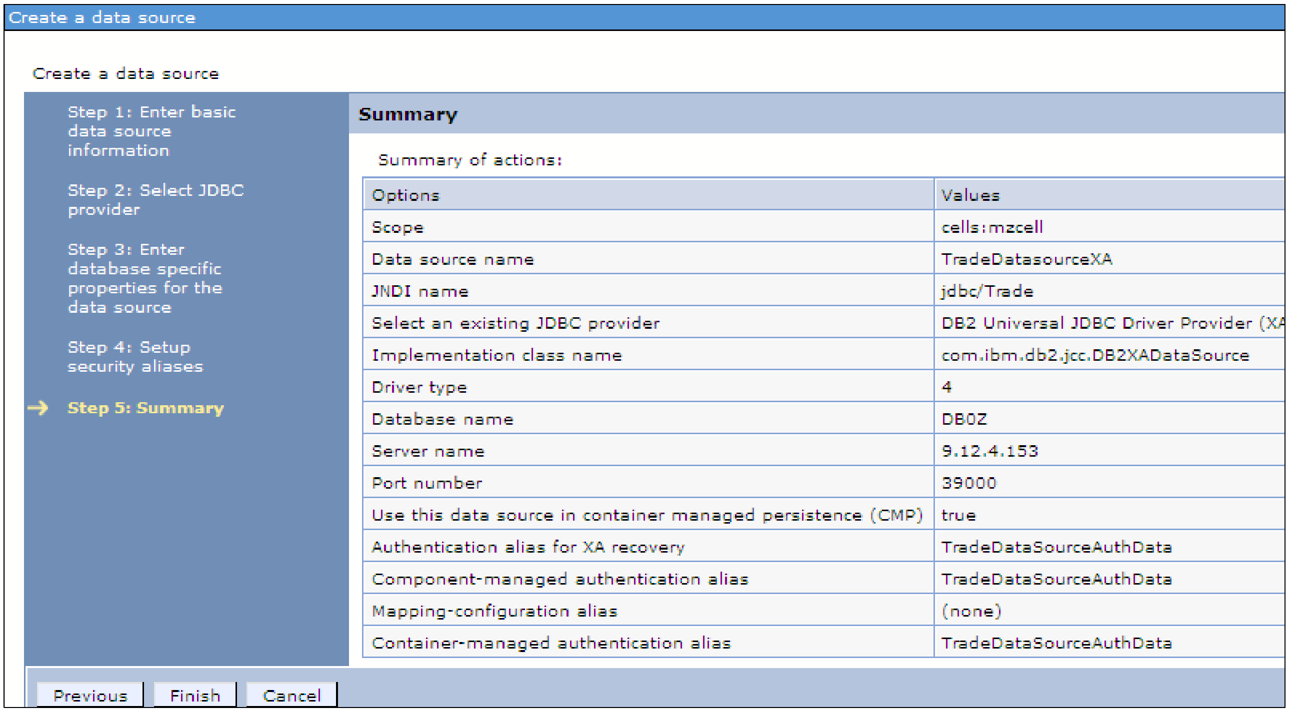


Figure 5-20 Summary of data source definition

This window is a summary window, which shows all the different values that you set. Click **Finish** and save the changes. You have a JDBC type 4 XA data source that is defined.

5.3 Configuring WebSphere Application Server for JDBC type 2 access

This section describes the following items:

- Defining a DB2 JDBC provider
- Defining environment variables to the location of the IBM Data Server Driver for JDBC and SQLJ classes for JDBC type 2 connectivity
- Defining a JDBC type 2 data source
- Configuring a subsystem ID on the data source

5.3.1 Defining a DB2 JDBC provider

To definite a DB2 JDBC provider, complete the following steps:

1. In the navigation window of the administration console of WebSphere Application Server, expand **Resources**. Under Resources, expand **JDBC** and you see the window that is shown in Figure 5-21.

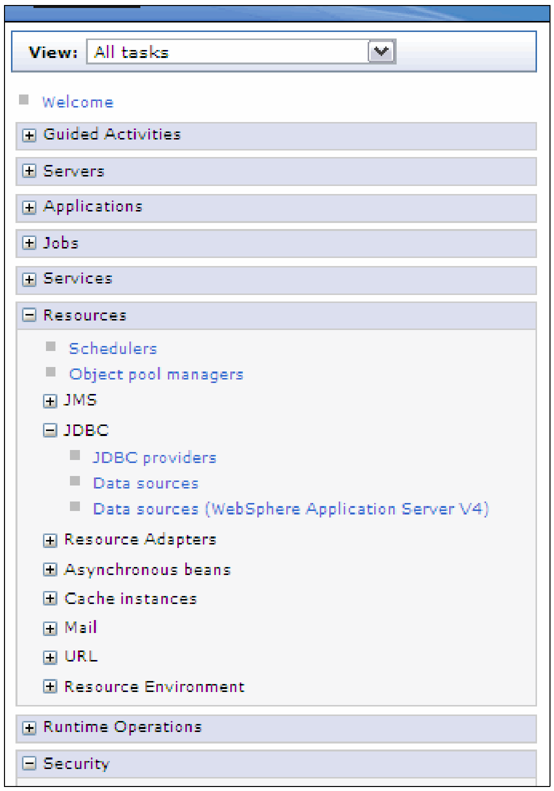


Figure 5-21 The administration console window of WebSphere Application Server

2. Double-click **JDBC providers** and the window that is shown in Figure 5-22 opens. This window shows a list of existing JDBC providers that are defined in your server.

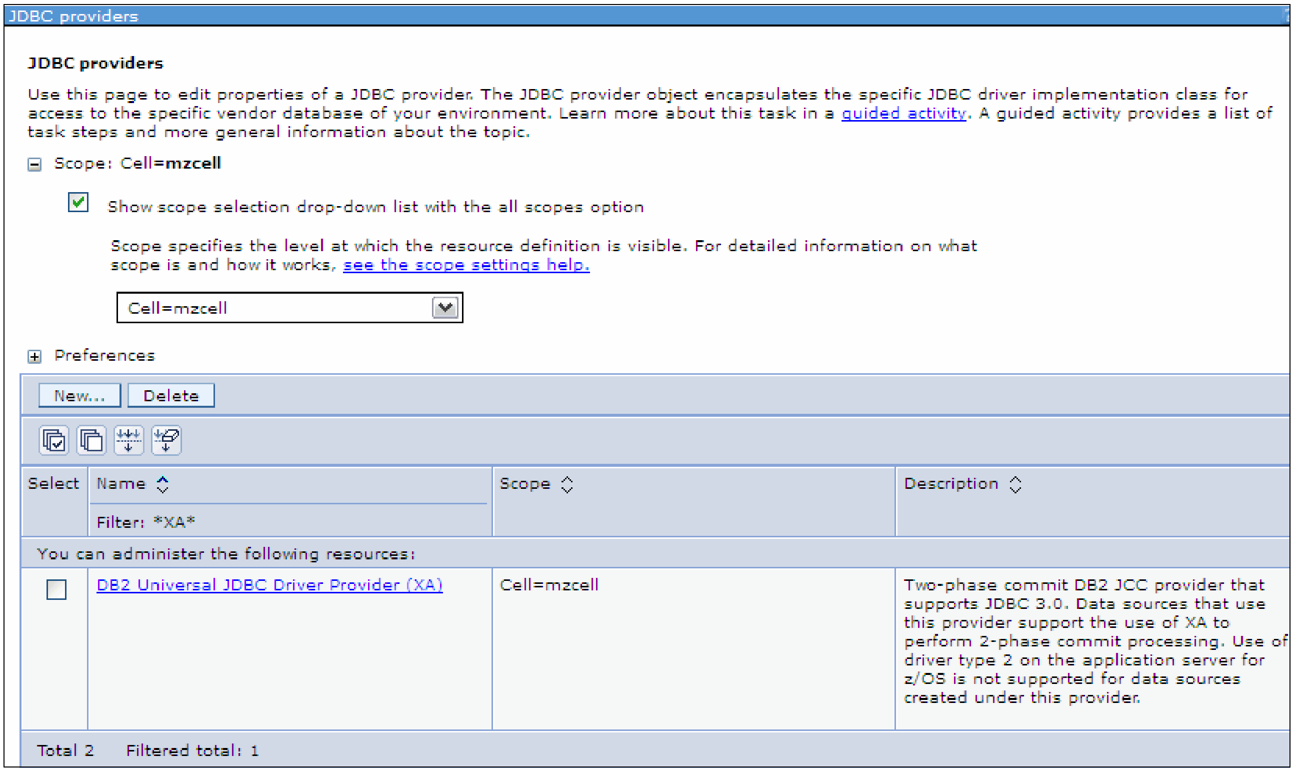


Figure 5-22 List of existing JDBC providers

3. The JDBC provider must be defined with the appropriate scope. See the scope note in Figure 5-22 on page 224. In this example, select the cell scope. Click **New** and the window that is shown in Figure 5-23 opens.

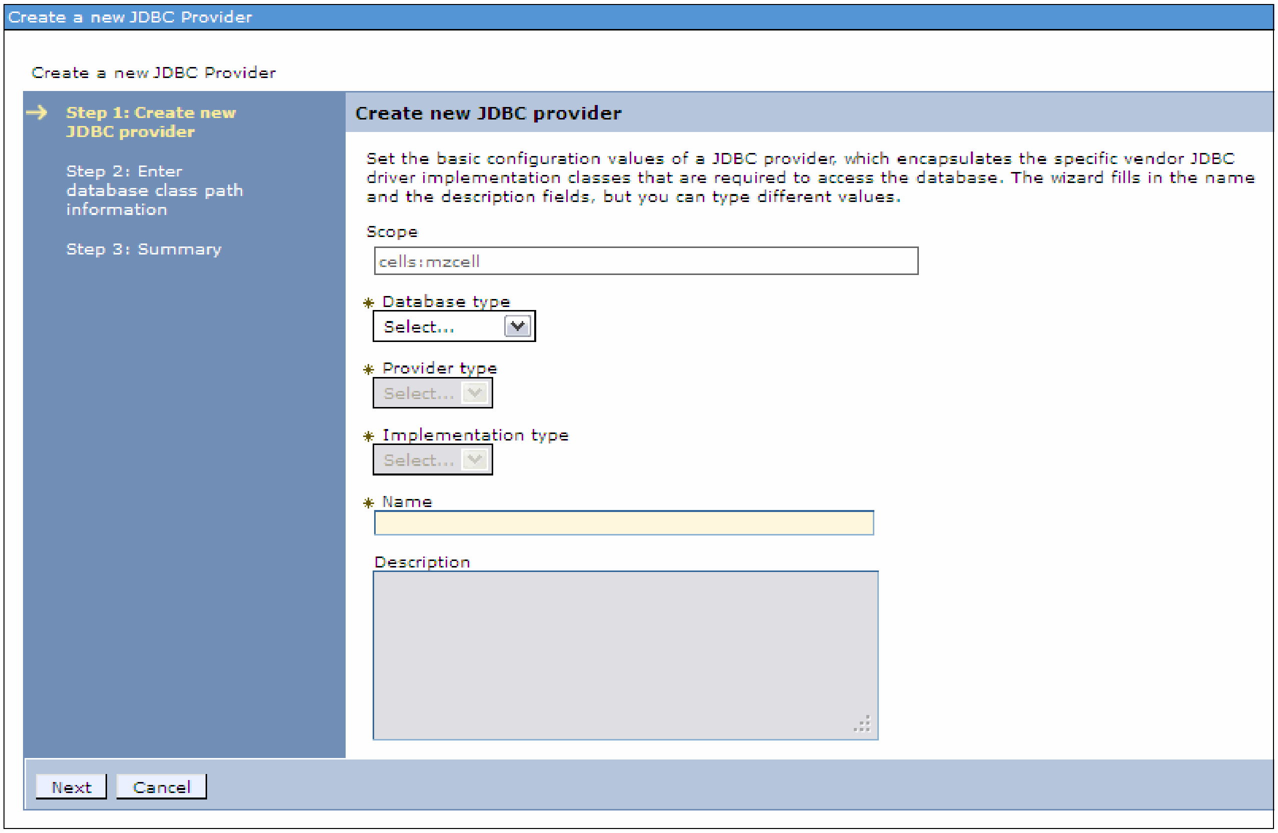


Figure 5-23 JDBC provider that is defined with the cell scope

In this window, complete the following steps, as shown in Figure 5-24:

a. Select **DB2** as the Database type from the drop-down menu.

b. Select **DB2 Universal JDBC Driver Provider** as the provider type from the drop-down menu.

c. Select **Connection pool** data source for the implementation type from the drop-down menu.

After you select the **Connection pool** data source as the implementation type, you can repeat data sources for the third time. Data sources that use this provider support only 1-phase commit processing, unless you use driver type 2 with the application server for z/OS.

If you use the application server for z/OS, driver type 2 uses RRS and supports 2-phase commit processing. The IBM Data Server Driver for JDBC and SQLJ has only one implementation class, which supports both 1-phase and 2-phase commit processing. Hence, it is not necessary to define separate JDBC providers, one each for 1-phase and 2-phase commit processing.

d. Enter a provider name. In this example, use **DB2 Universal JDBC Driver Provider**.

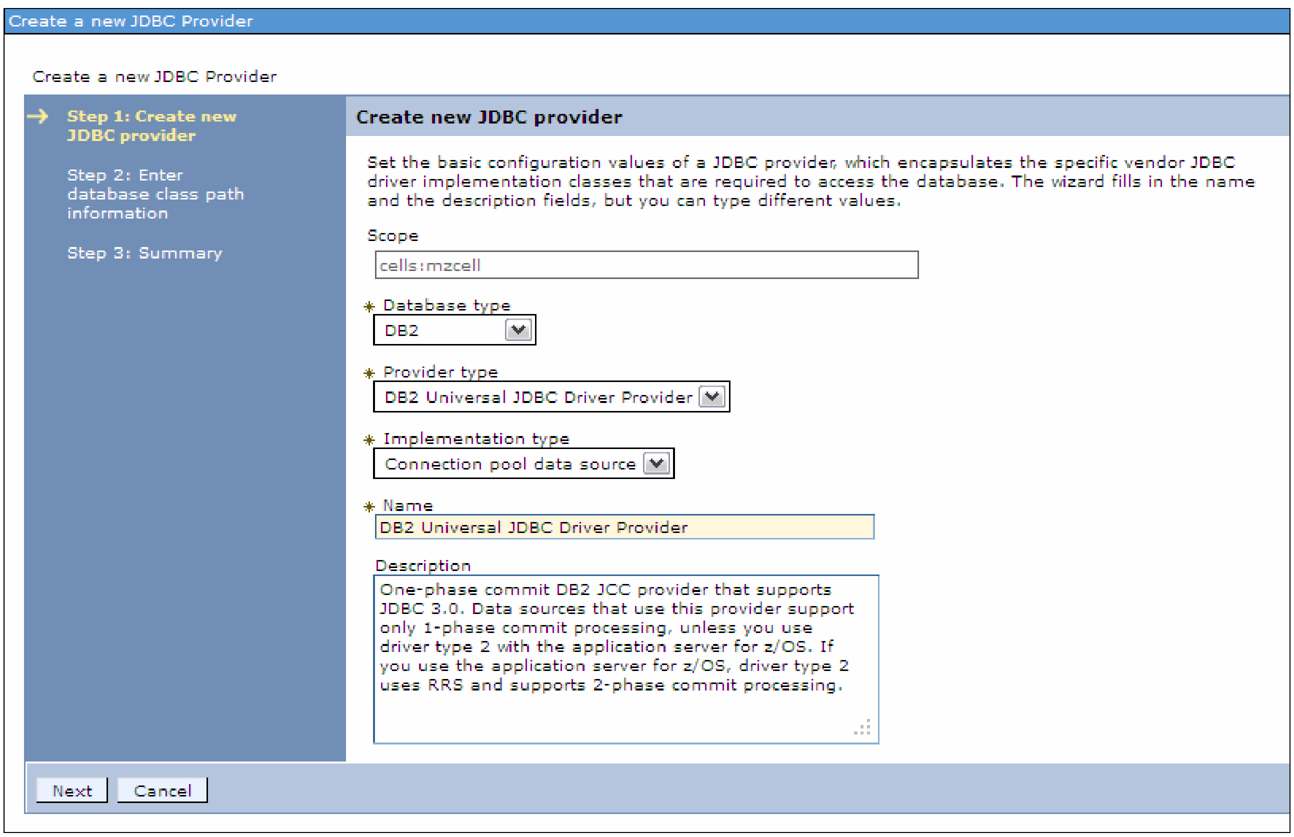


Figure 5-24 New JDBC provider definition

Click **Next**. The window that is shown in Figure 5-25 on page 227 opens.

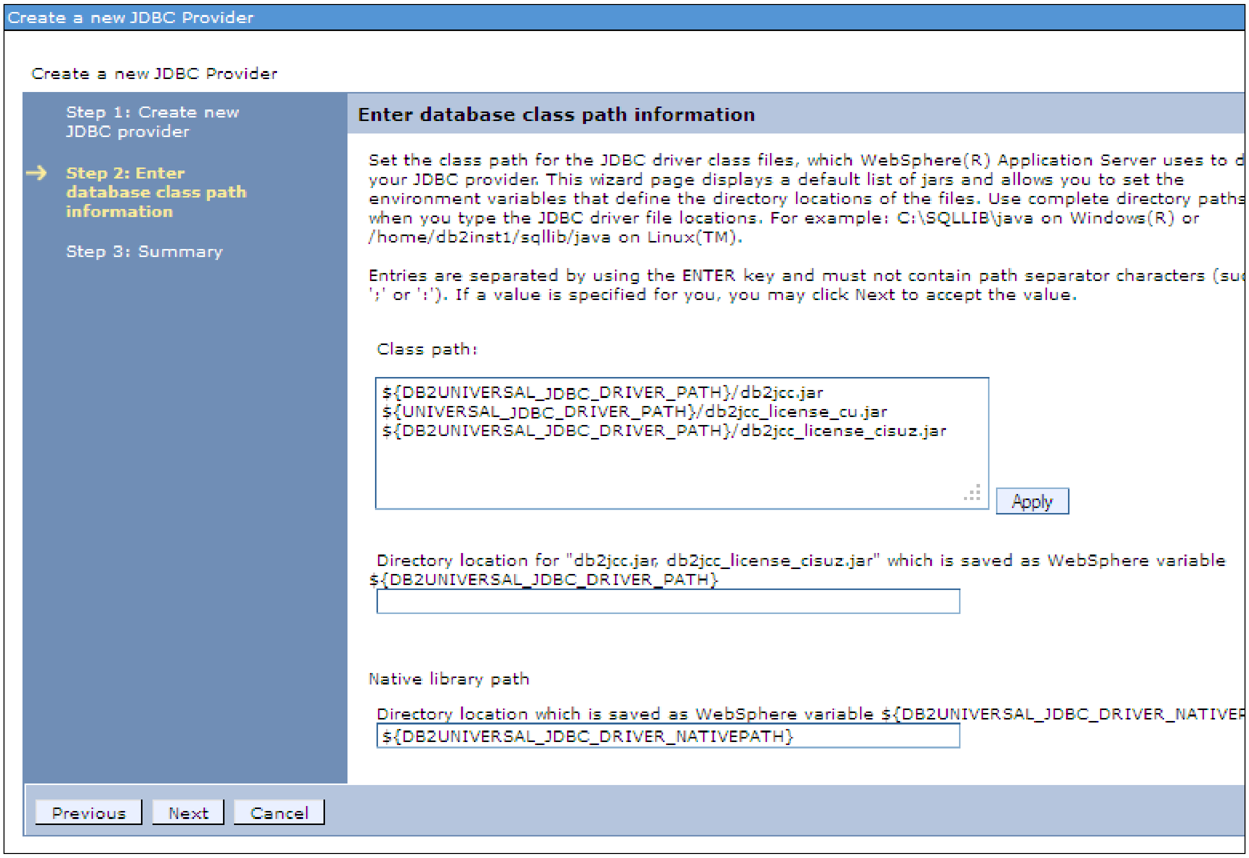


Figure 5-25 Driver classes location

4. The purpose of this window is to define the location of the IBM Data Server Driver for JDBC and SQLJ classes. The one difference with a JDBC type 2 connection on z/OS is the need to define the native library path. All of this is done by using variables.

   The usage of variables provides flexibility so that you can define the location in a single point and use that point for many JDBC providers that can be defined in a WebSphere Application Server. Write down the following variables from this window.

   – DB2UNIVERSAL_JDBC_DRIVER_PATH
   – UNIVERSAL_JDBC_DRIVER_PATH
   – DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH

   We show how to define these variables and their values later in this book.

Click **Next**. The summary window that is shown in Figure 5-26 opens.

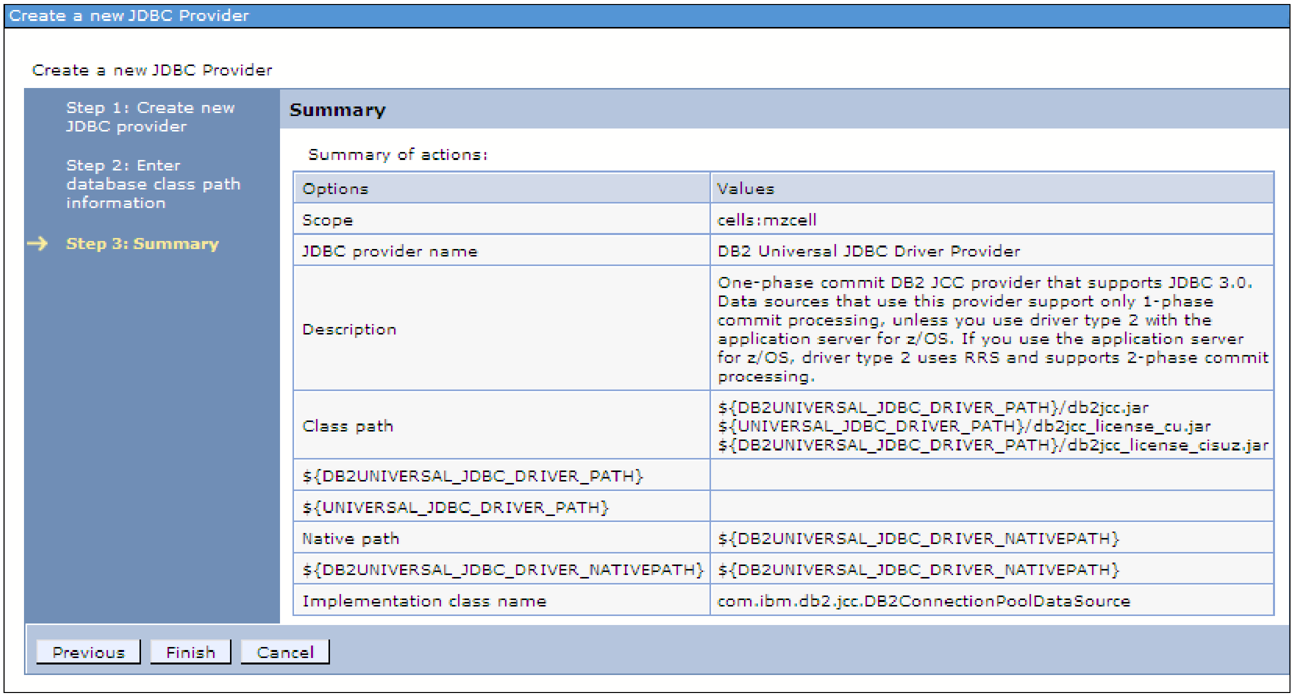


Figure 5-26 Summary of new JDBC provider definition

5. Click **Finish** and then save the changes.

You have created a DB2 Universal JDBC provider that is compatible with type 2 connectivity on z/OS.

5.3.2 Defining environment variables to the location of the IBM Data Server Driver for JDBC and SQLJ classes for JDBC type 2 connectivity

To define environment variables at the location of the IBM Data Server Driver for JDBC and SQLJ classes for JDBC type 2 connectivity, complete the following steps:

1. In the navigation window of the administrative console of WebSphere Application Server, which is shown in Figure 5-21 on page 223, expand **Environment**, as shown in Figure 5-27.

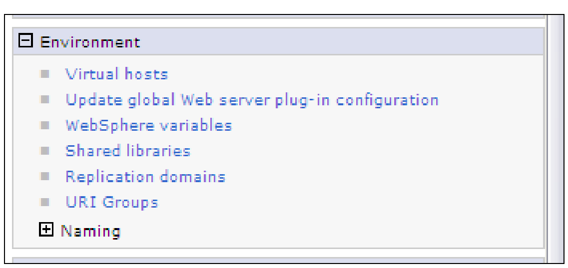


Figure 5-27 Environment window

2. Click **WebSphere variables**. The window that is shown in Figure 5-28 opens.

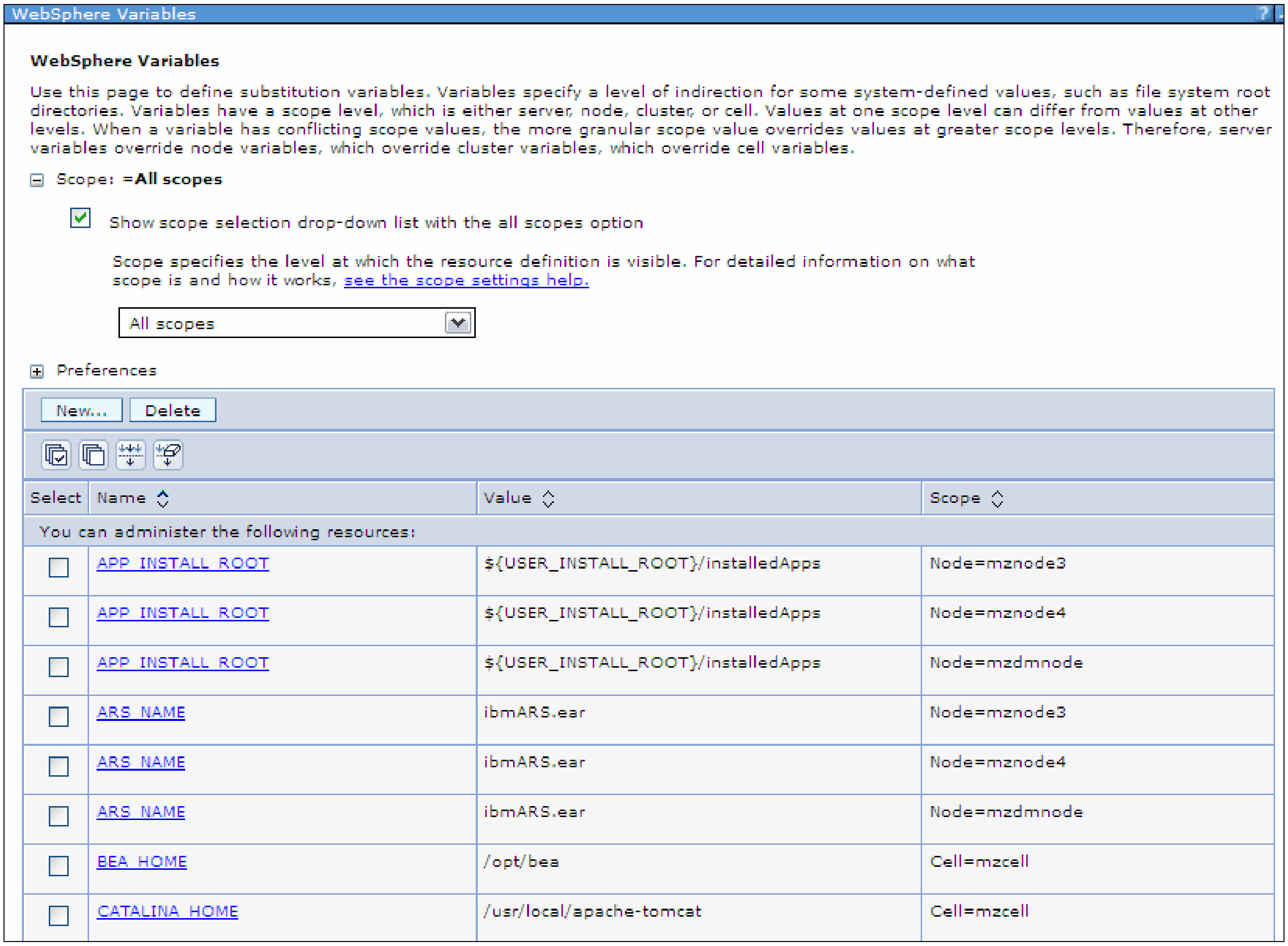


Figure 5-28 List of WebSphere variables

3. By default, the variables are defined to WebSphere Application Server at all scopes. The variables do not have the values defined by default. To see the variables, click the filter icon, as shown in Figure 5-29.

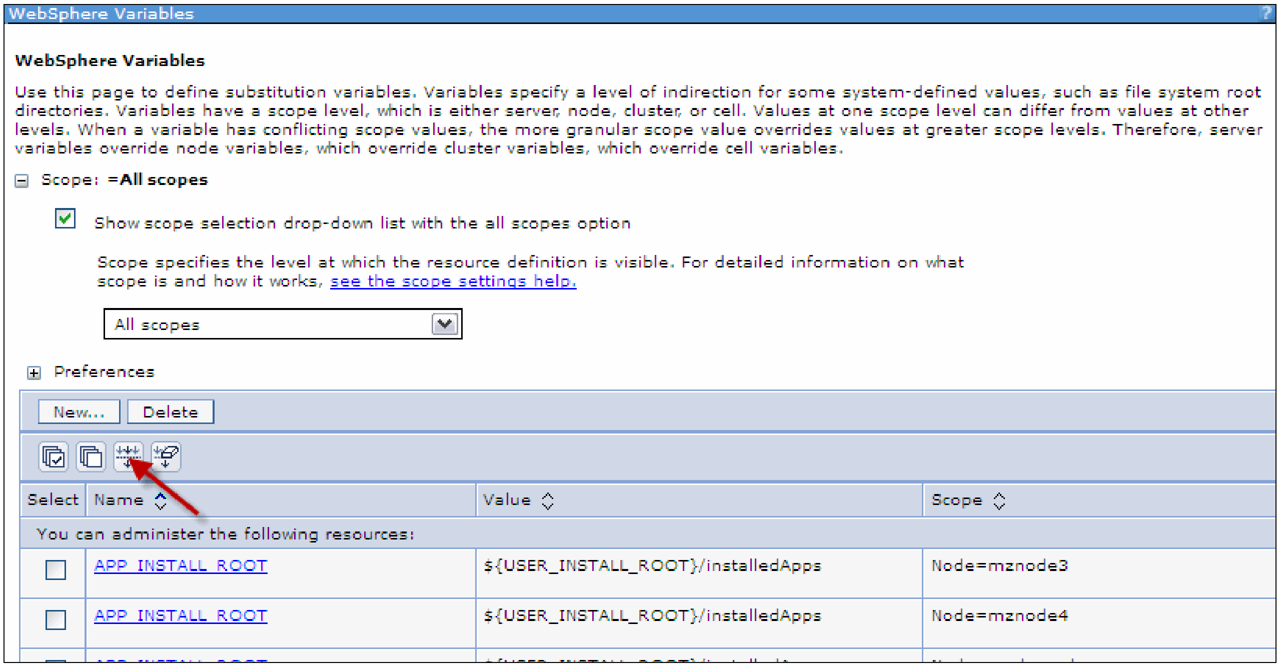


Figure 5-29 Filter variables

4. Enter DB2 in the search terms and click **Go**. A window that shows the default list of variables opens, as shown in Figure 5-30.

Figure 5-30 List of available variables

The variables are defined at all possible scopes in the cell. Pick the appropriate scope. In this example, pick the DB2UNIVERSAL_JDBC_DRIVER_PATH variable as the cell scope, as shown in Figure 5-31.

Figure 5-31 Variable cell scope mzcell

5. Double-click the variable name. The window that is shown in Figure 5-32 opens, where no value is set for the Type 2 driver.

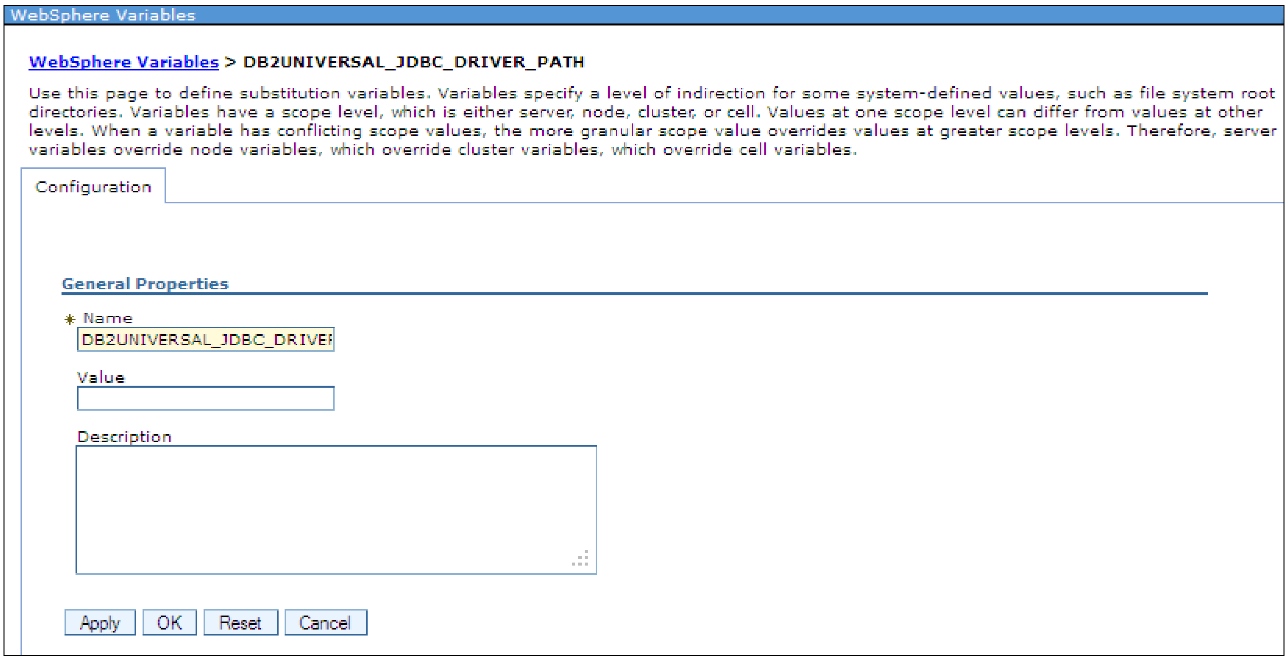


Figure 5-32 DB2UNIVERSAL_JDBC_DRIVER_PATH variable

6. Enter the location of the IBM Data Server Driver for JDBC and SQLJ classes in the value text box. In this example, enter /usr/lpp/db2/d0zg/jdbc/classes, as shown in Figure 5-33.

Figure 5-33 Location of the driver classes

7. Click **Apply** and then save the changes. Repeat the same steps for the UNIVERSAL_JDBC_DRIVER_PATH variable.
8. For JDBC type 2 connectivity, you must define the path of the native libraries by assigning a value to DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH variable, which points to the location of the native libraries.

   Double-click the DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH variable and the window that is shown in Figure 5-34 on page 233 opens. Enter the location of the native libraries, which in this example is `/usr/lpp/db2/d0zg/jdbc/lib/`.

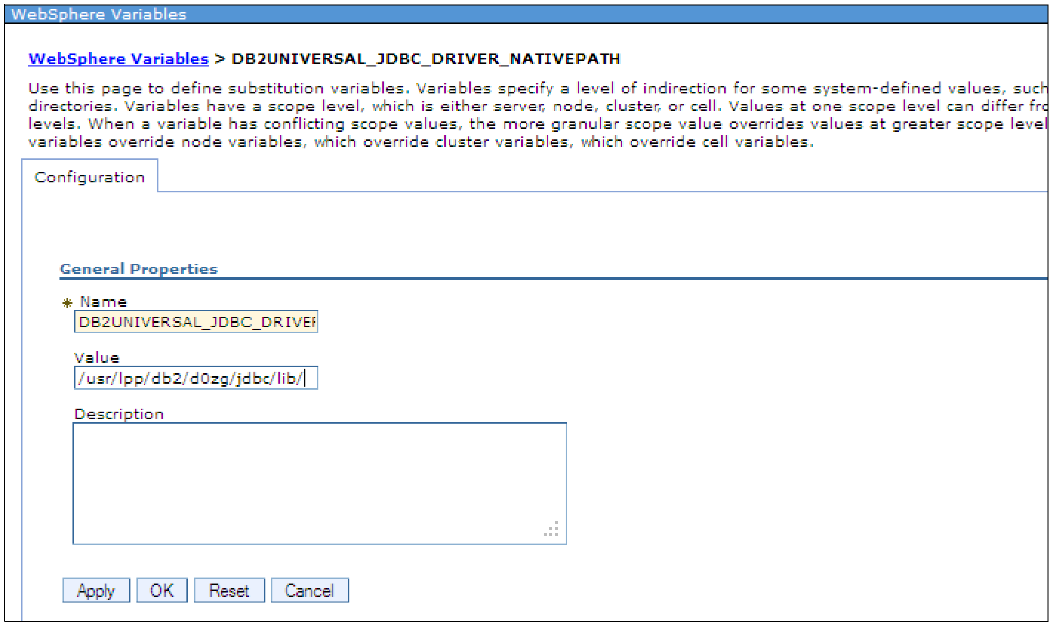


Figure 5-34 Location of the native libraries

9. Click **Apply** and save the changes.

5.3.3 Defining a JDBC type 2 data source

To define a JDBC type 2 data source, complete the following steps:

1. In the navigation window of the administrative console of the WebSphere Application Server, expand **Resources** and click **Data Sources**, as shown in Figure 5-35.

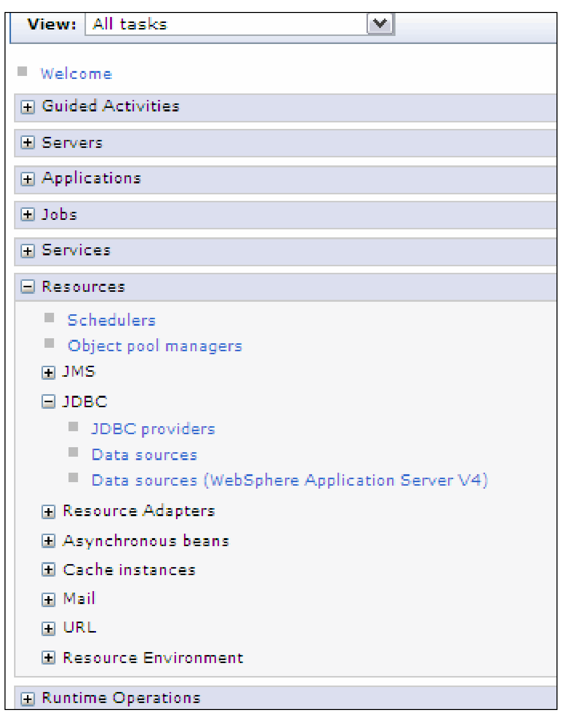


Figure 5-35 Administration window for WebSphere

The window that is shown in Figure 5-36 opens, which shows a list of the existing JDBC data sources that are defined in your environment.

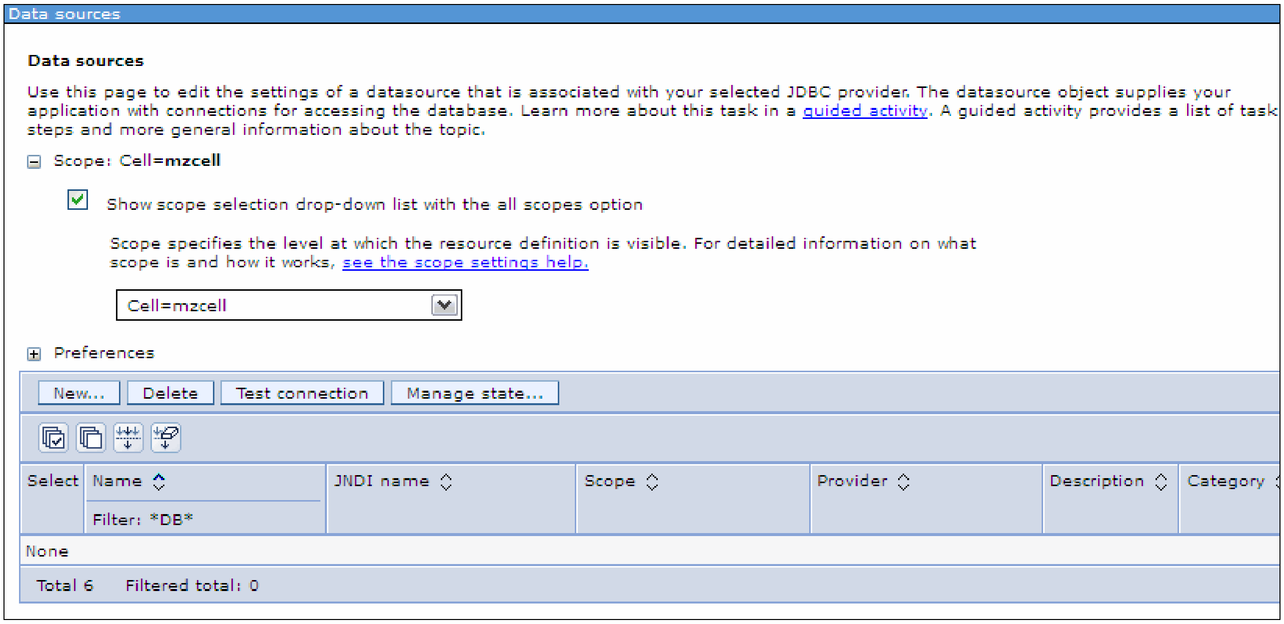


Figure 5-36 List of JDBC data sources

In this example, select the cell scope and click **New**. The window that is shown in Figure 5-37 opens.

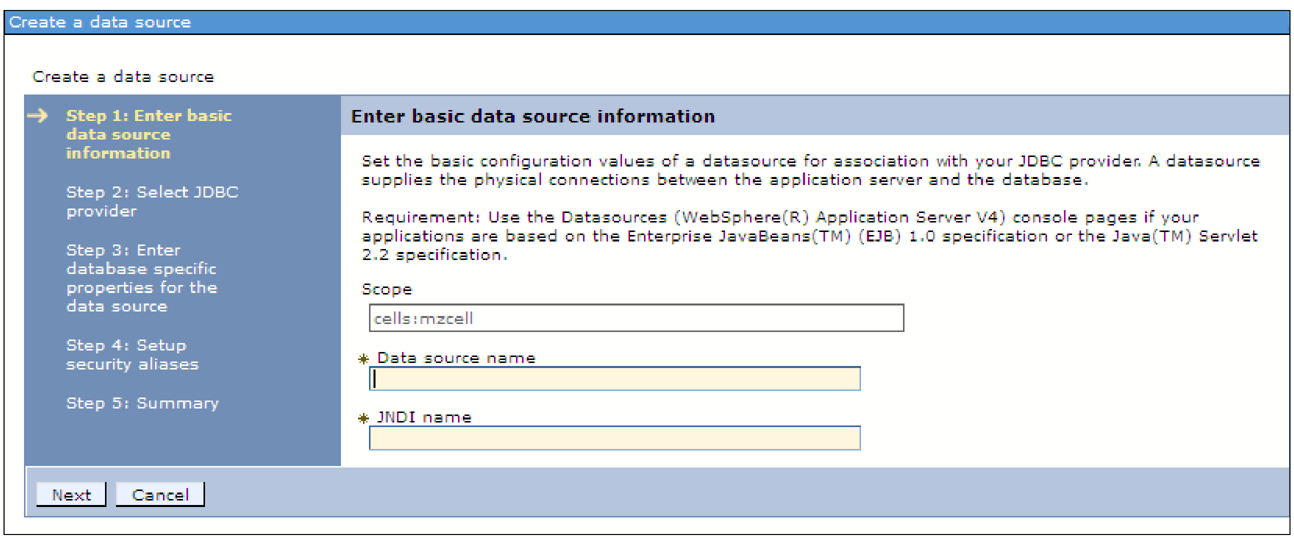


Figure 5-37 Window for entering data source information

2. In this window, enter a name for the data source and the JNDI names. In this example, enter TradeDatasourceType2 for the data source name and jdbc/Trade DataSourceType2 for the JNDI name, as shown in Figure 5-38.

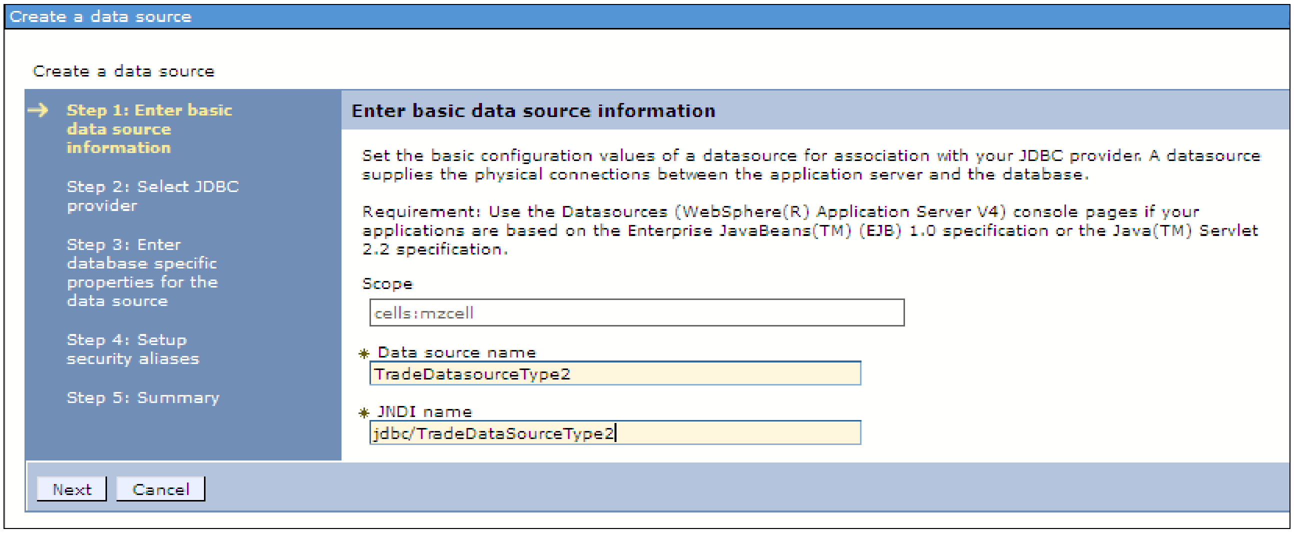


Figure 5-38 Defining the data source and JNDI names

Click **Next** and the window that is shown in Figure 5-39 opens.

3. In this window, select the DB2 Universal JDBC Driver Provider that was created earlier because you need a JDBC type 2 connection.

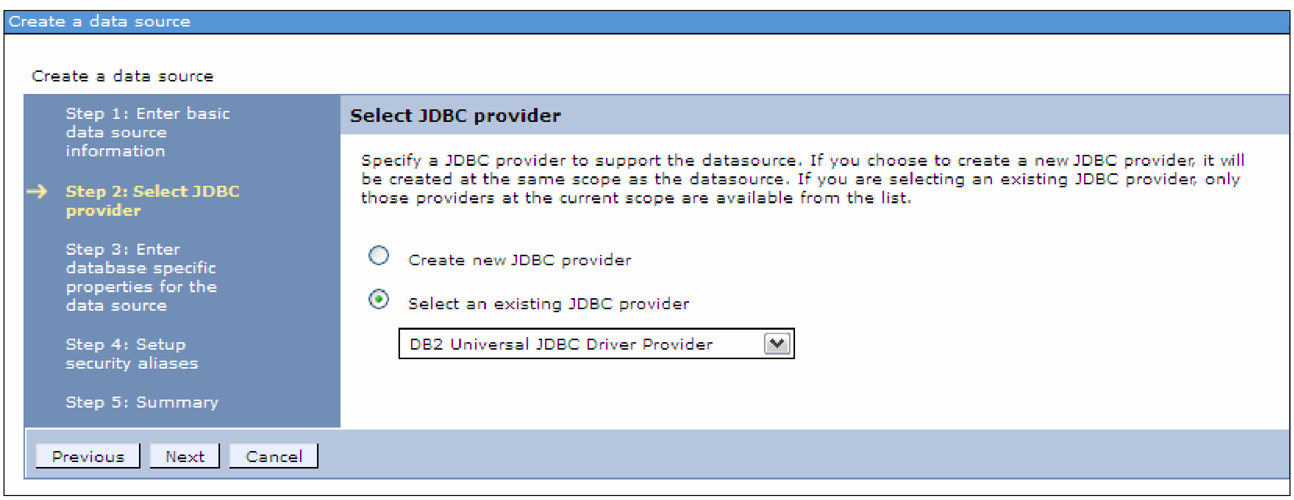


Figure 5-39 Selecting the JDBC type 2 Driver

Click **Next**. The window that is shown in Figure 5-40 opens.

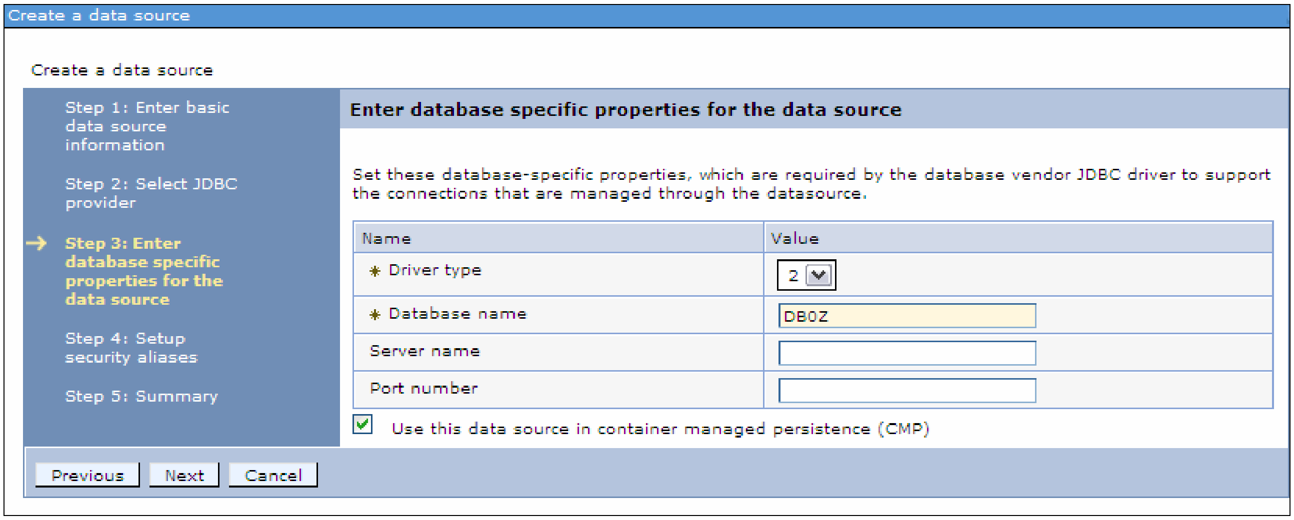


Figure 5-40 Database properties

4. In this window, enter the following values:
   - For driver type, select **2** from the drop-down menu, which directs WebSphere Application Server to use a JDBC type 2 connection to the database.
   - The DB2 for z/OS location name for the name of the database. The location name is *very* specific to DB2 for z/OS. In this example, enter DB0Z.
   - The server name and port number should be blank.

   You can obtain the values for the entries that are shown in Figure 5-40 by running **-DISPLAY DDF**. Example 5-2 shows the command and the values in the example z/OS system.

Example 5-2 - DISPLAY DDF command to verify DB2 definitions

```
DSNL080I  -D0Z2 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I DB0Z               -NONE             -NONE
DSNL084I TCPPORT=39000 SECPORT=0     RESPORT=39003 IPNAME=IPDB0Z
DSNL085I IPADDR=::9.12.4.153
DSNL086I SQL    DOMAIN=d0zg.itso.ibm.com
DSNL086I RESYNC DOMAIN=d0z2.itso.ibm.com
DSNL087I ALIAS              PORT  SECPORT STATUS
DSNL088I D0Z2               0     0       STARTD
DSNL089I MEMBER IPADDR=::9.12.4.142
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

Click **Next** and the window that is shown in Figure 5-41 on page 237 opens.

5. In this window, enter the authentication alias that should be used by WebSphere Application Server when it connects to DB2 for z/OS. This authentication alias must be defined beforehand. You have two options:

   – Component-managed Authentication Alias:

     This is the user ID/password that is used to access DB2 with component-managed security.

   – Container-managed Authentication Alias:

     This is the user ID/password that is used to access DB2 with container-managed security.

   These aliases are known as *J2C aliases*. They can be defined by using the administration console.

   By default, the user ID that WebSphere Application Server on z/OS runs under is used. This is possible only for a JDBC type 2 connection, which means that the user ID under which WebSphere Application Server runs under must have the appropriate access to the DB2 objects that are used in the application that uses this data source.

   We can also override that user ID and provide an authentication alias, which is then used.

   In this example, use the TradeDataSourceAuthData authentication alias.

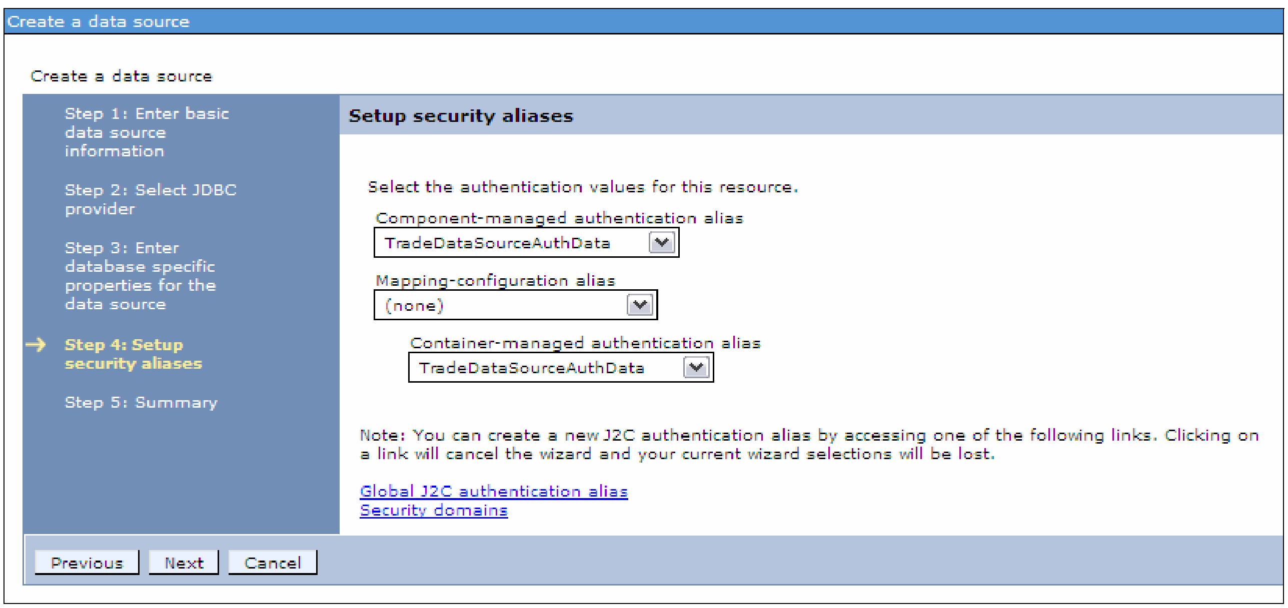


Figure 5-41 Security aliases

Click **Next**, which opens a summary window that shows all the different values you have set so far, as shown in Figure 5-42.

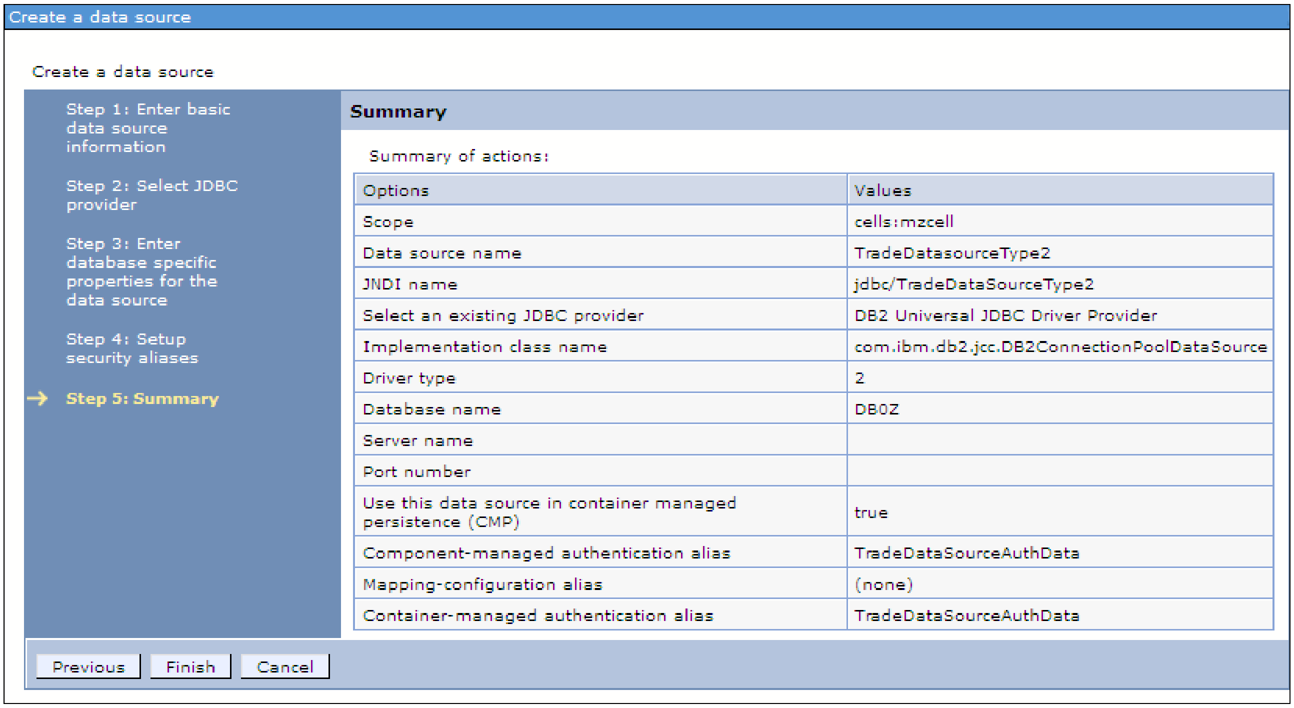


Figure 5-42 Summary of the type 2 Driver setup

6. Click **Finish** and then save the changes.

You have a JDBC type 2 data source that is defined.

5.3.4 Configuring a subsystem ID on the data source

WebSphere Application Server on z/OS, when it connects to DB2 for z/OS using a JDBC type 2 connection, does not connect to the DB2 using the location name that is provided by the connection application. It uses the value that is specified by a data source custom property called subsystem ID (*ssid*).

This property can be set to specify the DB2 subsystem identifier (not the DB2 location name) if the DB2 system is not part of a data sharing group. If DB2 is part of a data sharing group, then specifying the group attach name as the value is recommended because if customers have multiple members of a data sharing group in the same LPAR, specifying the group attach name as the value for the ssid property allows type 2 connections to fail over to the second DB2 member of the same data sharing group in the same LPAR if the one of the DB2 members fails.

The only time when ssid should be used instead of a group attach name is if there is a requirement that the WebSphere Application Server connect to only a specific DB2 subsystem.

JDBC type 2 connections do not workload balance between multiple DB2 members of a data sharing group in a single LPAR. The connections randomly pick one of the DB2 members to use for all connections and then, if that DB2 member fails, fail over to the second DB2 member of the same data sharing group in the same LPAR. This situation happens only if you specify a group attach name as the value of the ssid data source custom property in WebSphere Application Server.

If you use the ssid property when it is not provided, then the driver uses the ssid that it finds in the DSNHDECP load module. You load DSNHDECP by using the search sequence that specified in the STEPLIB environment variable or the //STEPLIB DD name concatenation. If that DSNHDECP load module does not accurately reflect the correct subsystem, or multiple subsystems are using a generic DSNHDECP, then there might be problems in connecting to DB2.

Another reason to use the ssid property for JDBC type 2 connections to DB2 from WebSphere Application Server on z/OS is so that a single WebSphere Application Server can connect to multiple DB2 subsystems. Then, different applications that are deployed in the same WebSphere Application Server can connect to different DB2 subsystems in the same LPAR if they use different data sources and the ssid is set on each data source.

To configure the ssid on the data source, complete the following steps:

1. In the navigation window of the administrative console of the WebSphere Application Server, expand **Resources**, and click **Data sources**, as shown in Figure 5-43.

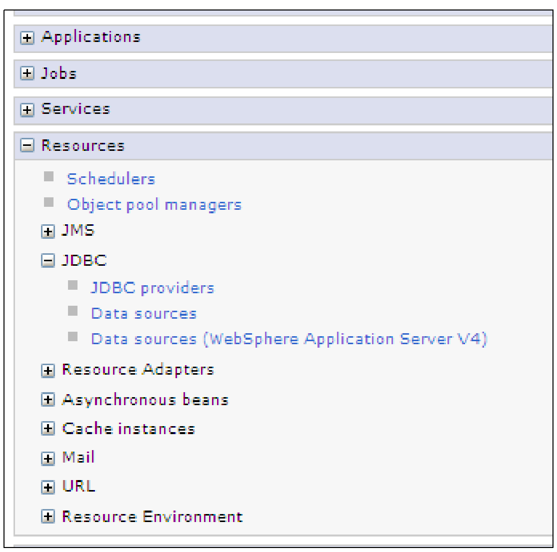


Figure 5-43 Administrative console of the WebSphere Application Server

2. The window that is shown in Figure 5-44 opens. This window shows a list of existing JDBC data sources that are defined in your environment. Click the **TradeDataSourceType2** JDBC type 2 data source.

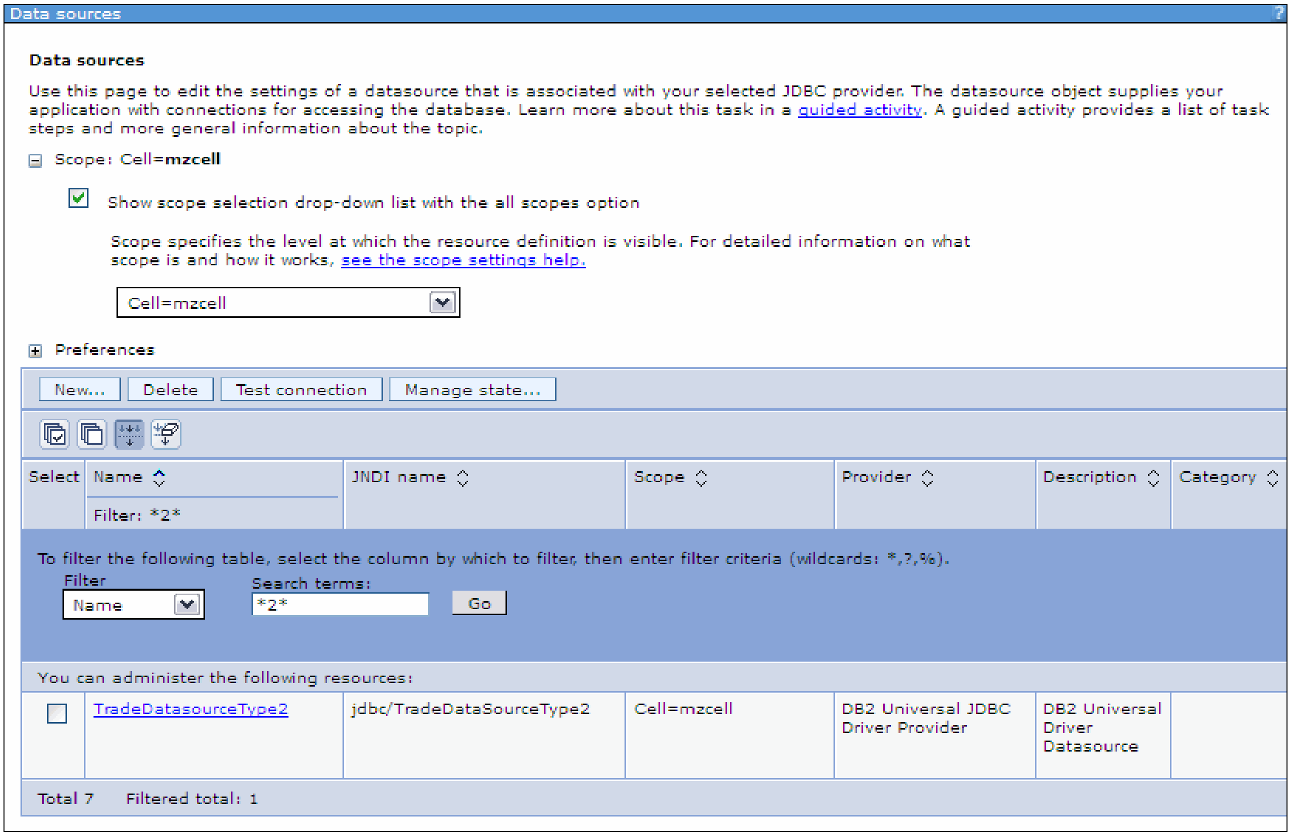


Figure 5-44 JDBC type 2 data source selection

3. The window that is shown in Figure 5-45 opens. Click **Custom properties** under the Additional properties section. The window that is shown in Figure 5-46 opens, which lists all the custom properties that are available to the data source.

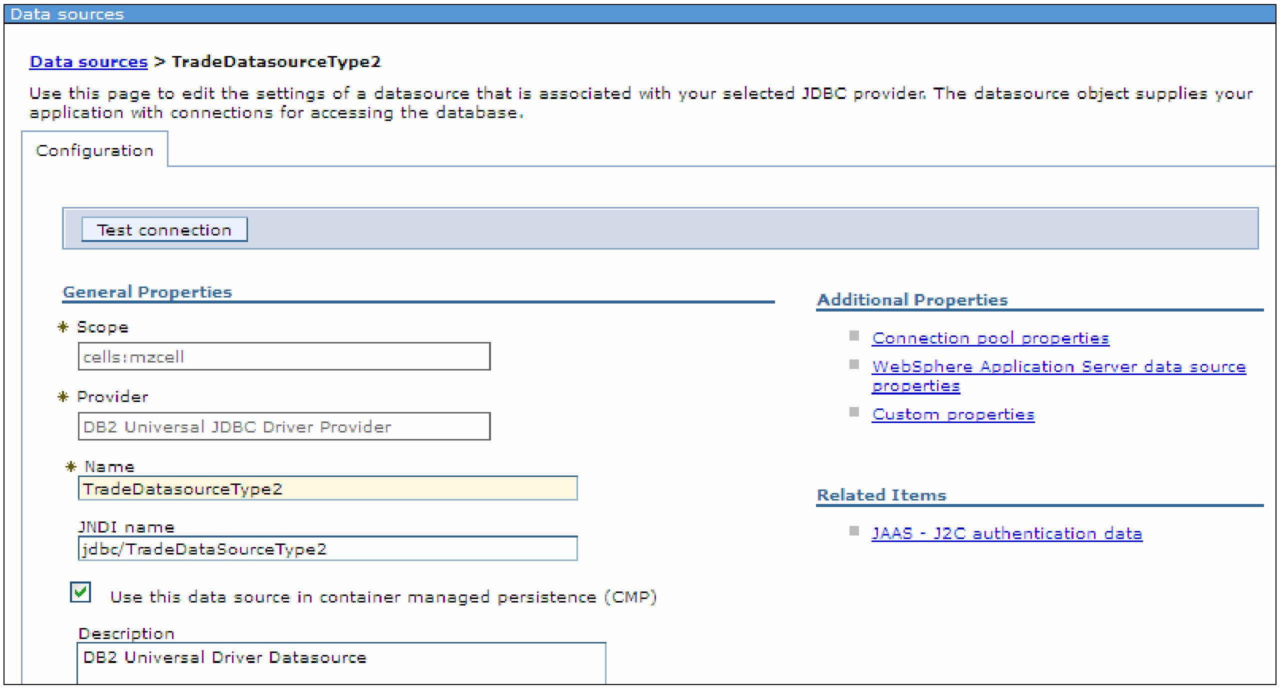


Figure 5-45 Selecting Custom properties

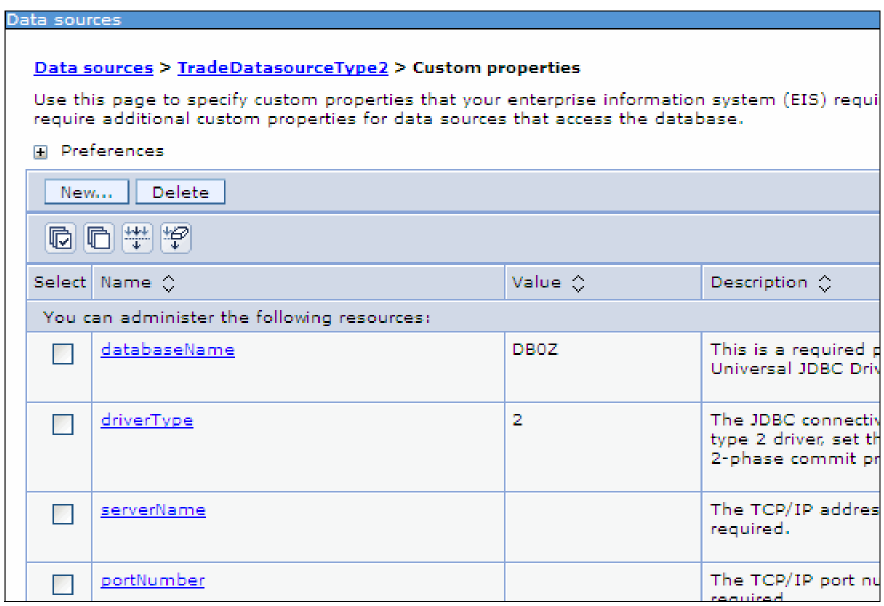


Figure 5-46 List of custom properties

4. The property ssid is not defined by default. You can define it. Click **New** and a new window opens. For the ssid, enter the group attach name or the subsystem ID. In this example, enter the group attach name of the DB2 data sharing group D0ZG.

The group attach name can be obtained by running **DISPLAY GROUP**. Example 5-3 shows the output from that command and the group attach name.

Example 5-3 DISPLAY GROUP command to verify that the group attach name

```
DSN7100I  -D0Z2 DSN7GCMD
 *** BEGIN DISPLAY OF GROUP(DB0ZG   ) CATALOG LEVEL(101) MODE(NFM )
                   PROTOCOL LEVEL(2)  GROUP ATTACH NAME(D0ZG)
 --------------------------------------------------------------------
 DB2                                    DB2 SYSTEM    IRLM
 MEMBER   ID  SUBSYS CMDPREF   STATUS   LVL NAME      SUBSYS IRLMPROC
 -------- --- ----   --------  -------- --- --------  ----   --------
 D0Z1       1 D0Z1   -D0Z1     ACTIVE   101 SC63      I0Z1   D0Z1IRLM
 D0Z2       2 D0Z2   -D0Z2     ACTIVE   101 SC64      I0Z2   D0Z2IRLM
 --------------------------------------------------------------------
 SCA   STRUCTURE SIZE:     8192 KB, STATUS= AC,   SCA IN USE:     4 %
 LOCK1 STRUCTURE SIZE:     8192 KB
 NUMBER  LOCK ENTRIES:     2097152
 NUMBER  LIST ENTRIES:        9324, LIST ENTRIES  IN USE:           7
 SPT01 INLINE LENGTH:       32138
 *** END DISPLAY OF GROUP(DB0ZG   )
 DSN9022I  -D0Z2 DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION
 ***
```

5. Enter the group name, as shown in Figure 5-47.

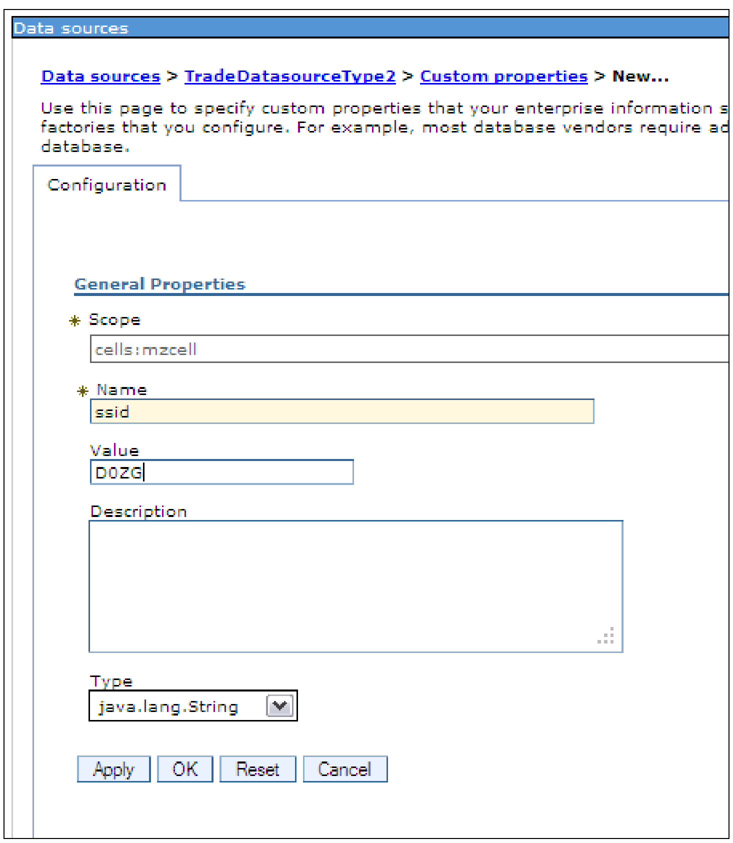


Figure 5-47 General properties definition

6. Click **Apply** and then save the changes.

Linking to the DB2 libraries

WebSphere Application Server on z/OS, when it is configured to use a JDBC type 2 connection to DB2 for z/OS, also requires access to three DB2 libraries:

- `DB2xx.SDSNEXIT`
- `DB2xx.SDSNLOAD`
- `DB2xx.SDSNLOD2`

WebSphere Application Server can access these libraries in three ways:

- The libraries can be placed in the LINKLIST of the z/OS operating system.
- The libraries can be added to the JCL of the startup procedure of the Application Server Servant by adding a STEPLIB.
- The libraries can be added by modifying the `STEPLIB` environment variable to include the DSNEXIT, DSNLOAD, and DSNLOD2 libraries.

This example uses the STEPLIB approach and adds the libraries to the servant region proclibs of the Deployment Manager and the WebSphere Application Server, as shown in Example 5-4. You must add to it to the Deployment Manager to test the connection.

Example 5-4 Application Server Servant libraries

```
//STEPLIB   DD DSN=DB0ZT.SDSNEXIT,DISP=SHR
//          DD DSN=DB0ZT.SDSNLOAD,DISP=SHR
//          DD DSN=DB0ZT.SDSNLOD2,DISP=SHR
```

You have completed all the required steps to configure WebSphere Application Server for JDBC type 2 access to DB2.

5.4 Configuring WebSphere Application Server for sysplex workload balancing

The section shows how to enable sysplex workload balancing for a JDBC type 4 connection to DB2. This feature is available for both XA and non-XA data sources. *This feature is not available on JDBC type 2 connections*. This example is for an XA data source.

To configure WebSphere Application Server for sysplex workload balancing, complete the following steps:

1. In the navigation window of the administrative console of the WebSphere Application Server, expand **Resources** and click **Data Sources**, as shown in Figure 5-48.

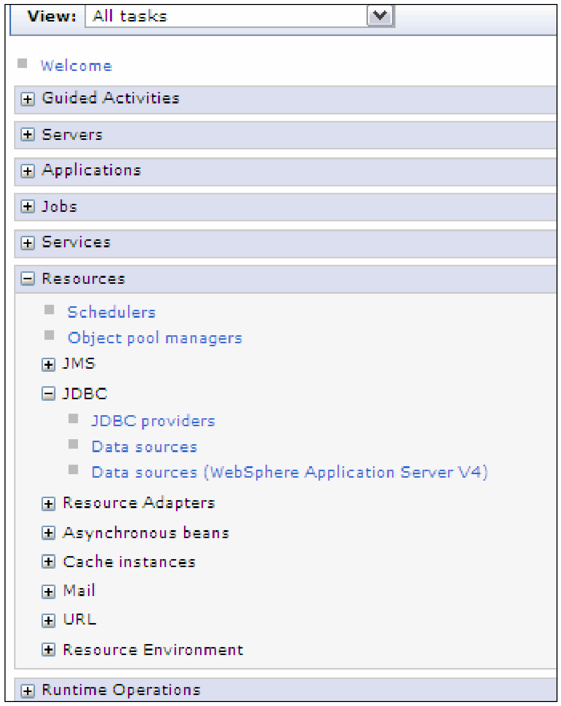


Figure 5-48 Administrative console of the WebSphere Application Server

The window that is shown in Figure 5-49 on page 245 opens. This window shows a list of existing JDBC data sources that are defined in your environment.

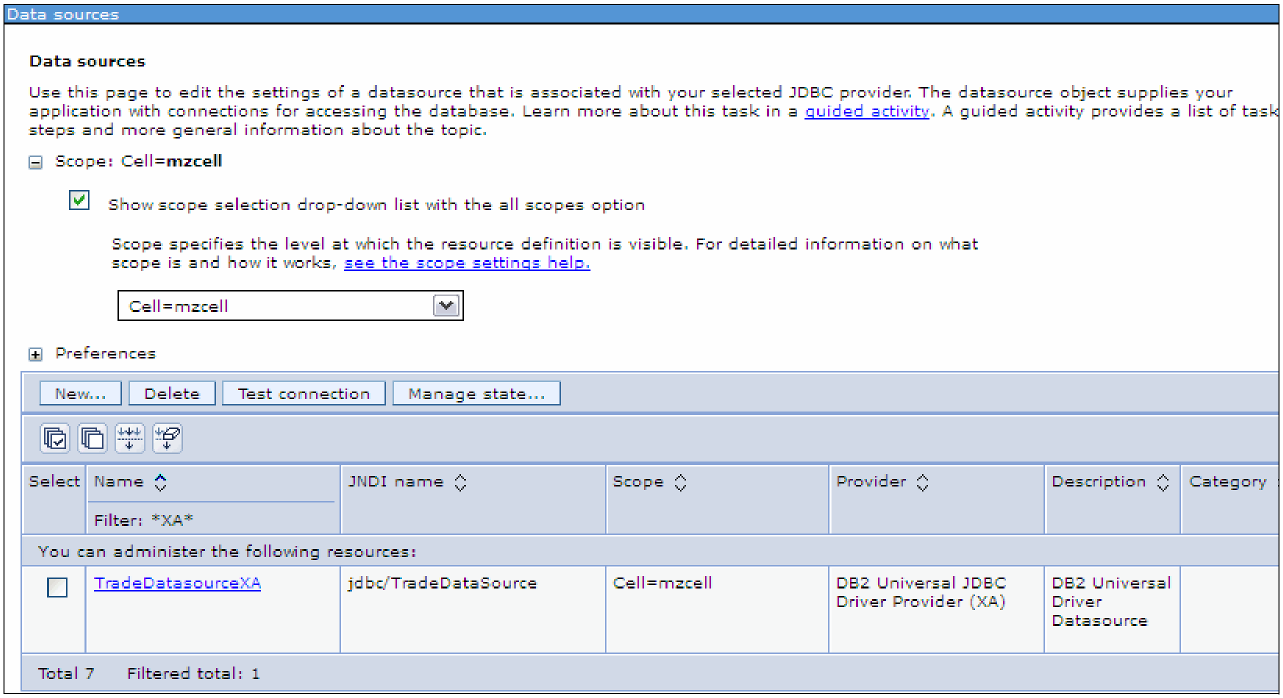


Figure 5-49 List of existing JDBC data sources

2. Click **TradeDatasourceXA** and the window that is shown in Figure 5-50 opens. This window lists all the custom properties that are available to the data source.

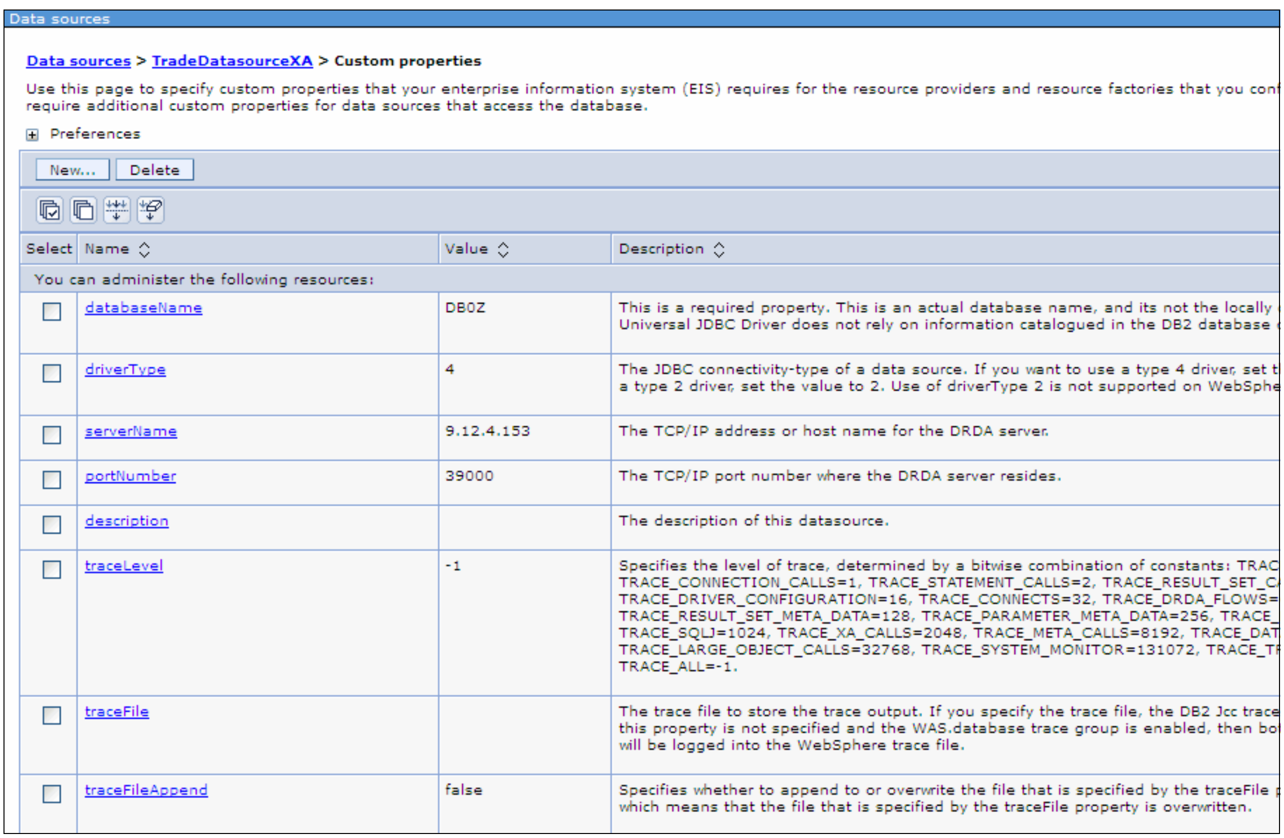


Figure 5-50 List of custom properties that are available to the data source

3. The enableSysplexWLB property that is required to enable sysplex workload balancing is not present by default. You can add this property by clicking **New**. The window that is shown in Figure 5-51 opens.

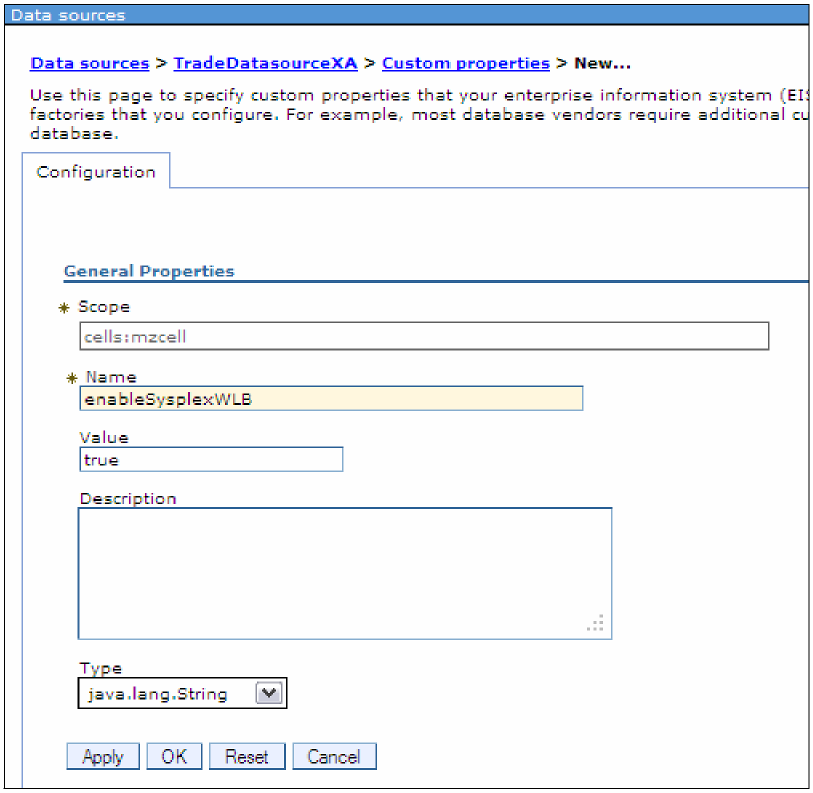


Figure 5-51 Adding the enableSysplexWLB property

Complete the following steps:

a. Enter enableSysplexWLB for the property name.

b. Enter true for the value.

4. Click **Apply** and then save the changes. The data source is now enabled for sysplex workload balancing.

5.5 Configuring client information in WebSphere Application Server

As more applications are written in Java that access data in DB2 for z/OS, the typical challenges that are faced by DBAs are that they often do not know which application the SQL statement comes from. To learn this information, you should set *client information* on the connection. This client information is passed to DB2 for z/OS and can be used to correlate requests. WebSphere Application Server (on all platforms) and DB2 (on all platforms) support this feature. The JDBC 4.0 specification adopted this functionality by providing an API

There are two places in WebSphere Application Server where you can set the client information properties.

- Data source custom properties
- Resource Reference extended data source properties

These properties can also be set in the application by using the JDBC 4.0 API setClientInfo. This action requires an application change and is typically required in situations in which client information settings can be determined and set only at run time. In all other situations, use data source custom properties or Resource Reference extended data source properties for easier system administration.

The following sections demonstrate how to set these properties in WebSphere Application Server. The approach is the same regardless of whether the application uses a JDBC type 2 or 4 connection (XA or non-XA).

5.5.1 Setting client information on a data source

To set client information on a data source, complete the following steps:

1. In the navigation window of the administrative console of the WebSphere Application Server, expand **Resources** and click **Data Sources**, as shown in Figure 5-52.

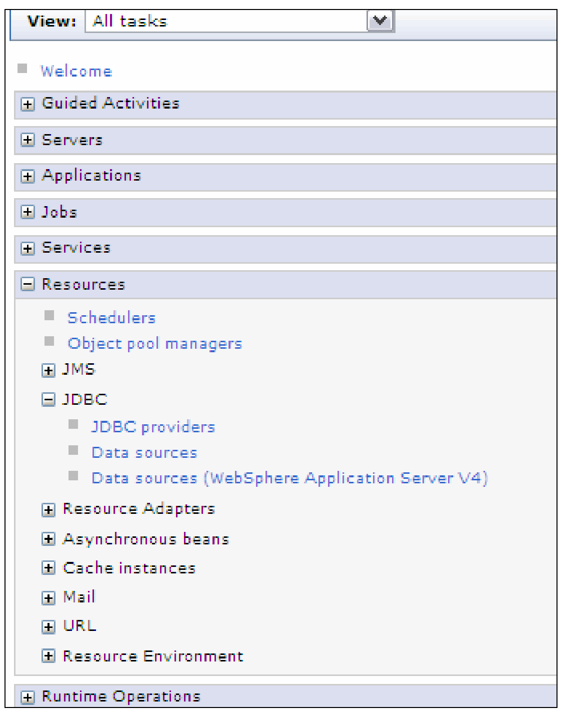


Figure 5-52 Administrative console of the WebSphere Application Server

The window that is shown in Figure 5-53 opens. This window shows a list of existing JDBC data sources that are defined in your environment.

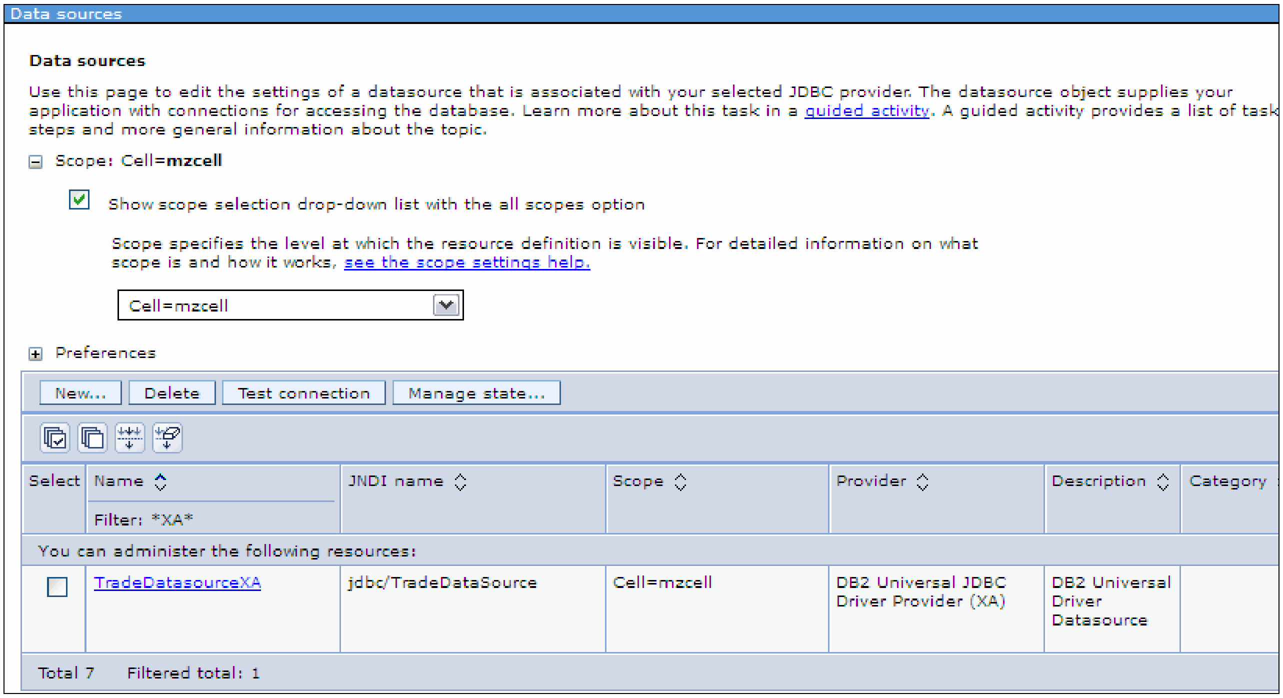


Figure 5-53 List of existing JDBC data sources

2. Click **TradeDatasourceXA** and the window that is shown in Figure 5-54 opens.

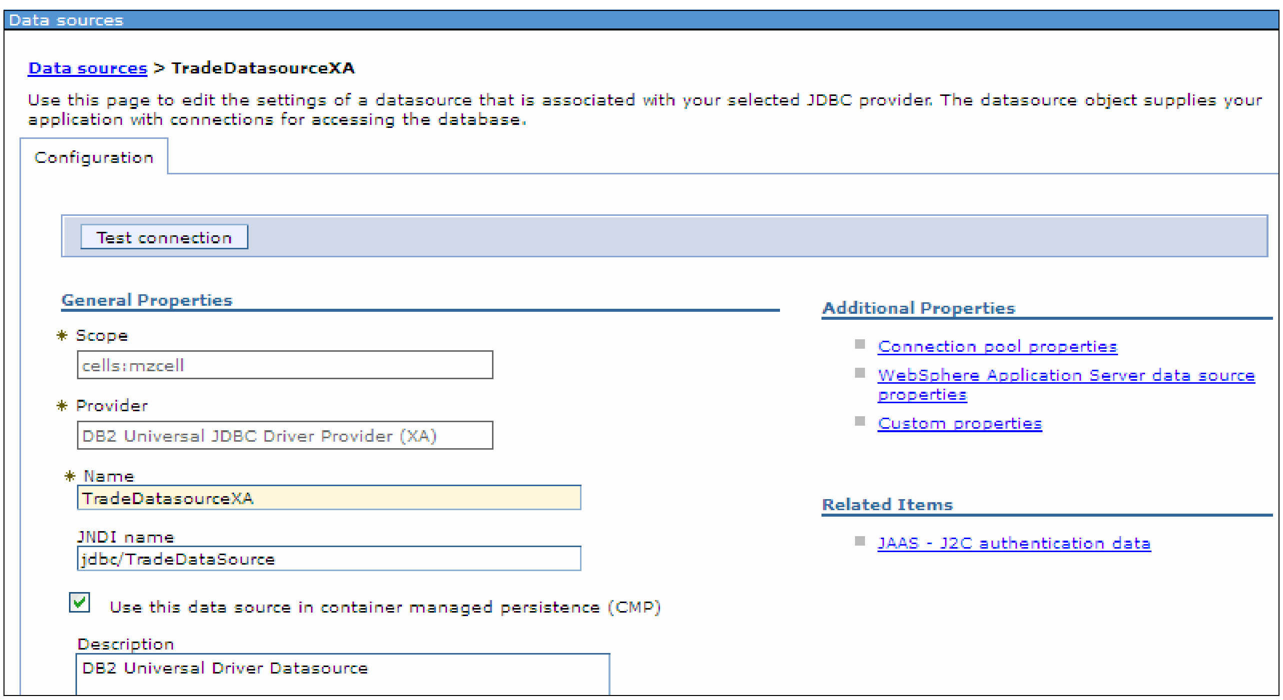


Figure 5-54 TradeDatasourceXA data source is accessed

3. Click **Custom properties**. The panel that opens lists all the custom properties that are available. By default, the properties that are available are the ones that are shown in Figure 5-55.

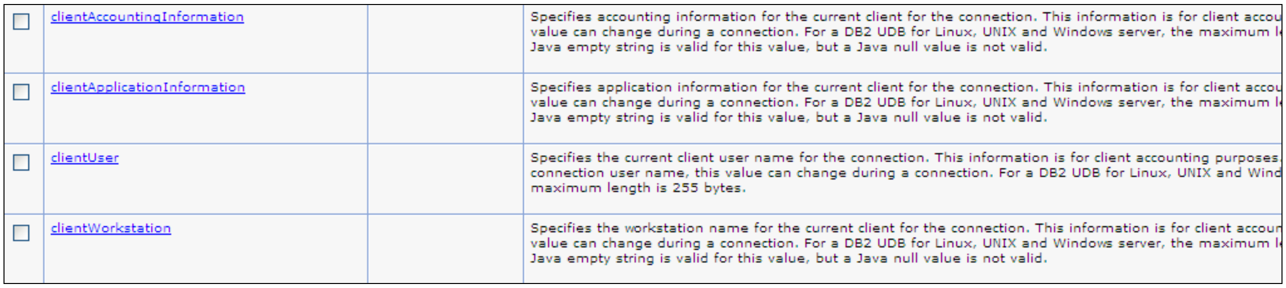


Figure 5-55 Available properties

4. By default, these properties do not have any values that are specified. You can set all the properties or any combination of them. In this example, set values for all of them. For example, to set a value for clientAccountingInformation, click the **clientAccountingInformation** property. The window that is shown in Figure 5-56 opens.

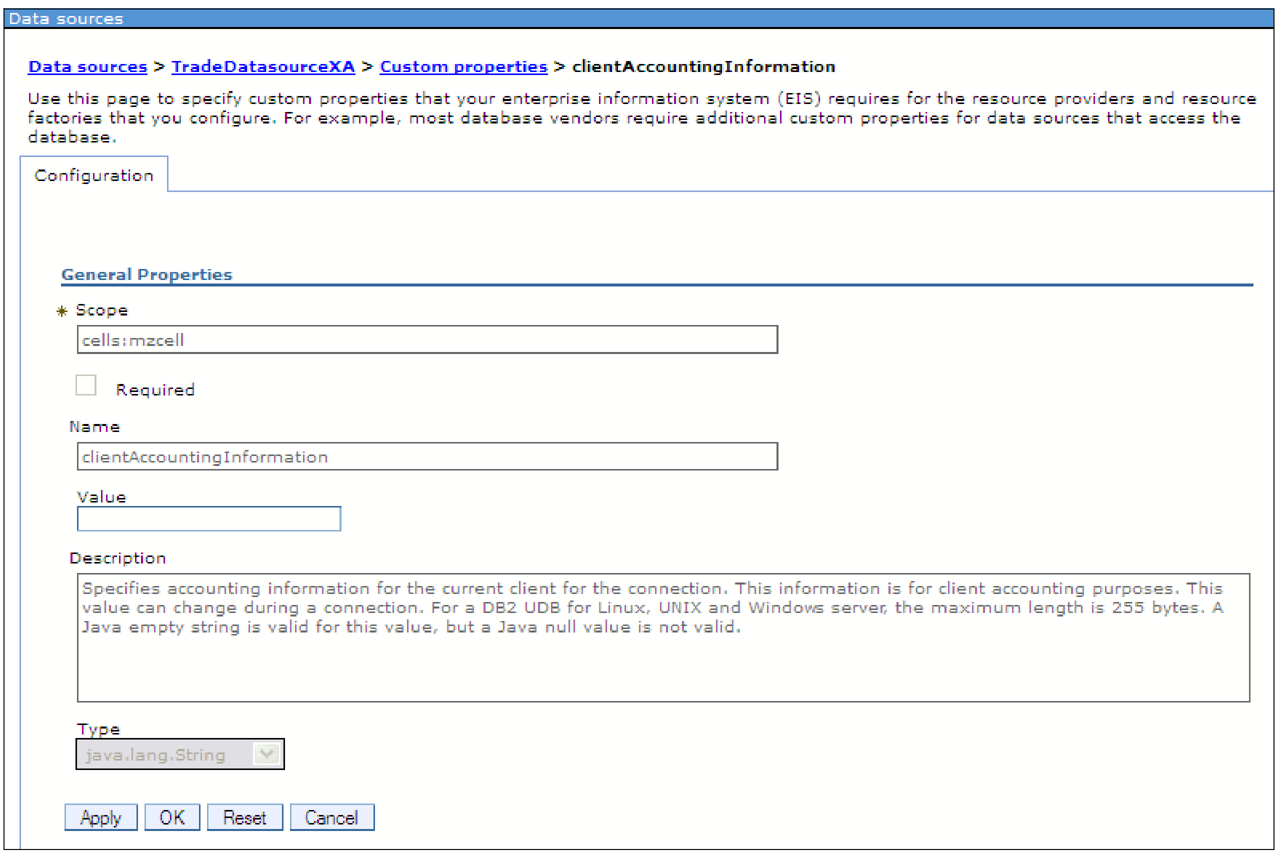


Figure 5-56 Set a value for clientAccountingInformation

5. Enter a string that identifies the application. In this example, use TradeClientAccountingInformation as the value, as shown in Figure 5-57.

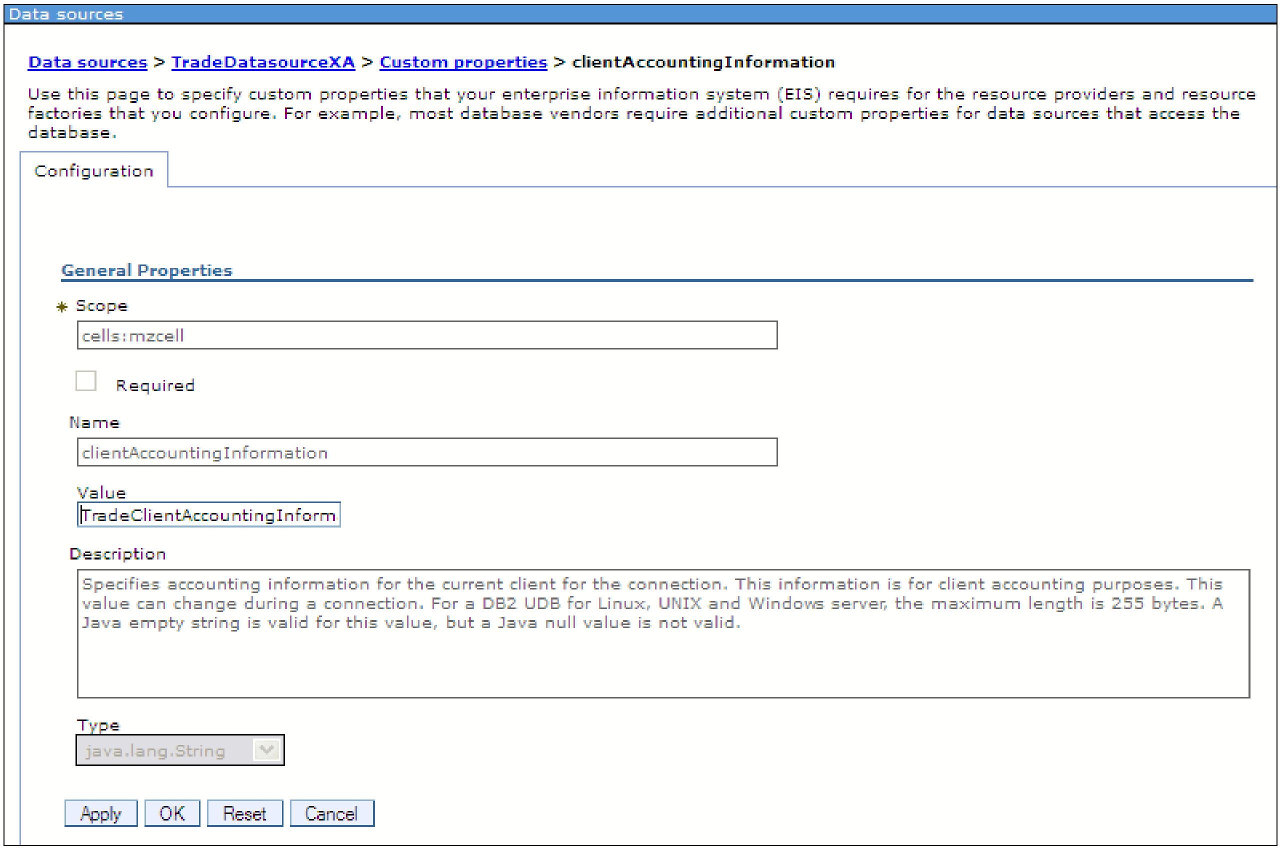


Figure 5-57 Application identification string

6. Click **Apply** and then save the changes. Figure 5-58 shows all the values of the properties, which we set by repeating the steps in this section.

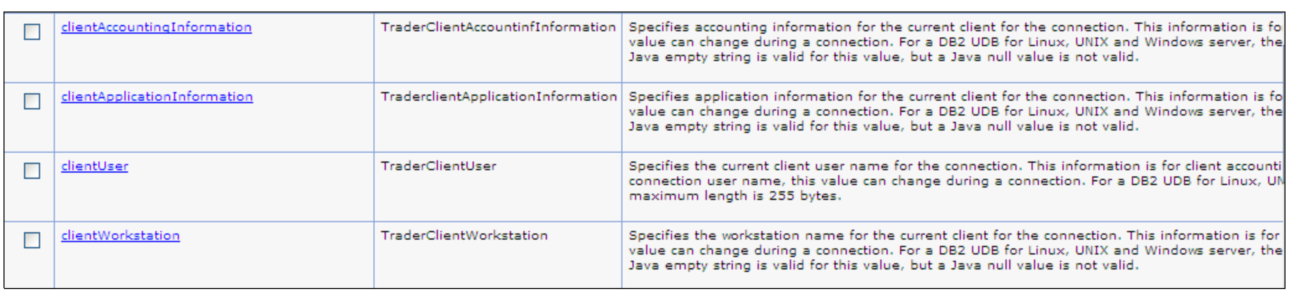


Figure 5-58 Properties values

5.5.2 Setting client information by using extended data source properties

In some customer environments, several applications share a single data source. This means the approach of setting client strings on the data source does not help identify the application. To address this issue, WebSphere Application Server allows individual applications to set these properties when they use resource references to access data sources. The approach is the same regardless of the application that is using a JDBC type 2 or 4 connection (XA or non-XA).

WebSphere Application Server requires your code to reference application server resources (such as data sources or J2C connection factories) through logical names, rather than access the resources directly in the Java Naming and Directory Interface (JNDI) name space. These logical names are called *resource references*.

WebSphere Application Server requires the usage of resource references for the following reasons:

- If application code looks up a data source directly in the JNDI naming space, every connection that is maintained by that data source inherits the properties that are defined in the application. Then, you create the potential for numerous exceptions if you configure the data source to maintain shared connections among multiple applications. For example, an application that requires a different connection configuration might attempt to access that particular data source, resulting in application failure.
- It relieves the programmer from having to know the name of the actual data source or connection factory at the target application server.

You can set the default isolation level for a data source through resource references. With no resource reference, you get the default for the JDBC driver that you use.

The extended properties are described in the WebSphere Application Server Information Center, which can be found at the following URL:

http://pic.dhe.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=%2Fcom.ibm.websphere.nd.multiplatform.doc%2Finfo%2Fae%2Fae%2Ftdat_heteropool.html

Using resource reference extended properties

To demonstrate setting the client information by using resource reference extended properties, we used a simple application named D0ZG_WASTestClientInfo, which uses a resource reference. The application uses a JDBC type 4 XA data source.

Complete the following steps:

1. In the navigation window of the administration console of the WebSphere Application Server, expand **Applications** and **Application Types**, as shown in Figure 5-59.

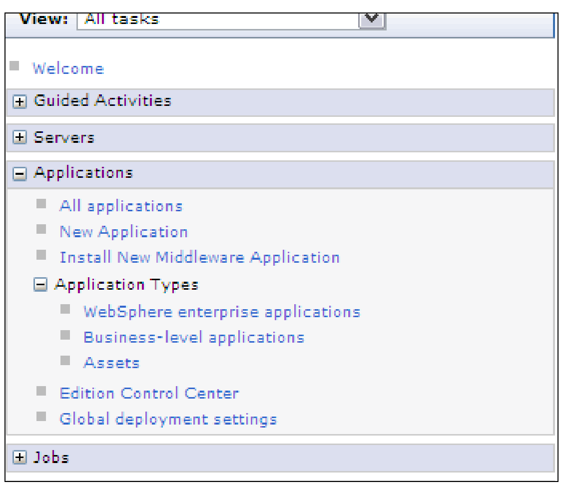


Figure 5-59 Administration console of the WebSphere Application Server

2. Click **WebSphere enterprise applications**. The window that is shown in Figure 5-60 opens. It has a list of all the installed applications in your environment.

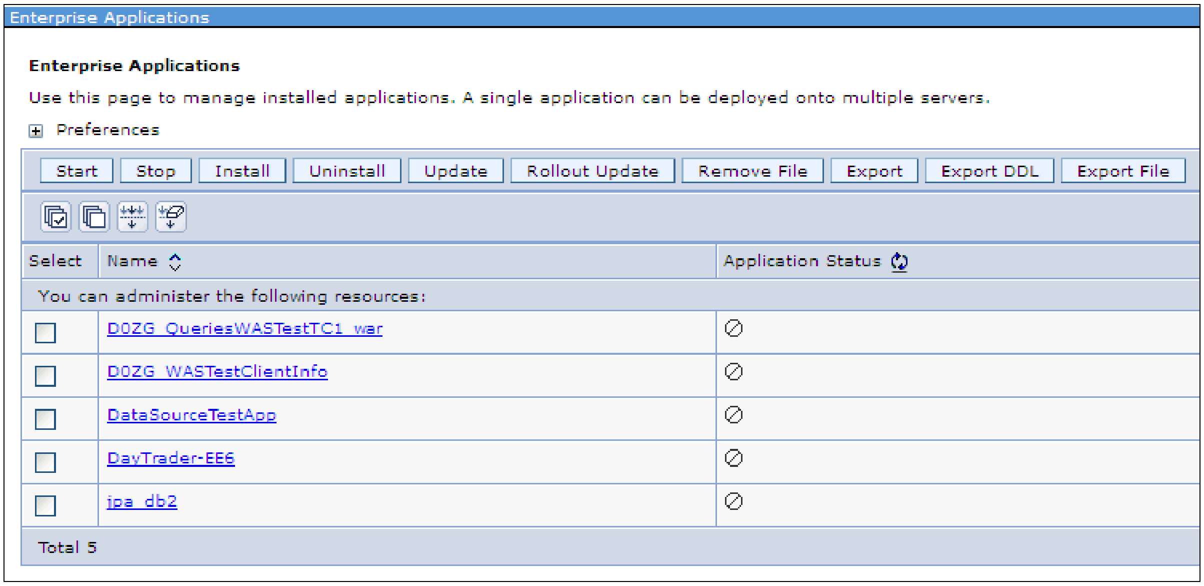


Figure 5-60 List all the WebSphere installed applications

3. Click the application on which you want to set the properties. In this example, click **D0ZG_WASTestClientInfo**. The window that is shown in Figure 5-61 opens and displays information about the application and all the artifacts that it uses.

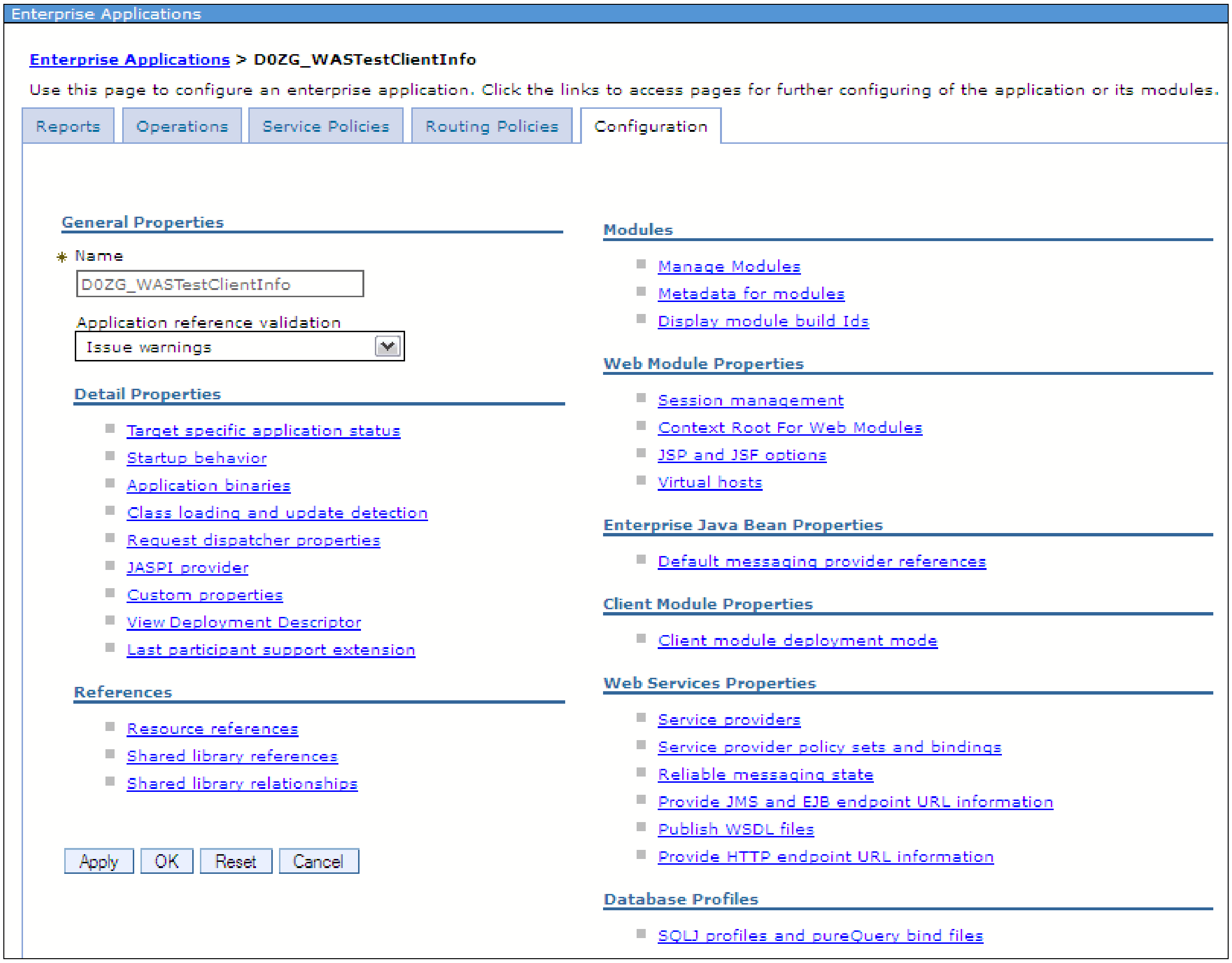


Figure 5-61 Information of the D0ZG_WASTestClientInfo application

4. Click **Resource references**. The window that is shown in Figure 5-62 opens. The window displays all the different resource references that are used by the applications. In this example, use only a data source reference.

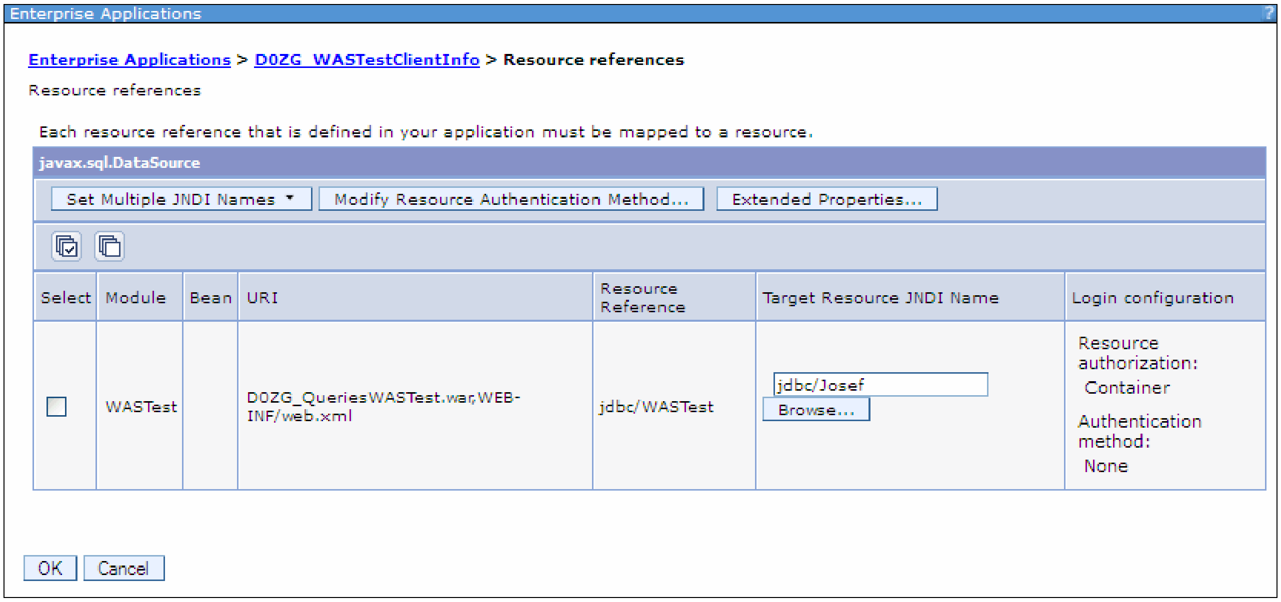


Figure 5-62 Resource reference for the chosen application

5. The example application uses jdbc/Josef, as shown in Figure 5-62. Select the module by selecting the **Select** check box, as shown in Figure 5-63.

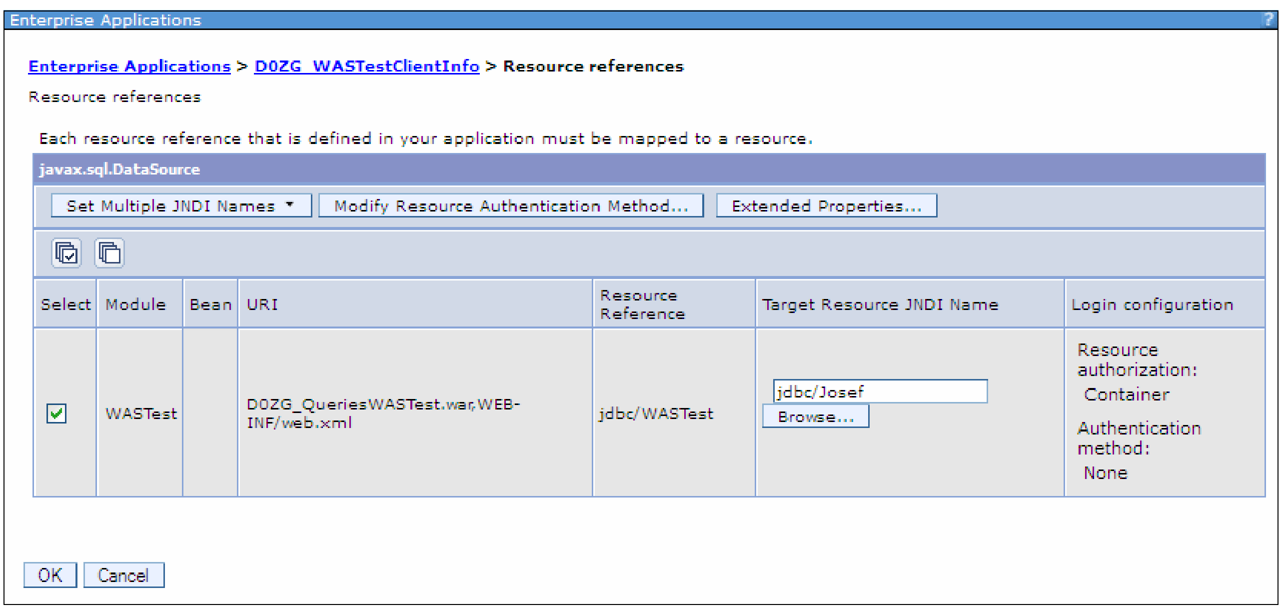


Figure 5-63 Selecting the module that is used by the application

6. Click **Extended Properties**. The window that is shown in Figure 5-64 on page 255 opens.

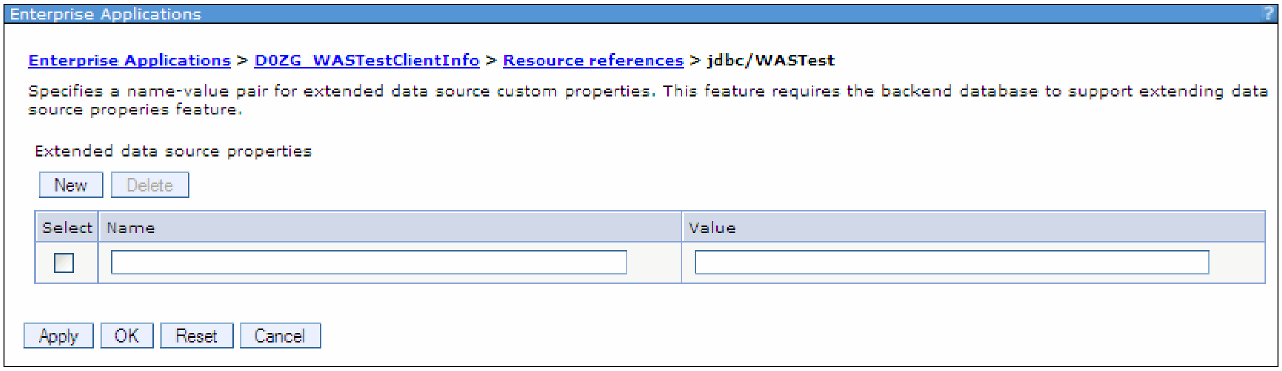


Figure 5-64 Extended properties panel

Enter the following information:

– Enter clientApplicationInformation for the Name
– Any string can be used as value. We used dwsClientinformation.

7. To add more properties, click **New**. A new field displays. In this example, enter clientWorkStation for Name and dwsClientWorkStation for the value, as shown in Figure 5-65. You can add the other properties, such as clientUser and clientAccountingInformation, as well.

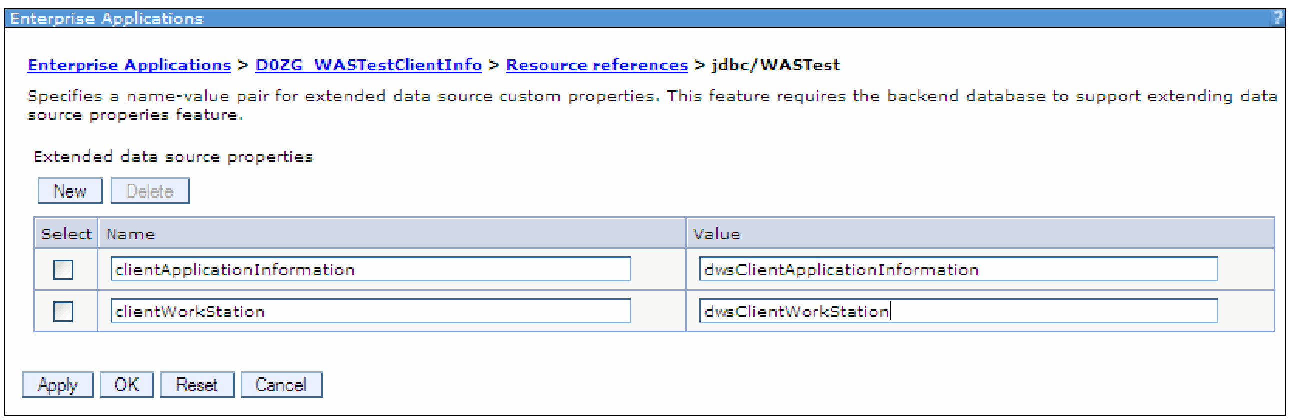


Figure 5-65 Entering the application properties

8. Click **Apply** and then **OK**. Save the changes. The application is configured and it is easy to identify the application in DB2 for z/OS.

5.5.3 Setting DB2 client information in a WebSphere Java application

If the DB2 client information settings that you must use can be determined only at run time, you might want to consider setting the DB2 client information in your Java application. You can set the DB2 client information from your WebSphere Java application by using the following options:

- Using the JDBC 4.0 setClientInfo Java API
- Using the Java API that is provided by IBM Data Server Driver JDBC and SQLJ
- Using the Java API that is provided by the WebSphere WSConnection class
- Calling the SYSPROC.WLM_SET_CLIENT_INFO stored procedure

For proof of technology (POT), we used Rational Application Developer for WebSphere Software to create the ClientInfo Dynamic Web project, which has the following servlets for setting DB2 client information:

- ClientInfoJDBC30API to use the Java interfaces that are provided by the DB2Connection class
- ClientInfoJDBC40API to use the java.sql.Connection.setClientInfo interface
- ClientInfoWSAPI to use the Java interfaces that are provided by the WebSphere WSConnection class
- ClientInfoWLM to use the SYSPROC.WLM_SET_CLIENT_INFO external stored procedure

If any applications set the DB2 client information fields, the values are not reset when the connection is returned to the connection pool. Applications must set these values at the beginning of the transaction to correctly collect and report data based on these fields.

The ClientInfo project servlets are illustrated in Figure 5-66.

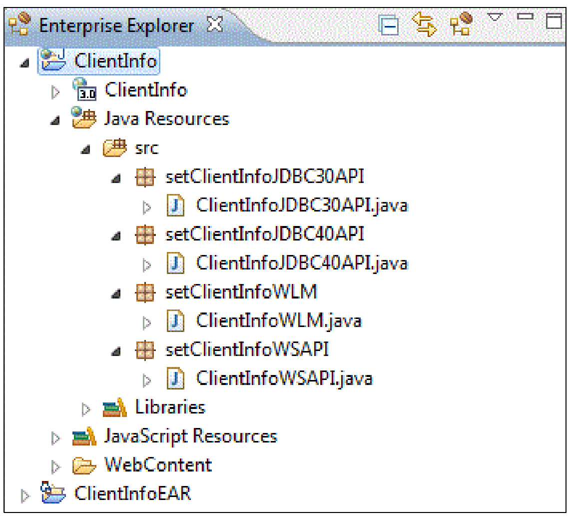


Figure 5-66 Rational Application Developer ClientInfo project

The Rational Application Developer ClientInfo project can be downloaded from the web. For more information, see Appendix H, "ClientInfo dynamic web project" on page 573.

The servlets that are illustrated in Figure 5-66 use the same program structure. Each servlet provides a setClientInformationFromJava subroutine to implement the particular code for setting DB2 client information by using the setClientInfo API, the Java interfaces provided by the DB2Connection class, or the WLM_SET_CLIENT_INFO stored procedure.

The servlet structure is illustrated in Example 5-5.

Example 5-5 General servlet structure set DB2 client information sample

```
package setClientInfoJDBC40API;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class ClientInfoJDBCAPI
 */
@WebServlet("/JDBC40API")
public class ClientInfoJDBC40API extends HttpServlet {
   private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ClientInfoJDBC40API() {
        super();
    }

   /**
    * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
    */
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
      PrintWriter pw = response.getWriter();
      response.setContentType("text/html");
      pw.println("Hello from ClientInfoJDBC40API Servlet <br/><br/> " );

      InitialContext ic = null;
      DataSource ds = null;
      try {
         ic = new InitialContext();
         ds = (DataSource) ic.lookup("jdbc/Josef"); 1
         pw.println("Successfully looked up jdbc/Josef JNDI entry<br/><br/>");
      } catch (NamingException e) {
         e.printStackTrace();
      }

      Connection conn = null;
      try {
         conn = ds.getConnection(); 2
         pw.println("Successfully got connection <br/><br/>");
         setClientInformationFromJava(conn,pw); 3
         pw.println("Running '"+returnSQL()+"' to retrieve current client info
settings<br/><br/>");
         PreparedStatement statement = conn.prepareStatement(returnSQL()); 4
          ResultSet rs = statement.executeQuery();
          while (rs.next()) { 5
              String clientaccounting = rs.getString(1);
```

```
                String clientapplication = rs.getString(2);
                String clientuserid = rs.getString(3);
                String clientworkstation = rs.getString(4);
                pw.println("CLIENT_ACCTNG=" + clientaccounting+"<br/>");
              pw.println("CLIENT_APPLNAME=" + clientapplication+"<br/>");
              pw.println("CLIENT_USERID=" + clientuserid+"<br/>");
              pw.println("CLIENT_WRKSTNNAME=" + clientworkstation+"<br/>");
              pw.println("<br/>");
               }
            pw.println("Running '"+returnSQLFunc()+"'<br/><br/>");
            PreparedStatement statementfunc =
conn.prepareStatement(returnSQLFunc()); 6
            ResultSet rsfunc = statementfunc.executeQuery();
            while (rsfunc.next()) {
                int    rowno    = rsfunc.getInt(1);
            String racfuser = rsfunc.getString(2);
                String racfgroup = rsfunc.getString(3);
                if (rowno == 1)
                pw.println("RACF user "+racfuser+" connected to the following
groups:<br/>");
                pw.println(rowno+" "+racfgroup+"<br/>");
              }
              conn.close(); 7
        }
        catch (SQLException e) {e.printStackTrace();} catch (Exception e) {
           e.printStackTrace();
        }
   }

   /**
    * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
    */
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
   }
   private String  returnSQL() { 8
      String sql = "SELECT CURRENT CLIENT_ACCTNG, " +
              " CURRENT CLIENT_APPLNAME ," +
              " CURRENT CLIENT_USERID, " +
              " CURRENT CLIENT_WRKSTNNAME " +
              "FROM SYSIBM.SYSDUMMY1";
      return sql;
   }

   private String  returnSQLFunc() { 9
      String sqlfunc =
            " WITH                                               " +
                  " Q1   (RES   ) AS                                   " +
                  " (SELECT f.GRACFGRP() FROM SYSIBM.SYSDUMMY1) ,      " +
                  " Q2            AS                                   " +
                  " (SELECT T.* FROM Q1,                               " +
                  "  XMLTABLE                                          " +
                  "  ('$D/GROUPS/GROUP'                                " +
                  "    PASSING XMLPARSE (DOCUMENT RES) AS D          " +
```

```
                "  COLUMNS                                              " +
                "   RACFUser    VARCHAR(08) PATH '../USER/text()'," +
                "   RACFGroup   VARCHAR(08) PATH './text()'       " +
                "  ) AS T )                                             " +
                " SELECT ROWNUMBER() OVER () AS ROWNO, Q2.* FROM Q2  " ;
      return sqlfunc;
   }
   public void setClientInformationFromJava(Connection conn, PrintWriter pw)
throws Exception 10
   { ....... Java code specific to the method for setting the DB2 client
information goes here ....
   }

}
```

Here are the processing steps:

1. Perform the JNDI lookup of jdbc/Josef.
2. Obtain the database connection.
3. Start the setClientInformationFromJava method to set the DB2 client information. The method contains Java code that is specific to the option that is chosen for setting the client information (JDBC 3.0, JDBC 4.0, WebSphere WSConnection class, or SYSPROC.WLM_SET_CLIENT_INFO procedure)
4. Prepare the SQL statement for reading the DB2 client information-related special registers CLIENT_ACCTNG, CURRENT CLIENT_APPLNAME, CURRENT CLIENT_USERID, and CURRENT CLIENT_WRKSTNNAME
5. Fetch and display the result set. This confirms whether setting the DB2 client information worked as expected.
6. Prepare the SQL statement for starting the GRACFGRP UDF. Before we did our testing, we stopped the UDF. As a result, the incoming UDF request was queued by DB2, giving us plenty of time to display the related DB2 thread attributes to confirm the DB2 client information settings.
7. Close the connection. This returns the connection to the application server for reuse.
8. returnSQL returns the first SQL statement to be dynamically prepared.
9. returnSQLFunc returns the second SQL statement to be dynamically prepared.
10. The setClientInformationFromJava contains the Java code that is specific to JDBC 3.0, JDBC 4.0, and WebSphere WSConnection class, or for invoking the SYSPROC.WLM_SET_CLIENT_INFO external stored procedure.

JDBC 4.0 setClientInfo Java API

If the WebSphere Application Server JDBC provider is configured to use the `db2jcc4.jar` file, you should use the java.sql.Connection.setClientInfo Java API to set the DB2 client information. The Java APIs that are described in “IBM Data Server Driver for JDBC and SQLJ Java API” on page 261 are deprecated in a JDBC 4.0 environment and should not be used. The Java code of the setClientInformationFromJava function that we used in the ClientInfoJDBC40API Java class is illustrated in Example 5-6.

Example 5-6 Using JDBC 4.0 setClientInfo Java API

```
public void setClientInformationFromJava(Connection conn, PrintWriter pw) throws
Exception
   {
```

```
      conn.setClientInfo("ClientUser","JDBC40API_clientuser"); 1
      conn.setClientInfo("ClientHostname","JDBC40API_clientworkstation");
      conn.setClientInfo("ApplicationName","JDBC40API_clientapplication");

conn.setClientInfo("ClientAccountingInformation","JDBC40API_clientaccounting");
      pw.println("successfully invoked setClientInfo JDBC 4.0 API for setting DB2
Client Info to the following values <br/><br/>" ); 2
      pw.println("  ClientUser=JDBC40API_clientuser<br/>" );
      pw.println("  ClientHostname=JDBC40API_clientworkstation<br/>" );
      pw.println("  ApplicationName=JDB40CAPI_clientapplication<br/>" );
      pw.println("
ClientAccountingInformation=JDBC40API_clientaccounting<br/><br/>" );
   }
```

The setClientInformationFromJava method that is shown in Example 5-6 on page 259 performs the following major steps:

1. Starts the JDBC 4.0 setClientUser API for setting the DB2 client information.
2. Returns confirmation messages to the browser application.

The ClientInfoJDBC40API servlet returned the processing result that is shown in Figure 5-67.

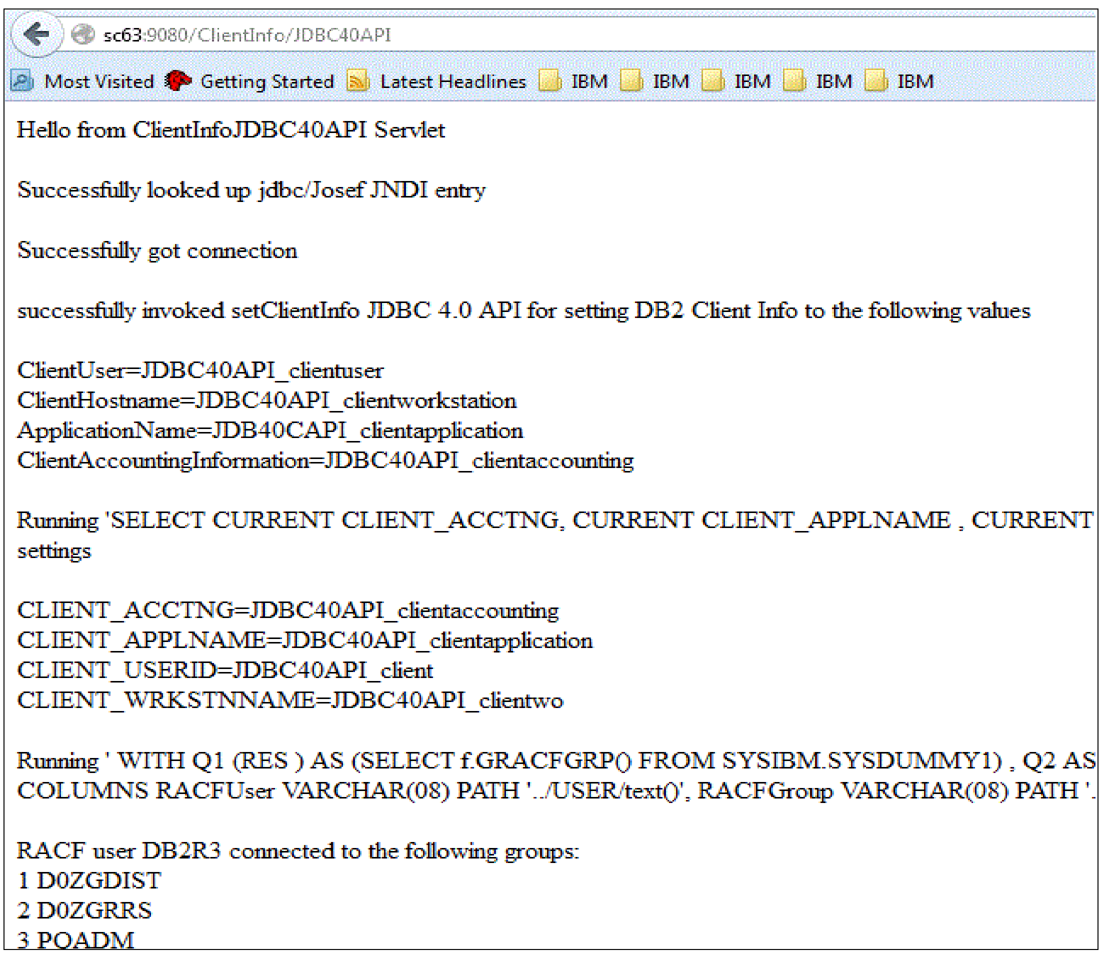


Figure 5-67 Servlet ClientInfoJDBC40API result

During servlet execution in our example, we used the display thread output that is shown in Figure 5-68 on page 261 to confirm the DB2 client information settings.

```
DSNV401I  -DOZ1 DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -DOZ1 ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
SERVER   SW *    12 db2jcc_appli DB2R3    DISTSERV 0083    45
 V437-WORKSTATION=JDBC40API_clientwo, USERID=JDBC40API_client,
      APPLICATION NAME=JDBC40API_clientapplication
 V429 CALLING FUNCTION=F.GRACFGRP,
```

Figure 5-68 Servlet ClientInfoJDBC40API display thread output

IBM Data Server Driver for JDBC and SQLJ Java API

The IBM Data Server Driver for JDBC and SQLJ combines type 2 and type 4 JDBC implementations. The driver is packaged in the following way:

- IBM Data Server Driver for JDBC and SQLJ Version 3.5x, JDBC 3.0 compliant. The `db2jcc.jar` and `sqlj.zip` files are available for JDBC 3.0 and earlier support.
- IBM Data Server Driver for JDBC and SQLJ Version 4.x, compliant with JDBC 4.0 or later. The `db2jcc4.jar` and `sqlj4.zip` files are available for JDBC 4.0 or later, and JDBC 3.0 or earlier support.

You control the level of JDBC support that you want by specifying the appropriate JAR files in the JDBC provider, as shown in Figure 5-25 on page 227. Both JAR files contain the DB2Connection class to support the following Java APIs for setting DB2 client information:

- setDB2ClientUser(String paramString)
- setDB2ClientWorkstation(String paramString)
- setDB2ClientApplicationInformation(String paramString)
- setDB2ClientAccountingInformation(String paramString)

Because these Java APIs are deprecated in JDBC 4.0, you might want to use the setClientInfo Java API in case your JDBC provider is configured to use the `db2jcc4.jar` file. For more information about how to use the setClientInfo API, see "JDBC 4.0 setClientInfo Java API" on page 259.

In our example, we use a pass-through mechanism that is provided by the WebSphere WSCallHelper class to invoke the APIs of the DB2Connection class that we need for setting the DB2 client information.

The Java code of the setClientInformationFromJava function that we used in the ClientInfoJDBC30API Java class is illustrated in Example 5-7.

Example 5-7 Using IBM Data Server Driver for JDBC and SQLJ set client information Java APIs

```
public void setClientInformationFromJava(Connection conn, PrintWriter pw) throws
Exception
{
      setWorkStationName(conn, "JDB30CAPI_clientworkstation")1;
      setApplicationName(conn,"JDBC30API_clientapplication");
      setAccounting(conn,"JDBC30API_clientaccounting");
      setEndUser(conn,"JDBC30API_clientuser");
      pw.println("successfully invoked JDBC 3.0 API for setting DB2 Client Info to
the following values <br/><br/>"); 5
      pw.println("  ClientUser=JDBC30API_clientuser<br/>");
      pw.println("  ClientHostname=JDB30CAPI_clientworkstation<br/>");
      pw.println("  ApplicationName=JDBC30API_clientapplication<br/>");
```

```
      pw.println("
ClientAccountingInformation=JDBC30API_clientaccounting<br/><br/>");
   }
public void setWorkStationName(Connection con, String work)
        throws SQLException, Exception {
      WSCallHelper
            .jdbcCall(null, con, "setDB2ClientWorkstation",
                  new Object[] { new String(work) },
                  new Class[] { String.class });
   }

   public void setApplicationName(Connection con, String appl) 2
         throws SQLException, Exception {
      WSCallHelper
            .jdbcCall(null, con, "setDB2ClientApplicationInformation",
                  new Object[] { new String(appl) },
                  new Class[] { String.class });
   }

   public void setAccounting(Connection con, String accounting) 3
         throws SQLException, Exception {
      WSCallHelper.jdbcCall(null, con, "setDB2ClientAccountingInformation",
            new Object[] { new String(accounting) },
            new Class[] { String.class });
   }

   public void setEndUser(Connection con, String endUser) throws SQLException, 4
         Exception {
      WSCallHelper.jdbcCall(null, con, "setDB2ClientUser",
            new Object[] { new String(endUser) },
            new Class[] { String.class });
   }
```

The setClientInformationFromJava method that is shown in Example 5-7 on page 261 performs the following major steps:

1. Invokes internal methods for further processing.
2. Uses the WSCallHelper method to invoke the DB2Connection.setDB2ClientApplicationInformation interface.
3. Uses the WSCallHelper method to invoke the DB2Connection.setDB2ClientAccountingInformation interface.
4. Uses the WSCallHelper method to invoke the DB2Connection.setDB2ClientUser interface.
5. Returns confirmation messages to the browser application.

The ClientInfoJDBC30API servlet returned the processing result that is shown in Figure 5-69 on page 263.

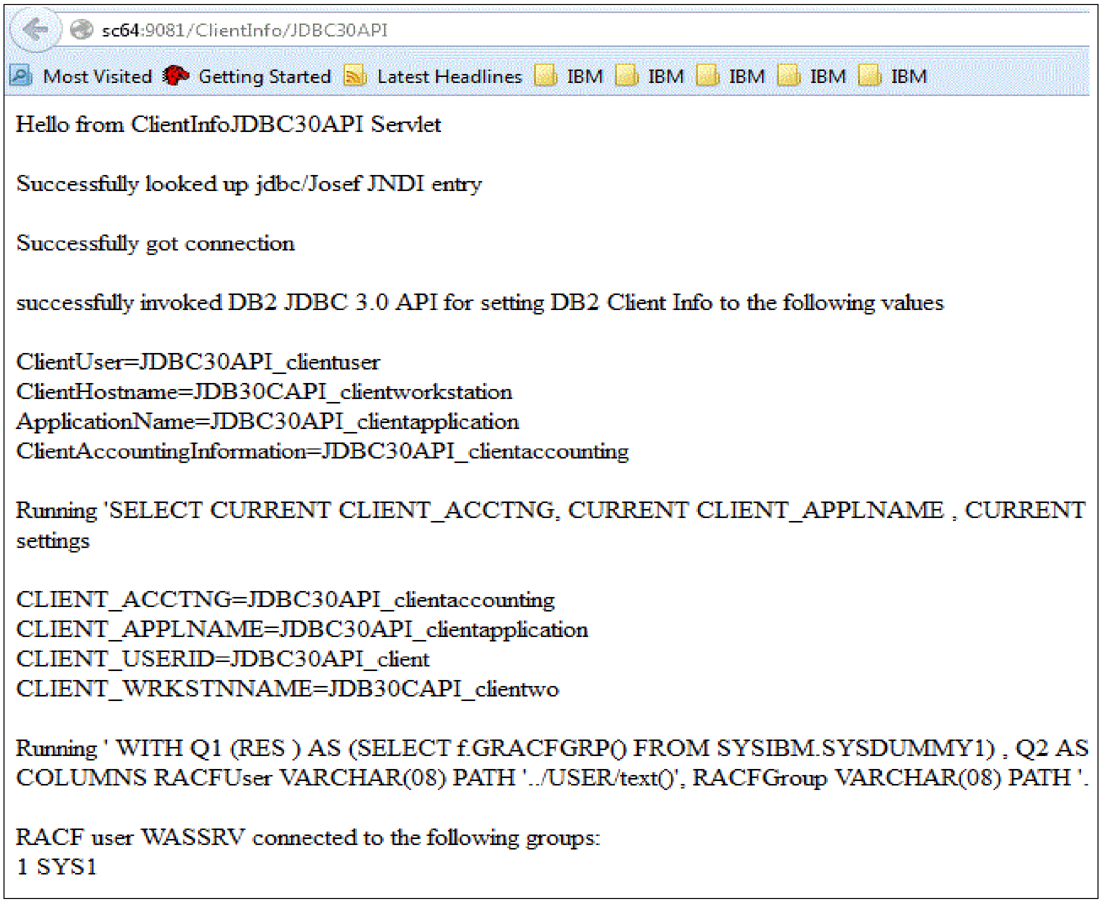


Figure 5-69 Servlet ClientInfoJDBC30API result

During servlet execution, we used the display thread output that is shown in Figure 5-70 to confirm the DB2 client information settings.

```
DSNV401I  -DOZ1 DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -DOZ1 ACTIVE THREADS -
NAME     ST A   REQ ID          AUTHID   PLAN     ASID TOKEN
SERVER   SW *    40 db2jcc_appli WASSRV   DISTSERV 0083    39
V485-TRUSTED CONTEXT=CTXWASTESTT4,
     SYSTEM AUTHID=WASSRV,
     ROLE=WASTESTDEFAULTROLE
 V437-WORKSTATION=JDB30CAPI_clientwo, USERID=JDBC30API_client,
      APPLICATION NAME=JDBC30API_clientapplication
 V429 CALLING FUNCTION=F.GRACFGRP,
      PROC=        , ASID=0000, WLM_ENV=DSNWLMDBOZ_GENERAL
```

Figure 5-70 Servlet ClientInfoJDBC30API display thread output

Using the Java API that is provided by the WebSphere WSConnection class

Instead of using the DB2Connection object (see "IBM Data Server Driver for JDBC and SQLJ Java API" on page 261), you can use the following Java interface:

com.ibm.websphere.rsadapter.WSConnection.setClientInformation(Properties arg0)

The Java code of the setClientInformationFromJava function that we used in the ClientInfoWSAPI Java class is illustrated in Example 5-8.

Example 5-8 Using the WSConnection class

```
public void setClientInformationFromJava(WSConnection conn, PrintWriter pw) throws 
Exception
   {
    Properties props = new Properties(); 1
    props.setProperty(WSConnection.CLIENT_ID, "WSAPI_clientuser"); 2
    props.setProperty(WSConnection.CLIENT_LOCATION, "WSAPI_clientworkstation");3
    props.setProperty(WSConnection.CLIENT_APPLICATION_NAME, 
"WSAPI_clientapplication"); 4
    conn.setClientInformation(props); 5
    pw.println("successfully invoked WSConnection APIs for setting DB2 Client Info 
to the following values <br/><br/>" );
    pw.println("  WSConnection.CLIENT_ID=WSAPI_clientuser<br/>" );
    pw.println("  WSConnection.CLIENT_LOCATION=WSAPI_clientworkstation<br/>" );
    pw.println(" 
WSConnection.CLIENT_APPLICATION_NAME=WSAPI_clientapplication<br/></br>" );
   }
```

In Example 5-8, the code performs the following actions:

1. Instantiates the properties object.
2. Sets the WSConnection.CLIENT_ID property.
3. Sets the WSConnection.CLIENT_LOCATION property.
4. Sets the WSConnection.CLIENT_APPLICATION_NAME property.
5. Invoke the WSConnection setClientInformation interface.

The ClientInfoWSAPI servlet returned the processing result shown Figure 5-71 on page 265.

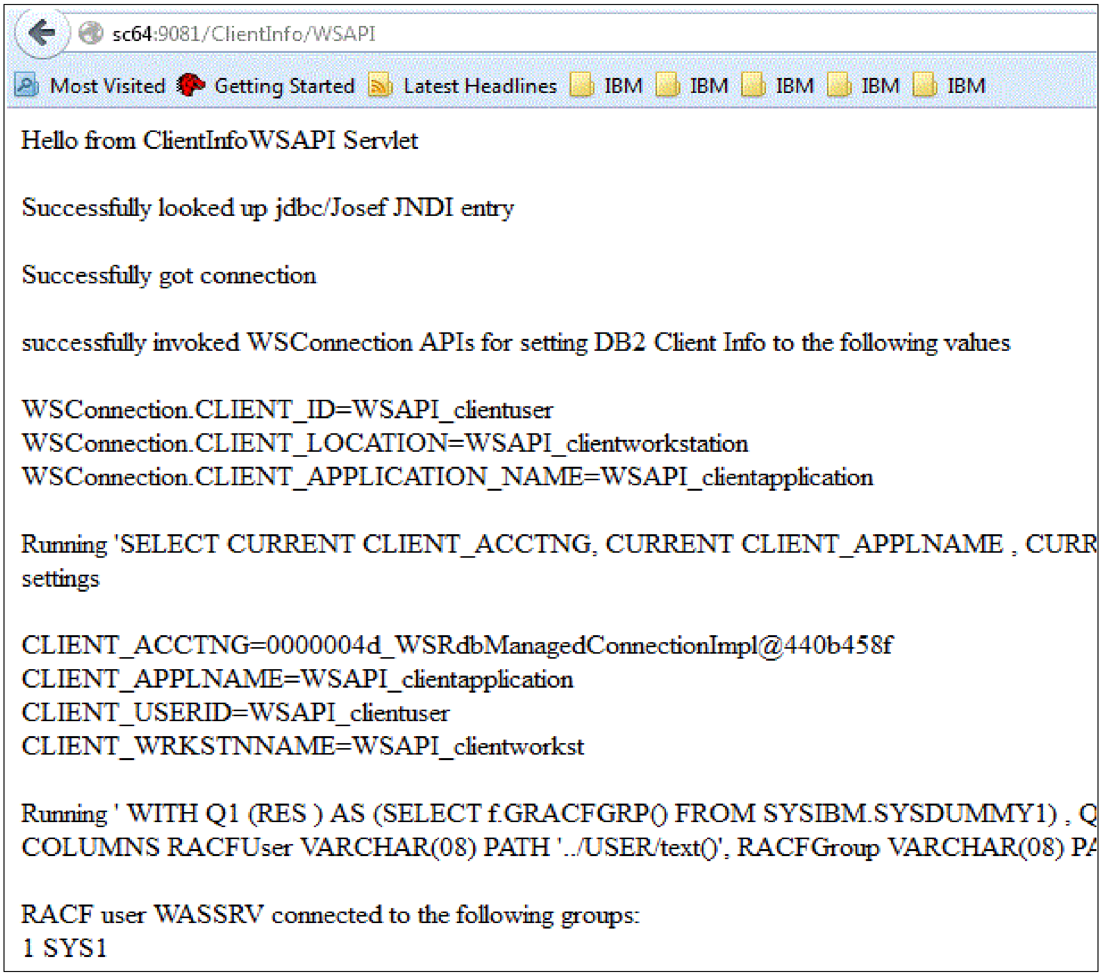


Figure 5-71 Servlet ClientInfoWSAPI result

During servlet execution, we used the display thread output that is shown in Figure 5-72 to confirm the DB2 client information settings.

```
DSNV401I  -D0Z1 DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -D0Z1 ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
SERVER   SW *    76 db2jcc_appli WASSRV   DISTSERV 0083    39
V485-TRUSTED CONTEXT=CTXWASTESTT4,
     SYSTEM AUTHID=WASSRV,
     ROLE=WASTESTDEFAULTROLE
 V437-WORKSTATION=WSAPI_clientworkst, USERID=WSAPI_clientuser,
      APPLICATION NAME=WSAPI_clientapplication
 V429 CALLING FUNCTION=F.GRACFGRP,
```

Figure 5-72 Servlet ClientInfoWSAPI display thread output

Calling the SYSPROC.WLM_SET_CLIENT_INFO stored procedure

The Java API that is available for setting the DB2 client information depends on the IBM Data Server Driver for JDBC and SQLJ JAR file that your JDBC Provider is configured for:

- If your JDBC provider uses the `db2jcc.jar` file, you can use the Java API that is described in "IBM Data Server Driver for JDBC and SQLJ Java API" on page 261.
- If your JDBC provider uses the `db2jcc4.jar` file, you can use the Java API that is described in "JDBC 4.0 setClientInfo Java API" on page 259 or "IBM Data Server Driver for JDBC and SQLJ Java API" on page 261. Considering that the Java APIs that are described in "IBM Data Server Driver for JDBC and SQLJ Java API" on page 261 are deprecated in JDBC 4.0, your development strategy might force you to change existing applications to use the JDBC 4.0 provided setClientInfo API.

Choosing the correct option for setting the DB2 client information can be difficult because the Java API you use depends on the JDBC driver level that your application is using. You can ignore this dependency by using the WLM_SET_CLIENT_INFO external stored procedure for setting the DB2 client information.

The WLM_SET_CLIENT_INFO external stored procedure load module DSNADMSI uses the RRS DSNRLI SET_CLIENT_ID function to set the client information that is associated with the current connection at the DB2 server. Using this method does not depend on the JDBC driver level, the JDK level, or the type or version of the application server that you are using.

The Java code of the setClientInformationFromJava function that we used in the ClientInfoWLM Java class is illustrated in Example 5-9.

Example 5-9 Using the SYSPROC.WLM_SET_CLIENT_INFO external stored procedure

```
public void setClientInformationFromJava(Connection conn, PrintWriter pw) throws
Exception
   {
      CallableStatement clientApplCall = null; 1
      clientApplCall = conn.prepareCall("CALL
SYSPROC.WLM_SET_CLIENT_INFO(?,?,?,?)"); 2
      clientApplCall.setString(1, "WLM_clientuser"); 3
      clientApplCall.setString(2, "WLM_clientworkstation");
      clientApplCall.setString(3, "WLM_clientapplication");
      clientApplCall.setString(4, "WLM_clientaccounting");
      clientApplCall.executeUpdate(); 4
      pw.println("successfully called SYSPROC.WLM_SET_CLIENT_INFO to set DB2
Client Info to the following values:<br/><br/>" ); 5
      pw.println("  ClientUser=WLM_clientuser<br/>" );
      pw.println("  ClientHostname=WLM_clientworkstation<br/>" );
      pw.println("  ApplicationName=WLM_clientapplication<br/>" );
      pw.println("  ClientAccountingInformation=WLM_clientaccounting<br/><br/>" );
   }
```

Here are the processing steps for calling the SYSPROC.WLM_SET_CLIENT_INFO:

1. Instantiate the clientApplCall CallableStatement object.
2. Dynamically prepare the CALL statement.
3. Use the java.sql.PreparedStatement.setString method to provide the necessary hostvariable values.
4. Run the **`SQL CALL`** statement.
5. Display the variable settings that are used in the **`CALL`** statement.

The ClientInfoWLM servlet returned the processing result that is shown in Figure 5-73.

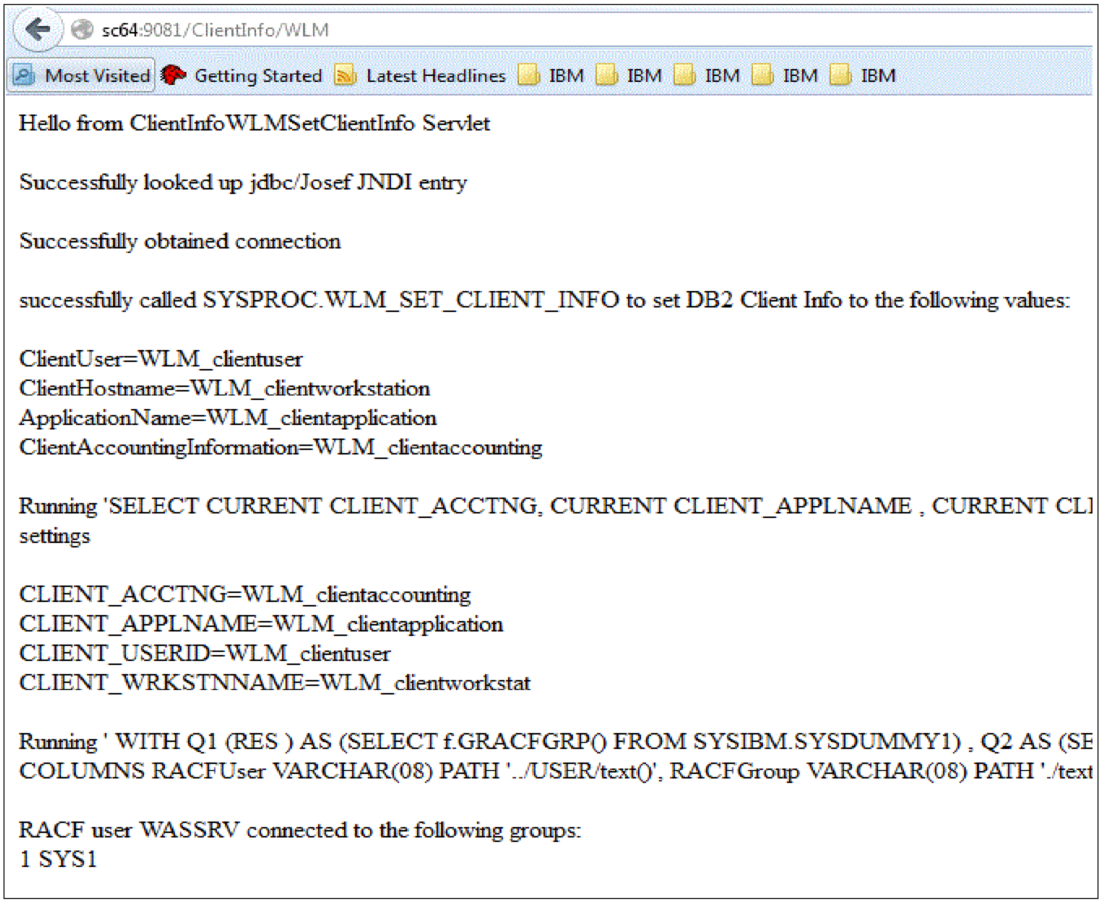


Figure 5-73 Servlet ClientInfoWLM result

During servlet execution, we used the display thread output that is shown in Figure 5-74 to confirm the DB2 client information settings.

```
DSNV401I  -DOZ1 DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -DOZ1 ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
SERVER   SW *    58 db2jcc_appli WASSRV   DISTSERV 0083    39
V485-TRUSTED CONTEXT=CTXWASTESTT4,
     SYSTEM AUTHID=WASSRV,
     ROLE=WASTESTDEFAULTROLE
 V437-WORKSTATION=WLM_clientworkstat, USERID=WLM_clientuser,
      APPLICATION NAME=WLM_clientapplication
 V429 CALLING FUNCTION=F.GRACFGRP,
      PROC=        , ASID=0000, WLM_ENV=DSNWLMDBOZ_GENERAL
```

Figure 5-74 Servlet ClientInfoWLM display thread output

5.6 Configuring the prepared statement cache in WebSphere Application Server

The WebSphere PreparedStatements cache does not store any DB2 specific information. The cache is used solely by WebSphere to reduce processor consumption when you create a Java object.

Complete the following steps:

1. In the navigation window of the administrative console of the WebSphere Application Server, expand **Resources** and click **Data sources**, as shown in Figure 5-75.

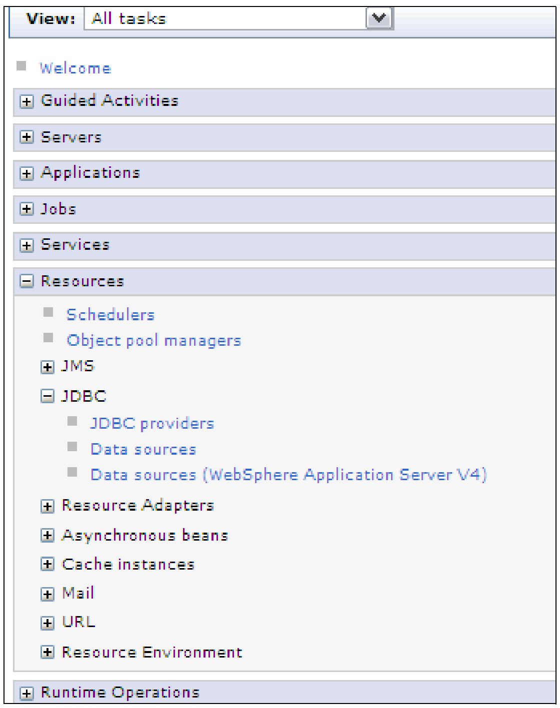


Figure 5-75 Administrative console of the WebSphere Application Server

The window that is shown in Figure 5-76 on page 269 opens. This window shows a list of the existing JDBC data sources that are defined in your environment.

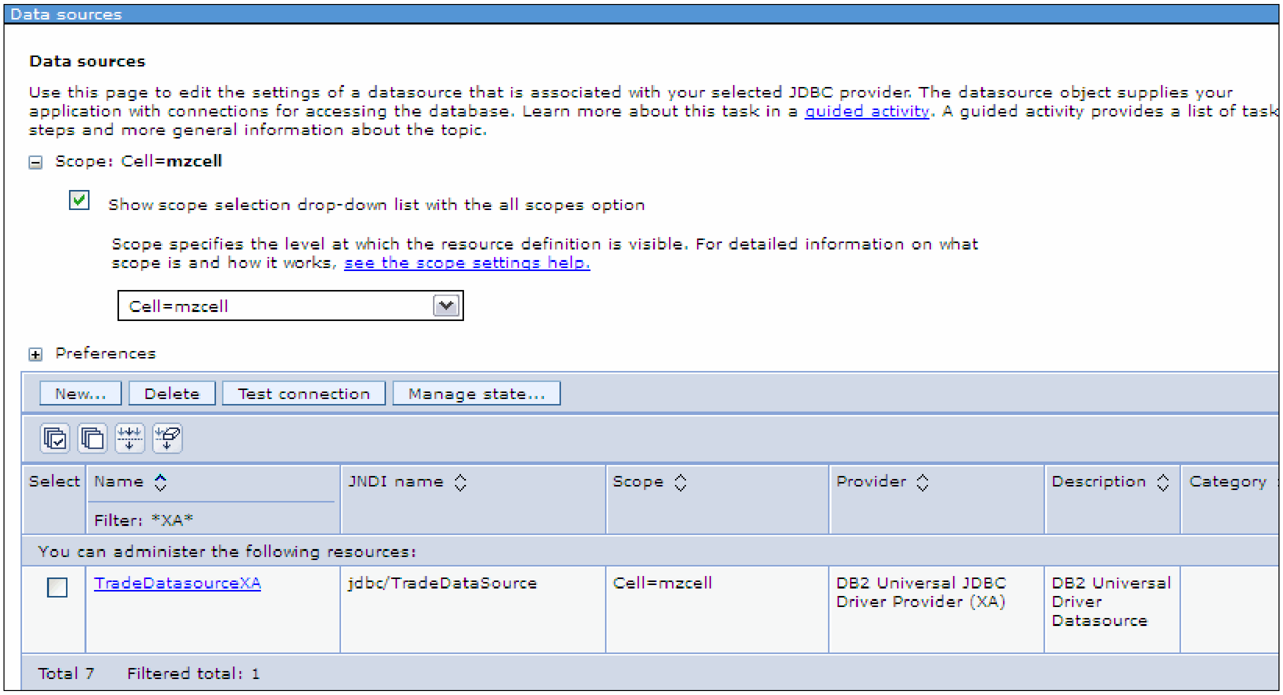


Figure 5-76 List of existing JDBC data sources

2. Click **TradeDatasourceXA** and the window that is shown in Figure 5-77 opens.

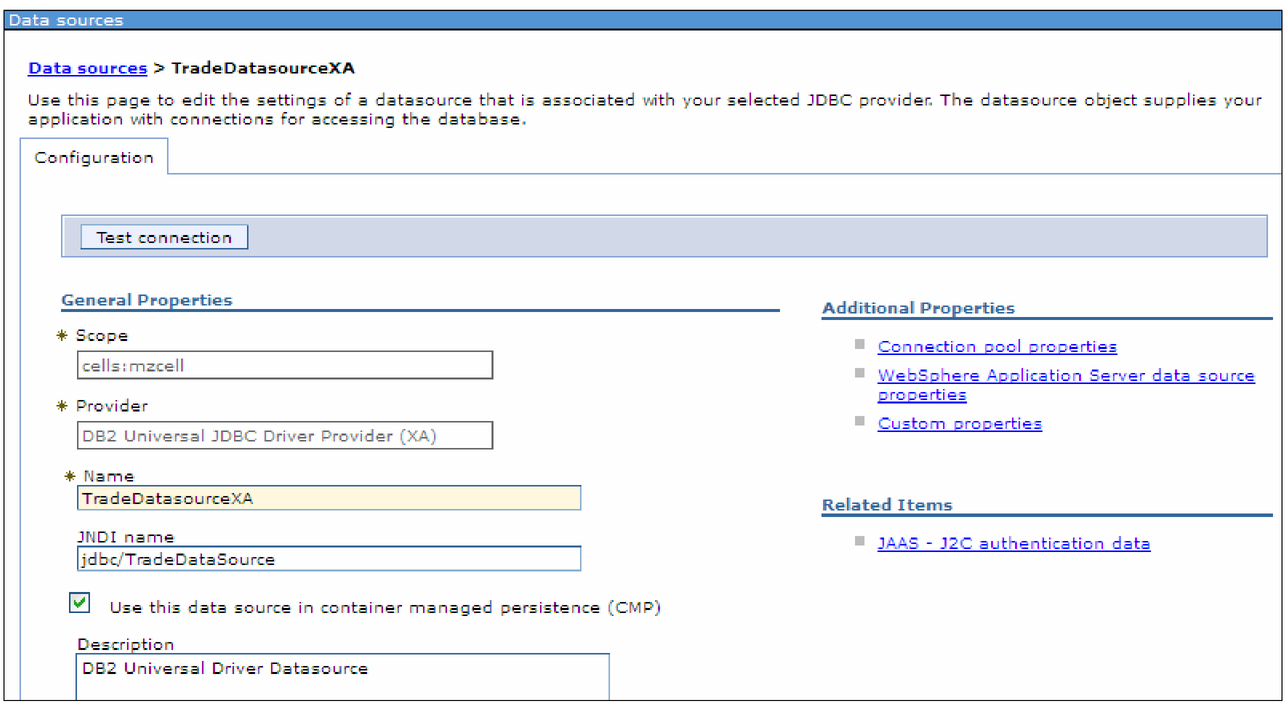


Figure 5-77 JDBC TradeDatasourceXA resource

3. Click **WebSphere Application Server data source properties**. The window that is shown in Figure 5-78 opens.

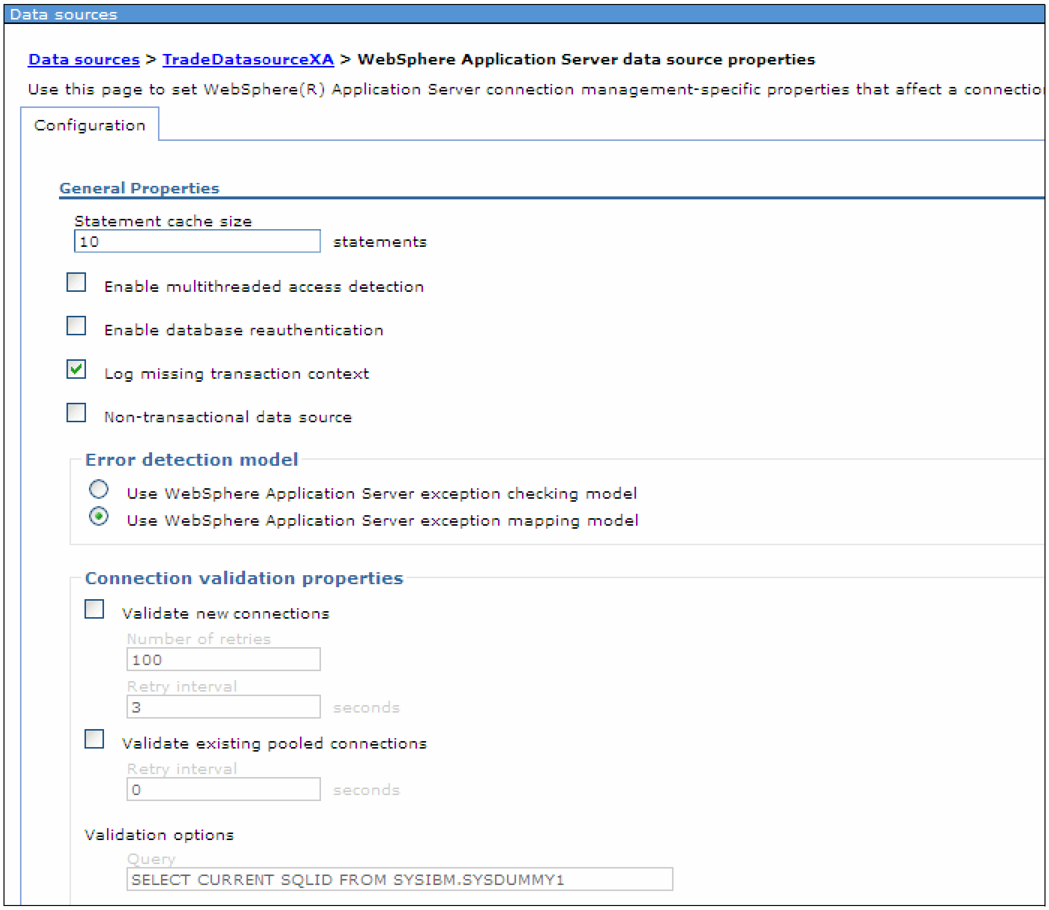


Figure 5-78 Data source properties window

The Statement cache size specifies the number of statements that can be cached per connection. The default size is 10. We used the default in our environment. Applications should configure the value based on how many SQL statements are used.

5.7 Configuring the J2C authentication alias

Java Authentication and Authorization Services (JAAS) is a Java API that is used to establish an authenticated user ID. This API can be invoked in several instances, in particular when connecting to DB2. This connection can be established on behalf of two environments:

- Container (using the user ID of the thread that is running)
- Component/application (The user ID is explicitly passed on the getConnection call.)

Complete the following steps:

1. In the navigation window of the administration console, expand **Security**, as shown in Figure 5-79 on page 271.

Figure 5-79 WebSphere navigation window

2. Click **Global security** and the window that is shown in Figure 5-80 opens.

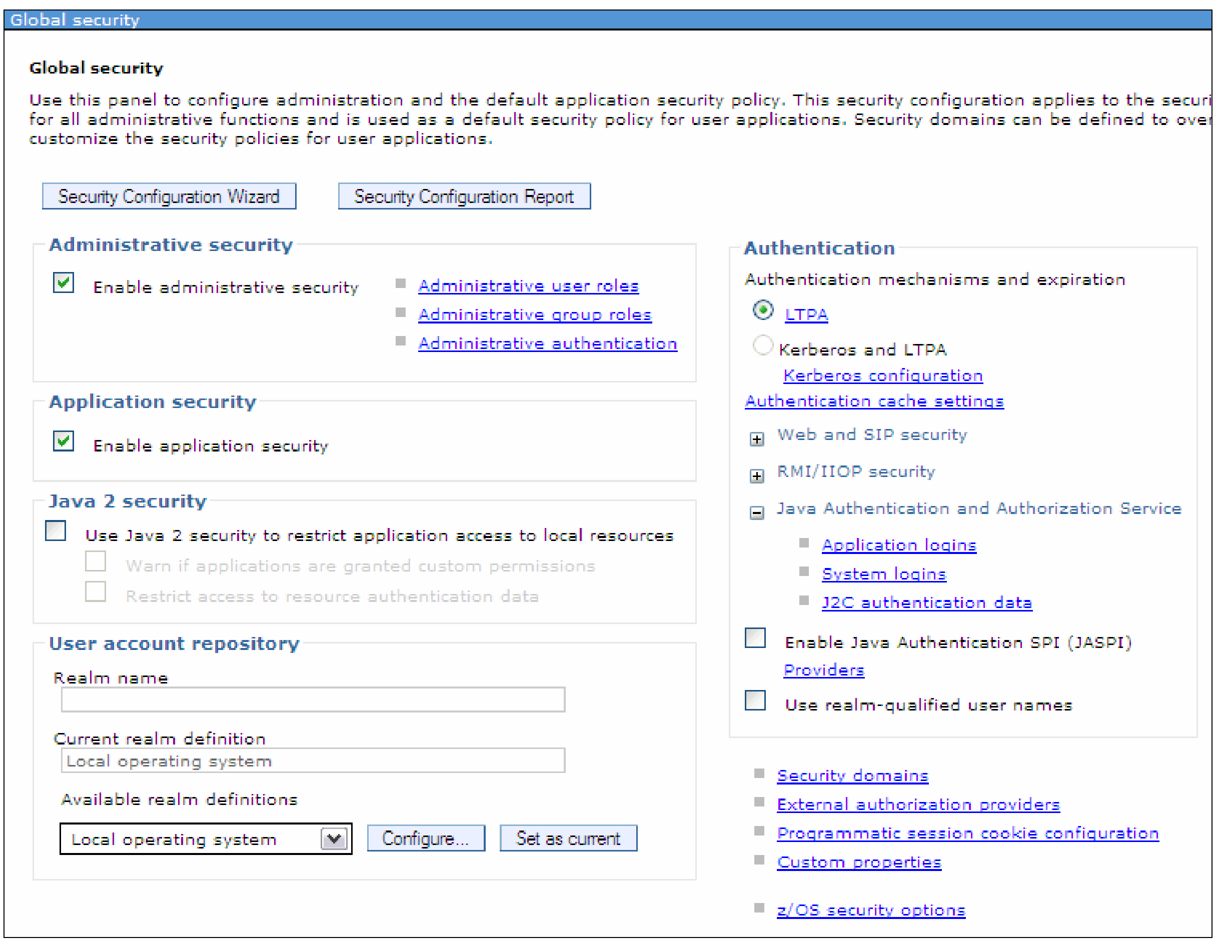


Figure 5-80 Global security

3. Expand **Java Authentication** and **Authorization Service** and click the J2C authentication data. The window that is shown in Figure 5-81 opens, which lists the existing J2C authentication aliases that are defined.

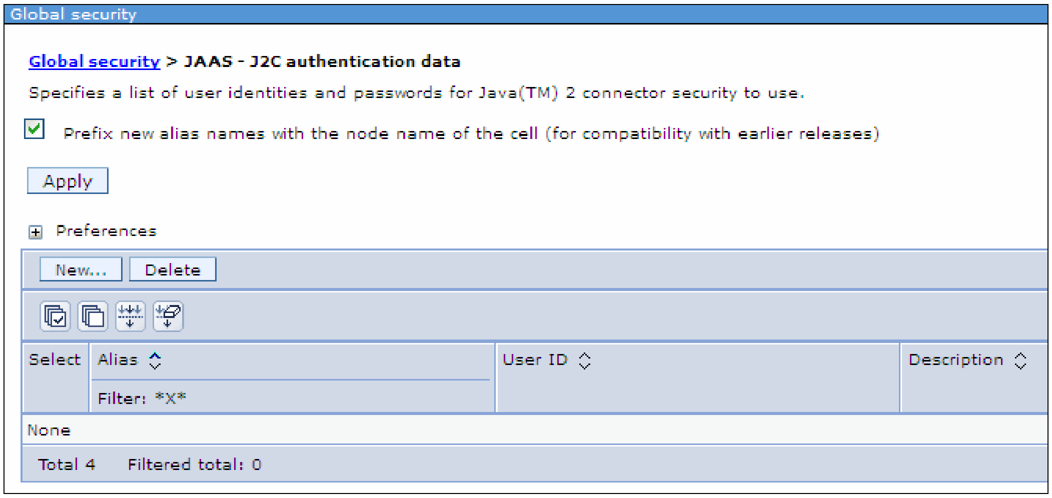


Figure 5-81   J2C authentication data

4. Click **New** and the window that is shown in Figure 5-82 opens.

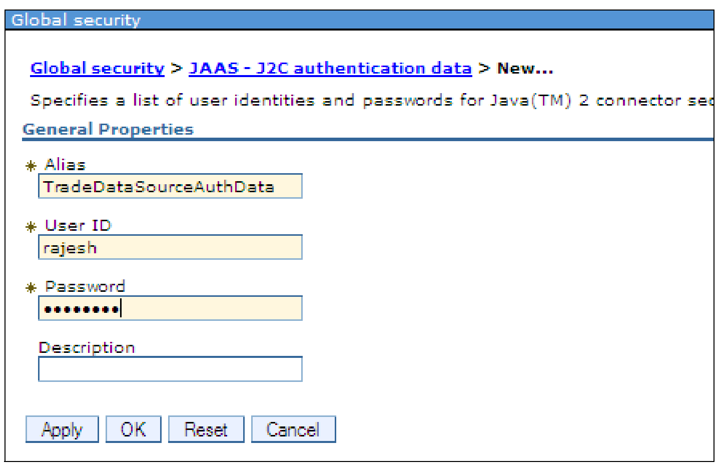


Figure 5-82   J2C authentication input definition

Enter the following information:

– A string for the alias. We used TradeDataSourceAuthData.

– The user ID to be used by the data source to connect to DB2 for z/OS. We used rajesh.

– The password.

5. Click **Apply**, then **OK**, and then save the changes. This J2C authentication alias can be used with either a JDBC type 2 or type 4 data source (XA or non-XA).

5.8 Configuring connection pool sizes on data sources in WebSphere Application Server

To configure connection pool sizes on data sources in WebSphere Application Server, complete the following steps:

1. In the navigation window of the administrative console of the WebSphere Application Server, expand **Resources** and click **Data sources**, as shown in Figure 5-83.

Figure 5-83 WebSphere navigation window

The window that is shown in Figure 5-84 opens. This window shows a list of existing JDBC data sources that are defined in your environment.

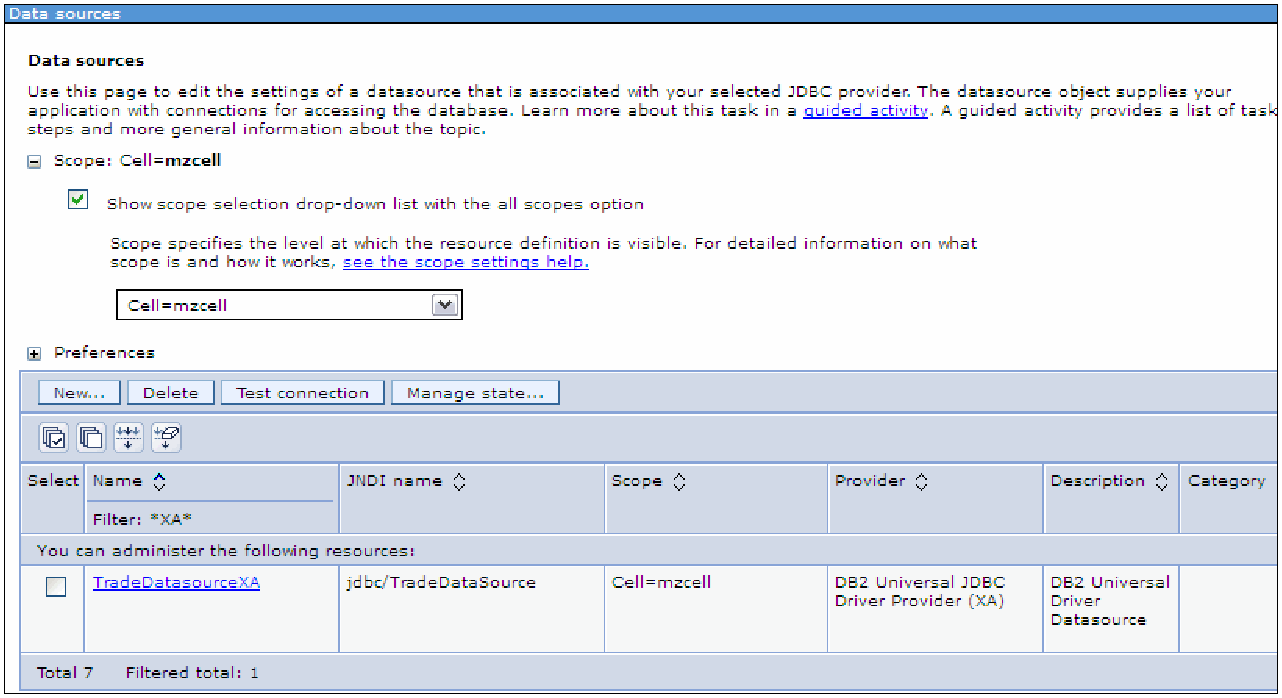


Figure 5-84 Data source and JDBC provider association

2. Click **TradeDatasourceXA** and the window that is shown in Figure 5-85 opens.

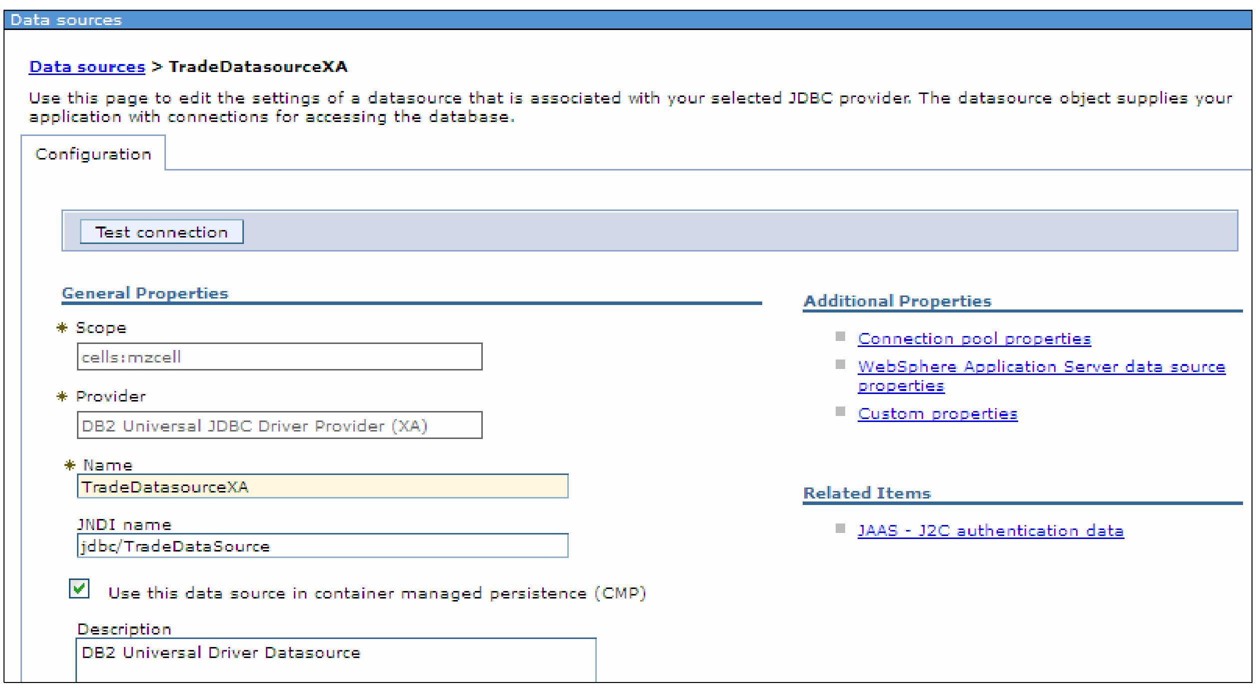


Figure 5-85 Data source and provider

3. Click **Connection pool properties**. The window that is shown in Figure 5-86 on page 275 opens.

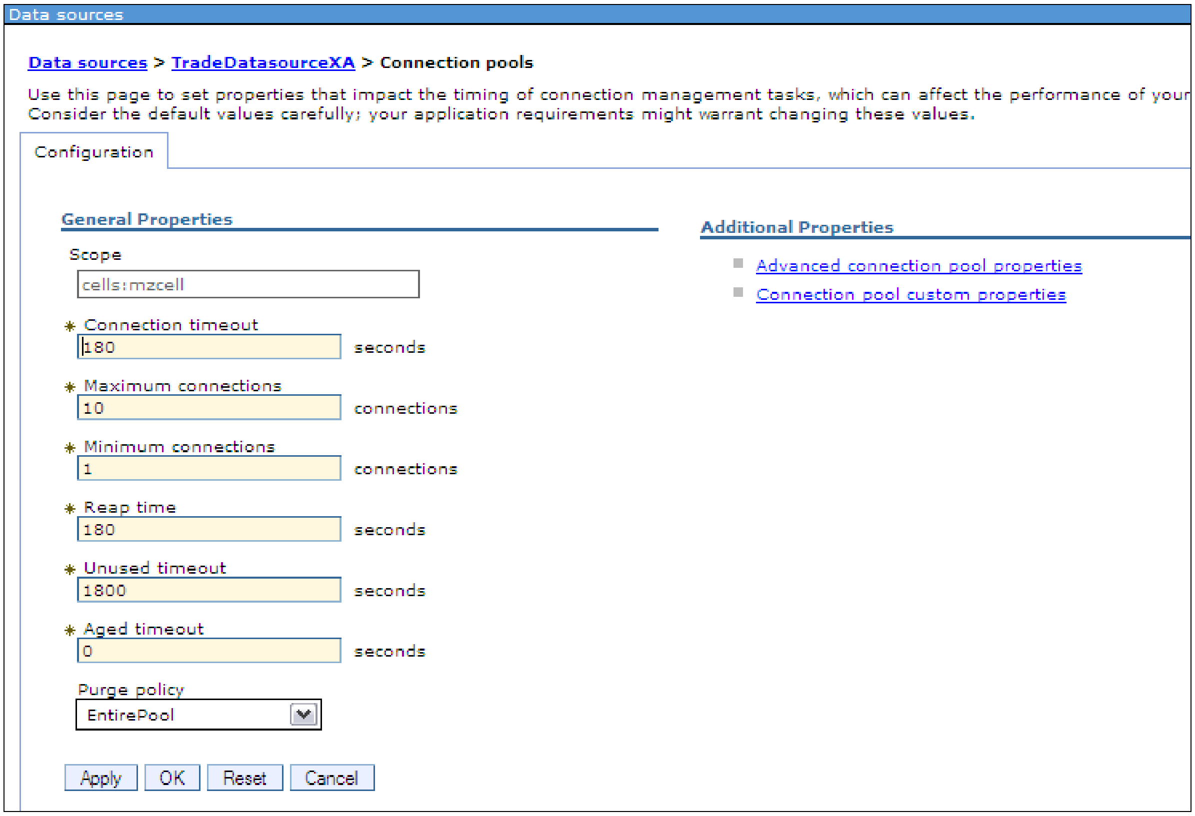


Figure 5-86 Connection pool properties

Connection pooling is a function of WebSphere Application Server. It is not a function of the IBM Data Server Driver for JDBC and SQLJ. The driver does not implement connection pooling.

Here is a brief description of the properties that are shown in Figure 5-86:

Connection Timeout How long to attempt connection creation before a timeout occurs

Max Connections The maximum connections from this JVM instance

Min Connections The minimum number of connections in a pool

Reap Time How often a cleanup of pool is scheduled, in seconds

Unused Timeout How long to let a connection sit in the pool unused

Aged Timeout How long to let a connection live before recycling

Purge Policy After StaleConnection, does the entire pool get purged or only individual connection

In the window that is shown in Figure 5-86, consider using the following preferred practices:

- Set the WebSphere Application Server connection unused timeout to a smaller value than the DB2 idle thread timeout to avoid stale connection conditions.
- Consider setting Min Connections to zero.
- In DB2 10, you can reduce processor usage by selectively binding the client package with the `RELEASE(DEALLOCATE)` option.
- Consider setting the WebSphere Application Server aged timeout to less than five minutes, such as 120 seconds, to reduce long-lived threads.

5.9 Enabling trusted context for applications that are deployed in WebSphere Application Server

A *trusted context* is an object that the database administrator defines that contains a system authorization ID and a set of trust attributes. The relationship between a database connection and a trusted context is established when the connection to the database server is created, and that relationship remains for the life of the database connection. This feature allows WebSphere Application Server to use the trusted DB2 connection under a different user without reauthenticating the new user.

Trusted context can be enabled at an application level in WebSphere Application Server.

For this example, we use a simple application, D0ZG_WASTestClientInfo, which uses a resource reference. The application uses a JDBC type 4 XA data source.

Complete the following steps:

1. In the navigation window of the administration console of the WebSphere Application Server, expand **Applications** and **Application Types**, as shown in Figure 5-87.

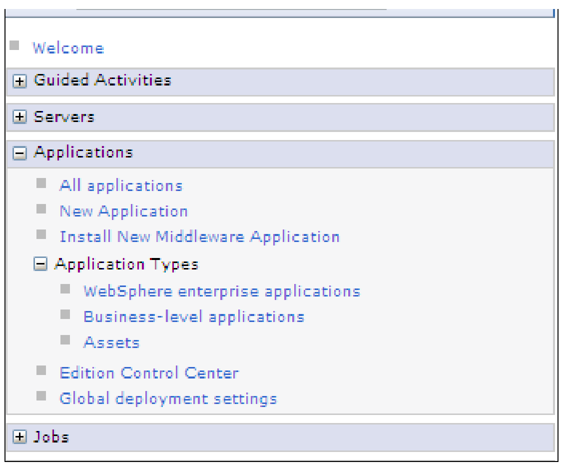


Figure 5-87 Administration console of WebSphere Application Server

2. Click **WebSphere enterprise applications**. The window that is shown in Figure 5-88 opens and shows all the installed applications in your environment.

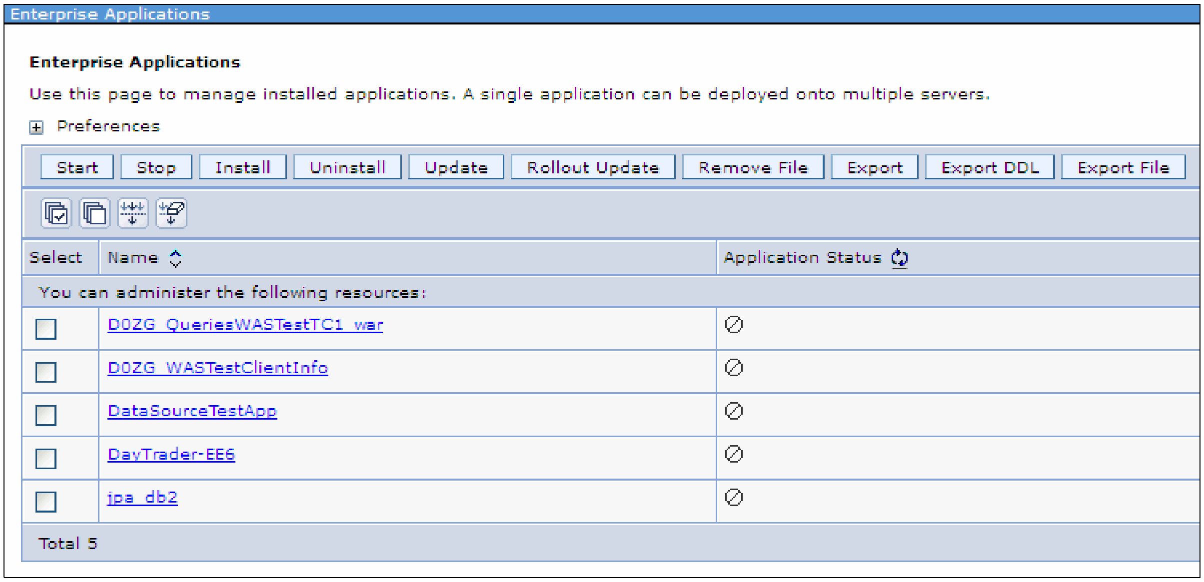


Figure 5-88 List of installed applications

3. Click the application that on which you want to set the properties. In this example, click **D0ZG_WASTestClientInfo**. The window that is shown in Figure 5-89 opens. This window displays information about the application and all the artifacts it uses.

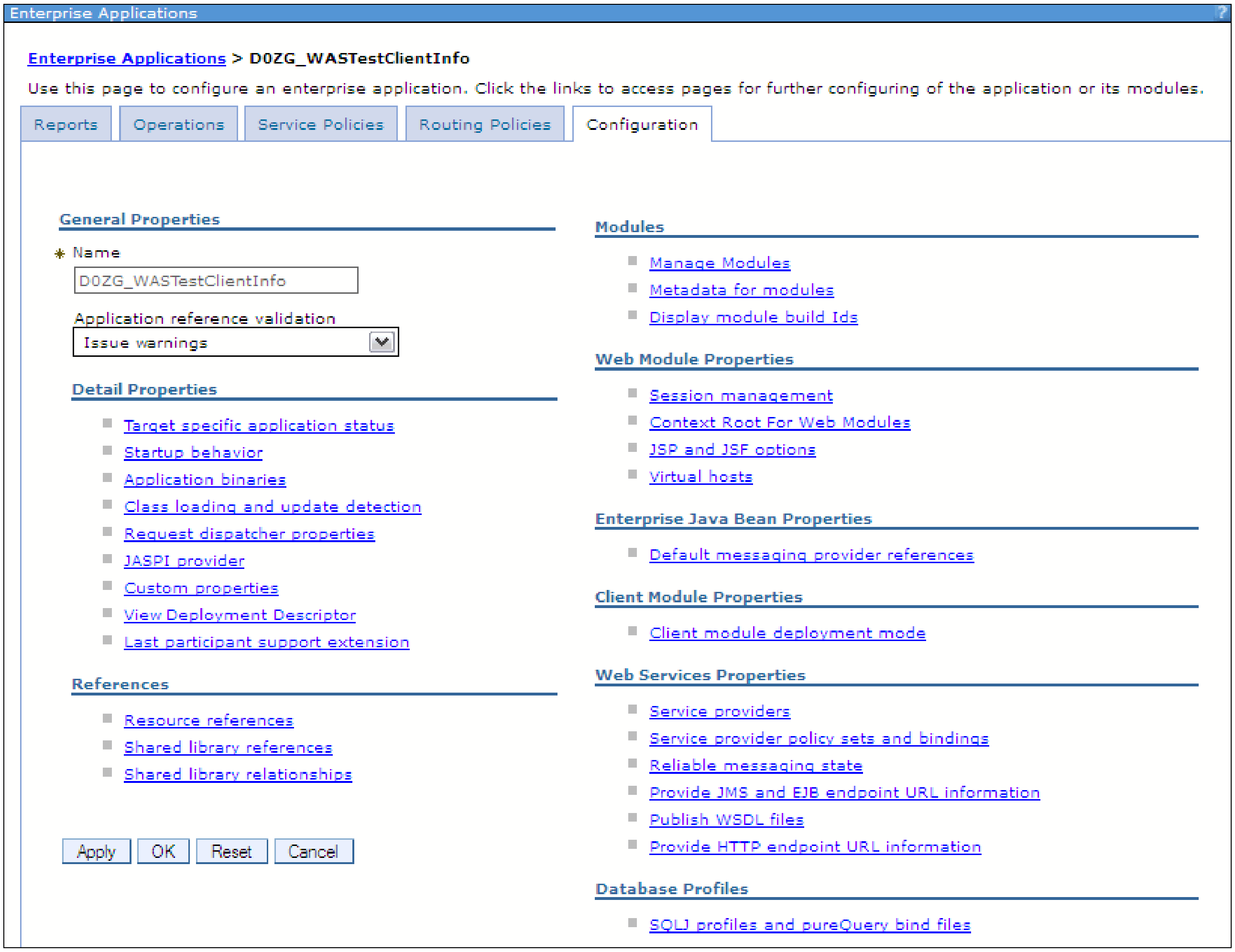


Figure 5-89 D0ZG_WASTestClientInfo.properties definition

4. Click **Resource references**. The window that is shown in Figure 5-90 on page 279 opens. The window lists all the different resource references that are used by the applications. In our example, we use only a data source reference.

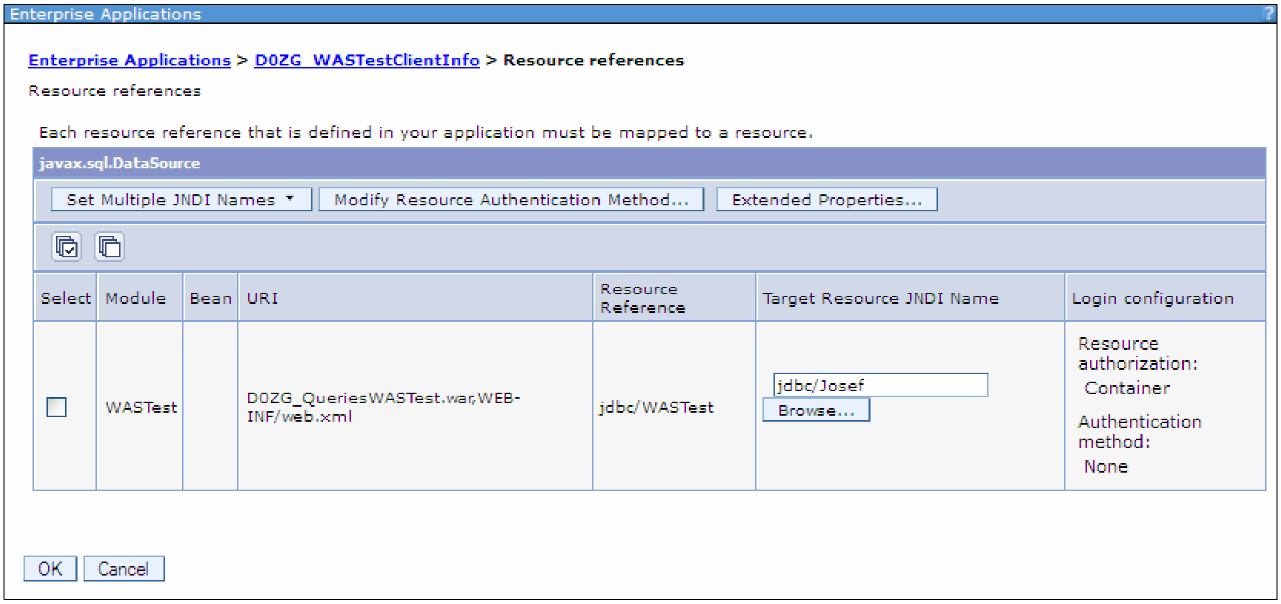


Figure 5-90 Resource reference

5. The example, application uses jdbc/Josef, as shown in Figure 5-90. Select the module by selecting the **Select** check box, as shown in Figure 5-91.

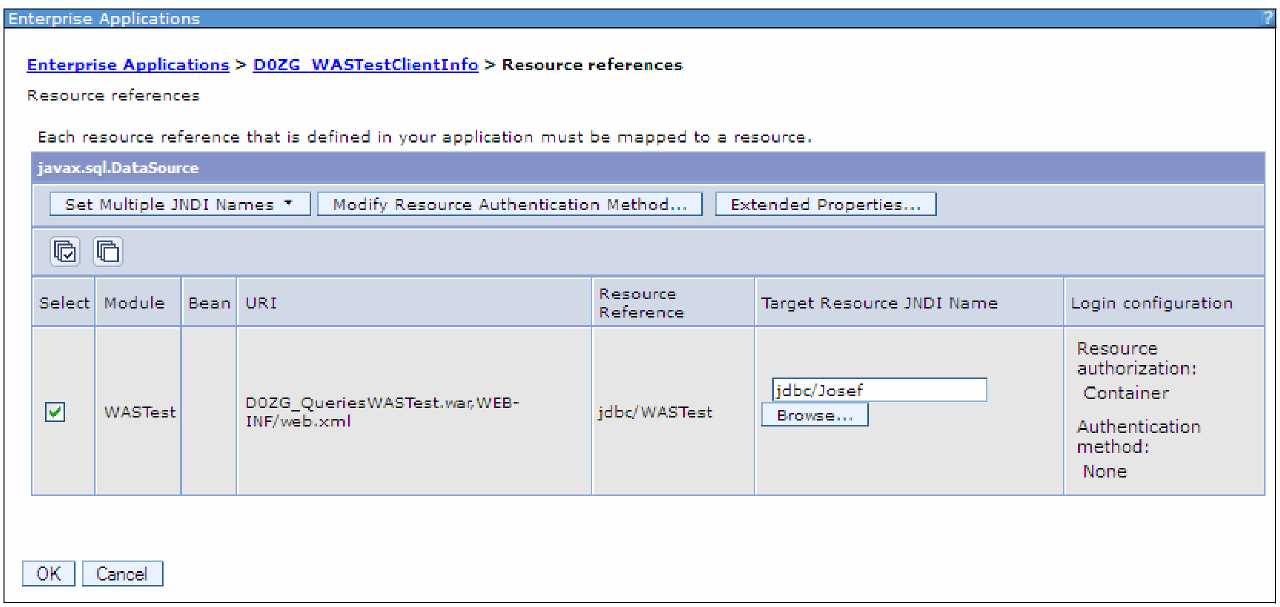


Figure 5-91 Selecting the jdbc/Josef module

6. Click **Modify Resource Authentication Method**. The window that is shown in Figure 5-92 opens.

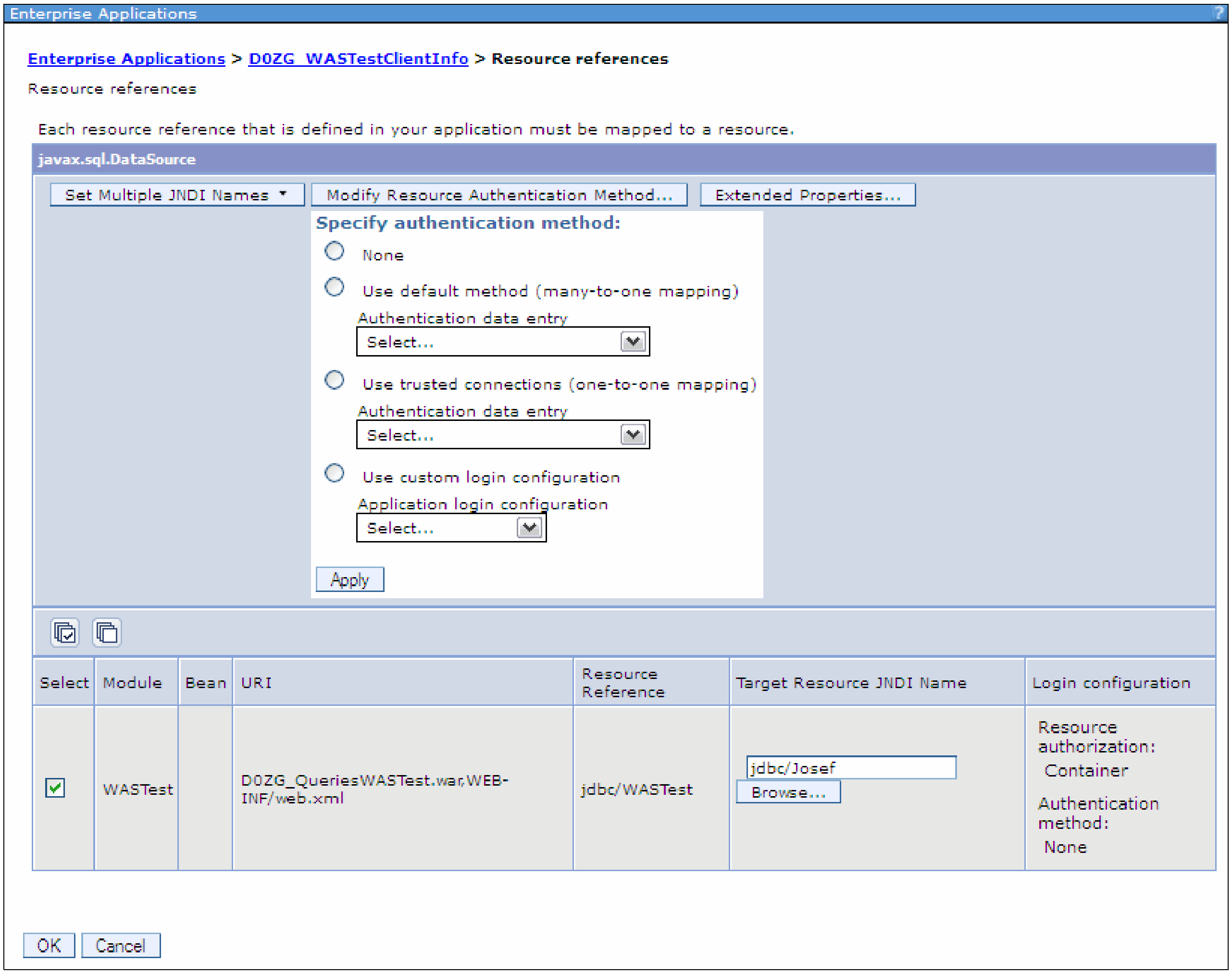


Figure 5-92 Resource Authentication definition

7. Select the **Use trusted connections** radio button. Then, select a JAAS alias in the drop-down menu, as shown in Figure 5-93. The user ID in the JAAS alias should have only connect privileges to DB2 for z/OS and should be defined as part of the trusted context definition in DB2. In our example, we created a JAAS alias named *trustedcontext*.

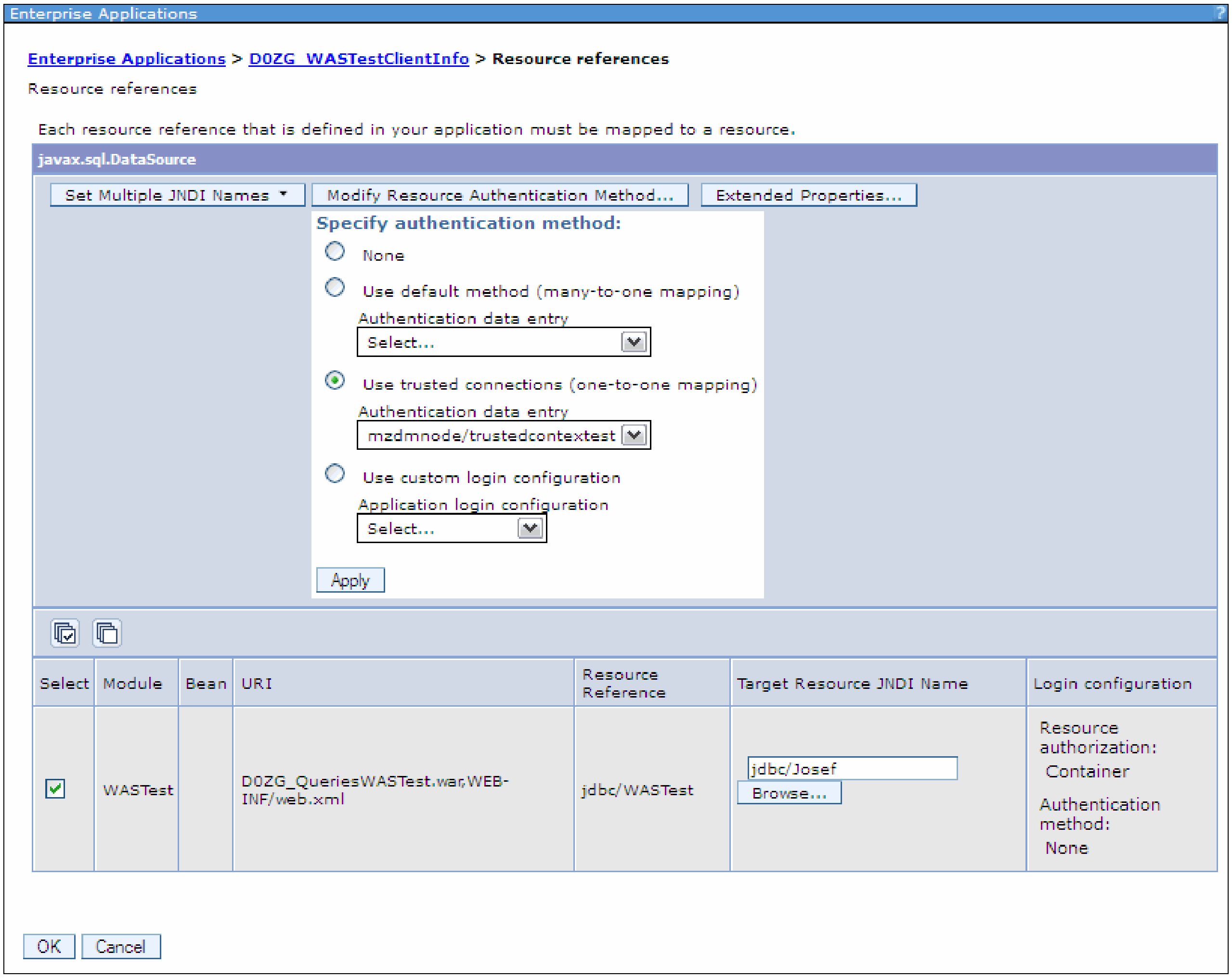


Figure 5-93 JAAS alias trusted connection

8. Click **Apply** and the window that is shown in Figure 5-94 opens.

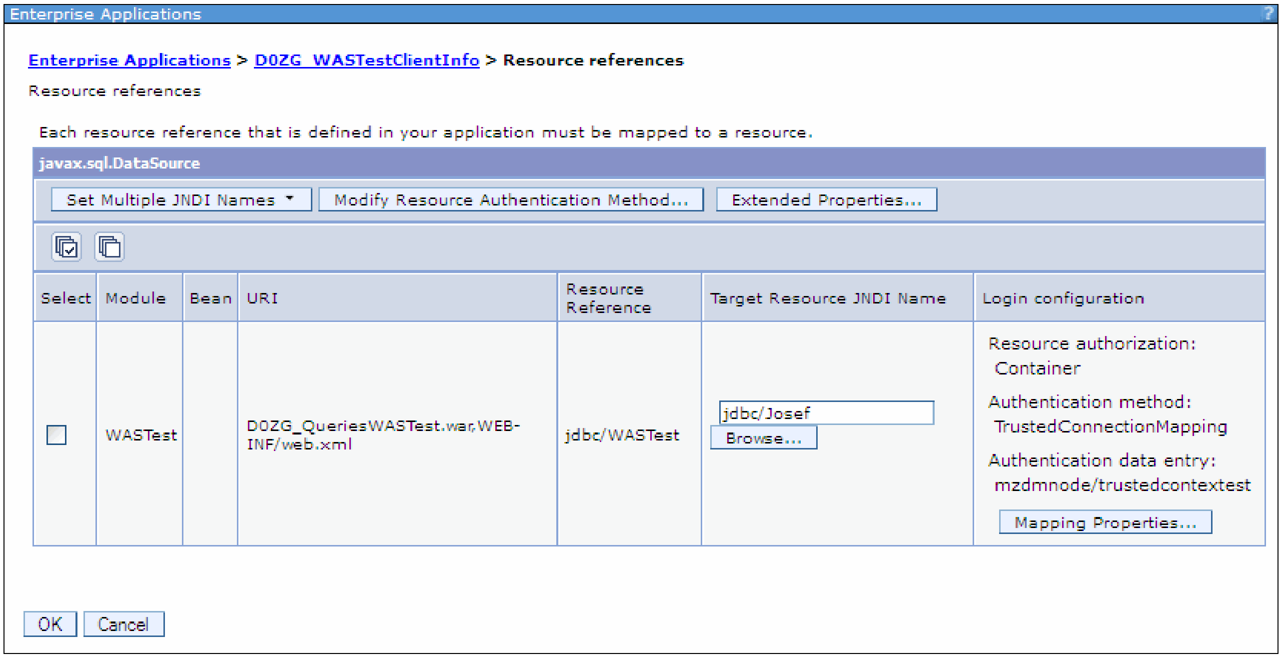


Figure 5-94 Trusted context enabled

9. Click **OK** and save the changes.

5.10 Configuring the JCC properties file in WebSphere Application Server

The IBM Data Server Driver for JDBC and SQLJ has many configuration properties. These properties apply to different application requirements. These properties can mostly be configured as WebSphere Application Server data source custom properties. There are a few properties that are considered global properties that can be specified only in a properties file, which is at the JVM level. This means that these properties apply to all the data sources in the WebSphere Application Server. This section describes how to define this property file for WebSphere Application Server.

Complete the following steps:

1. In the navigation window of the administration console of WebSphere Application Server, expand **Server Types**, as shown in Figure 5-95.

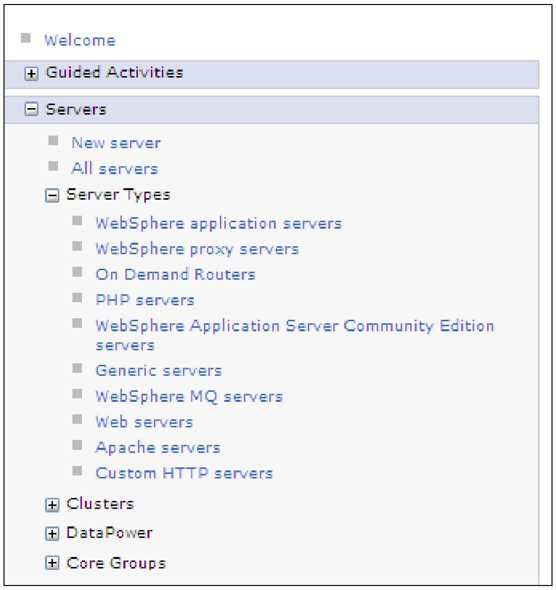


Figure 5-95 Administration console of WebSphere Application Server

2. Click **WebSphere Application servers** and the window Figure 5-96 opens and displays the servers that are defined in the environment. In the example environment, we had three servers. We focus on the MZSR014 server.

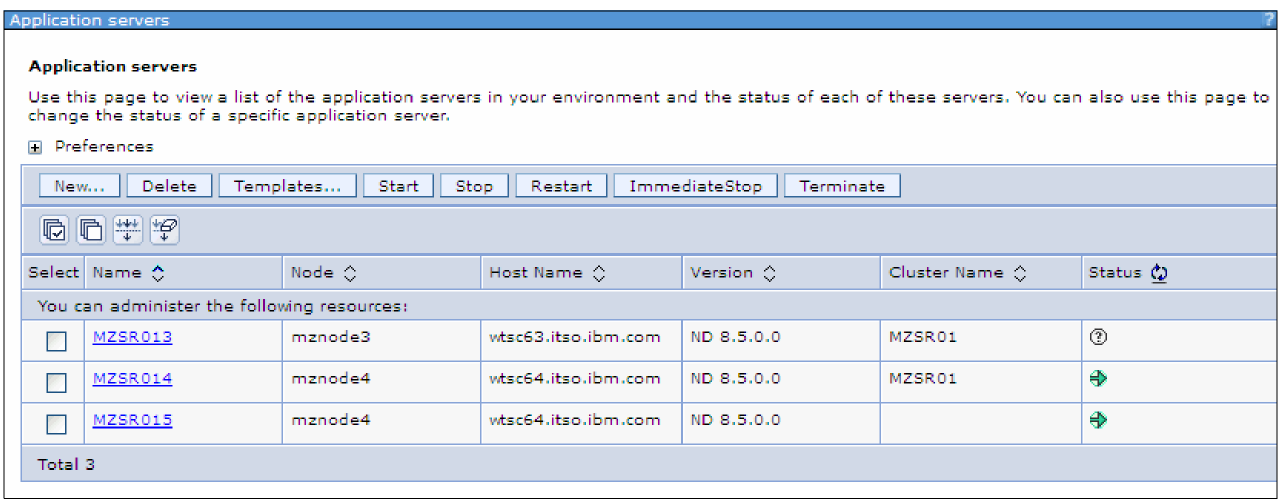


Figure 5-96 List of available servers

3. Click **MZSR014** and the window that is shown in Figure 5-97 opens.

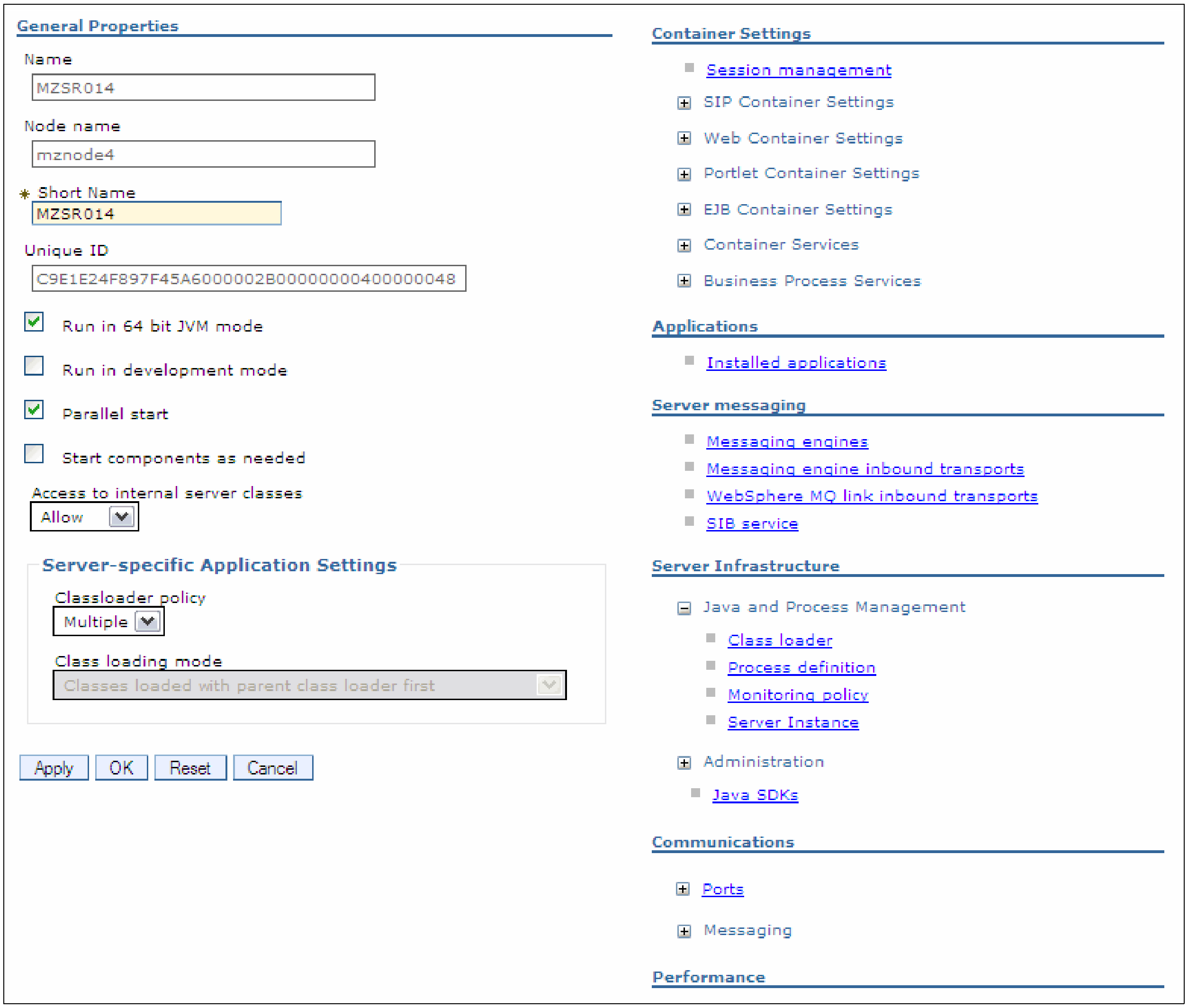


Figure 5-97 Properties of the MZSR014 server

4. Expand **Java and Process Management**. Click **Process definition**. The window that is shown in Figure 5-98 opens. This window is specific to WebSphere Application Server on z/OS.

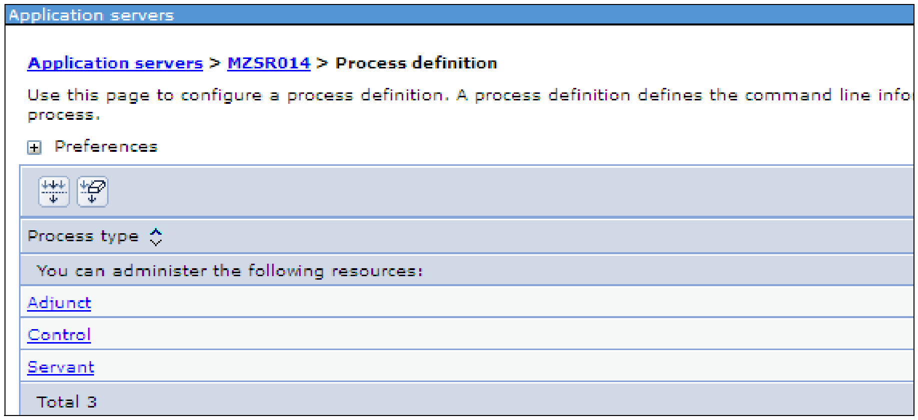


Figure 5-98 Server process definition

5. Click **Servant** and the window that is shown in Figure 5-99 opens.

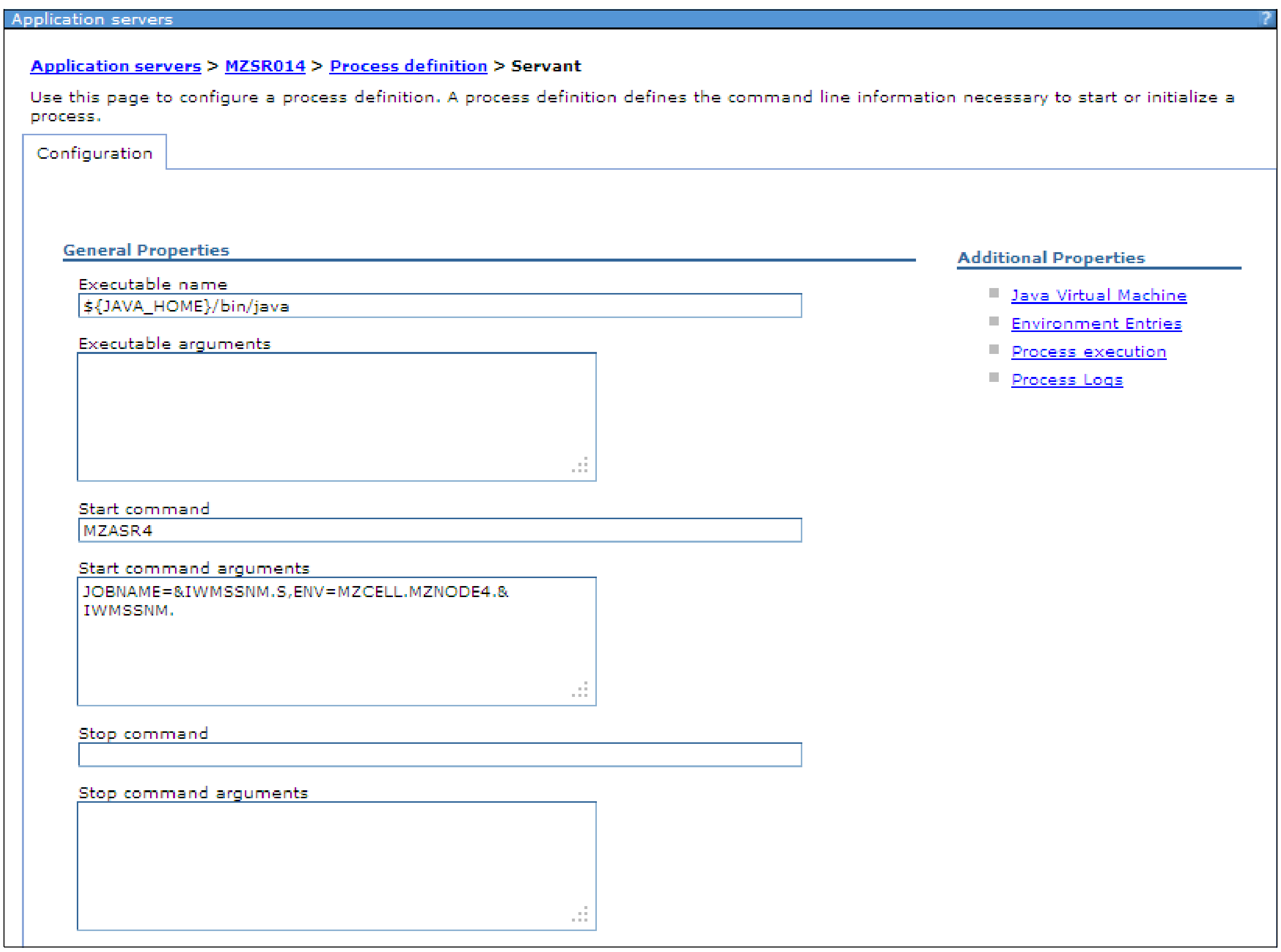


Figure 5-99 Configuring the process definition of the application server

6. Click **Java Virtual Machine** and the window that is shown in Figure 5-100 opens.

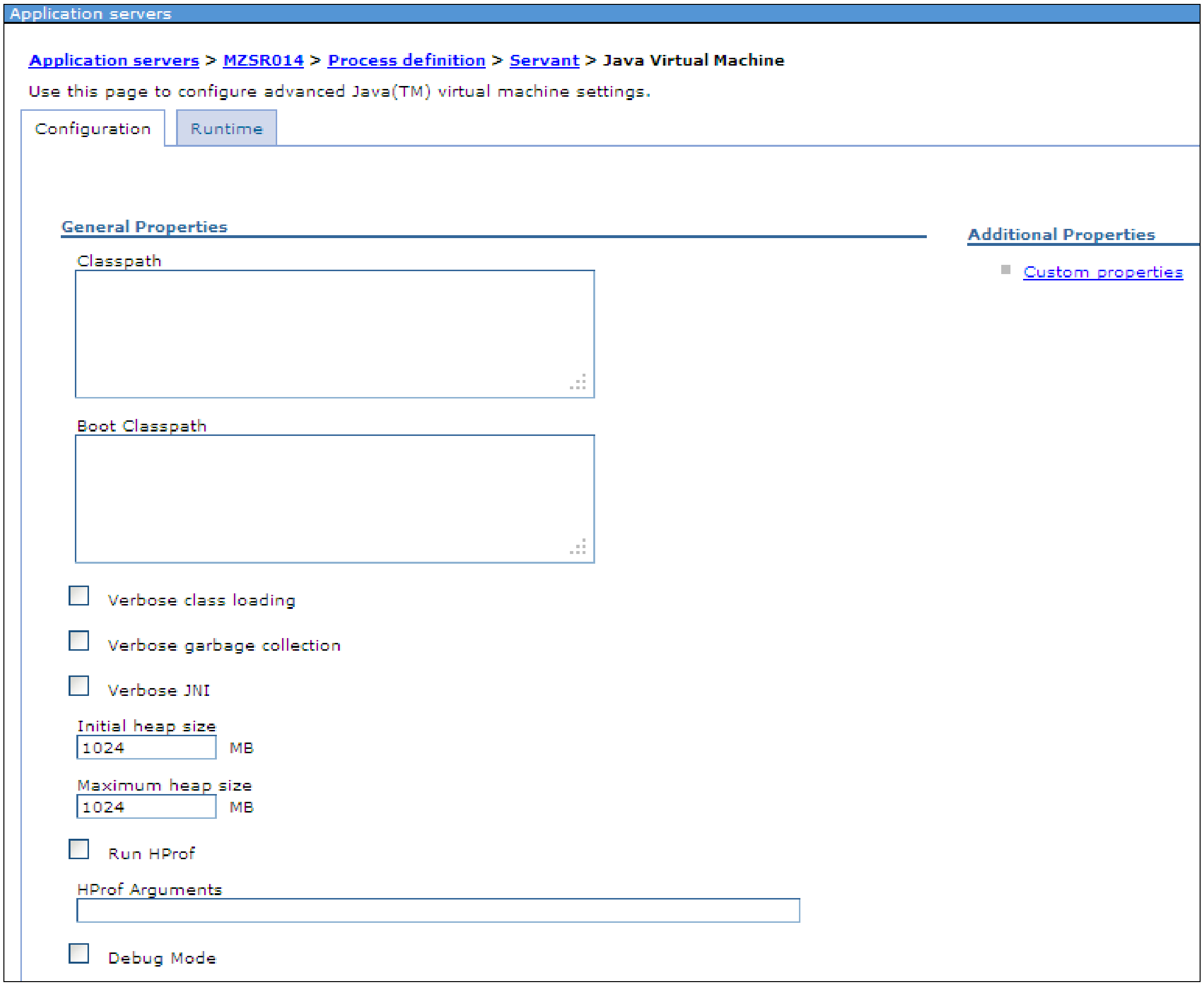


Figure 5-100 Java Virtual Machine for the application server

7. Click **Custom properties** and the window that is shown in Figure 5-101 opens.

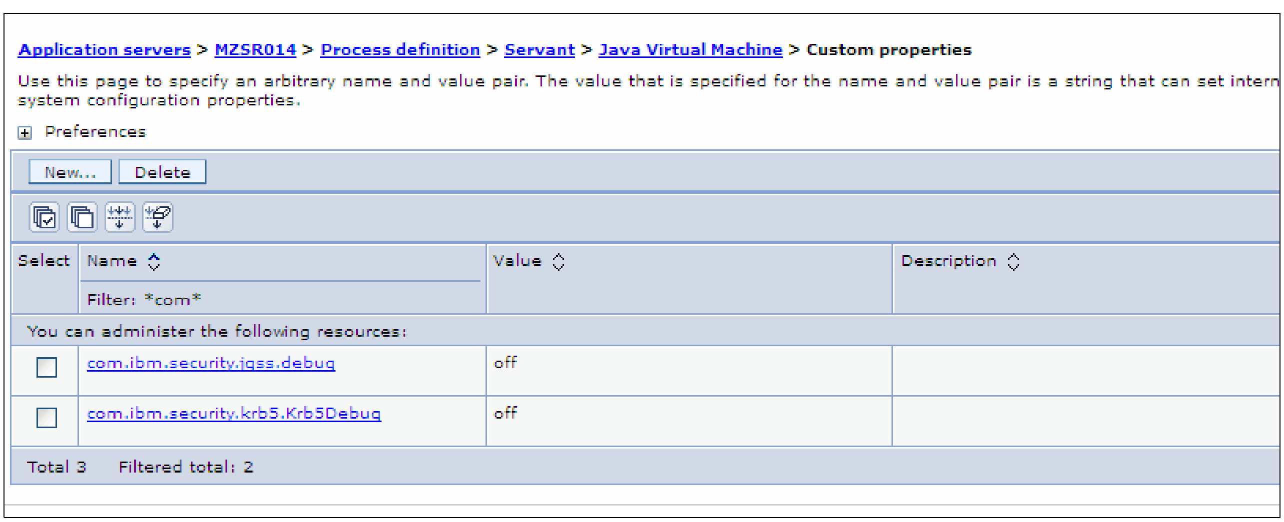


Figure 5-101 JVM custom properties

8. Click **New** and the window that is shown in Figure 5-102 opens. In the name field, enter db2.jcc.propertiesFile. In the value field, enter the location of the properties file. In our example, the properties file is named `jcc.properties`. It is stored in `/u/rajesh`.

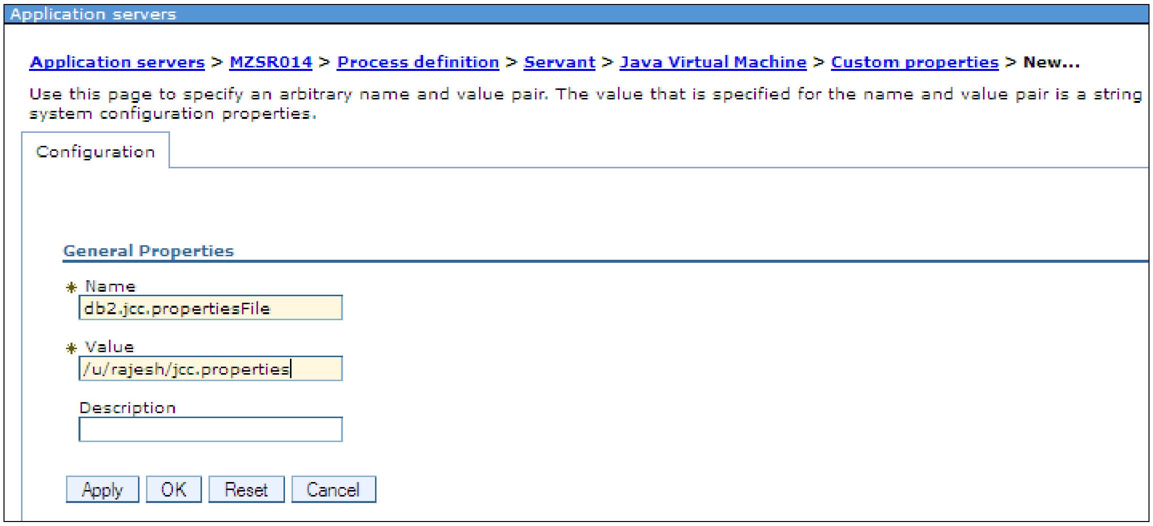


Figure 5-102 New custom property for JVM

Click **Apply**, then **OK**, and then save the changes. The window that is shown in Figure 5-103 opens.

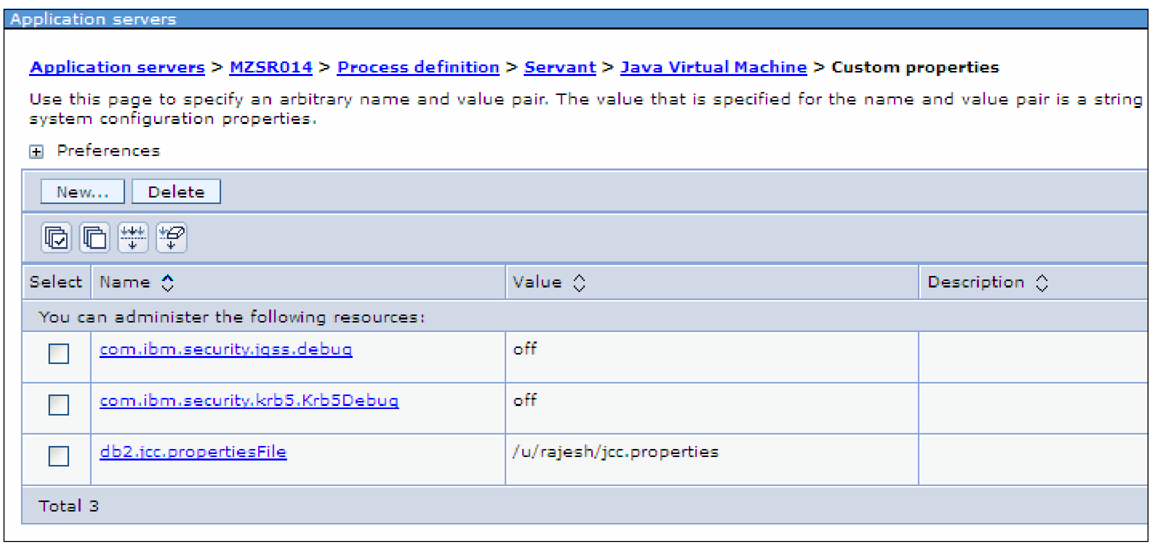


Figure 5-103 Application server defined

9. Now enter any required properties in the `jcc.properties` file and restart the server. You can validate that the `jcc.properties` file was acquired by looking at the following server log:

```
Trace: 2012/10/04 00:37:31.677 02 t=7E3AE8 c=UNK key=P8 tag= (13007004)
  SourceId: com.ibm.ws390.orb.CommonBridge.printProperties
ExtendedMessage: BBOJ0077I:  db2.jcc.propertiesFile = /u/rajesh/jcc.properties
```

When you see the message, you know that the server acquired the `jcc.properties` file.

5.11 Configuring data source properties (webSphereDefaultIsolationLevel, currentPackagePath, pkList, and keepDynamic)

In this section, we show how to set the following properties at a data source level in WebSphere Application Server:

- webSphereDefaultIsolationLevel
- currentPackagePath (for JDBC type 4 connection)
- pkList (for JDBC type 2 connections)
- keepDynamic

5.11.1 webSphereDefaultIsolationLevel

Complete the following steps:

1. In the navigation window of the administrative console of the WebSphere Application Server, expand **Resources** and click **Data Sources**, as shown in Figure 5-104.

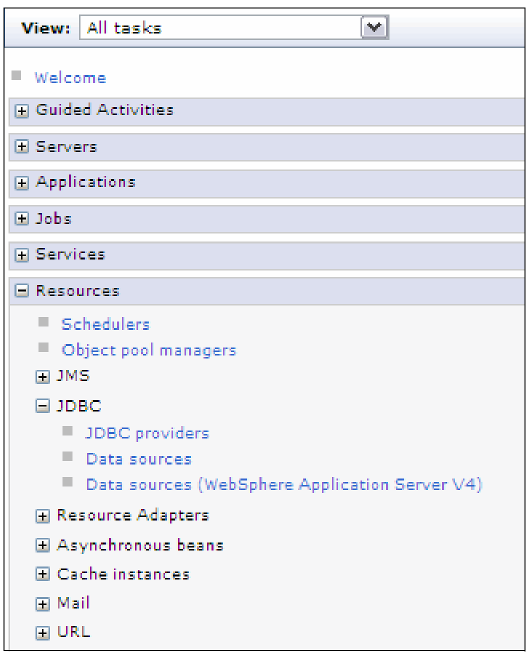


Figure 5-104 Administrative console of the WebSphere Application Server

The window that is shown in Figure 5-105 opens. This window shows a list of existing JDBC data sources that are defined in your environment.

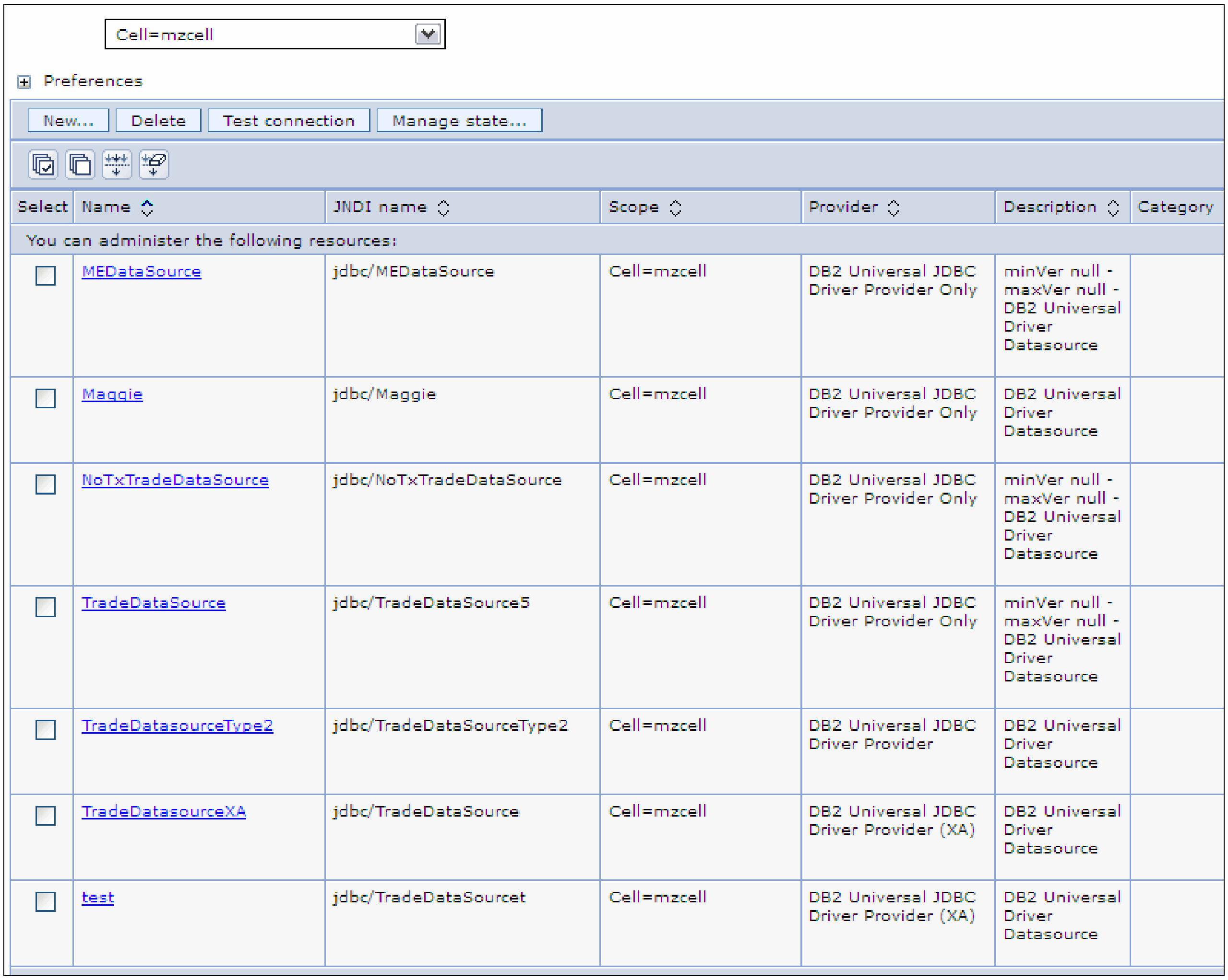


Figure 5-105 List of existing JDBC data sources

2. Select the data source on which you want to set the property. In this example, we select **TradeDatasourceXA** and the window that is shown in Figure 5-106 opens.

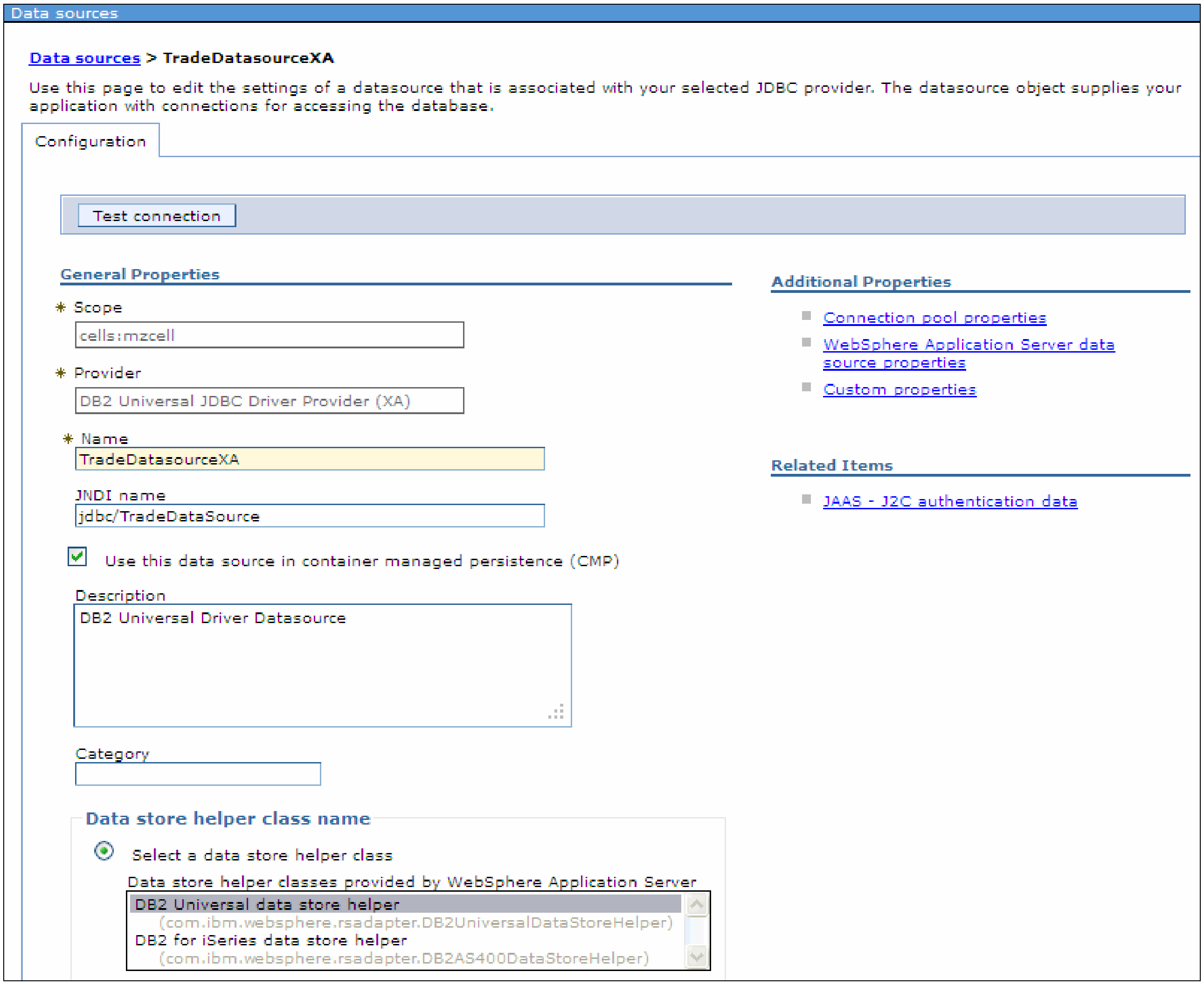


Figure 5-106 Data source TradeDatasourceXA

3. Click **Custom properties**. The window that is shown in Figure 5-107 on page 291 opens and lists all the custom properties that are available.

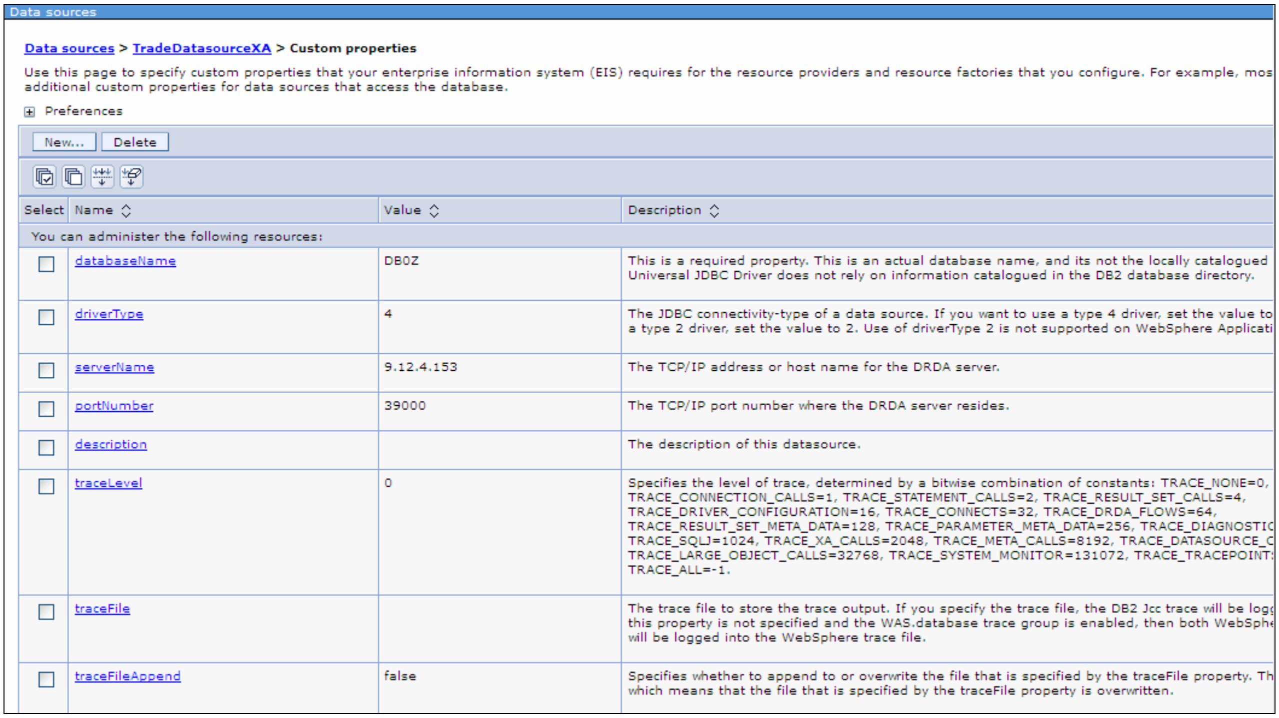


Figure 5-107 List of the custom properties

The webSphereDefaultIsolationLevel custom property is available by default, but the default value is not set. WebSphere Application Server uses JDBC TRANSACTION_REPEATABLE_READ, which maps to read stability (RS) in DB2 by default. Applications should choose the appropriate isolation level. In our example, we chose to use TRANSACTION_READ_COMMITTED, which maps to cursor stability (CS) in DB2, as shown in Figure 5-108.

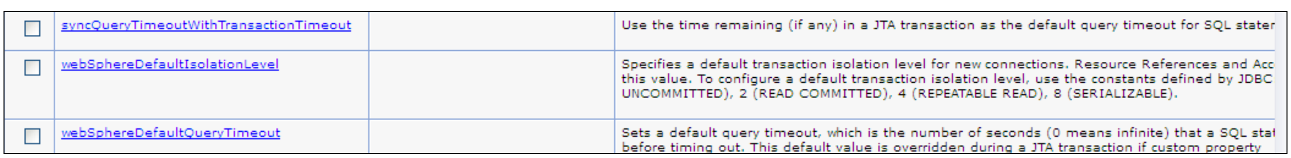


Figure 5-108 Isolation level definition

4. Click **webSphereDefaulIsolationLevel** and the window shown Figure 5-109 opens.

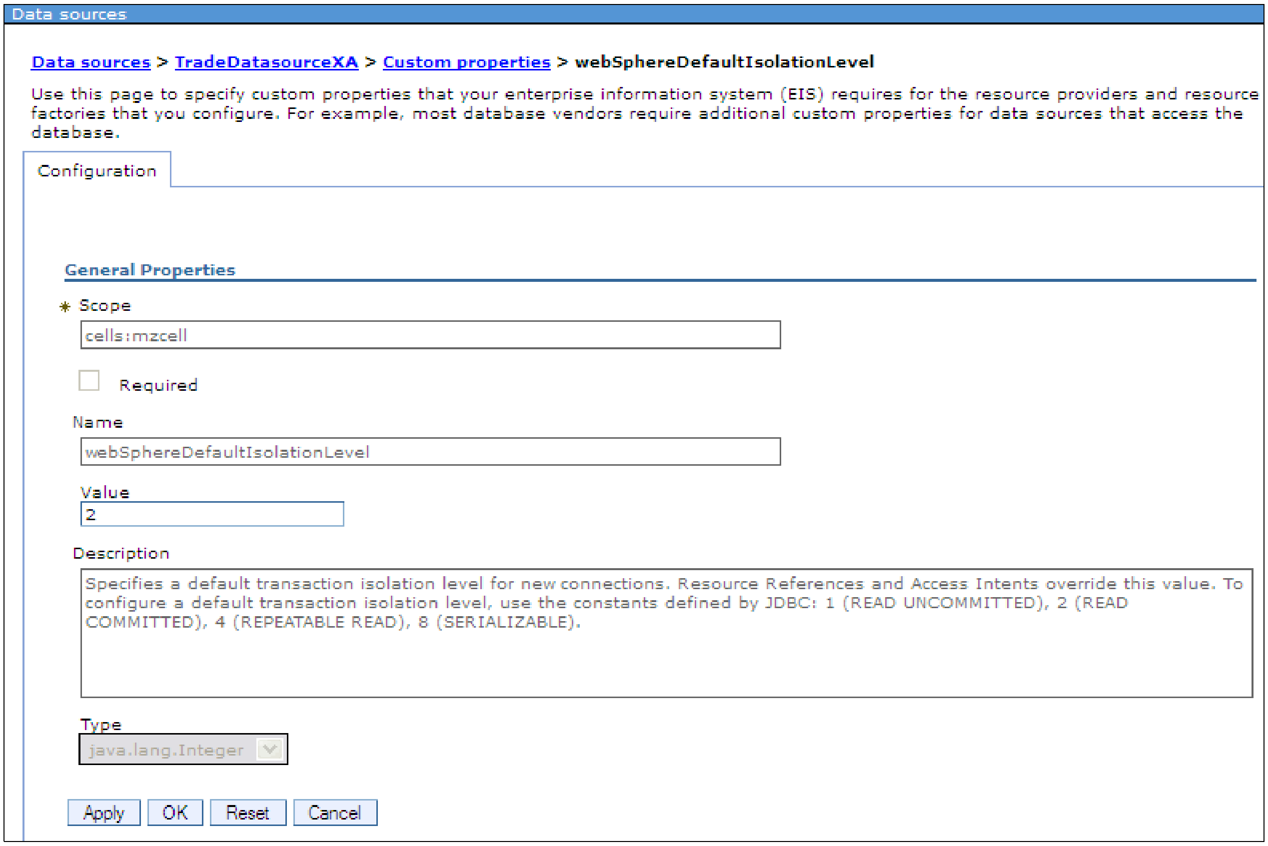


Figure 5-109 Custom property for default isolation level

5. Enter 2 for the value, which sets cursor stability in DB2. Click **Apply**, then **OK**, and then save the changes.

5.11.2 currentPackagePath

The currentPackagePath custom property is also available by default in WebSphere Application Server. It does not have any value, as shown in Figure 5-110. This property should be used under the following conditions:

- JDBC type 4 connectivity is used to connect to DB2 for z/OS.
- The application has multiple packages that must be accessed and those packages are bound to different collections.

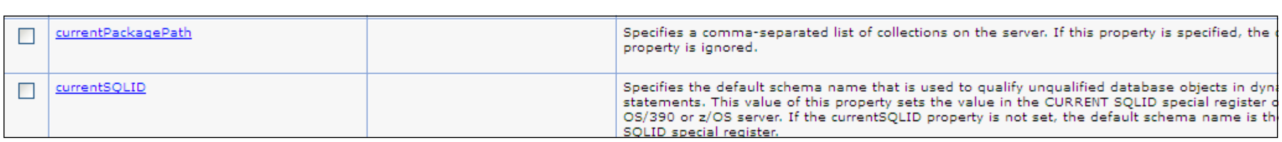


Figure 5-110 No default for currentPackagePath

Click **currentPackagePath** and the window that is shown in Figure 5-111 opens. Enter a comma-separated collection of names. In this example, the application used packages that were bound to collections MYCOLL1 and MYCOLL2.

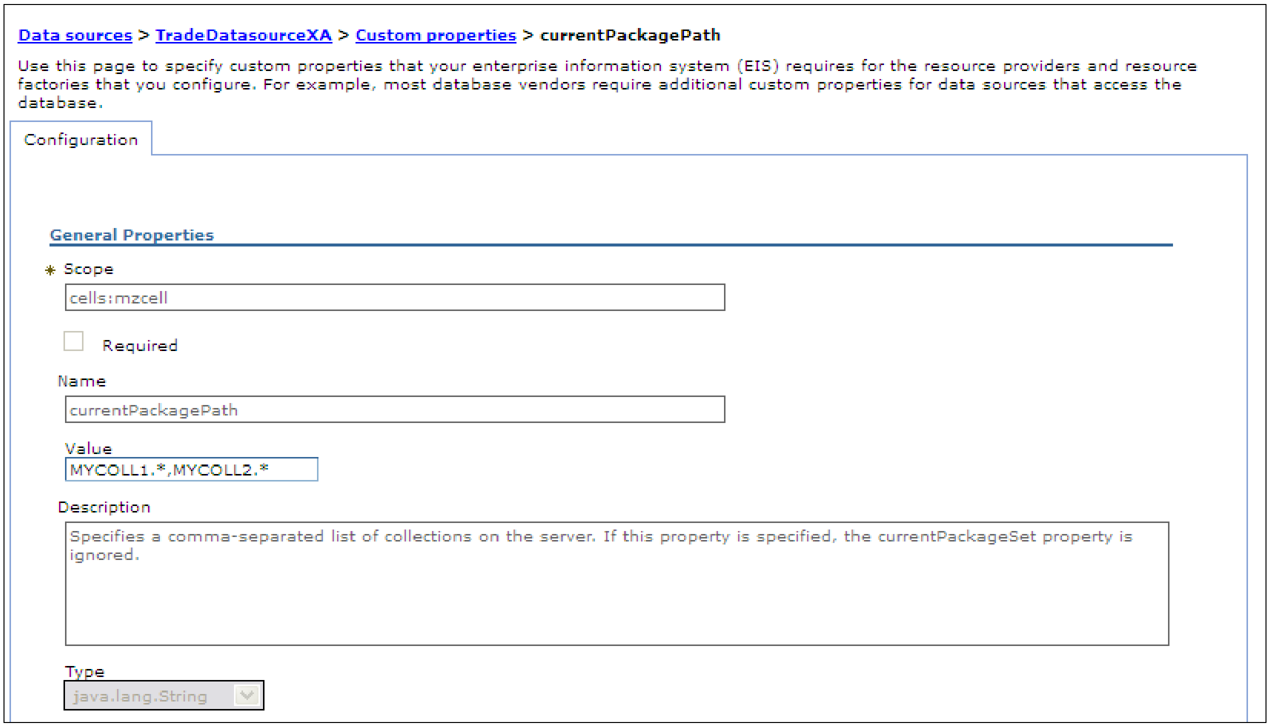


Figure 5-111 currentPackagePath

Click **Apply**, then **OK**, and then save the changes.

5.11.3 pkList

The pkList custom property is not available by default in WebSphere Application Server. This property should be used under the following conditions:

- JDBC type 2 connectivity is used to DB2 for z/OS.
- The application has multiple packages that must be accessed and those packages are bound to different collections.

Click **New** and the window that is shown in Figure 5-112 opens. Enter pkList for the name. Enter a comma-separated collection of names for the value. In this example, the application used packages that were bound to collections MYCOLL1 and MYCOLL2.

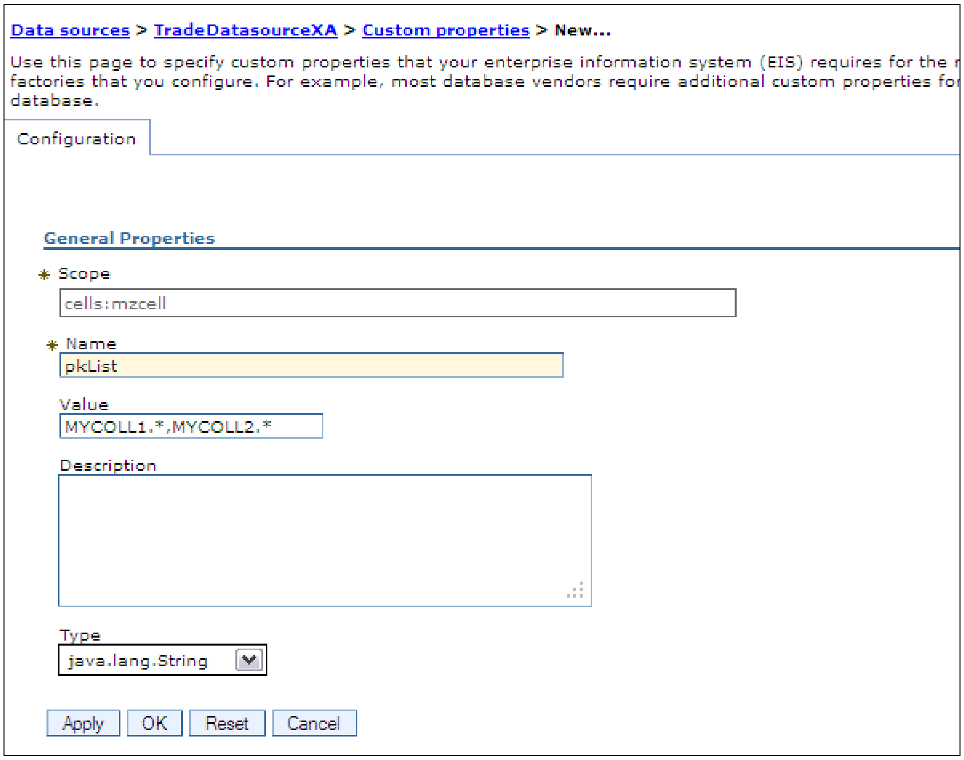


Figure 5-112 pkList

Click **Apply**, then **OK**, and then save the changes.

5.11.4 keepDynamic

The keepDynamic custom property is not available by default in WebSphere Application Server. The default behavior in WebSphere Application Server is to not use this property. This property should be used when you want to use a *local cache* in DB2, as shown in Figure 5-113.

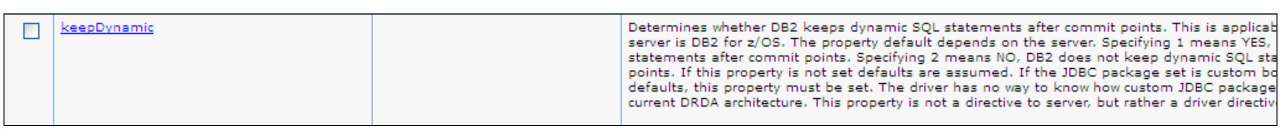


Figure 5-113 Property keepDynamic

For more information about keepDynamic, see “WebSphere Prepared Statement Cache and DB2 KEEPDYNAMIC option” on page 60.

Click **keepDynamic** and the window that is shown in Figure 5-114 opens. Enter a value of 1 to use the keepDynamic feature in DB2.

Figure 5-114 Custom property keepDynamic

Click **Apply**, then **OK**, and then save the changes.

6

Developing Java applications with DB2 for z/OS

This chapter provides an overview of DB2 10 for z/OS support for Java and describes selected topics about how to use that support. This chapter looks at DB2 support for drivers for Java applications and explains the design principles of dynamic and static SQL. This chapter also shows how to define and configure the IBM DB2 Driver for JDBC and SQLJ in various situations and demonstrates the usage of pureQuery for optimization of dynamic SQL.

This chapter also provides a short running sample of a stand-alone application that uses DB2 with JPA and explains the differences of stand-alone Java applications and applications in managed environments.

This chapter demonstrates how to get a good dynamic statement cache hit ratio and describes locking.

This chapter covers the following topics:

- Drivers for Java applications
- Dynamic SQL
- Static SQL
- PureQuery optimization
- DB2 support for Java stand-alone applications
- JDBC applications in managed environments
- Coding practices for a good DB2 dynamic statement cache hit ratio
- Locking

6.1 Drivers for Java applications

With DB2 support for Java, you can access relational databases from Java application programs. This is done by a driver that implements the Java Database Connectivity (JDBC) 4.0 standard, which is defined by the Java Specification Requests (JSR) 221. JDBC defines the standard application programming interface (API) for accessing relational database systems, such as DB2, from Java. Although it is used decreasingly and directly by programs because of the advent of more generic persistency frameworks such Hibernate and OpenJPA, this API is the fundamental building block for writing DB2 Java applications.

For more information about the specification, go to the following website:

http://www.jcp.org/en/jsr/detail?id=221

JDBC drivers are client-side adapters (although they could be clients in a server) that convert requests from applications through the usage of the API to a protocol that the database understands.

JDBC implementations normally implement two specification types:

- Type 2

  Drivers that are written partly in the Java programming language and partly in native code. They have no network connection and communicate with the database through interprocess communications. Their native code must be installed in the file system on the same machine as the database and can be used only to connect to a local database manager. The driver installation in this case is part of the database installation process. The application run time is notified about the location of the installed native code by looking at the LIBPATH environment variable. The native code can effectively work together with other components of the operating system, such as Workload Manager (WLM).

- Type 4

  Drivers that are written solely in Java and connect through TCP/IP to a local or remote database. However, even a connection to a local database remains a remote network connection.

The other JDBC types are not important regarding Java development. The type of a driver should not be confused with the specification level it implements. Type 4 means network driver and JDBC 4.0 means specification level 4.0.

IBM Data Server Driver for JDBC and SQLJ is a single driver that includes JDBC type 2 and JDBC type 4 behavior and that implements JDBC 4.0 and JDBC 3.0. Which type or version is used depends solely on the configuration options that are made while opening the connection to the database.

From an application point of view, there is no difference between the two types. The API is the same. The Java part of both drivers must be available to application clients in the class path. The application can make type 2 and type 4 connections by using this single driver instance. Type 2 and type 4 connections can be made concurrently.

To work with DB2 for z/OS, the license file `db2jcc_license_cisuz.jar` must be in the class path.

More information about the driver architecture and its configuration options can be found in Chapter 3, “DB2 configuration options for Java client applications” on page 81.

With the `sqlj4.zip` file in the class path, the IBM Data Server Driver for JDBC and SQLJ provides SQLJ functions that include JDBC 4.0 and later functions, and JDBC 3.0 and earlier functions.

6.2 Dynamic SQL

Since its first release with JDK 1.1 in 1997, the JDBC API as a generic database access technology was intended for dynamic SQL. Even the selection of a specific database is done dynamically because the JDBC driver for that database is assigned and loaded only at run time. According to this programming model, dynamic SQL statements are constructed and prepared also only at run time. They are not known to the application server or the database in advance. Sometimes even the programmer does not know what the results of a dynamic SQL generation routine will be exactly. The dynamic SQL string building can be bypassed and the SQL can be presented to the API as a constant.

Whether hardcoded or generated, the result is always a string with an SQL statement that then is given to the appropriate JDBC API. Both methods are considered dynamic because the database does not know the SQL in advance either way. The generation process can encompass the generation parameters as well or it can include parameter markers (question marks) that can be substituted later through an API call.

In the Java community, programming with dynamic SQL is the prevailing method. JDBC implements the dynamic SQL model. A major advantage is that application development is faster than with other techniques. All database vendors include a JDBC driver in their databases, making JDBC a universal technique that is known to almost every programmer. Although JDBC always uses the same programming principles, it does not allow a fully portable program. Among other advantages, persistency frameworks such as Hibernate or JPA address the portability problem. But even in the form of that new persistency layer, the underlying design schema remains dynamic SQL handled by a JDBC driver.

Although dynamic SQL with raw JDBC API statements is being used less often, there are some situations where it is the most suitable solution:

- The table structure is too complex for JPA entities.
- No entities are involved (for example, in mass updates).
- You are using maintenance or administrative programs.
- When the persistency framework is not powerful enough.

A short code snippet shows the principles of JDBC API coding. We do not go into too much detail because JDBC programming is widely known. Instead, we describe some important design issues that are relevant to other parts of this book.

As you can see in Example 6-1, the DriverManager.getConnection method with its parameters connects to the database. We could have used the Datasource interface as well, if we had used a predefined data source. We then ask the connection object to return a preparedStatement. Afterward, we present the SQL to the statement, leaving two places unclear. We code a “?” as a parameter marker instead. The parameter markers are filled afterward with a concrete value.

Example 6-1 Example of using PreparedStatement.executeQuery

```
Connection con = DriverManager.getConnection(url, properties);
PreparedStatement pstmt;
...
pstmt = con.prepareStatement(
   "UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?");
```

```
pstmt.setString(1,"4657");    // Assign value to first parameter
pstmt.setString(2,"000010");  // Assign value to second parameter
numUpd = pstmt.executeUpdate(); // Perform update
pstmt.close();
```

Then, the prepared statement is run by the database.

In addition, caches are filled. If you ran the statement inside WebSphere Application Server, the JVM's prepared statement cache is filled or, if the statement was already ran, you get a created statement object. The statement object is built around the statement `"UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?"`. The statement can have different parameters and a different case, but it remains the same statement. The cache includes only the dynamically created Java object for that specific statement. It does not include any DB2 related information.

On the DB2 side, a cache entry is also created if DB2 is defined that way. No Java object is stored, but access strategy-related information is stored. Both caches complement each other.

As an alternative, you can generate a complete SQL string without placeholders for the parameters. It would look like the following string:

```
"UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'";
```

This string results in a new Java statement object and new objects for other employees or phone numbers because you have a new statement instead of a parameter substitution. You can give this string to a prepareStatement for execution, But using a simple createStatement is sufficient, as shown in Example 6-2.

Example 6-2 Example of using createStatement

```
stmt = con.createStatement();
numUpd = stmt.executeUpdate(
   "UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'"
);
```

Only parameter markers allow DB2 to use the dynamic statement cache. Otherwise, a dynamic rebind for the mini-plan must be made. As of DB2 10, there are some additional capabilities to caching, as described in 6.7.4, "Literal replacement" on page 330.

6.3 Static SQL

Application development with DB2 with compiled languages is mostly done according to a static SQL model. Static means that the SQL statement is fixed at development time and known to the database. Only call parameters are variable at run time.

SQLJ, the name for static SQL in Java, is based on JDBC APIs by using embedded SQL to access the database. The database normally uses static SQL, but can use dynamic SQL in some cases. Because static SQL prepared in advance, performance is better compared to dynamic SQL. By contrast, dynamic SQL is not known by the system at compile time; parsing, validation, preparation of statements, and determination of the access path in the database is done only at run time. Errors or poorly performing statements might remain undetected until problems in production occur.

With SQLJ, the SQL statements are not part of the Java language. They are marked with `#sql` in the Java source code, but must be extracted before the Java compiler sees them or they cause Java compile errors. Therefore, the Java class is edited in a `<name>.sqlj` file that is then processed by the SQLJ translator. How an SQLJ statement is embedded in to the Java source code is illustrated in Example 6-3.

Example 6-3 Sample of an SQLJ statement

```
MyContext context = new MyContext();
String empno = "000021";
#sql [context] {
   SELECT FIRSTNME
      INTO :firstname
   FROM EMP WHERE EMPNO = :empno
   }
return firstname;
```

The SQLJ translator is the **sqlj** command. By default, It is in `/usr/lpp/db2/jdbc/bin` on z/OS and in `<install root>\SQLLIB/bin` on Windows. It replaces all `#sql` statements with generated Java code and creates new `<name>.java` files. The SQL is placed into SQLJ serialized profiles, which are `<name>.ser` files with extracted SQL that are used by the **db2sqljcustomize** utility. This utility creates (authorization is provided) four packages by default, one for each isolation level in the target DBRM.

This process can detect potential errors early.

Here are the advantages of static SQL:

- SQLJ commands are shorter and easier to read than dynamic SQL commands.
- The syntax is checked at compile time. Errors can be repaired early in the development process.
- Query results are type checked.
- Static SQL is less vulnerable to malicious SQL injection.
- A query plan is generated, which normally runs faster than dynamic access.
- A DBA and programmer can better interact with each other through the SQL.
- COBOL programmers are used to the programming model.
- Monitoring and tracing are easier because a selection can be made on the package name.
- The security model for SQLJ is different than for dynamic SQL. The connect user ID must be authorized for every DB2 object that is used by a program with dynamic SQL. If many different authorization IDs are used, this leads to many grants in DB2, making the security model disordered. With static SQL, authorization is made is on the package and not on the DB2 objects that are used by the package. This allows a more granular approach.

There are some SQLJ sample programs that come with the DB2 product. They are in `<install root>\SQLLIB\samples\java\sqlj` for DB2 on Windows or in `/usr/lpp/db2/samples/java` on z/OS.

Despite all these advantages, only a few Java projects use SQLJ. The more complex application build process might be one reason why this is so. Another reason is that SQLJ remains basically JDBC and has no support for object-relational mapping (ORM). A programmer's productivity and application maintainability seem to be more important for many projects than advantages in performance and security.

Here are the disadvantages of the static SQL model:

- More build steps are necessary, which might span multiple departments and areas of responsibility.
- Good support is needed, for example, the SQLJ editing tools with IBM Rational Application Developer or IBM Data Studio. For Maven, the build manager for Java projects, some additional build steps must be included.
- The SQLJ programming model might not be known by Java programmers. Only few samples exist.
- There is no support for ORM.
- Portability is reduced because not every database supports static SQL.

WebSphere Application Server offers support for static SQL for Enterprise Java Beans (EJB) 2.x and later entity beans with the **`ejbdeploy`** SQLJ option. In EJB 3.0 and later, container-managed (CMP) Enterprise beans are replaced by JPA entities. Although EJB 2.x could be used in later versions of WebSphere Application Server, it is unlikely.

With JPA, this feature is offered through pureQuery and offers you the advantages of both dynamic and static SQL.

6.4 PureQuery optimization

PureQuery is a high performance Java data access platform that helps manage applications that access data.

It has the following features:

- APIs that are built for ease of use and to simplify the usage of preferred practices.
- Development tools, which are delivered in IBM Data Studio full client, for Java and SQL development.
- A run time for optimizing database access and simplifying management tasks.

All three features can be used or omitted independently from each other.

pureQuery provides an alternative set of APIs. They are similar in concept to JPA and can be used instead of Java Database Connectivity (JDBC) to access DB2. Even if these APIs are not used in your application, the pureQuerys Client optimization feature makes it possible to take advantage of static SQL for existing JDBC applications without modifying existing dynamic source code.

Figure 6-1 on page 303 shows the flow between pureQuery and the database.

The general concept is to collect all dynamic SQL statements of your application at development or deployment time by using pureQuery. The application developer does not need to be involved in this process. The collected statements are then bound into packages in the database. At execution time, the pureQuery run time uses the static SQL from the packages instead of the dynamic SQL to work with DB2. Where dynamic SQL statements cannot be collected or converted, the run time continues to use dynamic SQL.

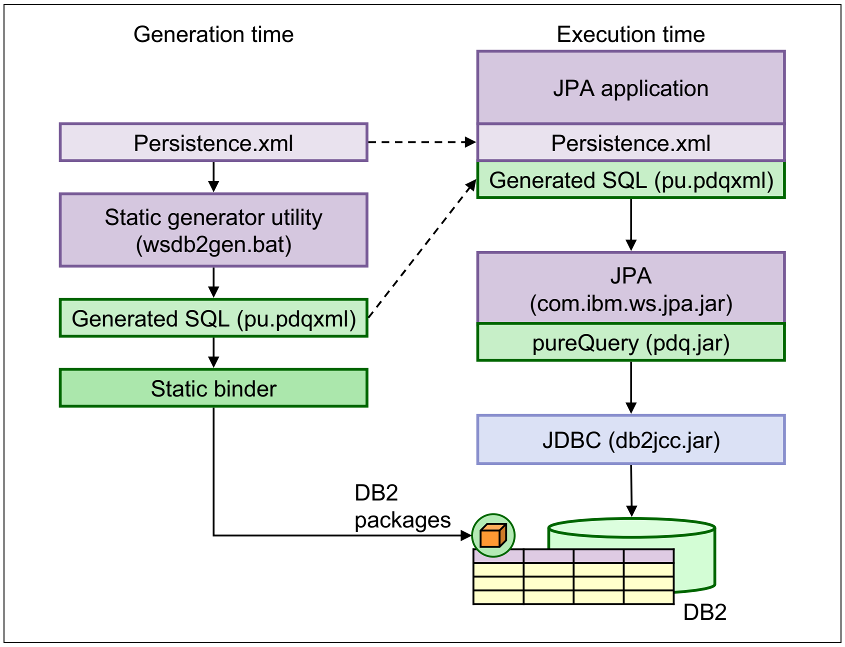


Figure 6-1 The flow from application to database using pureQuery

There are two ways of collecting the SQL:

- If JPA is used, then WebSphere Application Server or pureQuery is used to examine persistence-units in the `persistence.xml` file of an application module. Only SQL from named queries is detected.
- SQL statements can be traced and captured at run time. All SQL statements are detected.

To understand the functionality, look at the way SQL is collected for JPA. Either a command or IBM Data Studio can be used.

The **`wsdb2gen`** command is in the `/bin` directory of WebSphere Application Server. To run it, extend the WebSphere class path by using the `pdq.jar`, `pdqmgmt.jar` and `db2jcc4.jar` files that come with IBM Data Studio. A sample command is shown in Example 6-4.

Example 6-4 An example wsdb2gen command

```
wsdb2gen -pu jpa_db2 -url jdbc:db2://d0zg.itso.ibm.com:39000/DB0Z -user
 db2r1 -pw passw0rd
```

The utility uses the persistence unit name as input, along with other parameters, and generates an output file that contains SQL statements that are required by all entity operations, including persist, remove, update, and find. It also generates the SQL statements that are needed in the running of JPA named queries. Other dynamic SQL cannot be found and us not included in the output.

The ANT task **`WsJpaDBGenTask`** provides an alternative to the **`wsdbgen`** command.

The output of the command is a file that contains the persistence unit name followed by a suffix of `.pdqxml`. The `pdqxml` file is written to the same directory as your `persistence.xml` file. Alternatively, by using IBM Data Studio, pureQuery tools can be added to your JPA project.

The jpa_db2_web project (see Appendix I, “Additional material” on page 587) is a small project that illustrates the pureQuery functionality. It simply lists the DB2 Department Table in the SAMPLE database in the browser.

To enable pureQuery support for your project in IBM Data Studio, go to the **Java Perspective** and right-click the jpa_db2_web project. Then, select **Data Access Management** → **Add Data Access Development support**. The window that is shown in Figure 6-2 opens.

Select the **Add pureQuery support to project** check box, which adds the pureQuery runtime libraries to your build path. The run time has five JAR files with names that start with pdq. The WebSphere Application Server run time must be in the class path as well.

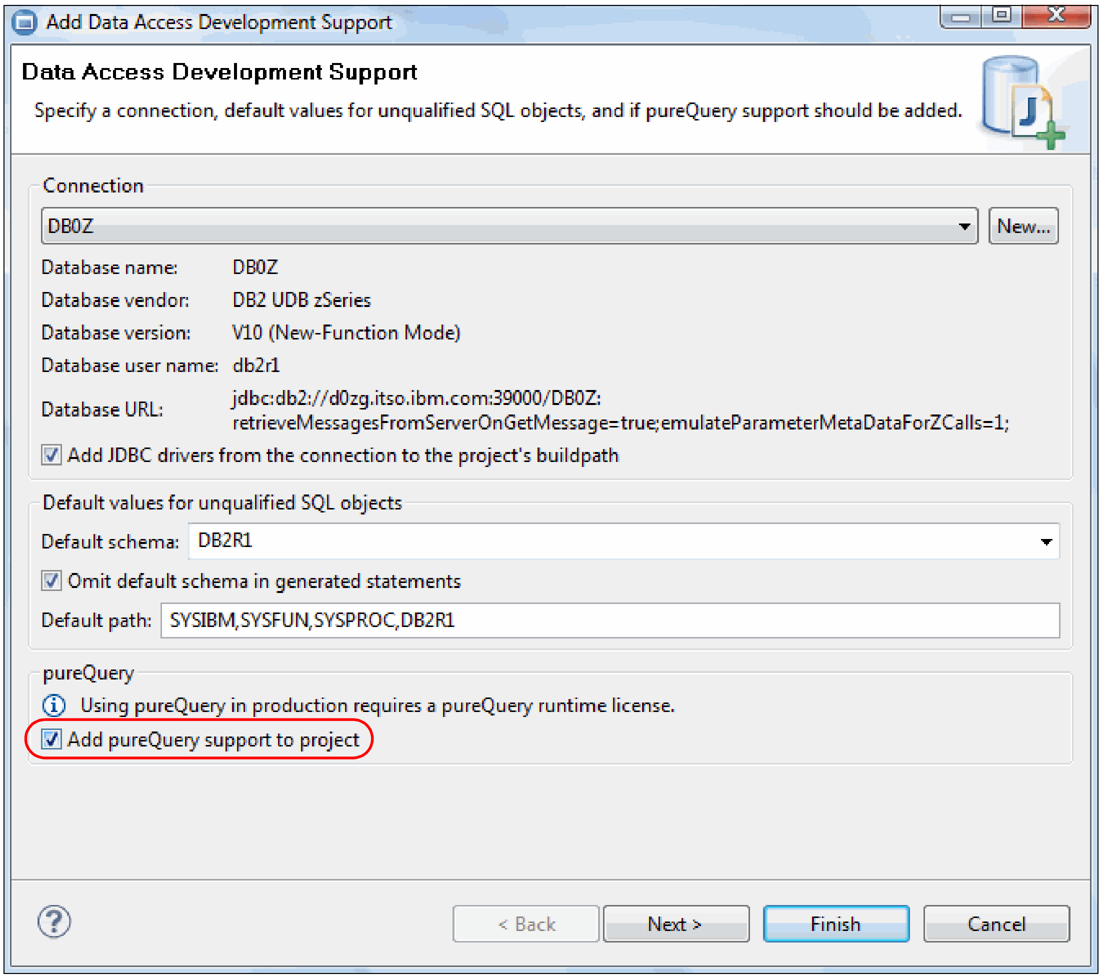


Figure 6-2 Add data access management support to the project

You must define a database connection to the SAMPLE database in this window. It is used to check the SQL statements and prefix table names with the provided schema name in the generated output statements.

The `pdqxml` file then is generated by right-clicking the `persistence.xml` file of your project in the Java Perspective. Then, select **Data Access Development** → **Generate pureQueryXML File**, as shown in Figure 6-3. A file named `jpa_db2.pdqxml`, which is named after the persistence-unit name used in that project, is generated.

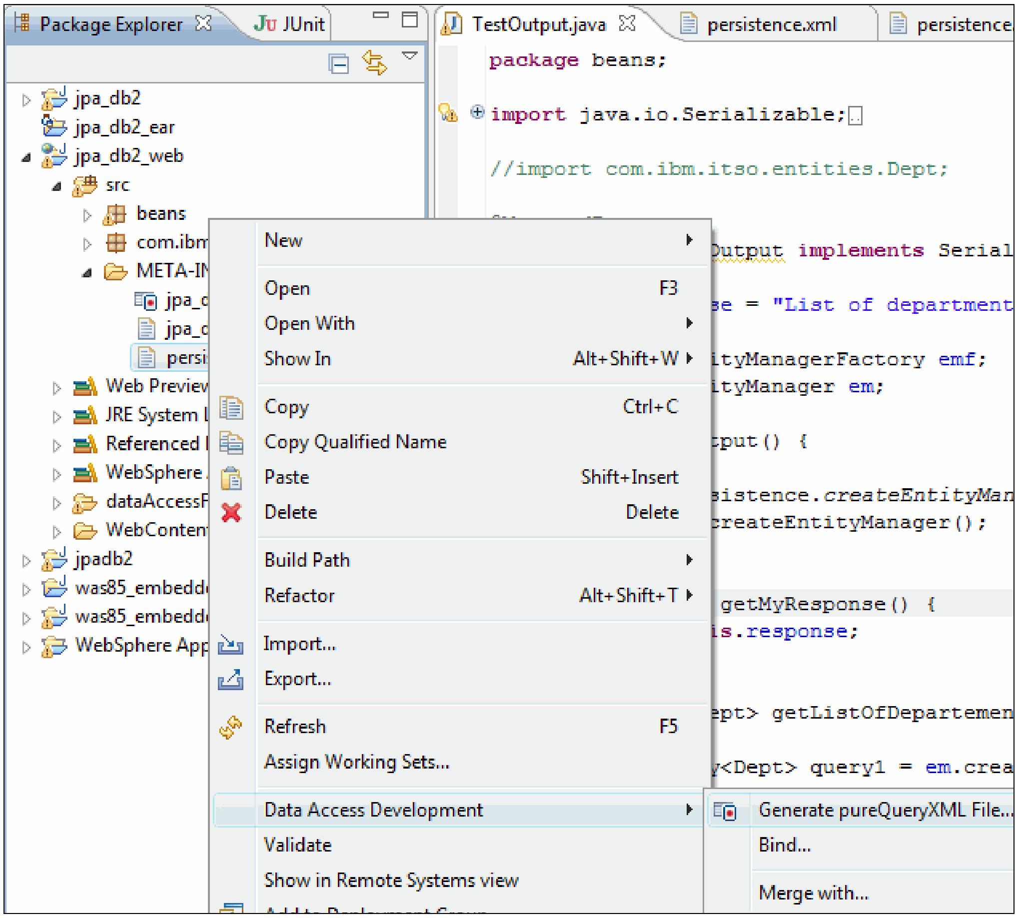


Figure 6-3 Generate pdqxml files with IBM Data Studio

The pdqxml file can be checked afterward by using a special view that is provided by IBM Data Studio, as shown in Figure 6-4. Collected SQL statements can be run against the defined database, changed, or cleared from the bind process. Then, the generated SQL could be optimized in collaboration with the database administrators.

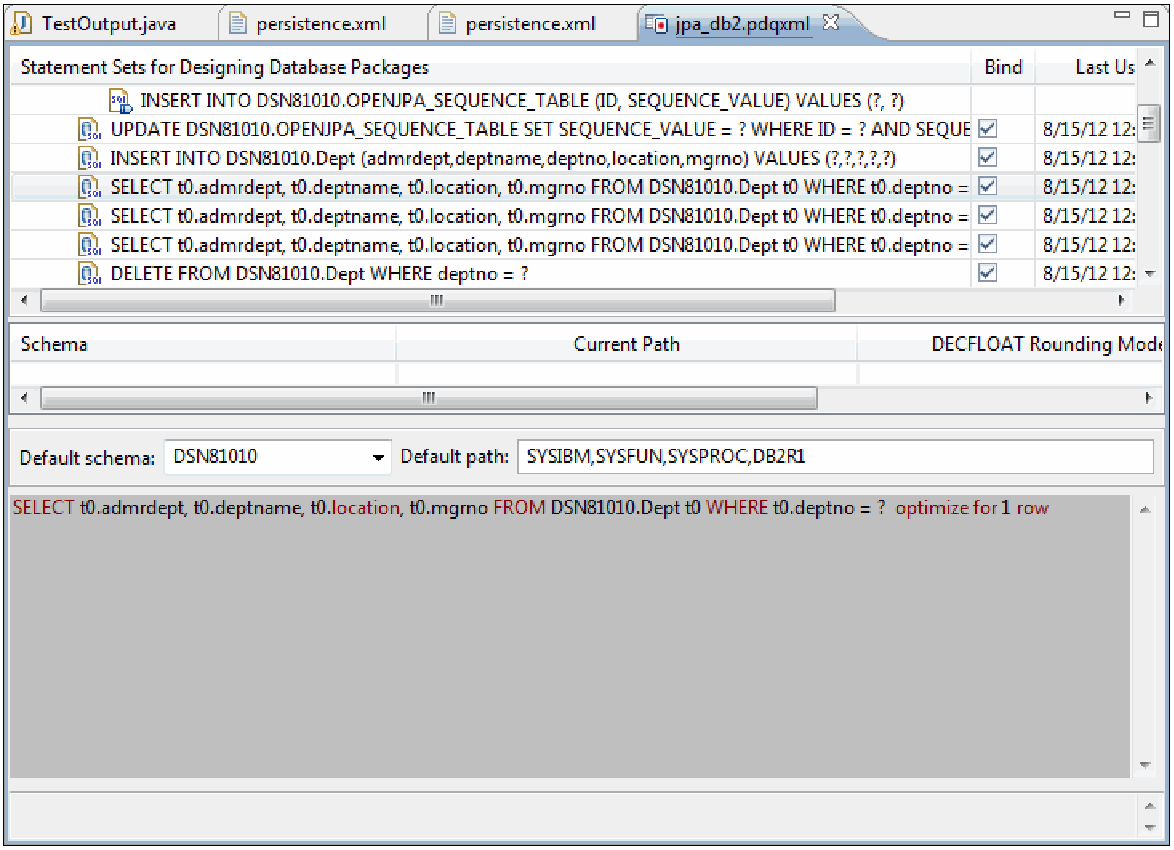


Figure 6-4 Work with jpa_db2.pdqxml after generation

The pdqxml file must be packaged inside your archive file in the same location as the persistence.xml configuration file, usually the META-INF directory of the module.

The application can now be deployed to the server. However, it works with dynamic SQL unless you bind the database packages. To bind the packages, in the WebSphere Application Server console, click **WebSphere enterprise applications**, click the application name, and click **SQLJ profiles and pureQuery bind files**. Alternatively, you can use the **AdminTask** command, as shown in Example 6-5.

Example 6-5 Bind the packages

```
AdminTask.processSqljProfiles('[-appName jpa_db2 -url
jdbc:db2://d0zg.itso.ibm.com:39000/DB0Z -user db2r1 -password ******** -options
-classpath
[/u/db2r1/pureQuery/pdq.jar:/u/db2r1/pureQuery/pdqmgmt.jar:/usr/lpp/db2/d0zg/jdbc/
classes/db2jcc4.jar ] -profiles
[jpa_db2_web.war/WEB-INF/classes/META-INF/jpa_db2.pdqxml ]]')
```

Be sure that you grant execution authority on the package to public or to the user that is defined for the data source in WebSphere Application Server.

The pureQuery integration that is delivered with WebSphere Application Server requires the addition of the Data Studio pureQuery run time to the JDBC provider, as shown in Example 6-6. It must be purchased separately. In the WebSphere environment, you place the pureQuery JAR files `pdq.jar` and `pdqmgmt.jar` in to the DB2 JDBC Driver Provider class path.

Example 6-6 Add the pureQuerey run time to JDBC providers class path

```
${DB2_JCC_DRIVER_PATH}/db2jcc4.jar
${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar
${DB2_JCC_DRIVER_PATH}/db2jcc_license_cisuz.jar
${PUREQUERY_PATH}/pdq.jar
${PUREQUERY_PATH}/pdqmgmt.jar
```

In WebSphere Application Server, you must use a JPA for the WebSphere Application Server persistence provider. Only this JPA uses static SQL support by using the DB2 pureQuery feature. This is the default in WebSphere. The original Apache OpenJPA driver does not support pureQuery optimization. Be sure not to overwrite this default with a `provider` statement in your `persistence.xml` file.

If you ran your application in server MZSR015, you could verify that your SQL is static by activating a trace in the server:

```
/F MZSR015,TRACEJAVA='JPA=all: openjpa=all: SystemErr=all: SystemOut=all:
com.ibm.pdq=all'.
```

Reset the trace by running `/F MZSR015,TRACEINIT`.

You can find another pureQuery optimization example at the following website:

http://www.ibm.com/developerworks/websphere/techjournal/0812_wang/0812_wang.html#resources.

6.5 DB2 support for Java stand-alone applications

An application running in WebSphere Application Server almost always uses a managed data source that is predefined in the server. If so, JDBC driver parameters are defined in the application server through the WebSphere data source properties. If you do not run in a managed container, you must set up the connection to the database in the program itself.

Java stand-alone applications are used often. On z/OS, the traditional batch job often is developed in Java.

As an example, Java development cannot be done without frequent JUnit tests, which are Java stand-alone applications. Today, every Java class has a corresponding test class that checks all the methods of the class. A framework that is called JUnit (http://www.junit.org) organizes the tests. After development, the program must build, normally after all its components are checked out of a source code version control system, such as Concurrent Versions System (CVS). The build process includes the creation of Java archives in which the application is packaged. Java archives (JARs), web application archives (WARs), and enterprise archives (EARs) must be built. Many dependencies to other Java archives must be resolved during that process. Then, the application then is deployed automatically to a Java Platform, Enterprise Edition server.

Today, Apache Maven is the open source project that is usually used for this work. It is a Java stand-alone application that runs several times a day.

During the development cycle, database definitions must be provided at several points. Unit tests must check data that comes from a database or the packaging or deployment process must include the preconfigured JDBC driver.

This section shows you some ways of dealing with different configuration options for the usage of IBM DB2 Driver for JDBC and SQLJ for stand-alone applications.

6.5.1 Alternatives for setting the JDBC driver parameters

Even if you develop a Java Platform, Enterprise Edition application that is used with a full server, you most likely use stand-alone Java applications, for example, in unit tests for your JPA entities or other JDBC classes. It is preferable to have the JDBC driver configured the same way for these tests. Hence, this section shows you some possibilities about changing the driver properties.

For example, the currentSchema property is often defined as a JDBC driver property outside of the Java program. This way, the Java class can be used for multiple database schema without having to change the code. This situation also applies to the defaultIsolationLevel property.

You can specify driver properties in the following ways:

- As Java system properties during the startup of the JVM. They are called IBM Data Server Driver for JDBC and SQLJ *configuration properties* because every connection to DB2 on this JVM inherits this configuration.
- Specify IBM Data Server Driver for JDBC and SQLJ properties during the setup of a specific connection.
- Specify connection and runtime properties for JPA programs in the `persistence.xml` file.
- Change settings for a single unit of work in your program.

Specification at connection setup

There are three different ways to set driver parameters during connection setup:

- Set the `java.util.Properties` value in the `info` parameter of a `DriverManager.getConnection` call, as shown in Example on page 309.

Example 6-7 Setting JDBC driver parameters with java.util.Properties

```
Properties properties = new Properties();
properties.put("user", "db2r1");
properties.put("password", "pwpwpw");
properties.put("currentSchema", "DSN81010");
properties.put("defaultIsolationLevel", new Integer(
      java.sql.Connection.TRANSACTION_READ_COMMITTED).toString());
String url = "jdbc:db2://wtsc63.itso.ibm.com:39000/DB0Z";
Connection conn = DriverManager.getConnection(url, properties);
```

- Set a `java.lang.String` value in the `url` parameter of a `DriverManager.getConnection` call, as shown in Example 6-8 on page 309.

- Use `setXXX` methods, where XXX is the unqualified property name, with the first character capitalized when using subclasses of `com.ibm.db2.jcc.DB2BaseDataSource`. For example, to change the `defaultIsolationLevel` property, you use the method `ds.setDefaultIsolationLevel()` before establishing the connection. In this case, the class is no longer portable because you are using the IBM Data Server Driver for JDBC and SQLJ interfaces directly.

These examples focus on the `defaultIsolationLevel` and `currentSchema` properties because they are frequently needed during connection setup.

You can find a full list of the properties in *DB2 10 for z/OS Application Programming Guide and Reference for Java*, SC19-2970.

The isolation level constant in the `java.sql.Connection` class is an integer. For the properties dictionary, it must be converted to a string.

No implementation-specific classes are used in this example. Thus, portability is ensured.

The IBM Data Server Driver for JDBC and SQLJ supports a number of isolation levels, which correspond to database server isolation levels. Table 6-1 shows the equivalency of standard JDBC and DB2 isolation levels.

Table 6-1 Equivalency of JDBC and DB2 isolation levels

JDBC value	DB2 isolation level
`java.sql.Connection.TRANSACTION_SERIALIZABLE`	Repeatable read
`java.sql.Connection.TRANSACTION_REPEATABLE_READ`	Read stability
`java.sql.Connection.TRANSACTION_READ_COMMITTED`	Cursor stability
`java.sql.Connection.TRANSACTION_READ_UNCOMMITTED`	Uncommitted read

As of WebSphere Application Server V8.5, the default isolation level is read stability. For a stand-alone JPA, the default is cursor stability.

The driver parameters are set as shown in Example 6-8.

Example 6-8 Setting JDBC driver parameters in the connection url

```
String user = new String("db2r1");
String password = new String("pwpwpw");
String currentSchema = new String("DSN81010");
String defaultIsolationLevel = new Integer(
      java.sql.Connection.TRANSACTION_SERIALIZABLE).toString();
String url = "jdbc:db2://wtsc63.itso.ibm.com:39000/DB0Z:";
String url2 = "user=" + user + ";"
      + "password=" + password + ";"
      + "defaultIsolationLevel=" + defaultIsolationLevel + ";"
      + "currentSchema=" + currentSchema + ";";
Connection conn = DriverManager.getConnection(url + url2);Connection conn =
DriverManager.getConnection(url +url2);
```

The resulting connection `url` string from Example 6-8 on page 309 is shown in Example 6-9.

Example 6-9 Connection url string

```
jdbc:db2://wtsc63.itso.ibm.com:39000/DB0Z:user=db2r1;password=pwpwpw;defaultIsolat
ionLevel=8;currentSchema=DSN81010;
```

In the connection `url` string, all text after the last “:” is treated as JDBC driver properties, which are optional. If you provide JDBC driver properties in the connection string, do not forget the last “;” because otherwise it will not work.

Specification at JVM start

IBM Data Server Driver for JDBC and SQLJ configuration properties all start with `db2.jcc`. They are specified as JVM system properties, that is, as `-D` parameters when starting the JVM. For example, `-Ddb2.jcc.currentSchema=DSN81010` defines the default `currentSchema` for all connections coming from that JVM.

Alternatively, you can point `-Ddb2.jcc.propertiesFile=/home/myJcc.properties` to a file that contains the properties, for example, `db2.jcc.currentSchema=DSN81010` and other properties that you want to be valid for that Java run. If you use a `DB2JccConfiguration.properties` file without pointing to it at JVM startup, you must include the directory that contains that file in the class path. It is only searched by the driver if `-Ddb2.jcc.propertiesFile` is not set.

Definition of JDBC properties for JPA applications

With JPA, the `META-INF/persistence.xml` file is the location where the JPA implementation expects to find its runtime definitions. Provider-specific parameters must be defined here.

For our Java stand-alone example, we use the Apache OpenJPA implementation (`http://openjpa.apache.org`) because the IBM JPA implementation in WebSphere Application Server is based on OpenJPA. In Example 6-10, you see a `persistence.xml` file for use in a Java SE environment, as indicated by `transaction-type="RESOURCE_LOCAL"`. In contrast, in a `persistence.xml` file for use in WebSphere Application Server, it is `transaction-type="JTA"`. In WebSphere Application Server, almost no property is defined, but in Java SE, you must specify connection parameters.

Example 6-10 Example of an OpenJPA persistence.xml file

```
<?xml version="1.0"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
   <persistence-unit name="jpadb2-zos" transaction-type="RESOURCE_LOCAL">
      <provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
      <class>ibm.itso.entities.Dept</class>
      <class>ibm.itso.entities.Emp</class>
      <properties>
         <property name="openjpa.RuntimeUnenhancedClasses" value="unsupported" />
         <property name="openjpa.jdbc.Schema" value="DSN81010" />
         <property name="openjpa.ConnectionDriverName"
value="com.ibm.db2.jcc.DB2Driver" />
         <property name="openjpa.ConnectionProperties"
value="username=db2r1,password=pwpwpw" />
         <property name="openjpa.ConnectionURL"
value="jdbc:db2://d0zg.itso.ibm.com:39000/DB0Z:clientApplicationInformation=jpaDB2
Tests;defaultIsolationLevel=4;" />
```

```
        <property name="openjpa.Log" value="DefaultLevel=ERROR, SQL=TRACE" />
        <property name="openjpa.DataCache" value="false" />
        <property name="openjpa.QueryCache" value="false" />
        <property name="openjpa.jdbc.DBDictionary" value="db2(batchLimit=100)" />
        <property name="openjpa.jdbc.QuerySQLCache" value="false" />
        <property name="openjpa.ConnectionFactoryProperties"
value="PrettyPrint=true, PrettyPrintLineLength=72"/>
      </properties>
   </persistence-unit>
   </persistence-unit>
</persistence>
```

The default schema is defined in `<property name="openjpa.jdbc.Schema" value="DSN81010" />`.

The default isolation level here is part of the connection url. During our tests, the definition `openjpaConnectionProperty` had no effect.

With OpenJPA, every property starting with `openjpa` is proprietary.

If the parameter name `defaultIsolationLevel` has a spelling error, no error message is given. The parameter would be ignored instead and set to the default value.

Here are the connection parameters with a short description of each one:

- javax.persistence.jdbc.driver: Fully qualified name of the driver class
- javax.persistence.jdbc.url: Driver-specific connection URL
- javax.persistence.jdbc.user: User name that is used by the connection
- javax.persistence.jdbc.password: Password that is used for the connection

Specification for a single unit of work

JDBC isolation levels can be set for a unit of work within a JDBC program by using the `Connection.setTransactionIsolation` method.

6.5.2 Java batch considerations with DB2

For almost all companies in every industry, batch processing is still a fundamental, mission-critical component. It might be tempting for Java developers to reuse classes they have developed for their online transaction processing (OLTP) in batch programs. In some cases, even online transactions are called from batch. This can be successful if there is a dedicated batch window when no users are online and the numbers of transactions are not high.

But when you plan a batch process with millions of database updates, there are things to consider. OLTP is triggered by a user with a direct response. To initiate OLTP, users typically complete an entry form or perform other actions through a user interface application component. The user interface component then initiates the associated online transaction with the business logic in the background. When the transaction is complete, the same user interface or other user interface component presents the result of the transaction to the user. The response can be data or can be a message regarding the success or failure of the processing of the input data. The transaction has high priority in the system and normally gets system resources at once. Data is committed after every transaction.

In contrast, batch processes require no user activity. Most batch programs read data from various sources (for example, databases, files, and message queues), process that data, and then store the result. The speed of a single update is not that important but the overall throughput is. The priority in the system is lower because the online work must not be disturbed. The work follows an input - processing - output pattern where the input from one process is often needed for input for another process. If there are abends, restarts of the same job must be possible at any time. In order to not have to start from the beginning, checkpoints should be built in to the application.

A checkpoint is one of the key features that distinguishes bulk jobs from OLTP applications, in that data is *committed* in chunks, along with other required housekeeping to maintain recovery information for restarting jobs. An extreme example is doing a checkpoint after every record, which equates to how OLTP applications typically work. At the other extreme is doing a checkpoint at the end of the job step, which might not be feasible in most cases because recoveries can be expensive and too many locks can be held in the database for too much time. A checkpoint is somewhere between these two extremes. The checkpoint interval can vary depending on a number of factors, such as whether jobs are run concurrently with OLTP applications, how long locks can be held, the amount of hardware resources available, the SLAs to be met, and the time that is available to meet deadlines. Depending on the technology that is used, there are static ways of setting these checkpoint intervals, but ideally checkpoint intervals can be set dynamically as well.

The application logic should take the following items into consideration:

- Database commits should, if possible, not occur after a single update but only after a group of updates.
- Plan checkpoints at which an application restart can occur.
- If transactions must be made in an OLTP server from a batch program, use WLM service classes that prevent the normal online transactions from being constrained.
- A JDBC batch statement.
- Consider using a WebSphere embeddable EJB container for your batch. It is especially useful if you can then avoid connecting to the WebSphere Application Server that is used for online work. The batch can be assigned to a special WLM service class. All database services such persistence service with JPA, transactions with EJBs, and bean validation are available.
- Consider using a WebSphere Extended Deployment Compute Grid. You can process business transactions cost-effectively by sharing resources and development skills between batch and online transactions (OLTP).

6.5.3 Portability

When you search for a sample of a JDBC program, you many of them that start with the following string:

```
// Load the driver
Class.forName("com.ibm.db2.jcc.DB2Driver");
```

This string couples the Java class unnecessarily to a specific implementation and prevents portability. As of JDBC 4, you do not need to load the drive if you have the driver implementation classes in your class path; in DB2, these are in `db2jcc4.jar`. The `java.sql.DriverManager` methods find the implementation classes that are using the service location mechanisms. If the connection URL starts with `jdbc:db2`, the IBM Data Server Driver for JDBC and SQLJ is found.

JDBC 4.0 Drivers must contain the `META-INF/services/java.sql.Driver` file. This file points to the correct implementation class; for DB2, it is `com.ibm.db2.jcc.DB2Driver`.

6.5.4 Sample Java SE stand-alone application with JPA and DB2

This section shows a simple Java stand-alone application. It shows the basic things that you need to start with your own application and how the tools in IBM Data Studio can support you. The application is based on the Java Persistence API (JPA) instead of raw JDBC or SQLJ because JPA is the most common approach. Samples using direct JDBC statements can be found in *DB2 for z/OS and WebSphere: The Perfect Couple*, SG24-6319.

Later in the book, you see a more complex application that runs inside a Java Platform, Enterprise Edition container (see “A short Java Platform, Enterprise Edition example” on page 346). There you can find more background for programming with JPA.

To run the example yourself, you need a local DB2 or a DB2 on z/OS system with an installed SAMPLE database, the IBM DB2 Driver for JDBC and SQLJ (`db2jcc4.jar`), the license JAR for the specific platform, the OpenJPA implementation `openjpa-all-2.2.0.jar`, and the logging framework `slf4j-simple-1.6.6.jar`. For more information about obtaining these items, see Appendix I, “Additional material” on page 587. Some of the JAR files can be found in a WebSphere Application Server installation. You can get one, for example, if you augment IBM Data Studio with the WebSphere Application Server test environment described Appendix C, “Setting up a WebSphere Application Server test environment on IBM Data Studio” on page 523.

Complete the following steps:

1. In IBM Data Studio in the Data Perspective window, click **Data Source Explorer** create a DB connection. The example that is shown in Figure 6-5 shows a connection to DB2 on z/OS. However, the example should work for any DB2 system that has the sample database installed.

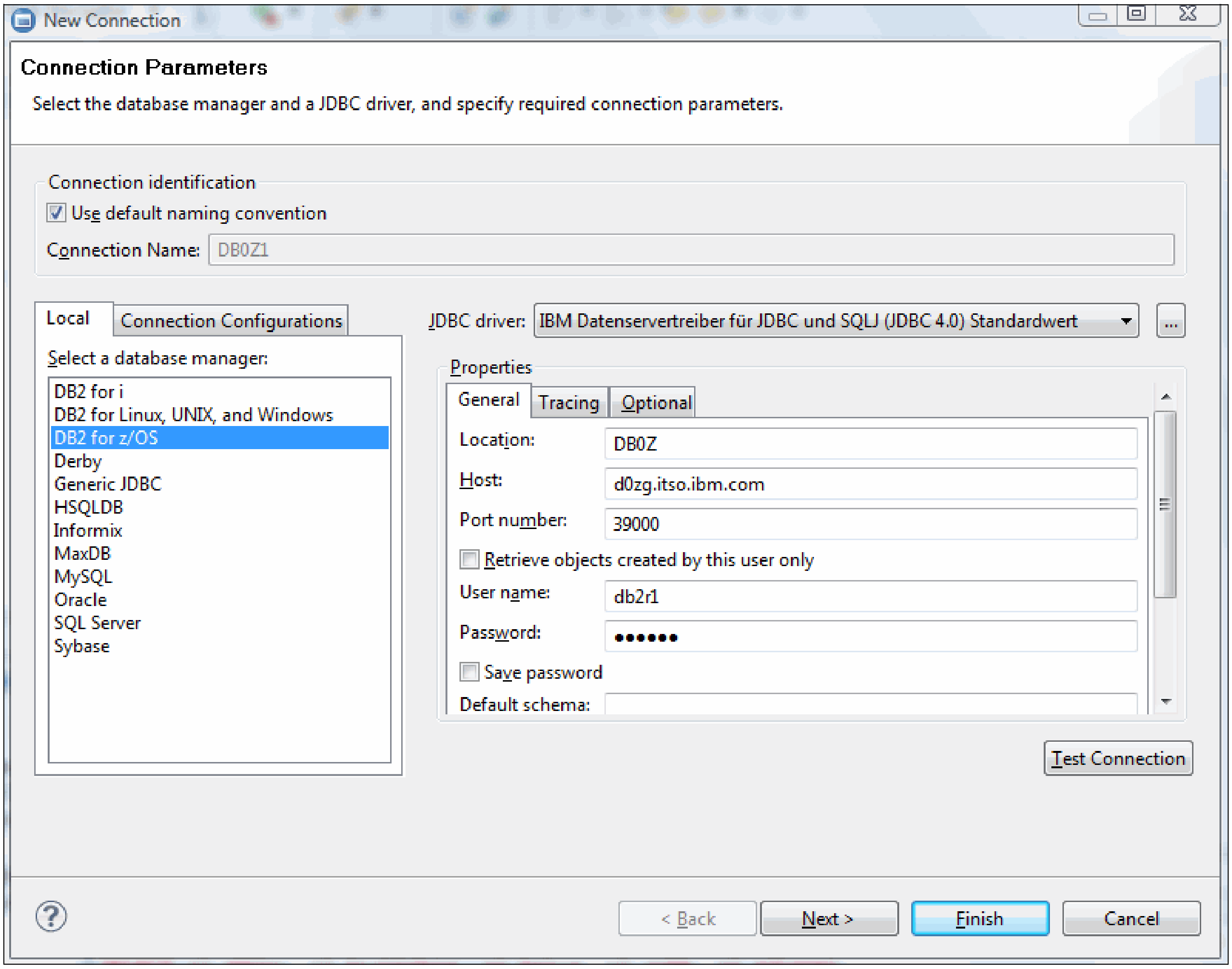


Figure 6-5 Create a data source connection

Use the valid connection parameters for your system. The data source connection inside IBM Data Studio is needed so that you can use the Data Studio tools for the generation of Java JPA entities. If you defined the Java code by typing the class definitions, this step is not needed.

2. Check whether the sample DB is present by scrolling through the hierarchy that opens after you establish the connection, as shown in Figure 6-6. You need the DB to create entities and for the test runs.

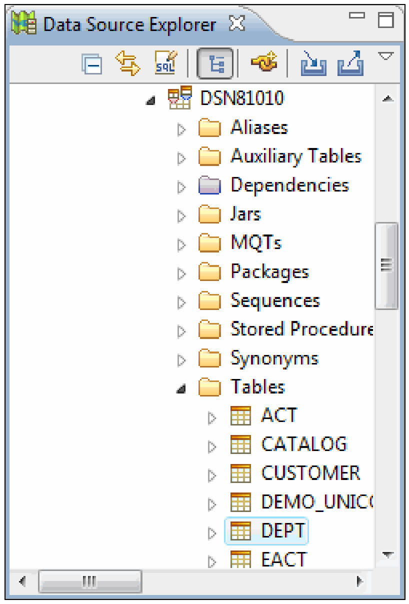


Figure 6-6 Check whether you can connect to the sample database

3. In the Java Perspective in the Package Explorer window, create a JPA project named `jpa_db2`, as shown in Figure 6-7. You can use the default location, and do not need to select a target run time. Check whether the configuration shows Minimal JPA 2.0 configuration. The project does not need to be added to an EAR.

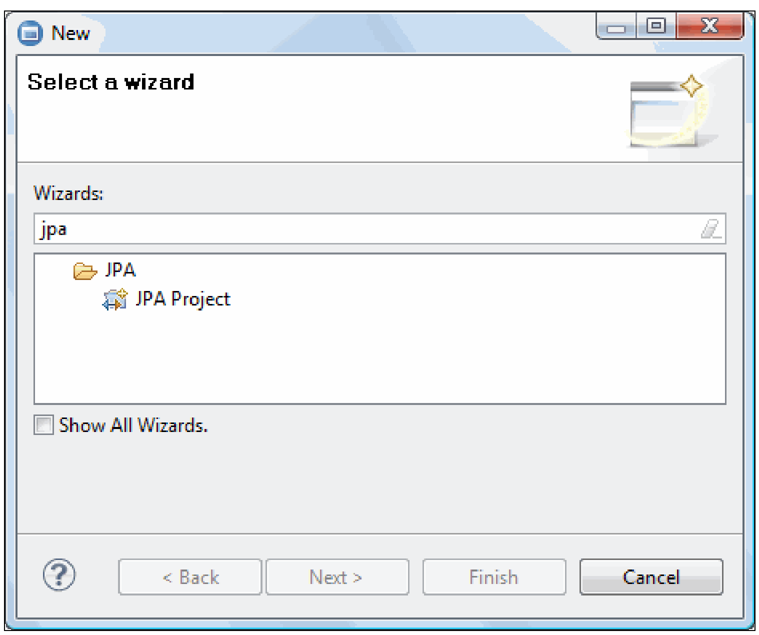


Figure 6-7 Create a JPA project

This should give you a project with the structure shown in Figure 6-8:

– A Java project with a source folder.

– A `META-INF` directory with a default `persistence.xml` file.

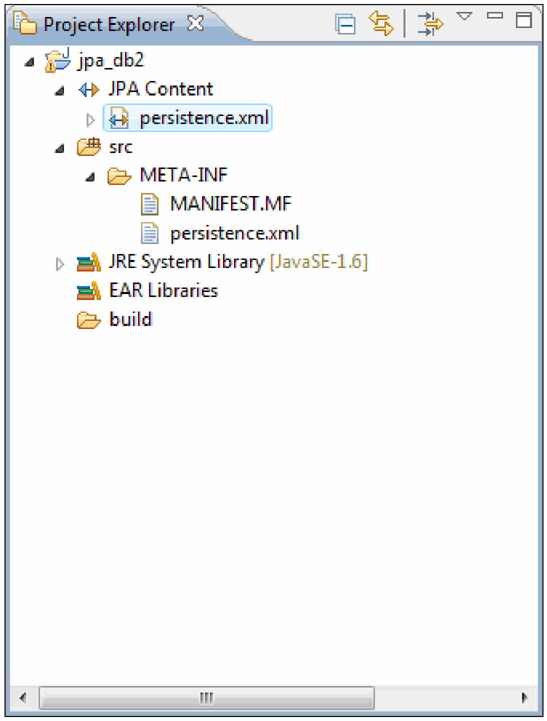


Figure 6-8 Project structure

In the JPA Perspective window, the Project Explorer provides a special view of the `persistence.xml` file that is in the `META-INF` directory. There are default contents that are already generated for the still empty `persistence-unit` that is named after your project name. In the Java Perspective window, it is shown only as a normal file in `META-INF`.

Example 6-11 shows the generated JPA definition file.

Example 6-11 Generated persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
   <persistence-unit name="jpa_db2">
   </persistence-unit>
</persistence>
```

We are now going to generate a Java JPA entity from an existing database table. In JPA terms, this is what is known a bottom-up approach. Complete the following steps:

1. Right-click the project name and select **JPA Tools** → **Generate Entities from Tables**, as shown in Figure 6-9.

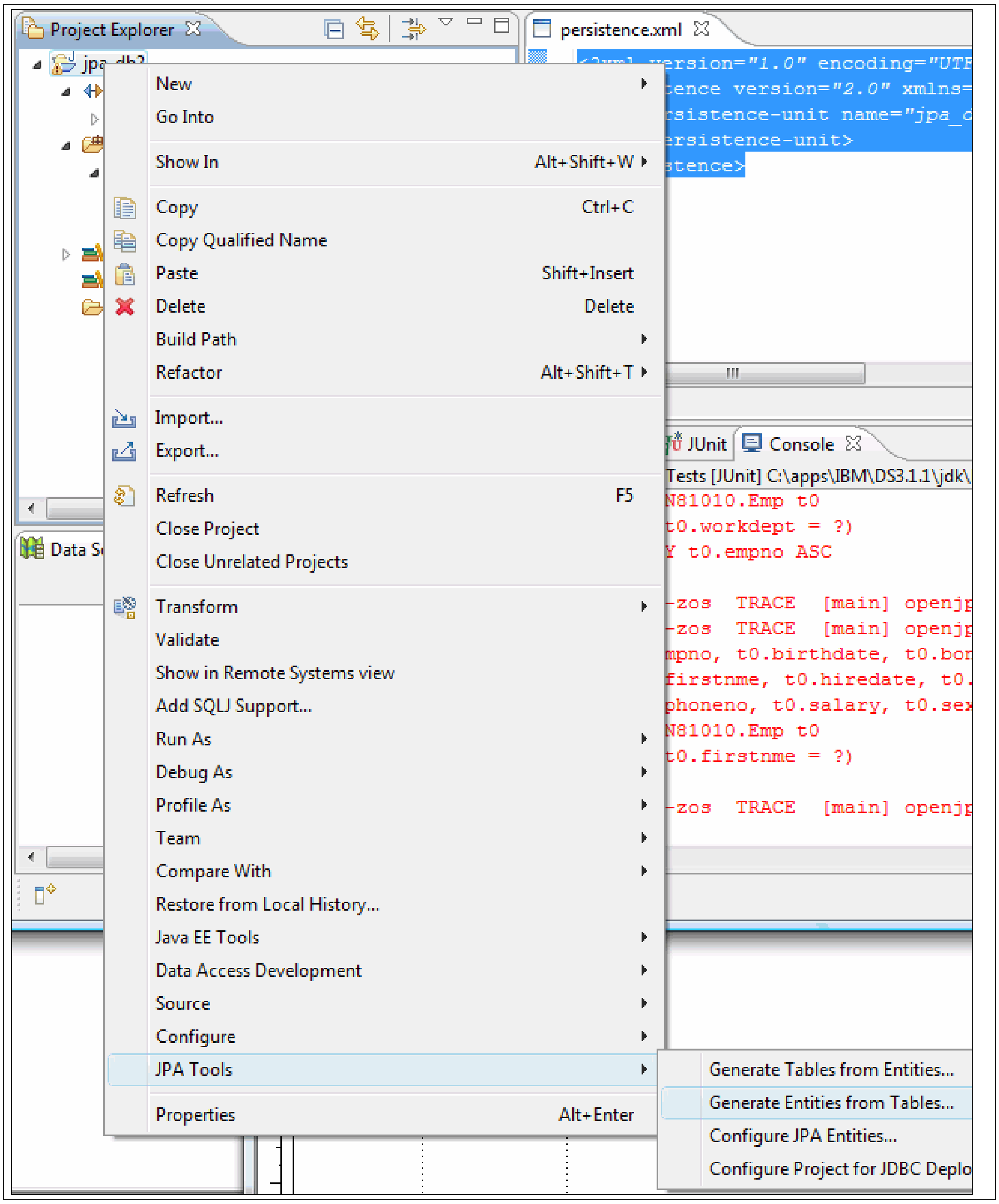


Figure 6-9 Select a table for the generation of JPA entities

2. Select the correct database connection. It is the one that created in step 1 on page 314. Next, you must select the schema under which the sample database is defined. In this example, it is DSN81010. The table names then display.

3. Select the DEPT table, as shown in Figure 6-10.

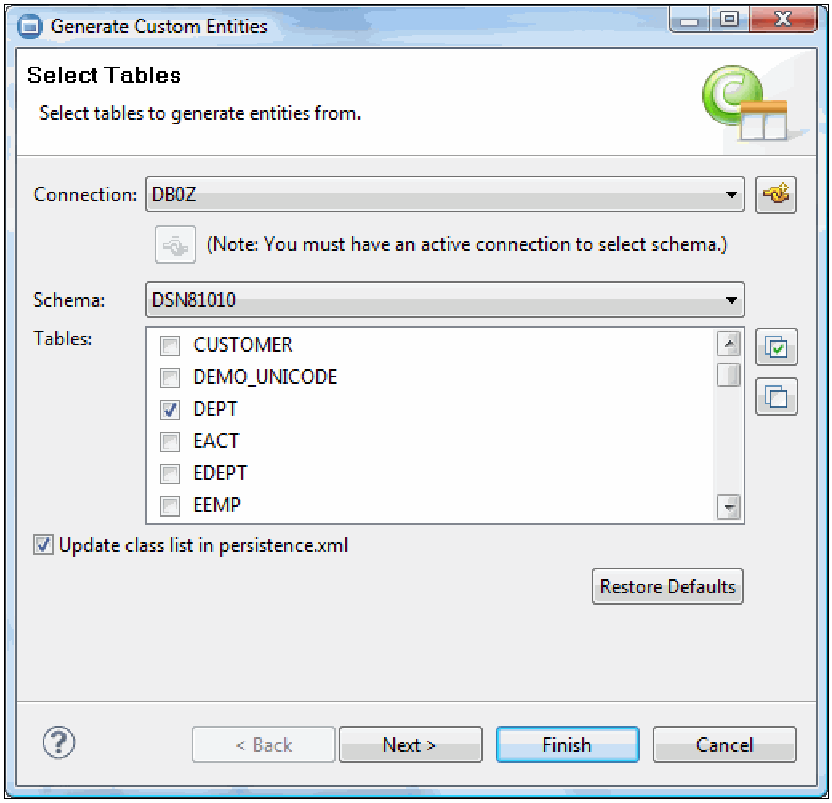


Figure 6-10 Select the DEPTtable in the DSN81010 schema

Leave the settings in the Table Associations window at the defaults. If there are relationships among other tables, you can define their associations here. Because you have only one table in our sample, you do not need to specify anything, as shown in Figure 6-11.

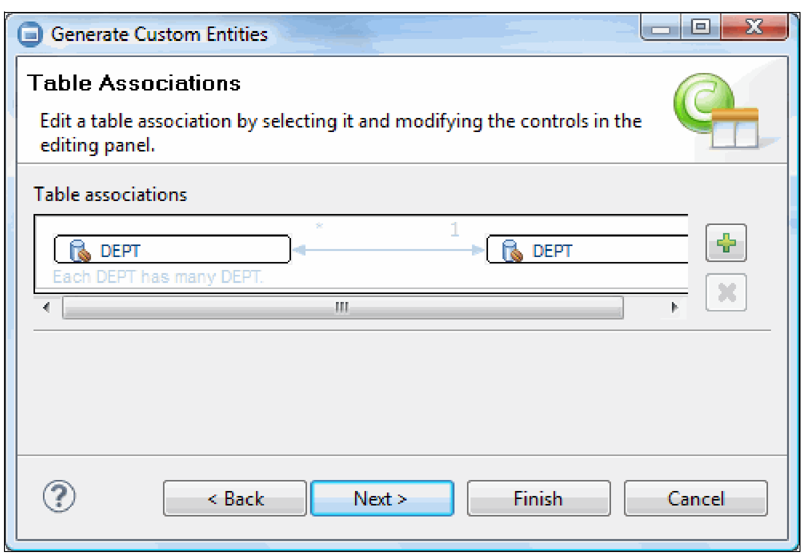


Figure 6-11 Relationships to other classes

4. In the Customize Default Entity Generation window, set **Key generator** to auto. This inserts the annotation `@GeneratedValue(strategy=GenerationType.AUTO)` in to your generated Java class for the key field deptno. Specify `com.ibm.itso.entities` in the Packages field, as shown in Figure 6-12.

   You do not have to specify a class name because the table name is used as a class name by default. The default behavior can be changed afterward by using special Java annotations.

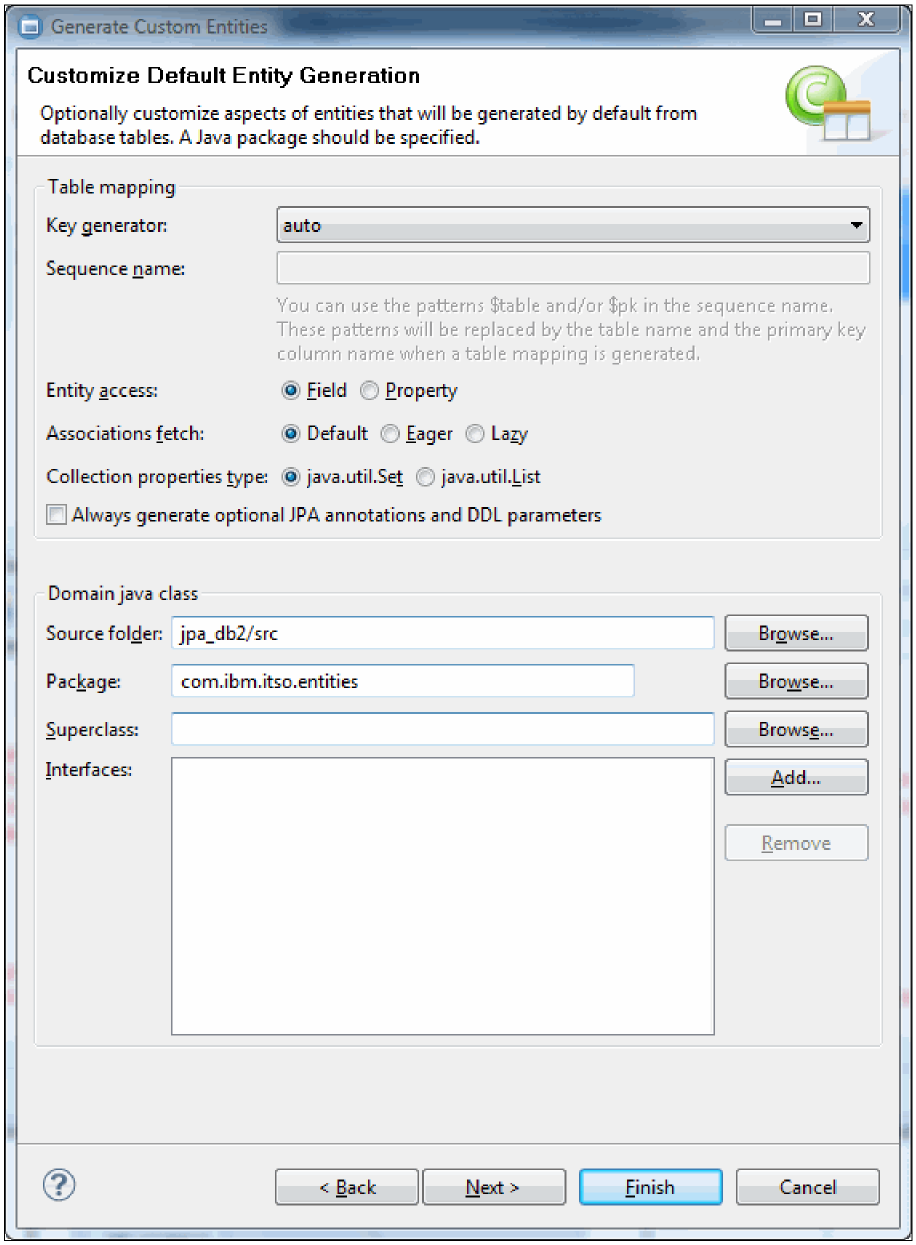


Figure 6-12 Generated class characteristics

This key generator generates a Java package named `com.ibm.itso.entities` with a class named `Dept.java`, which is named after the table name. In addition, the `persistence.xml` file is expanded by one entry, which declares the `Dept` class as an entity, as shown in Example 6-12.

Example 6-12 Added Dept class in the persistence.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
   <persistence-unit name="jpa_db2">
      <class>com.ibm.itso.entities.Dept</class>
   </persistence-unit>
</persistence>
```

The `Dept.java` file still has syntax errors because the class path is missing some important libraries. We are now going to fix this.

a. Switch to the Java perspective, right-click the project name, and select **Build Path** → **Add External Archives**. Add the following archives:
   - slf4j-simple-1.6.6.jar
   - openjpa-all-2.2.0.jar
   - db2jcc4.jar
   - db2jcc_license_cu.jar or if you test with DB2 on z/OS db2jcc_license_cisuz.jar

Even after the class path contains the correct libraries, the project cannot be built because one error remains:

```
Class "com.ibm.itso.entities.Dept" is included in a persistence unit, but is not mapped.
```

This seems to be an Eclipse-related error and it can be fixed easily.

b. Click **Project** → **Clean** and clean the whole workspace or your project.

c. Click **Project** and verify that **Build Automatically** is selected so that the project is compiled and rebuilt after the cleaning.

The generated Java source for the Dept entity is shown in Example 6-13. The names for the class and the fields are all taken from the table and column names of the database.

Example 6-13 Generated Dept.java entity

```
package com.ibm.itso.entities;

import java.io.Serializable;
import javax.persistence.*;


/**
 * The persistent class for the DEPT database table.
 *
 */
@Entity
public class Dept implements Serializable {
   private static final long serialVersionUID = 1L;

   @Id
```

```
@GeneratedValue(strategy=GenerationType.AUTO)
private String deptno;

private String admrdept;

private String deptname;

private String location;

private String mgrno;

 public Dept() {
 }

public String getDeptno() {
   return this.deptno;
}

public void setDeptno(String deptno) {
   this.deptno = deptno;
}

public String getAdmrdept() {
   return this.admrdept;
}

public void setAdmrdept(String admrdept) {
   this.admrdept = admrdept;
}

public String getDeptname() {
   return this.deptname;
}

public void setDeptname(String deptname) {
   this.deptname = deptname;
}

public String getLocation() {
   return this.location;
}

public void setLocation(String location) {
   this.location = location;
}

public String getMgrno() {
   return this.mgrno;
}

public void setMgrno(String mgrno) {
   this.mgrno = mgrno;
}
```

If the program had only the Dept class, the program would be ready to run. But the program cannot run because it is a simple Java POJO without a main method. To run the program, you must code a JUnit testdriver. Enable the project to run JUnit tests, as shown in Figure 6-13. Click **Build Path** → **Configure Build Path** → **Add Library** → **JUnit** and select the JUnit library with the version JUnit4 and add the unit test run time to the class path.

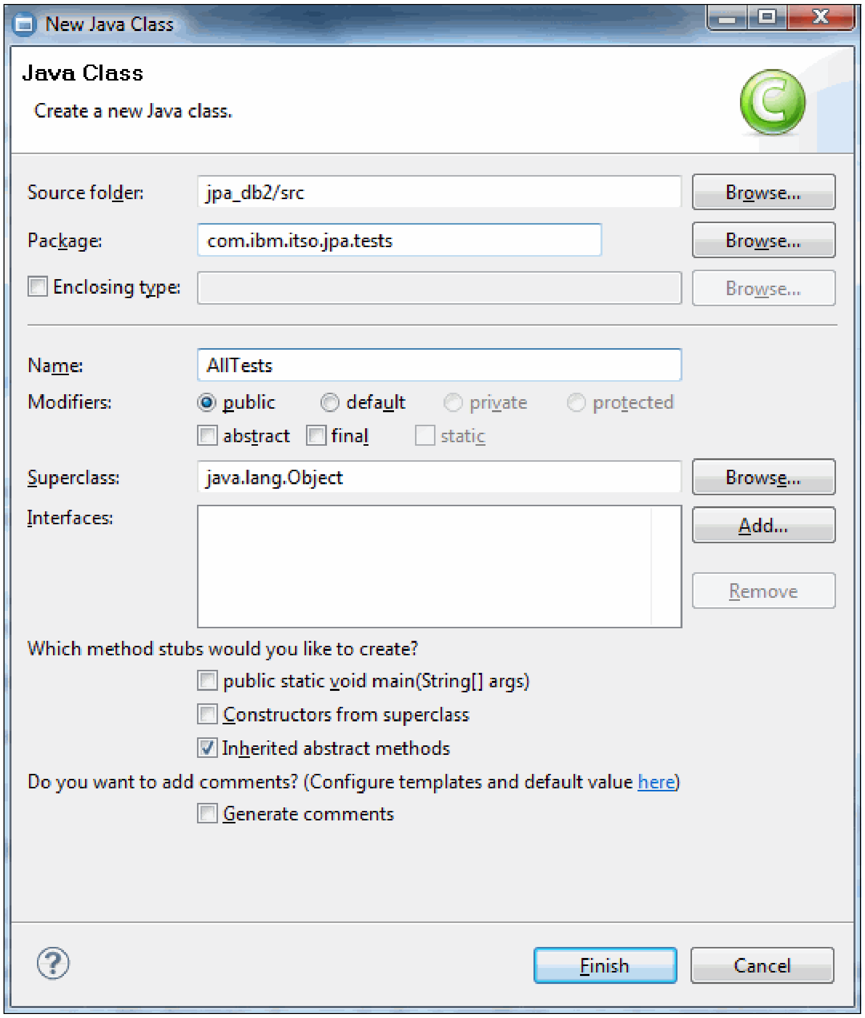


Figure 6-13 Creation of the JUnit test class

Now you are ready to create the test class. Complete the following steps:

1. Right-click the project and select **New** → **Class**.
2. For the package name, specify com.ibm.itso.jpa.tests, and for the class name, AllTests.
3. Replace the contents of the Java source for `AllTests.java` with the contents of Example 6-14.

Example 6-14 Sample test driver class

```
package com.ibm.itso.jpa.tests;

import static org.junit.Assert.assertEquals;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
```

```
import javax.persistence.TypedQuery;
import org.junit.Before;
import org.junit.Test;
import com.ibm.itso.entities.Dept;

public class AllTests {

   protected EntityManagerFactory emf;
   protected EntityManager em;

   @Before
   public void initEmfAndEm() {

      emf = Persistence.createEntityManagerFactory("jpa_db2");
      em = emf.createEntityManager();
   }

   @Test
   public void getDeptResultListSize() {

      TypedQuery<Dept> query1 = em.createQuery("Select d from Dept d",
            Dept.class);
      List<Dept> results = query1.getResultList();
      assertEquals(results.size(), 14);
   }

   @Test
   public void getListOfDepartements() {

      TypedQuery<Dept> query1 = em.createQuery("Select d from Dept d",
            Dept.class);
      List<Dept> results = query1.getResultList();
      for (Dept dept : results)
            System.out.println(dept.getDeptname());
   }
}
```

4. Replace the contents for the `META-INF/persistence.xml` file with the contents of Example 6-15.

Example 6-15 META-INF/persistence.xml update

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
   <persistence-unit name="jpa_db2">

<provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
      <class>com.ibm.itso.entities.Dept</class>
      <properties>
         <property name="openjpa.RuntimeUnenhancedClasses" value="unsupported"
/>
         <property name="openjpa.ConnectionDriverName"
value="com.ibm.db2.jcc.DB2Driver" />
```

```
        <property name="openjpa.jdbc.Schema" value="DSN81010" />
        <property name="openjpa.ConnectionProperties"
value="username=db2r1,password=pwpwpw" />
        <property name="openjpa.ConnectionURL"
value="jdbc:db2://d0zg.itso.ibm.com:39000/DB0Z" />
        <property name="openjpa.Log" value="DefaultLevel=ERROR" />
      </properties>
   </persistence-unit>
</persistence>
```

This action adds connection-specific information properties to the file. You must use your own names.

To start the run, complete the following steps:

1. Right-click `AllTests.java` in the Package Explorer in the Java Perspective. Click **Run AS** → **JUnit Test**.

   A new view named JUnit should open. A green or red bar shows you the success or failure of the two tests that are defined in our `AllTest.java` test driver.

   The first test shows a red bar or an exception. The failure has a special reason. Because you run JPA outside of a managed environment, there is something missing called JPA entity enhancement, which is described later in this chapter. This enhancement is normally done by an Java Platform, Enterprise Edition application server automatically. For now, enable it only for your stand-alone environment. The first test run created an entry for the run configuration, which facilitates this action.

2. Click **Run** → **Run Configurations** and select **JU Alltests**.
3. In the Arguments tab, enter `-javaagent:C:\apps\apache-openjpa-2.2.0\openjpa-2.2.0.jar` with the right path to `openjpa-2.2.0.jar` for your environment, as shown in Figure 6-14 on page 325.

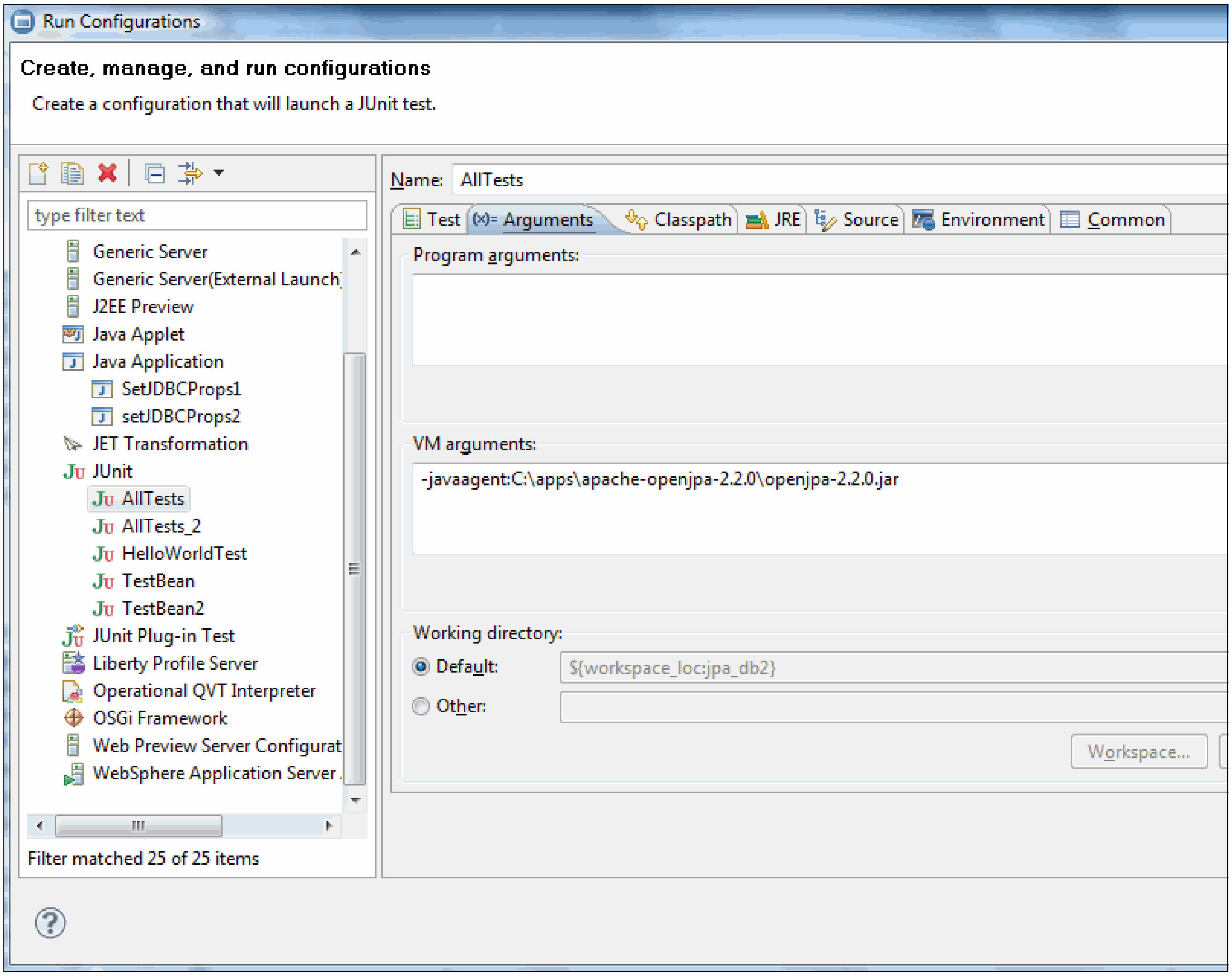


Figure 6-14 Specify the JPA enhancement javaagent for the unit test

4. Right-click `AllTests.java` in the Package Explorer again. Click **Run AS** → **JUnit Test**.

The JUnit run time now inspects the Alltests class for methods that are annotated with @Test and runs them. Because the DEPT table has 14 rows, the `assertEquals(results.size(), 14)` statement in the `getDeptResultListSize()` method succeeds. The second test in the `getListOfDepartements()` method is not a real test (it has no assert). It prints only the DEPTNAME column of the result set just to show that the objects are created from the database.

The overall result of the test is successful, as shown in Figure 6-15.

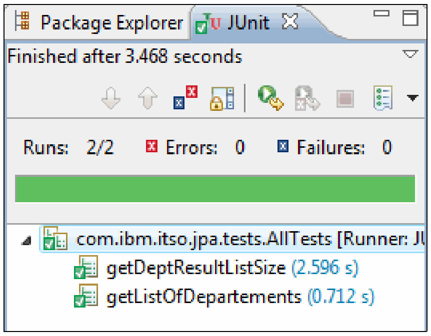


Figure 6-15 JUnit test success message

In addition, you should see a list of Department Names in the Console window, as shown in Example 6-16.

Example 6-16 JUnit test run console output

```
SPIFFY COMPUTER SERVICE DIV.
PLANNING
INFORMATION CENTER
DEVELOPMENT CENTER
MANUFACTURING SYSTEMS
ADMINISTRATION SYSTEMS
SUPPORT SERVICES
OPERATIONS
SOFTWARE SUPPORT
BRANCH OFFICE F2
BRANCH OFFICE G2
BRANCH OFFICE H2
BRANCH OFFICE I2
BRANCH OFFICE J2
```

6.6 JDBC applications in managed environments

A run time is *managed* if all the resources that your program deals with are defined in the container that encloses the application. Normally, this is an application server such as WebSphere Application Server, but it does not have to be. IMS or CICS also provide a managed infrastructure. Managed environments differ from unmanaged environments in several ways. Generally, in unmanaged environments, you must provide the following items in your application program:

- The loading and configuration of the JDBC-Driver
- The definition of connections
- A security information provision
- Usage of transaction support

Java programs that have this information hardcoded into their classes can run in managed environments because any Java class can use the full capability of the JVM and bypass the server provided functions. This is not a preferred practice, though.

In a managed environment such as WebSphere Application Server, resources such as data sources are predefined in the server environment. They are assigned with a JNDI Name. The database connection in an application program is done by first looking up this JNDI name in the server. The server then gives back a Datasource object that is used by the application or the persistency framework to make the connection. This name (a string) must be coded in the Java program and should be a logical name that is used only inside Java. It should not directly use the JNDI name that is defined in a specific server for the data source, although it works. Instead, it should be a reference to this name that must be mapped at deployment time. This act of association is called binding the resource reference to the data source.

For example, the string `private String dbName = "java:comp/env/jdbc/sampleRef"`; is used to look up the reference `jdbc/sampleRef`, as declared in `web.xml` or `ebj-jar.xml`. `java:comp/env/` is an indicator for the server to use a reference and not the real name. The name is valid only inside the Java code and does not need to refer to any existing data sources JNDI name.

Example 6-17 shows a sample declaration of a data source in a `web.xml` deployment descriptor. This way, the application is coded independently from any information in the server, which ensures portability.

Example 6-17 Resource reference declaration in web.xml

```
<resource-ref>
      <description />
      <res-ref-name>jdbc/sampleRef</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
      <res-sharing-scope>Shareable</res-sharing-scope>
   </resource-ref>
```

The real data source to be used by this application is declared only at deployment time or, as a special case, in an embedded configuration file. This file, for which you can see an example in Example 6-18, contains the required binding information. Unless it is embedded in the application package, this file is normally generated at deployment time. Its name is `ibm-web-bnd.xml` or `ibm-ejb-jar-bnd.xml` and contains the binding-name attribute. It comes from an administrator who defined the server resources.

Example 6-18 Web application bindings file ibm-web-bnd.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-bnd
   xmlns="http://websphere.ibm.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://websphere.ibm.com/xml/ns/javaee
      http://websphere.ibm.com/xml/ns/javaee/ibm-web-bnd_1_1.xsd"
   version="1.1">
   <virtual-host name="default_host" />
   <resource-ref name="jdbc/sampleRef" binding-name="jdbc/sample" />
</web-bnd>
```

As of Java EE5, resources can be injected into your program by using Java annotations. The annotation that is used is `javax.annotation.Resource`. The process of binding references to data sources remains basically the same. Instead of scanning the `web.xml` file during deployment in search for unresolved references, the server examines the annotations. The reference does not need to be declared in `web.xml` any more.

The annotation of `@Resource(name="jdbc/AccountDB")` is equivalent to the traditional `java:comp/env/jdbc/AccountDB"` lookup and must be mapped at deployment time. In this case, the name is a logical reference.

`@Resource(mappedName="jdbc/definedAccountDB")` on the other side directly points to the defined resource in the target runtime server without mapping. This non-portable solution works if the resource is defined, but it is not a preferred practice.

6.6.1 Data source connection tests on z/OS

When you define a data source or try to find errors with a data source, connection tests are useful. You test the physical connection and verify that the correct security settings are in place for that data source, as shown in Figure 6-16.

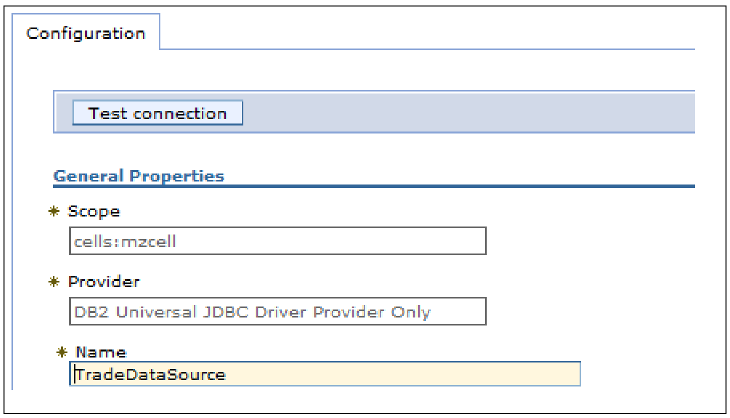


Figure 6-16 Data source connection test

Although this is a well-known feature, there are some implications to using it with WebSphere Application Server on z/OS. While in normal operation the respective servant connects to the database itself, connection tests are sometimes done by other WebSphere Application Server address spaces, depending on the scope that is defined for the data source. The correlation of data source and test connection locality is shown in Table 6-2.

Table 6-2 Correlation of data source scope with the test connection JVM

Data source scope	JVM where the test connection operation occurs
Cell	Deployment manager.
Node	Node agent process (of the relevant node).
Cluster	Node agent for each node that contains a cluster member.
Server	Server. If the server is unavailable, the test connection operation is tried again in the node agent for the node that contains the application server.

Consideration: In a network deployment implementation of the application server, you cannot test connections for the following data sources at the node level or cluster level:

- IBM Data Server Driver for JDBC and SQLJ data source with driver type 2
- DB2 Universal JDBC Driver Provider data source with driver type 2

The application server issues the following exception for a test connection at the node level:

```
java.sql.SQLException: Failure in loading T2 native library db2jcct2DSRA0010E: SQL
state = null, Error Code = -99,999
```

Therefore, when you create these data sources at the node scope or cluster scope, you might want to temporarily create the same configurations at a server scope for testing purposes. Run the test connection operation at the server level to determine whether the data source settings are valid for your overall configuration.

6.7 Coding practices for a good DB2 dynamic statement cache hit ratio

Saving prepared statements in a dynamic statement cache can avoid unnecessary preparation processes and thus improve performance. Besides the DB2 dynamic statement cache setting, you should pay attention to SQL program coding, which affects the hit ratio of statement cache.

6.7.1 Eligible SQL statements for caching

SELECT, **UPDATE**, **INSERT**, **DELETE**, and **MERGE** statements can be saved in the cache.

If JDBC packages are bound with **REOPT(ALWAYS)**, statements cannot be saved in the cache. If JDBC packages are bound with **REOPT(ONCE)** or **REOPT(AUTO)**, statements can be saved in the cache.

Statements that are sent to an accelerator server cannot be saved in the cache.

6.7.2 SQL comments considerations

There are two types of SQL comments:

- Simple comments: Introduced with two consecutive hyphens (--) and end with the end of a line.
- Bracketed comments: Introduced with /* and end with */. A nested bracketed comment means that the comment contains another bracketed comment, for example, /* /* */ */.

The following types of SQL statement text with SQL comments can be saved in the dynamic statement cache:

- SQL statement text with SQL bracketed comments within the text.
- SQL statement text that begins with SQL bracketed comments that are unnested. No single SQL bracketed comment that begins the statement can be greater than 258 bytes.

6.7.3 Conditions for prepared statement reuse

Suppose that S1 and S2 are source statements, and P1 is the prepared version of S1. P1 is in the dynamic statement cache. The following conditions must be met before DB2 can use statement P1 instead of preparing statement S2:

- The authorization ID or role that was used to prepare S1 must be used to prepare S2.

  For the conditions that a statement is eligible for reuse, see the following website:

  http://pic.dhe.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db2z10.doc.perf/src/tpc/db2z_conditionsstmtsharing.htm

- S1 and S2 must be identical (The exception is literal constant if the **PREPARE ATTRIBUTES** clause **CONCENTRATE STATEMENTS WITH LITERALS** is enabled). The statements must pass a character by character comparison and must be the same length. If the **PREPARE** statement for either statement contains an **ATTRIBUTES** clause, DB2 concatenates the values in the **ATTRIBUTES** clause to the statement string before comparing the strings. For example, if A1 is the set of attributes for S1 and A2 is the set of attributes for S2, DB2 compares S1||A1 to S2||A2.

- When the plan or package that contains S2 is bound, the values of these bind options must be the same as when the plan or package that contains S1 was bound:

  - **CURRENTDATA**
  - **DYNAMICRULES**
  - **ISOLATION**
  - **SQLRULES**
  - **QUALIFIER**
  - **EXTENDEDINDICATOR**

- When S2 is prepared, the values of the following special registers must be the same as when S1 was prepared:

  - **CURRENT DECFLOAT ROUNDING MODE**
  - **CURRENT DEGREE**
  - **CURRENT RULES**
  - **CURRENT PRECISION**
  - **CURRENT REFRESH AGE**
  - **CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION**
  - **CURRENT LOCALE LC_CTYPE**

6.7.4 Literal replacement

Before DB2 9, if a dynamic SQL statement is run frequently but the literal constants in it vary, it cannot get the performance benefit of cached statement reuse.

DB2 10 introduces a way for users to get higher cache reuse from dynamic statements that reference literal constants. You can specify the **PREPARE ATTRIBUTES** clause **CONCENTRATE STATEMENTS WITH LITERALS**, or set the JDBC driver connection property `statementConcentrator=YES` to enable it.

If DB2 prepares a SQL statement and **CONCENTRATE STATEMENT** is enabled, DB2 replaces certain literal constants in the SQL statement text with the ampersand character ('&'), and inserts the modified statement into the dynamic statement cache.

When DB2 runs subsequent dynamic SQL statements, if the first search of the cache does not find an exact match by using the original statement text, DB2 substitutes the ampersand character ('&') for literal constants in the SQL statement text and searches the cache again to find a matching cached statement that also has '&' substituted for the literal constants. If that statement text comparison is successful, DB2 determines whether the literal reusability criteria between the two statements allows for the new statement to share the cached statement.

The reusability criteria includes, but is not limited to, the immediate usage context, the literal data type, and the data type size of both the new literal instance and the cached literal instance. If DB2 determines that the statement with the new literal instance cannot share the cached statement because of incompatible literal reusability criteria, DB2 inserts, into the cache, a new statement that has both '&' substitution and a different set of literal reusability criteria. This new statement is different from the cached statement, even though both statements have the same statement text with ampersand characters ('&'). Now, both statements are in the cache, but each has different literal reusability criteria that makes these two cached statements unique.

Here is an example:

Assume that DB2 prepares the following SQL where column X is data type decimal:

```
SELECT X, Y, Z FROM TABLE1 WHERE X < 123 (no cache match)
```

After the literals are replaced with '&', the cached statement is as follows:

```
SELECT X, Y, Z FROM TABLE1 WHERE X < & (+ lit 123 reuse info)
```

Assume that the following new instance of that statement is now being prepared:

```
SELECT X, Y, Z FROM TABLE1 WHERE X < 1E2
```

According to the literal reusability criteria, the literal value 1E2 does not match the literal data type reusability of the cached statement. Therefore, DB2 does a full cache prepare for this **SELECT** statement with literal 1E2 and inserts another instance of this '&' **SELECT** statement into the cache as follows:

```
SELECT X, Y, Z FROM TABLE1 WHERE X < & (+ lit 1E2 reuse info)
```

Now, given the two '&' **SELECT** statements that are cached, attempt to prepare the same **SELECT** statement again but with a different literal value instance from the first two cases:

```
SELECT X, Y, Z FROM TABLE1 WHERE X < 9
```

DB2 fails to find an exact match for the new **SELECT** statement with literal '9', replaces literal '9' in the **SELECT** statement with '&', and does a second search. Both of the two cached statements are reusable with literal value '9', therefore, simply by order of statement insertion into the cache, cached statement for literal 123 is the first cached statement found that satisfies the literal reusability criteria for the new literal value '9'.

6.8 Locking

Here are the factors that influence locking:

- Isolation level
- Lock avoidance
- Optimistic locking

6.8.1 Isolation level

WebSphere and DB2 naming conventions for the isolation level do not explicitly map. The translation between the two levels is listed in Table 6-1 on page 309.

The isolation level settings are listed below in order from most to least restrictive. In combination with the executed SQL, these modes determine the lock mode and duration of the locks that are acquired for the transaction.

- `TRANSACTION_SERIALIZABLE` (Repeatable Read) acquires locks on all rows read by an SQL statement whether they qualify for the result set or not. The locks are held until the transaction is ended through a commit or rollback. Other transactions cannot insert, delete, or update rows that are accessed by an SQL statement executing with RR.
- `TRANSACTION_REPEATABLE_READ` (Read Stability) acquires locks on all stage 1 qualifying rows and maintains those locks until the application issues a commit or rollback. With RS, other transactions cannot update or delete rows that qualified (during stage 1 processing) for the statement because locks are held. If the application attempts to re-reference the same data later in the transaction, the results will not have been updated or deleted. However, other applications can insert more rows, which is known as a *phantom read* because subsequent selects against the same data within the same transaction might result in extra rows being returned.
- `TRANSACTION_READ_COMMITTED` (Cursor Stability) ensures that all data that is returned is committed. When SELECTing from the table, locks are not held for rows or pages for which a cursor is not positioned. DB2 tries to avoid taking locks on non-qualifying rows. If an application attempts to re-reference the same data later in the transaction, there is no guarantee that data has not been updated, inserted, or deleted.
- `TRANSACTION_READ_UNCOMMITTED` (Uncommitted Read) means that locks are not acquired for queries (**`SELECT`**), and the application may return data from another transaction that has not yet been committed or rolled back.

6.8.2 Lock avoidance

Locking carries a cost both for concurrency and processing. Provided certain conditions are met, DB2 can avoid requesting a Read or Share lock on behalf of the application process. This function, which applies only to low-level locks, is referred to as *lock avoidance*.

Prerequisite of lock avoidance

Users have no direct control over the usage of lock avoidance. Lock avoidance occurs in the following situations:

- There is a read-only or ambiguous cursor with ISOLATION(CS) and CURRENTDATA(NO).
- For any nonqualifying rows that are accessed by queries that are bound with ISOLATION CS or RS.
- When DB2 system managed referential integrity (RI) checks for dependent rows when either the parent key is updated or if the parent key is deleted and the **`DELETE RESTRICT`** option is defined.

DB2 supports three types of cursors:

- Read-only cursors
- Updatable cursors
- Ambiguous cursors

If a cursor is defined with the clauses **FOR FETCH ONLY** or **FOR READ ONLY**, it is a read-only cursor. If a cursor is defined with the clause **FOR UPDATE OF**, it is an updatable cursor. A cursor is considered ambiguous if DB2 cannot tell whether it is used for update or read-only purposes. For more information about these three types of cursors, see *DB2 9 for z/OS: Resource Serialization and Concurrency Control*, SG24-4725.

In a JDBC application, the declaration and processing of a cursor occurs with a different syntax, but the concept is basically the same. Instead of processing a cursor, a PreparedStatement is created and a ResultSet is used to process the results. Example 6-19 shows an updatable cursor, which is a cursor that is not eligible for lock avoidance.

Example 6-19 Updatable cursor in a JDBC application

```
PreparedStatement p1 = con.prepareStatement("SELECT ACTKWD, ACTDESC FROM " +
"DSN8A10.ACT WHERE ACTNO = 180",ResultSet.TYPE_FORWARD_ONLY,ResultSet.CONCUR_UPDATABLE);
ResultSet rs1 = p1.executeQuery();
String s1 = null;
String s2 = null;
while (rs1.next())
{
s1 = rs1.getString(1);
s2 = rs1.getString(2);
if (s1.compareTo("DOC   ") == 0)
{
rs1.updateString(2,"Make A Document");
rs1.updateRow();
}
System.out.println("Active Description Is "+rs1.getString(2));
}
```

Lock avoidance control

The following options can be specified when you bind a JDBC driver:

- **CURRENTDATA**: With **CURRENTDATA(NO)**, DB2 uses lock avoidance techniques to access the data. Lock avoidance is not considered for qualifying rows if the application is bound with **CURRENTDATA(YES)**.
- **ISOLATION**: Cursor Stability (CS) increases the concurrency and also the possibility of lock avoidance.

If you use a read-only result set with **CURRENTDATA(NO)**, the stability of the qualifying rows is not protected by the lock. When the row qualifies under the protection of a data page latch, the row is passed to the application, and the latch is released. Therefore, the content of the qualified row might have changed immediately after it was passed to the application. To continue processing further rows in a page, DB2 must latch the page again.

Impact on block fetch and parallelism

Table 6-3 summarizes the impact of the **CURRENTDATA** option for parallelism and block fetch for distributed applications.

Table 6-3 Impact of CURRENTDATA option

Current data required	Ambiguous cursor	Read-only cursor
YES	Lock avoidance is not considered for ISOLATION(CS) applications. I/O and CP parallelism are not allowed. Block fetching does not apply.	Lock avoidance is not considered for ISOLATION(CS) applications. I/O and CP parallelism are allowed. Block fetching applies.
NO	Lock avoidance is considered for ISOLATION(CS) applications. I/O and CP parallelism are allowed. Block fetching applies for distributed applications.	

If your business logic allows, use the `CONCUR_READ_ONLY` result set (this is the JDBC equivalent for the `DB2 'FOR READ ONLY'` clause) if there is no update that is intended, along with ISOLATION(CS) and CURRENTDATA(NO).

6.8.3 Optimistic locking

Lock avoidance can improve concurrency and reduce processor consumption, but applications with positioned update intention are not eligible for it. For example, if the data is read from the tables and presented to the users before the update, to make sure that there is data integrity, the lock should be held from read to commit. In Example 6-20, a thread wants to read a record and then update it. If DB2 releases the lock after the **SELECT**, other threads may modify this record, and then the update might fail to make the changes to the specified row.

Example 6-20 Read a record and then update

```
SELECT ACTDESC INTO :desc FROM DSN81010.ACT WHERE ACTKWD = 'DOC';
-- Other processing
UPDATE DSN81010.ACT SET ACTDESC = 'MAKE DOCUMENT' WHERE AND ACTKWD = 'DOC';
COMMIT;
```

To ensure data integrity and reduce locking, you can use optimistic concurrency control.

When an application uses optimistic concurrency control, locks are obtained immediately before the read operation and released immediately after the read. The update locks are obtained immediately before an update operation and held until the end of the process. It minimizes the time for which a resource is unavailable for use by other transactions. Optimistic concurrency control uses the RID and a row change token to test whether data was changed by another transaction since the last read operation, so it can ensure data integrity while limiting the time that locks are held.

Eligible applications for optimistic locking

If an application uses optimistic concurrency control but resource contentions happen frequently, then the update fails, and you must reprocess the failed record. Reprocessing hurts overall performance compared to the performance savings achieved by avoiding the locks.

In general, optimistic concurrency control is appropriate for application processes that do not have concurrent updates on the same resource, such as information only (read-only) web applications, single user applications, or pseudo-conversational OLTP applications, where the data is read from the tables and presented to the users before performing the updates. Optimistic concurrency control is also appropriate for applications accessing tables that are defined with page level locking or higher level lock size when the concurrently running processes are accessing different sets of data.

Using ROW CHANGE TIMESTAMP

To implement optimistic concurrency control, you can establish a row change time stamp column with a **CREATE TABLE** statement or an **ALTER TABLE** statement. The column must be defined with one of the following null characteristics:

- NOT NULL GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
- NOT NULL GENERATED BY DEFAULT FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP

After you establish a row change time stamp column, DB2 maintains the contents of this column. When you want to use this change time stamp as a condition when making an update, you can specify an appropriate predicate for this column in a WHERE clause, as shown in Example 6-21.

Example 6-21 Implement optimistic concurrency control by using ROW CHANGE TIMESTAMP

```
ALTER TABLE DSN81010.ACT ADD COLUMN RCT
NOT NULL GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP;

--REORG TABLESPACE

SELECT ACTDESC, ROW CHANGE TIMESTAMP FOR ACT INTO :desc, :rct FROM DSN81010.ACT
WHERE ACTKWD = 'DOC';

-- Other processing
UPDATE DSN81010.ACT SET ACTDESC = 'MAKE DOCUMENT'
WHERE ROW CHANGE TIMESTAMP FOR ACT = :rct AND ACTKWD = 'DOC';

-- Other processing
COMMIT;
```

In this example, DB2 completes the following steps:

1. Acquires a minimal lock before the read to ensure data integrity. The best option is to read with **ISOLATION(CS)** and **CURRENTDATA(NO)** to get lock avoidance without sacrificing data integrity.
2. Selects **ROW CHANGE TIMESTAMP** along with other pertinent information from the table by employing lock avoidance techniques.
3. Releases the lock immediately after the read or employ lock avoidance techniques by using the **ISOLATION (CS)** with **CURRENTDATA (NO)** bind options.
4. Saves the data, particularly the **ROW CHANGE TIMESTAMP**, for future comparison.
5. Acquires the exclusive locks immediately before the update and holds on to the update until the process ends or commits.
6. During the update, checks whether the data read was changed by another process since it was last read, by comparing the current row change time stamp with that of the saved values.

7. The update succeeds only when it is verified that the **ROW CHANGE TIMESTAMP** values match the saved ones; otherwise (in case another process changed the value), the update fails with a return code of +100 (row not found for update).
8. The application must reprocess the failed record, if needed.

Note: You can use **ROW CHANGE TOKEN** instead of **ROW CHANGE TIMESTAMP** in SQL. It takes the last 8 bytes of the DB2 time stamp and returns it as BIGINT.

7

Java Platform, Enterprise Edition with WebSphere Application Server and DB2

When dealing with a database, Java enterprise programs in the past used the JDBC driver APIs either for dynamic SQL or for static SQL (SQLJ) directly. As an alternative, the usage of Container-Managed Persistence (CMP) was specified and implemented in almost all Java enterprise servers, such as WebSphere Application Server. *DB2 for z/OS and WebSphere: The Perfect Couple*, SG24-6319 describes this situation and gives multiple examples of how to write DB2 applications with dynamic JDBC, SQLJ, and EJB (CMP).

This chapter gives a database administrator (DBA) enough background information so that the DBA can assess what is behind the newer Java enterprise concepts. For the Java programmer, this chapter provides samples that could serve as a starting point to working with DB2 on z/OS from inside and outside of managed application server environments.

This chapter covers the following topics:

- Java Platform, Enterprise Edition with WebSphere Application Server and DB2
- Implementation version of JPA inside WebSphere Application Server
- Preferred practices of Java Platform, Enterprise Edition and DB2
- Known issues with OpenJPA 2.2 and DB2

7.1 Java Platform, Enterprise Edition with WebSphere Application Server and DB2

The Java programming environment on z/OS is established. Both Java stand-alone applications and Java Platform, Enterprise Edition applications inside WebSphere Application Server are used. DB2 is their prime persistent storage.

The first generation of Java applications commonly uses driver-specific JDBC statements for dynamic SQL or, following a more traditional development path, static SQLJ in a similar way to how you include SQL into COBOL programs. Because the driver is associated to a particular database and has database-specific statements, your code is tied to that database.

However, to fulfill driver-specific requirements is a violation of one of the primary goals of Java: portability. Although SQL is standardized among different systems and databases, raw JDBC programming error handling, for example, remains database specific. In addition, with raw JDBC or SQLJ programming, boilerplate code must be written to make Java objects work with the information from the database. You must fill the class attributes for every database field with separate statements. The JDBC API is not designed to store Java objects directly into relational databases.

The EJB 2.0 specification was a trial run to hide all the platform-specific details and delegate the arduous task of mapping the database information in Java objects to a standardized application server. Container-managed persistence (CMP) EJBs help the programmer by supporting automatic transaction handling and security services.

EJBs were not well received by the Java community for numerous reasons, mostly to do with the shortcomings of the specifications. Unit tests of EJB entities are nearly impossible because EJBs need an enterprise container to run in. The mapping of the state of Java objects to a relational representation is insufficient in the EJB model. It misses important aspects of object-oriented programming, such as inheritance.

Other approaches were more successful than the EJB 2.0 persistency specification, which is part of Java 2 Enterprise Edition 1.4. Hibernate, iBATIS, and EclipseLink are examples of successful persistency frameworks that often are used in enterprise applications instead of the EJBs that are offered by standard Java Platform, Enterprise Edition application servers.

Things have changed since the advent of Java Platform, Enterprise Edition 5, though. This Enterprise Java specification now includes the Java Persistence API (JPA). JPA 1.0 is part of Java Platform, Enterprise Edition 5, and JPA 2.0 is part of Java Platform, Enterprise Edition 6. The concepts that made Hibernate and the other persistency frameworks successful are now included in the Java enterprise standard. EJB Container-managed persistence (CMP) beans are replaced by JPA entity beans. EJBs now provide transaction support only; they do not provide persistency any more.

JPA was defined within the Java EE specification for Enterprise JavaBeans (EJB) 3.0. With JPA 2.0, the JPA specification is defined separately in *Java Specification Request (JSR) 317: Java Persistence API, Version 2.0.*

WebSphere Application Server V8.5 conforms to Java Platform, Enterprise Edition 6 and supports JPA 2.0. The JPA implementation inside WebSphere Application Server is based on the Apache OpenJPA project. Although you can use this implementation directly in WebSphere Application Server, the WebSphere Application Server default is to use the JPA for WebSphere Application Server persistence provider. There are some enhancements in the WebSphere Application Server version of the JPA provider over the original Apache version. The support of pureQuery client optimization is one example.

7.2 Implementation version of JPA inside WebSphere Application Server

You can see the version of both implementations by running **wsjpaversion**, as shown in Example 7-1.

Example 7-1 The wsjpaversion command

```
C:\Programme\ibm\WebSphere\AppServer\bin>wsjpaversion.bat
WSJPA 2.2.1-SNAPSHOT
Versions-ID: WSJPA-2.2.1-SNAPSHOT-r1119:2559
Überarbeitung der WebSphere-JPA-Unterversion: 1119:2559

OpenJPA 2.2.1-SNAPSHOT
Versions-ID: openjpa-2.2.1-SNAPSHOT-r422266:1325904
Überarbeitung der Apache-Unterversion: 422266:1325904

os.name: Windows Vista
os.version: 6.0
os.arch: x86

java.version: 1.6.0
java.vendor: IBM Corporation

java.class.path:
        C:\Program Files\ibm\WebSphere\AppServer\dev\JavaEE\j2ee.jar
        C:\Program Files\ibm\WebSphere\AppServer\plugins\com.ibm.ws.jpa.jar
        C:\Program
Files\ibm\WebSphere\AppServer\plugins\com.ibm.ws.prereq.commons-collections.jar
        C:\Program Files\IBM\WebSphere Studio Workload
Simulator\jsoap\iwlJSoap.jar
```

Strictly speaking, JPA is every thing that you need for persistence for new projects. JPA is now considered the standard approach for Object to Relational Mapping (ORM) and can replace all the preceding ORM frameworks.

7.2.1 The goals of the Java Persistence API

The goal of JPA is to enable the Java programmer to handle only the main constituents of his program: Java objects. In a program, all the objects that are used are related in some way, and make up an object model. The object model reflects real objects of a business or other things the program should deal with. From a developer's point of view, the application logic and the object model is what matters.

In addition, there is the relational model, which is based on tables. Their correlation to each other is mathematically verified and optimized in a normalization process. Special skills are necessary to accomplish this task. It requires a deep knowledge of the database and its organization to accomplish this task effectively. Table design and definition are normally not a task the Java developer wants to deal with. It does not directly solve his problem. Both models must be coordinated with each other and mapped.

To leave the Java programmer free to work with his object model, the task of mapping his model to the relational model is delegated to the JPA infrastructure. Ideally, the Java programmer does not have to know which database is used and how the data in this database is dealt with.

JPA implementations allow simple Java classes or Plain Old Java Objects (POJOs) to be persisted. A POJO is considered a simple Java class because there is nothing that it depends on (not even on the code that makes nearly automatic persistence possible). To add the persistence behavior, Java annotations are added to the Java class. Java annotations do not change the program logic of the class but only give information to runtime environments that need this information. Only JPA uses `javax.persistence.*` annotations. Otherwise, runtime annotations are ignored. Thus, the Java class remains a POJO.

Conversely, the EJB 2 CMP specification requires classes to implement interfaces or methods that makes the class dependent on other classes or a server run time.

Because JPA has no dependencies on other containers, run times, or servers, it can be used as a stand-alone POJO persistence layer or it can be integrated in to any Java EE compliant container and many other lightweight frameworks.

This situation cannot be reached in more complex situations. In practice, the Java programmer and the DBA must communicate with each other and adjust their respective models. In many cases, most parts of the data exist, even for new applications or applications that are supposed to be migrated to JPA. Most companies have their data organized in to databases. Here, the Java programmer must follow the structure of existing tables because they are used by other programs as well and cannot easily be changed for the new one. JPA entities must be designed according to the relational data. This is called a bottom-up ORM approach. To accomplish this task, JPA gives you a rich set of annotations (or the XML equivalent) that allows you to customize each part of the mapping.

JPA entity customizing

Whether you must connect to an existing database or must follow strict database naming conventions, you can customize your JPA entities in many ways. You could start with your data model, which includes the database schema, and then work upwards to your entity classes. The **`wsreversemapping`** tool can help with this approach. It is used to perform reverse (bottom-up) mappings of database tables to entities. The generated Java files from the **`wsreversemapping`** tool might require some editing before they can be used in an application. Also, generated files do not contain annotations. Annotations can be added manually.

The JPA solution for WebSphere Application Server provides several tools that help with developing JPA applications. Combining these tools with IBM Rational Application Developer or IBM Data Studio provides a solid development environment for either Java EE or Java SE applications. IBM Rational Application Developer or IBM Data Studio include GUI tools to insert annotations, a customized `persistence.xml` file editor, a database explorer, and other features.

The customization of the ORM involves the following areas:

- Elementary mapping rules
  - Table name or names of additional tables to be used to map an entity
  - Single key column or composite keys for that entity
  - Rules for generating the key value
  - Attribute data types and table column characteristics

- Relationship mapping rules
  – Collections of Java types and their representation in tables (through foreign key or join tables)
  – Unidirectional and bidirectional mapping
  – One-to-one, one-to-many, many-to-one, many-to-many mapping
- Inheritance mapping
  – Single-table-per-class hierarchy strategy
  – Joined-subclass strategy
  – Table-per-concrete-class strategy

You can see from the volume of options that you need much experience and a good knowledge of theory and background of the mapping patterns to successfully work with ORM.

Automatic database schema generation

If the new application does not have to work with legacy data and can start from scratch, you can go the opposite way: the top-down approach. In that approach, the domain model dictates the relational schema. If you are using a top-down mapping of the object model to the relational model, you develop the entity classes first and then use the OpenJPA functionality to generate the database tables that are based on the entity classes. The **`wsmapping`** tool helps with this approach. You can use the **`wsmapping`** tool to create database tables. As an alternative, by specifying the buildSchema parameter to the `openjpa.jdbc.SynchronizeMappings` of your `persistence.xml` property, the mapping tool provides the default mapping that matches the database schema automatically during the run of your application. You are not required to run the batch mapping tool if the default mapping satisfies the necessary database schema.

Most persistence providers, including OpenJPA, allow you to generate the database automatically from the entities. Automatic table creation by JPA does not need a great deal of configuration, though. JPA follows a configuration-by-exception mapping strategy. Nearly everything is taken from the existing definitions in the Java class.

However, sticking to the defaults and relying only on the automatic generation of the database tables might lead to problems in more complex situations. The generated relational model should be reviewed because a normalized schema with too many tables might be the result. Bad performance could be a consequence, and maintenance might become more difficult.

Entity handling with the Entity Manager

The Entity Manager is the unit through which your program works with JPA. Entity manipulation, such as persist, read, change, and delete actions, are done by invoking methods on the entity manager. For create, read, update, and delete operations of simple entities, JPA provides the Java Persistence Query Language (JPQL). JPQL is syntactically similar to SQL, but is object-oriented rather than table-oriented. More complex tasks can use commands that the database provides natively. JPQL uses SQL and resembles raw JDBC coding.

The Entity Manager can deal with four types of commands:

- Dynamic query

  A string with JPQL statements is given as an argument to the Entity Manager for execution. The string can be a simple select or a more complex query by using joins and other selection criteria. It also can be an update or delete statement that is given to the **createQuery** method. Example 7-2 is an example in which an array of employee objects are returned. The query selects Java objects and not table rows with JPQL.

  Example 7-2 Example of a dynamic JPA query

```
em = emf.createEntityManager();
TypedQuery<Employee> query1 = em.createQuery(
        "Select d from Employee d", Employee.class);
```

- Static query

  This query must not be confused with static SQL. It still translates to dynamic SQL in the JDBC driver. Static here means that the query is already coded at build time and inspected by the JPA run time before the program actually uses it. It can have variable parameters. Query templates can be statically declared by using the NamedQuery annotation, as shown in Example 7-3. They are coded in the same Java source as the class they deal with. Many NamedQuery templates are a sort of table with statements that are prepared for later use by other parts of the program.

  Example 7-3 Example of a static JPA query

```
Declared in the entity class:
@NamedQuery(name="DeleteEmpAThiele", query="DELETE FROM Employee e " +
      "where e.lastname = 'Thiele'")
Use in a different class:
Query delete1 = em.createNamedQuery("DeleteEmpAThiele");
      delete1.executeUpdate();
```

- Native query

  Similar to the JDBC method **prepareStatement()**, a SQL string is given as a parameter with optional arguments. In addition, the second parameter says that the result list is expected to be of the type `Magazine`, as shown in Example 7-4.

  Example 7-4 Example of a native SQL query in JPA

```
Query query = em.createNativeQuery("SELECT ISBN, TITLE, PRICE, "
   + "VERS FROM MAG WHERE PRICE > ?1 AND PRICE < ?2", Magazine.class);
   query.setParameter(1, 5d);
   query.setParameter(2, 10d);
List<Magazine> results = (List<Magazine>) query.getResultList();
```

- Stored procedure call

  A stored procedure is similar to a native query. OpenJPA supports stored procedure invocations as SQL queries. OpenJPA assumes any SQL that does not begin with the **SELECT** keyword (ignoring case) is a stored procedure call, and starts it as such at the JDBC level. See Example 7-5.

  Example 7-5 Example call of a stored procedure

```
Query query = em.createNativeQuery("CALL MY_STOREDPROCEDURE(?)");
query.setParameter(1, arg1);
query.executeUpdate());
```

You can use native queries and stored procedure calls in cases where the JPA defaults are not enough and a generated table model does not fit your demands. The usage of JPA native queries can help you migrate raw JDBC applications to JPA or help you avoid raw JDBC in cases where the JPA defaults lead to problems. Generated SQL sometimes cannot use the full potential that the database normally provides. With native queries, you are able to use the database power inside JPA.

7.2.2 OpenJPA and JDBC interaction

OpenJPA interacts with the database by using the normal Java DataBase Connectivity (JDBC) APIs that are provided by the driver. OpenJPA uses the configuration of the JDBC driver that is in place through the definition of a data source or other properties unless the JPA provider is told to set specific configuration properties. In these cases, JPA calls the respective JDBC API for reconfiguration. Great care should be taken to not interfere with the configuration of the data source in an application server.

OpenJPA Entity Manager handles all the communication that is needed with the JDBC driver, for example, when the JDBC driver is requested to provide a connection to the database.

OpenJPA obtains JDBC connections on an as needed basis and releases them as fast as possible. A connection is made for each query. The connection is closed and given back to the pool. The connection is open only during a data store transaction or if a JDBC `ResultSet` is still active.

All this is transparent to the programmer and the Java program and normally this is the best behavior. In rare cases, you can configure OpenJPA's usage of JDBC connections through the `openjpa.ConnectionRetainMode` configuration property.

7.2.3 Agile JPA development with a WebSphere Application Server embeddable EJB container and DB2

Agile development has become the prevailing paradigm for nearly every enterprise Java project today. The development of every new Java production class starts by writing a test driver for that class. Test drivers are normally written as JUnit tests. Unit tests are integrated in development tools, such as IBM Rational Application Developer for WebSphere, Eclipse, Ant, or Maven. The number of unit tests grows as your application grows. A test for a new class normally includes all the tests for previously developed classes. This ensures that the new class does not interfere with the rest of the application.

The developer runs unit tests often, for example, once a minute or after even minor changes of a class. This ensures that the application remains in a consistent state.

The tests run inside a Java stand-alone test-driven environment. They must provide every service that the test depends on. Often, these services are configured as part of the test environment itself. For example, many databases and their JDBC driver are written in Java and can be included in to the application class path of the test run. Similar to the concept of the embeddable EJB container, these databases are embeddable databases. They start in the same JVM with the application and define their databases and tables at run time. Often, they are defined as in-memory databases. They and their contents vanish after the test run. One advantage is that every programmer has his own database; no coordination with other programmers is necessary. The database is reset to a known state for each test. The Apache Derby embeddable JDBC driver has such a capability. DB2 does not have an embeddable database.

The downside of dynamic databases is that you must define the database infrastructure and the test data for every run on your own. There are tools that help in that situation. JPA can generate the required tables automatically based on the definition of the Java classes. In addition, DbUnit (http://www.dbunit.org) is a JUnit extension that puts your database into a known state between test runs. It can be used for in-memory databases and for normal data stores. Provided that the Java programmer has sufficient access rights, the Java programmer can use DbUnit to reset the DB2 test database.

Problems with test runs arise when the Java class under test requires special services that are only provided in a full-blown Java Platform, Enterprise Edition server. Examples are security, transaction, or persistency services, which normally cannot be included in the tests. As a circumvention, these services are delegated to serve as mock-ups of objects that typically return hardcoded values from method invocations.

The Spring Container (http://www.springsource.org) addresses this and other problems. Spring in combination with Hibernate (http://www.hibernate.org) became a strong competitor to Java Platform, Enterprise Edition servers such as WebSphere Application Server.

The EJB 3.1 specification now includes an JSE-friendly embeddable container that is ideally suitable for agile Java development. As of WebSphere Application Server V8.0, this embeddable container is available. It does have some limitations, but can speed up development in a Java Platform, Enterprise Edition environment.

The WebSphere Application Server embeddable EJB container is a container for enterprise beans that does not require a Java Platform, Enterprise Edition server to run. The EJB programming model and the EJB container services are now available for Java Platform, Standard Edition (Java SE) servers.

The EJB container can be used for the following functions:

- EJB unit testing: Developers can test their enterprise beans without needing a full server installation of WebSphere Application Server in their development environment. It is an ideal environment for quickly developing and testing applications that might eventually run in the application server. It starts within seconds and is sufficiently configurable for the main tasks for applications that do not need a full Java Platform, Enterprise Edition server.
- Embedding enterprise beans in Java SE applications, for example, in batch applications, if the client that uses the EJBs is in the same JVM as the embeddable container.

Embeddable EJB container functions

According to the Enterprise JavaBeans (EJB) 3.1 specification, all embeddable EJB containers that vendors use must at least implement the EJB Lite subset of EJB functionality. It includes the following items:

- Local (and no-interface) session beans with synchronous methods only, which include stateless, stateful, and singleton bean types
- Declarative and programmatic security
- Interceptors
- Support for annotations or XML deployment descriptors and the `ejb-jar.xml` file
- Java Persistence Architecture (JPA) 2.0

- WebSphere Application Server 8.5 adds the following features to that EJB Lite subset:
  – Java Database Connectivity (JDBC) data source configuration, usage, and dependency injection.
  – Bean validation: To use bean validation with the embeddable EJB container, the javax.validation classes must exist in the class path. This can be achieved by including `com.ibm.ws.jpa.thinclient_8.0.0.jar` in the class path.

Here are the limitations when you use the embeddable container:

- Inbound RMI/IIOP calls are not supported, which means that all EJB clients must exist within the same Java virtual machine (JVM) as the embeddable container.
- Message driven beans (MDB) are not supported.
- The embeddable container cannot be clustered for high availability.

Embeddable EJB container configuration

Because the embeddable EJB container runs inside your application or your unit test as a separate container, the configuration is different from the normal WebSphere configuration. It relies on a file named `embeddable.properties` in the current work directory or a property file that the Java system property `com.ibm.websphere.embeddable.configFileName` points to.

In this file, you define data sources, as shown in Example 7-6. The example shows two data sources that are bound to the JNDI-namespace under the names `jdbc/TxDSz` and `jdbc/NoTxDSz` at container startup.

Example 7-6 DB2 data source definitions for the WebSphere embeddable EJB container

```
# JPA Transactional data source definition
DataSource.db2_1.name=jdbc/TxDSz
DataSource.db2_1.className=com.ibm.db2.jcc.DB2XADataSource
DataSource.db2_1.driverType=4
DataSource.db2_1.databaseName=DB0Z
DataSource.db2_1.serverName=d0zg.itso.ibm.com
DataSource.db2_1.portNumber=39000
DataSource.db2_1.user=DB2R1
DataSource.db2_1.password=db2r1pw

# JPA non-Transactional data source definition
DataSource.db2_2.name=jdbc/NoTxDSz
DataSource.db2_2.className=com.ibm.db2.jcc.DB2DataSource
DataSource.db2_2.driverType=4
DataSource.db2_2.databaseName=DB0Z
DataSource.db2_2.serverName=d0zg.itso.ibm.com
DataSource.db2_2.portNumber=39000
DataSource.db2_2.user=DB2R1
DataSource.db2_2.password=db2r1pw
DataSource.db2_2.transactional=false
```

For a JPA application, it is preferred practice to define both data sources to allow the full JPA functionality, such as automatic entity identity generation. This is done in the no-transactional data source.

The number of configuration parameters are limited compared to the number of configuration options you have with WebSphere Application Server. The following WebSphere Information Center contains a list of all data source definitions:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.zseries.doc/ae/rejb_emconproperties.html

There is no configuration option for the current schema; it must be defined by an `persistence.xml` property statement:

```
<property name="openjpa.jdbc.Schema" value="DSN81010" />
```

A short Java Platform, Enterprise Edition example

In this section, a short example shows the usage of the database definition that is shown in Example 7-6 on page 345 in a Java Platform, Enterprise Edition application. It is a basic program that serves only as a starting point for Java Platform, Enterprise Edition -based applications with DB2. The application deals with the EMPLOYEE table in the SAMPLE database of DB2. The SAMPLE database on z/OS is slightly different from the one on Linux UNIX, and Windows. On z/OS, we define an alias of EMPLOYEE for the table DSN81010.EMP to be able to run the example on both databases. The table EMPLOYEE exists without alias on DB2 for Linux, UNIX, and Windows.

The Java equivalent of the EMPLOYEE table is the Java class Employee, as shown in Example 7-7.

The server and JPA run time knows at class load time that this class corresponds to the database table EMPLOYEE because it is annotated with the `@Entity` tag. By default, the Java names for the class and the fields are directly taken by the JPA run time as names for use with the database. In addition, the source file defines a `@NamedQuery` for later use by other parts of the application.

Example 7-7 The Employee class

```
package com.ibm.itso.entities;

import java.io.Serializable;
import javax.persistence.*;
import java.math.BigDecimal;
import java.util.Date;

@Entity
@NamedQuery(name="DeleteEmpAThiele", query="DELETE FROM Employee e
   where e.lastname = 'Thiele'")

public class Employee implements Serializable {
   private static final long serialVersionUID = 1L;

   @Id
   private String empno;
    @Temporal( TemporalType.DATE)
   private Date birthdate;
   private BigDecimal bonus;
   private BigDecimal comm;
   private short edlevel;
   private String firstnme;
    @Temporal( TemporalType.DATE)
   private Date hiredate;
```

```
   private String job;
   private String lastname;
   private String midinit;
   private String phoneno;
   private BigDecimal salary;
   private String sex;
   private String workdept;
    public Employee() {
    }

   public String getEmpno() {
      return this.empno;
   }

   .... more getters and setters .......

}
```

The application shows how Java Platform, Enterprise Edition components, such as transactional EJBs and JPA entities, can be included in agile development.

For that reason, the application is called from a JUnit test inside IBM Data Studio. Test1 does a **SELECT** on the EMPLOYEE table and checks whether all 42 Employee objects are returned in the result list. Test2 does an **INSERT** of a new Employee into the table and checks afterward that the number of table rows has increased to 43. Test 3 deletes the added row and checks the correct number of rows afterward. The tests that are shown in Example 7-8 do not belong to the application, and are only for development.

Example 7-8 JUnit test driver

```
package tests;

import static org.junit.Assert.assertEquals;

import org.junit.After;
import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;
import ibm.itso.ejbs.EmpBean;
import java.util.List;

import javax.ejb.embeddable.EJBContainer;
import javax.naming.NamingException;

import com.ibm.itso.entities.Employee;

public class TestEmpBean {

   EJBContainer ec = null;
   Employee Emp1 = null;
   EmpBean EmpBean = null;

   @Before
   public void initEmbeddableContainerAndTestData() throws NamingException {

      // Create the embeddable container
      ec = EJBContainer.createEJBContainer();
      // Use the container context to look up the EmpBean
```

```
        EmpBean = (EmpBean) ec.getContext().lookup(
              "java:global/bin/EmpBean!ibm.itso.ejbs.EmpBean");
        // Create some test data
        Emp1 = new Employee();
        Emp1.setFirstnme("Andreas");
        Emp1.setLastname("Thiele");
        Emp1.setMidinit("A");
        Emp1.setEmpno("999999");
        Emp1.setWorkdept("A00");
     }

     @Test
     public void testNumberOfEmployeeRows() {

        // Query Employee table and verify it has 42 rows
        List<Employee> Emps = EmpBean.getEmployeeResultList();
        assertEquals(42, Emps.size());
     }

     @Test
     public void insertNewEmployeeBean() {

        try {
        EmpBean.persistNewEmployee(Emp1);
        }
        catch (Exception e) {
           System.out.println("Exception persisting Employee:\n" +e);
        }
        // Number of rows has increased to 43
        List<Employee> Emps = EmpBean.getEmployeeResultList();
        assertEquals(43, Emps.size());
     }

     //@Ignore
     @Test
     public void deleteInsertedEmpAgain() {

        try {
        EmpBean.deleteInsertedEmpAgain();
        }
        catch (Exception e) {
           System.out.println("Exception deleting Employee:\n" +e);
        }
        // Number of rows should be 42 again
        List<Employee> Emps = EmpBean.getEmployeeResultList();
        assertEquals(42, Emps.size());
     }

     @After
     public void shutDown() {
        ec.close();

     }
}
```

The JUnit tests use a stateless session EJB, `EmpBean`, that is part of the application.

As you can see in Example 7-9, the EJB consists of three transactional methods for **SELECT**, **INSERT**, and **DELETE** for JPA entity objects that are mapped to the EMPLOYEE table. The work with the database is done by the `javax.persistence.EntityManager` by using the persistence unit `EmpPU`. No JDBC statement is used to do the work, and no column name used. Even a table name is not given. All this is derived by the JPA container at run time from the Java class that the entity manager is asked to deal with, for example, `em.persist(employee);`.

In the EJB, only resource references that must be mapped to real names inside the server are used.

Example 7-9 Sample session EJB for SELECT, INSERT, and DELETE of a JPA entity

```
package ibm.itso.ejbs;

import static javax.ejb.TransactionAttributeType.SUPPORTS;
import static javax.ejb.TransactionAttributeType.REQUIRED;

import java.util.List;

import javax.annotation.Resource;
import javax.annotation.Resources;
import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;
import javax.persistence.TypedQuery;
import javax.sql.DataSource;

import com.ibm.itso.entities.Employee;

@Resources({ @Resource(name = "jdbc/TxDSref", type = DataSource.class),
      @Resource(name = "jdbc/NoTxDSref", type = DataSource.class) })

@Stateless
public class EmpBean {

   @PersistenceContext(unitName = "EmpPU")
   private EntityManager em;

   @TransactionAttribute(SUPPORTS)
   public List<Employee> getEmployeeResultList() {

      TypedQuery<Employee> query1 = em.createQuery(
            "Select d from Employee d", Employee.class);
      return query1.getResultList();
   }

   @TransactionAttribute(REQUIRED)
   public void persistNewEmployee(Employee employee) {
      em.persist(employee);
   }

   @TransactionAttribute(REQUIRED)
   public void deleteInsertedEmpAgain() {
      Query delete1 = em.createNamedQuery("DeleteEmpAThiele");
```

```
      delete1.executeUpdate();
   }
}
```

The persistence unit is defined in a short `persistence.xml` file, as shown in Example 7-10. It shows a `transaction-type="JTA"`, declaring that everything is handled within the server. This is different from a persistence-unit with `transaction-type="RESOURCE_LOCAL"`, where all the database connection definitions must be made. Only resource references are used, so the file remains portable.

Example 7-10 Persistence.xml of the sample program

```
<?xml version="1.0"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
<persistence-unit name="EmpPU" transaction-type="JTA">
   <jta-data-source>java:comp/env/jdbc/TxDSref</jta-data-source>
   <non-jta-data-source>java:comp/env/jdbc/NoTxDSref</non-jta-data-source>
  <class>com.ibm.itso.entities.Employee</class>
<properties>
   <property name="openjpa.Log" value="DefaultLevel=INFO" />
    <property name="openjpa.jdbc.Schema" value="DSN81010" />
 </properties>
</persistence-unit>
</persistence>
```

The references are resolved in the definition of the container definition file `embeddable.properties`. For every EJB that uses database resources, a `Bean<bean_name>.ResourceRef.BindingName.jdbc` statement must be included. This definition is then assigned to a bean by the container after an EJB is found.

At start, the container looks for enterprise beans in the class path, that is, it looks for Java classes that are annotated, for example, with the `@Stateless` annotation. The EJBs found are then further examined for resource references. They are declared in the EJBs by annotations like `@Resource(name = "jdbc/TxDSref", type = DataSource.class)`, which are resource references.

Resource references must be bound to names in the servers namespace at deployment time. The name in `@Resource(name = "jdbc/TxDSref"` resolves to `java:comp/env/jdbc/TxDSref`, as any resource reference would be named in Java. This reference is likewise defined in the `persistence.xml` file for the JPA container.

Because there is no deployment in this case, the relationship between the resource reference and the real JNDI name for the resource in the server must be defined in the embeddable containers definition file. The following example shows how to accomplish this task:

```
Bean.#bin#EmpBean.ResourceRef.BindingName.jdbc/TxDSref=jdbc/TxDSz
```

The bean named `EmpBean` in the `/bin` directory uses a resource reference named `jdbc/TxDSref` that resolves to the servers JNDI name `jdbc/TxDSz`.

The beans must be registered in the namespace as well so that they can be looked up by clients, such as the `TestEmpBean.java` JUnit test driver. The embeddable container does this task, like any other Java Platform, Enterprise Edition application server, in the `java.global` namespace. The name under which the bean can be found is the following one:

```
java:global/bin/EmpBean!ibm.itso.ejbs.EmpBean
```

The name `ibm.itso.ejbs.EmpBean` is the fully qualified class name of the class in the class path and `/bin/EmpBean` is the location where it can be found. `/bin` in this case is the output folder for compiled classes in the current directory (the project directory in Rational Application Developer). Alternatively, this can be the name of a JAR file containing `@Stateless` annotated classes (without the .jar in the name), which then is taken as the EJB module name.

To run the unit test, the project must have the following JAR files in its class path. Some of the JAR files can be found in a WebSphere Application Server installation. You can get one, for example, if you augment IBM Data Studio with the WebSphere Application Server test environment, as described in Appendix C, “Setting up a WebSphere Application Server test environment on IBM Data Studio” on page 523.

- `com.ibm.ws.ejb.embeddableContainer_8.5.0.jar`
- `com.ibm.ws.jpa.thinclient_8.5.0.jar`
- `db2jcc_license_cu.jar or db2jcc_license_cisuz.jar` for connections to DB2 for z/OS
- `db2jcc4.jar`

Run `TestEmpBean.java` as a JUnit test, which creates a run configuration that must be updated afterward because you must specify a Java agent in your Java system properties to enhance the JPA entities. For the `TestEmpBean` JUnit test run, click **Run** → **Run Configurations**.

An example of how to do this task for JUnit tests is shown in Figure 6-14 on page 325. For the run with the embeddable EJB container, use the following statement:

```
-javaagent:"C:\Programme\ibm\WebSphere\AppServer\runtimes\com.ibm.ws.jpa.thinclien
t_8.5.0.jar
```

Run `TestEmpBean.java` as a JUnit test a second time. This time you should see the green bar for a successful test, as shown in Figure 7-1.

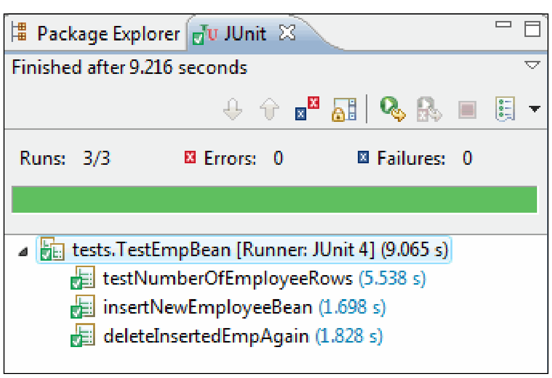


Figure 7-1 Insert and delete a table row with embeddable EJB container - successful test

Despite your success, you might see the following error message during the unit test:

```
NMSV0307E: A Java: URL name was used, but Naming was not configured to handle
Java: URL names. The likely cause is a user in error attempting to specify a Java:
URL name in a non-J2EE client or server environment. Throwing
ConfigurationException.
```

This error message is explained at the following web page:

http://www.ibm.com/support/docview.wss?uid=swg21580598

Why JPA enhancement

You do not need to be concerned about enhancement if you deploy an application into a Java EE 5 compliant application server, such as WebSphere Application Server, because it enhances your entities automatically at run time. Thus, enhancement can be an issue only for Java stand-alone applications, such as with JUnit testing.

What is enhancement? If a Java class is annotated as a JPA entity (`@Entity`), then all its non-transient fields are traced by the JPA run time. Changing a field marks it as *dirty*, which means it must be persisted. Similar monitoring occurs with variables that are annotated with `FetchType.LAZY`, where a special access strategy must be prepared. The class does this work by "enhancing" the setters of applicable fields with newly generated Java code. This can be done at build time by using the `org.apache.openjpa.ant.PCEnhancerTask` utility. It is more common to change the entity at class load time dynamically through a Java agent.

The concept of Java agents was introduced in JDK5 and works by specifying a JAR file with the agent class in the **`-javaagent`** keyword at JRE start time. The `META-INF/MANIFEST.MF` file of this JAR file has the **`Premain-Class`** keyword, which specifies the agent class.

The agent is intercepted in front of your main method. It can configure the runtime environment before your application runs. The agent can then manipulate the class loaders to add JPA code to your classes.

Java agents for JPA enhancement are provided by both the `openjpa-2.2.0.jar` and `com.ibm.ws.jpa.thinclient_8.5.0.jar` files, which can be found in the `runtimes` directory of WebSphere Application Server.

If you run the application in WebSphere Application Server, you can obtain a small performance benefit if you can enhance your entities when you build the application. The application does not attempt to enhance entities that are already enhanced.Enhance the entity classes by using the JPA enhancer tool, **`wsenhancer`**, which can be found in the `bin` directory of WebSphere Application Server.

On a Windows development system where all your entity classes are in the `build` directory, the command to enhance all the entities on the class path looks like Example 7-11.

Example 7-11 wsenhancer command

```
C:\myproject\cd build
C:\myproject\build>%profile_root%\bin\wsenhancer.bat
```

Summary

With WebSphere Application Server, embeddable EJB container agile Java EE development becomes feasible.

7.2.4 Use of alternative JPA persistence providers

The default persistence provider in WebSphere Application Server is the JPA for the WebSphere Application Server persistence provider that is implemented in the `com.ibm.websphere.persistence.PersistenceProviderImpl` class. Alternatively, the Apache OpenJPA persistence provider can be used. These two providers are built into the server and installed automatically during the server installation.

Although they are built from the Apache OpenJPA persistence provider, the JPA for WebSphere Application Server persistence provider contains the following enhancements and differences:

- Static SQL support using the DB2 pureQuery feature.
- Access intent support.
- Enhanced tracing support.
- Version ID generation.
- WebSphere product-specific commands and scripts.
- Translated message files.

Check in-memory caches for lazily loaded many-to-one or one-to-one relationships. Setting the `wsjpa.BrokerImpl` property to `true` specifies that the JPA implementation attempts to load lazy fields from memory at run time if the foreign key data for the lazy fields are available.

If no JPA provider is configured in the `<provider>` element of the `persistence.xml` file within an Enterprise JavaBeans (EJB) module, the default JPA provider that is configured for this server is used. The product is packaged with the JPA for WebSphere Application Server persistence provider that is defined as the default provider. However, it is possible to override this default and specify a different default through the administrative console, as shown in Figure 7-2. To do so, click **Application servers**, select your server, and click **Container Services → Default Java Persistence API settings**.

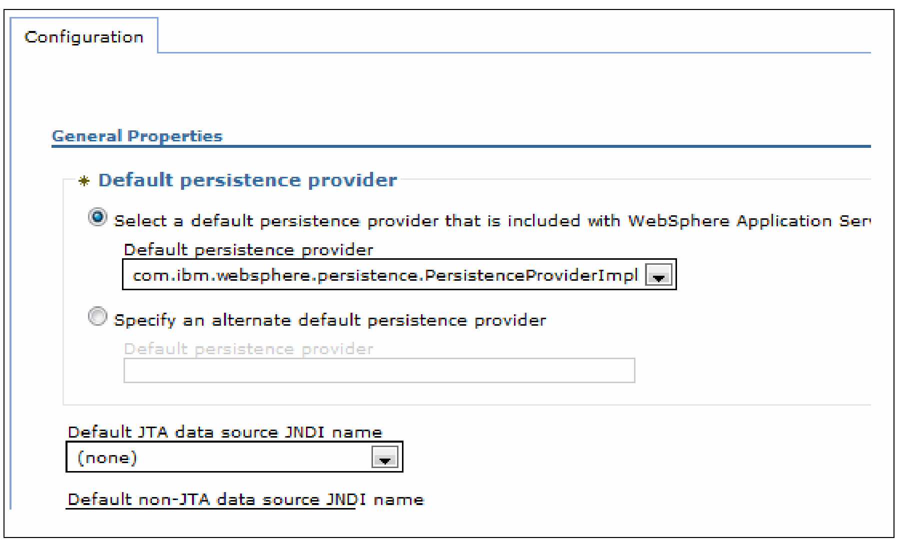


Figure 7-2 Specify an alternative default persistence provider

Depending on your requirements, you can embed the implementation classes of an alternative persistence provider inside an application, or place the persistence provider into a shared library.

7.2.5 Usage of Non-JTA data sources

Some JPA entity features require that a non-JTA data source be specified. An example of this is automatic entity identity generation. Ensure that a non-JTA data source is configured to match your application needs. A non-transactional data source must be defined in WebSphere Application Server for that purpose. To accomplish this task, click **Data sources**, click your data source, click **WebSphere Application Server data source properties**, and select the **Non-transactional datasource** check box, as shown in Figure 7-3.

The application server does not enlist the connections from this data source in global or local transactions. Non-JPA applications must explicitly call **`setAutoCommit(false)`** on the connection if they want to start a local transaction on the connection, and they must commit or roll back the transaction that they started.

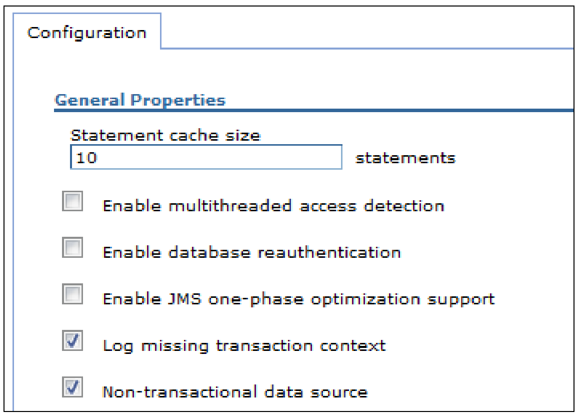


Figure 7-3 Non-transactional data source

7.2.6 Data source resource definition in applications

In support of the Java Enterprise Edition (Java EE) 6 specification, applications can define data sources in annotations or in the deployment descriptor, as shown in Example 7-12.

Example 7-12 Data source definition with Java annotations

```
@DataSourceDefinition(
        name         = "java:comp/env/jdbc/db2",
        className    = "com.ibm.db2.jcc.DB2DataSource",
        databaseName = "SAMPLEDB",
        serverName   = "localhost",
        portNumber   = 50000,
        properties   = { "driverType=4" },
        user         = "user1",
        password     = "pwd1"
        )
```

7.2.7 Definition of the IBM DB2 Driver in WebSphere Application Server V8.5 Liberty Profile

The Liberty profile is a new dynamic profile of WebSphere Application Server V8.5 that provisions only the features that are required by the applications. For example, if an application requires a servlet engine, a Liberty profile can be configured to start only the WebSphere Application Server kernel, the HTTP transport, and the web container. This improves the server start time and results in a small footprint because it does not use the full Java Enterprise Edition stack. Furthermore, if the application needs additional features such as database connectivity, the Liberty profile configuration can be dynamically modified to include the JDBC feature without the needing a server restart.

The name of the product suggests that the server might be just another profile of the WebSphere Application Server product. This is misleading. The Liberty Profile is a new product that is different from WebSphere Application Server. For example, you do not need the Profile Management Tool (PMT) to create a new server. The code may be shared in many cases with the normal application server, but the packaging is different. In addition to the binary files, which have only a less than 50 MB footprint, you need just one XML file to configure a server.

This section cannot show all the details of that server. You can find a detailed description of the Liberty Profile in *WebSphere Application Server V8.5 Administration and Configuration Guide,* SG24-8056. Here, we focus on the definition of the IBM Data Server Driver for JDBC and SQLJ driver and the way to configure a data source.

To run the sample application `db2_jpa_web`, you must define the WebSphere Application Server Liberty Profile `server.xml` file as shown in Example 7-13.

Example 7-13 Server and data source definitions for Liberty Profile

```
<server description="ITSO DB2R1">

   <!-- Enable features -->
   <featureManager>
      <feature>jsp-2.2</feature>
      <feature>jsf-2.0</feature>
      <feature>localConnector-1.0</feature>
      <feature>jpa-2.0</feature>
      <feature>jdbc-4.0</feature>
   </featureManager>

   <httpEndpoint host="localhost"
         httpPort="9080"
         httpsPort="9443"
         id="defaultHttpEndpoint"/>

   <jdbcDriver id="DB2T4" libraryRef="DB2T4LibRef"/>
   <library id="DB2T4LibRef">
      <fileset dir="C:/aps/IBM/SQLLIB/java/"
      includes="db2jcc4.jar db2jcc_license_cu.jar"/>
   </library>

   <dataSource beginTranForResultSetScrollingAPIs="false"
         connectionSharing="MatchCurrentState"
         id="sample_ds"
         isolationLevel="TRANSACTION_READ_COMMITTED"
```

```
      jdbcDriverRef="DB2T4" jndiName="jdbc/sample"
      statementCacheSize="20">
    <connectionManager
      agedTimeout="30m"
      connectionTimeout="10s"
      maxPoolSize="20"
      minPoolSize="5"/>
    <properties.db2.jcc databaseName="DB0Z"
      driverType="4"
      password="db2r1pw"
      portNumber="39000"
      serverName="d0zg.itso.ibm.com"
      currentSchema="DSN81010"
      user="db2r1"/>
  </dataSource>

    <applicationMonitor updateTrigger="mbean"/>
</server>
```

To run the WebSphere Application Server Liberty Profile, you can either install a single server run time or you can augment IBM Data Studio with the WebSphere Application Server test environment, as described in Appendix C, “Setting up a WebSphere Application Server test environment on IBM Data Studio” on page 523, which describes how to install the Liberty Profile in IBM Data Studio. For more information about the data source definition, see the Information Center found at the following website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.wlp.nd.doc%2Ftopics%2Frwlp_ds_appdefined.html

7.2.8 LOB streaming

JPA 2 with DB2 supports LOB streaming. Large amounts of data can be streamed into and out of persistent fields without ever holding that data in memory.

To use LOB streaming, either annotate a `java.io.InputStream` or a `java.io.Reader` property with an `@Persistent` annotation, as shown in Example 7-14.

Example 7-14 LOB streaming

```
@Entity
public class Employee {
...
@Persistent
private InputStream photoStream;
```

There is a known issue with LOB data streaming and DB2 for very large streams. You might have to switch progressive streaming off. For more information, see 7.4, “Known issues with OpenJPA 2.2 and DB2” on page 359.

7.2.9 XML JPA column mapping

DB2 is one of only a few databases that support XML column types, XPath queries, and indexes over these columns. As of DB2 9, mapping of an entity property that is mapped to an XML column is supported by OpenJPA.

With WebSphere Application Server V8.5, column mapping is no longer a server extension feature, but is provided directly by OpenJPA. Therefore, you can find information regarding XML mapping in the Apache OpenJPA documentation directly at the following website:

http://openjpa.apache.org/builds/latest/docs/docbook/manual.html#ref_guide_xmlmapping

Here is an example of this feature. As always with JPA, the process is about mapping Java objects to database columns. In the case of mapping to an XML column, the standard mapping routine cannot be used. Instead, you must specify a third-party mapping tool, which is done by annotating the field containing the JAXB object that is persisted as XML with a strategy handler, as shown in Example 7-15.

The handler knows how to deal with Java Architecture for XML Binding (JAXB) annotations. With JAXB, a Java object can be marshalled or unmarshalled to an XML structure as defined by JAXB annotations. This is analogous to what JPA does with database objects.

Example 7-15 Applying a third-party XML mapping tool using JPA annotations

```
...
@Persistent
@Strategy("org.apache.openjpa.jdbc.meta.strats.XMLValueHandler")
@Persistence(fetch=FetchType.LAZY)
private MyXMLObject xmlObject;
...
```

A sample Java object that is converted to its XML equivalent and is included in the JPA entity that is shown in Example 7-15 looks like Example 7-16.

Example 7-16 Sample JAXB object to be included into a JPA entity

```
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
....
public class MyXMLObject {
    @XmlElement(name = "field1", required = true)
    protected String field1;
   @XmlElement(name = "list1", required = true)
    protected List<String> list1;
    .....
```

The XML structure is built by JAXB and then persisted by JPA. The JAXB JAR files must be on the application class path (`jaxb-api.jar`, `jaxb-impl.jar`, `jsr173_1.0_api.jar`, or the equivalent).

For more information about how WebSphere Application Server is involved in this process, see the following website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.express.doc%2Finfo%2Fexp%2Fae%2Ftejb_jpaColMap.html

7.3 Preferred practices of Java Platform, Enterprise Edition and DB2

This section provides samples of Preferred practices of Java Platform, Enterprise Edition and DB2.

7.3.1 Using resource references

Even today, many classes use data sources directly instead of resource references. Theoretically, there are two ways of direct access:

1. Connection attributes are defined in the `DriverManager.getConnection()` properties and virtually hardcoded. This is an oversimplification for demonstration purposes.
2. The application class uses a managed data source but specifies its JNDI name in the server directly. No reference is used.

Coding infrastructure information in the Java code is a breach of the separation of concerns (SoC) principle and prevents portability. Although this works in many cases, it can lead to some problems in others.

Section 6.6, “JDBC applications in managed environments” on page 326 provides details about resource references.

The application server requires the usage of resource references for the following reasons:

- If application code looks up a data source directly in the JNDI naming space, every connection that is maintained by that data source inherits the properties that are defined in the application. Then, you create the potential for numerous exceptions if you configure the data source to maintain shared connections among multiple applications. For example, an application that requires a different connection configuration might attempt to access that particular data source, resulting in application failure.
- It relieves the programmer from having to know the name of the data source or connection factory at the target application server.
- You can set the default isolation level for a data source through resource references. With no resource reference, you get the default for the JDBC driver that you use.

7.3.2 Providing a JDBC driver in your application libraries

You should not provide a JDBC driver in your application libraries.

For a large Java Platform, Enterprise Edition project, it is normal that the application is built several times a day from a central repository. Hundreds and even thousands of program artifacts are checked out and combined in several deployable archives like .ear, .jar, and .war files. This process mostly is done by specialized builder programs such as Maven.

This process normally must be done for several environments, such as unit tests, integration environments, or quality assurance systems. Some might have predefined database connections, and some might not. Unit tests normally run unmanaged, so they must provide their own database connectivity. In these cases, you need the JDBC driver in your `/lib` directory, but in production you must not have it there, as wrong packaging can easily occur.

There are problems when these “forgotten” drivers interfere with the installed driver in the application server. This is especially the case when your application is deployed with the class loading policy `parent last`. Parent last means that everything in your application is loaded before the classes in the application server.

This has the same effect as a **STEPLIB** in your JCL. Every program in the **STEPLIB** overcomes the one that the system provides, which is not wanted behavior in a production environment.

7.3.3 Resetting the database for each test run

During the development of the application, every developer should have one set of database test data to prevent data corruption. To obtain this set, setting up the database infrastructure with many schemata.

You should always avoid creating tests that depends on the results of preceding tests. The entire database might not need to be reinitialized, but the parts you use should be.

7.3.4 Optimizing generated SQL from persistence frameworks

Persistence frameworks such as JPA or Hibernate by default do not use DB2 capabilities fully because they produce simple but not always performant SQL. If you want to see how JPA functions, enable a WebSphere Application Server trace by running the following string:

```
/F MZSR015,TRACEJAVA='JPA=all: openjpa=all: SystemErr=all: SystemOut=all:
com.ibm.pq=all'
```

You see all the generated dynamic SQL statements. You can see how a change of the JPA class annotations is reflected in the SQL. If the results are not satisfactory, you might have to use native queries where you have full control over the SQL.

Reset the trace by running the following string:

```
/F MZSR015,TRACEINIT
```

7.4 Known issues with OpenJPA 2.2 and DB2

The OpenJPA 2.2. Reference Guide reports some known issues with DB2. Here are the known issues that result from a connection to DB2 for z/OS:

- Floats and doubles might lose their precision when stored.
- Empty char values are stored as NULL.
- Fields of type BLOB and CLOB are limited to 1M. This number can be increased by extending DB2Dictionary.
- The usage of DB2 on z/OS with the IBM Data Server driver requires the **DESCSTAT** subsystem parameter value to be set to `'YES'`. If this parameter is set to `'NO'`, the mapping tool fails with a persistence exception that has this error

  “`Invalid parameter: Unknown column name TABLE_SCHEM"`

  After changing the value of **DESCSTAT**, DB2 metadata tables must be re-created by running the **DSNTIJMS** job.

- When using LOBs with persistent attributes of a streaming data type (for example, `java.io.InputStream`) in the case of a very large LOB, the DB2 Data Server driver automatically uses progressive streaming to retrieve the LOB data. If you get an `LobClosedException`, you might have to set the following string:

  ```
  fullyMaterializeLobData=true;progressiveStreaming=NO
  ```

8

Monitoring WebSphere Application Server applications

As with other applications, you want to know how your WebSphere Application Server and DB2 applications are performing in terms of elapsed time, response time, and transactions per second. There are many tools and techniques that are available that you can use to capture performance-related information in a WebSphere Application Server / DB2 environment.

Before you dive into the different tools and traces that can be used to collect and analyze performance data, it is important to establish a performance analysis strategy or framework in your installation.

Capturing information is the first step; you also must analyze and interpret the data. As performance data can be captured on both the WebSphere Application Server and the DB2 side, it is also important to be able to correlate the data that is collected on both sides.

This chapter covers the following topics:

- Performance monitoring
- Correlating performance data from different sources
- Monitoring from WebSphere Application Server
- Monitoring from the DB2 side
- Using the performance database
- Monitoring from the z/OS side with RMF

8.1 Performance monitoring

Performance monitoring and analysis is a broad subject. The focus in this publication is on monitoring connections into DB2 for z/OS that come from WebSphere Application Server applications. This book looks at the WebSphere Application Server side, z/OS side, and the DB2 side, with an emphasis on DB2.

Before you dive into the different tools and traces that can be used to collect and analyze performance data, it is important to establish a performance analysis strategy or framework in your installation. This typically consists of two components:

- Continuous monitoring
- Detailed monitoring

8.1.1 Continuous monitoring

First, you must decide what performance data to collect, either continuously or at regular intervals, to determine how your applications are performing on a day to day basis.

For DB2 for z/OS, this data is typically DB2 statistics and accounting trace records, and for WebSphere Application Server, the SMF 120 records. When using dynamic SQL, which is used by JDBC applications, it might be a good idea to capture information from the dynamic statement cache at regular intervals to track the performance of individual SQL statements over time. For more information about which DB2 information to capture, see 8.4.1, “Which information to gather” on page 395.

You can use this information to establish a profile for your applications that you can track over time.

You can use this information to understand how your applications perform on a day to day basis, and to determine what has changed if performance deteriorates.

8.1.2 Detailed monitoring

Normally, this type monitoring is used only when there is a problem with the application’s performance. These traces typically introduce a performance impact so you do not want to turn on these traces permanently.

There are many types of traces in all components of the application (WebSphere Application Server, Data server driver, JVM, and DB2) that you have at your disposal. You must understand what detailed traces are available to you and in which cases they can be useful.

8.2 Correlating performance data from different sources

Gathering performance data or trace data in the different components that are involved in running a transaction is one thing, but correlating the data from different components is something else. When you look at the overall performance of the system, this situation is not really an issue, but when you drill down to the transaction level, you should be able to tie together the data that is gathered by the different components. This publication focuses on correlating WebSphere Application Server data, DB2 performance data, and WLM and RMF data for accounting, workload management, or debugging purposes.

From a DB2 side, people traditionally use the planname, authorization ID, or transaction name (correlation name) to identify transactions. In a WebSphere Application Server environment, those items are not always available, or are often the same for all work coming from the application server, and are therefore not helpful.

For example, when you use a type 4 connection using JDBC, there is not really a DB2 plan (other than the generic DISTSERV plan that is used by everybody). Therefore, using the planname is not useful for discovering how specific applications are performing. The same situation applies to the usage of the DB2 authorization ID. In many cases, the application server uses a single authorization ID for all work that is being sent to DB2.

Therefore, to be able to identify WebSphere Application Server applications and correlate them with the information in DB2 and RMF, you typically must use different identifiers and different techniques than the DB2 planname and authorization ID.

8.2.1 Using client information strings for correlating data

A convenient way to establish a "link" between the WebSphere Application Server application and the work inside the database engine is to use *client information strings*[1], which are also called *user agent strings*. They provide extended information about the client to the server to allow better accounting, workload management, or debugging. The extended client information is sent to the database server when the application performs an action that accesses the server, such as running SQL statements.

Setting the client information in your application

You can use IBM Data Server Driver for JDBC and SQLJ-only methods to provide extended client information to the data source and additional information about the client to the server. However, in the IBM Data Server Driver for JDBC and SQLJ Version 4.0 or later, the IBM Data Server Driver for JDBC and SQLJ-only methods are deprecated. You should use `java.sql.Connection.setClientInfo` instead.

Therefore, the IBM Data Server Driver for JDBC and SQLJ-only methods (using `com.ibm.db2.jcc.DB2Connection`) in Table 8-1 are listed for reference only.

Table 8-1 Setting client information through Data Server Driver only methods

Method	**Information provided**
`setDB2ClientAccountingInformation`	Accounting information
`setDB2ClientApplicationInformation`	Name of the application that is working with a connection
`setDB2ClientDebugInfo`	The **`CLIENT DEBUGINFO`** connection attribute for the Unified debugger
`setDB2ClientProgramId`	A caller-specified string that helps the caller identify which program is associated with a particular SQL statement
`setDB2ClientUser`	User name for a connection
`setDB2ClientWorkstation`	Client workstation name for a connection

[1] This Java public class, Class BrokerClientInfo, is a data structure that is used to describe client information.

The IBM Data Server Driver for JDBC and SQLJ Version 4.0 and later supports the usage of client information properties that are part of the JDBC 4.0 standard. Use those properties instead of the 'IBM DB2-only' implementation. An application can also use the `Connection.getClientInfo` method to retrieve client information from the database server, or use the `DatabaseMetaData.getClientInfoProperties` method to determine which client information the IBM Data Server Driver for JDBC and SQLJ driver supports.

Table 8-2 lists the client information property values that the IBM Data Server Driver for JDBC and SQLJ returns for DB2 for z/OS when the connection uses *type 4 connectivity*.

Table 8-2 Client properties that are set by the driver when using a type 4 connection to DB2 for z/OS

Name	MAX_LEN (bytes)	DEFAULT_VALUE	Description
`ApplicationName`	32	`clientProgramName` property value, if set; `"db2jcc_application"` otherwise.	The name of the application that is using the connection. This value is stored in DB2 special register `CURRENT CLIENT_APPLNAME`.
`ClientAccountingInformation`	200	A string that is the concatenation of the following values: ► `"JCCnnnnn"`, where nnnnn is the driver level, such as 04000. ► The value that is set by `DB2Connection.setDB2ClientWorkstation`. If the value is not set, the default is the host name of the local host. ► `applicationName` property value, if set; 20 blanks otherwise. ► `clientUser` property value, if set; eight blanks otherwise.	The value of the accounting string from the client information that is specified for the connection. This value is stored in the DB2 special register `CURRENT CLIENT_ACCTNG`.
`ClientHostname`	18	The value that is set by `DB2Connection.setDB2ClientWorkstation`. If the value is not set, the default is the host name of the local host.	The host name of the computer on which the application that is using the connection is running. This value is stored in the DB2 special register `CURRENT CLIENT_WRKSTNNAME`.
`ClientUser`	16	The value that is set by `DB2Connection.setDB2ClientUser`. If the value is not set, the default is the current user ID that is used to connect to the database.	The name of the user on whose behalf the application that is using the connection is running. This value is stored in the DB2 special register `CURRENT CLIENT_USERID`.

Table 8-3 lists the client information property values that the IBM Data Server Driver for JDBC and SQLJ returns for DB2 for z/OS when the connection uses *type 2 connectivity*.

Table 8-3 Client properties that are set by the driver when using a type 2 connection to DB2 for z/OS

Name	MAX_LEN (bytes)	DEFAULT_VALUE	Description
ApplicationName	32	Empty string	The name of the application that is using the connection. This value is stored in the DB2 special register CURRENT CLIENT_APPLNAME.
ClientAccountingInformation	200	Empty string	The value of the accounting string from the client information that is specified for the connection. This value is stored in the DB2 special register CURRENT CLIENT_ACCTNG.
ClientHostname	18	Empty string	The host name of the computer on which the application that is using the connection is running. This value is stored in the DB2 special register CURRENT CLIENT_WRKSTNNAME.
ClientUser	16	Empty string	The name of the user on whose behalf the application that is using the connection is running. This value is stored in the DB2 special register CURRENT CLIENT_USERID.

Specifying the client information inside the application has the advantage that each application can set its own setting and allows a high degree of granularity, making detailed monitoring of applications and component possible.

The disadvantage of this approach is that you rely on the programmer to provide this information, and monitoring is often not the number one priority, which might result in this information not being passed, which can result in the program running with the wrong priority and missing its service levels when client information is used to classify work in WLM.

Setting the client information in WebSphere Application Server

Specifying the client information at the application server level removes the burden from the programmer and also allows you to dynamically change the settings without having to change the application code.

Specifying client information at the data source level

Figure 8-1 and Figure 8-2 on page 367 show how to specify the client information strings at the data source level.

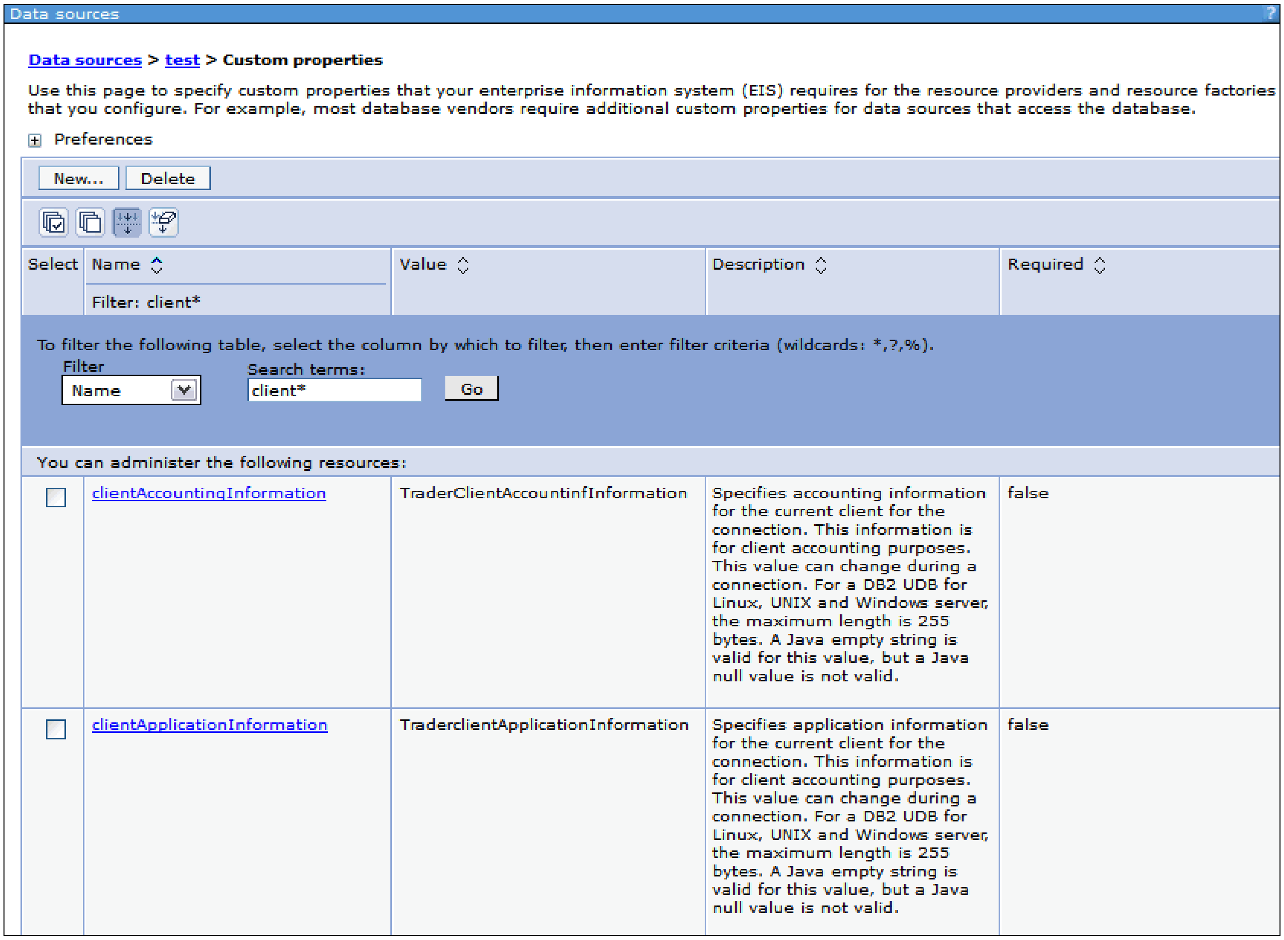


Figure 8-1 Data source Custom properties

They must be entered as custom properties at the data source level. In this example, we use TradeClientUser for the `clientUser` property. When the application connects to DB2 through this data source, this results in the `CURRENT CLIENT_USERID` special register being set to `TradeClientUser`.

You can administer the following resources:				
☐	clientAccountingInformation	TraderClientAccountinfInformation	Specifies accounting information for the current client for the connection. This information is for client accounting purposes. This value can change during a connection. For a DB2 UDB for Linux, UNIX and Windows server, the maximum length is 255 bytes. A Java empty string is valid for this value, but a Java null value is not valid.	false
☐	clientApplicationInformation	TraderclientApplication	Specifies application information for the current client for the connection. This information is for client accounting purposes. This value can change during a connection. For a DB2 UDB for Linux, UNIX and Windows server, the maximum length is 255 bytes. A Java empty string is valid for this value, but a Java null value is not valid.	false
☐	clientUser	TraderClientUser	Specifies the current client user name for the connection. This information is for client accounting purposes. Unlike the JDBC connection user name, this value can change during a connection. For a DB2 UDB for Linux, UNIX and Windows server, the maximum length is 255 bytes.	false
☐	clientWorkstation	TraderClientWorkstation	Specifies the workstation name for the current client for the connection. This information is for client accounting purposes. This value can change during a connection. For a DB2 UDB for Linux, UNIX and Windows server, the maximum length is 255 bytes. A Java empty string is valid for this value, but a Java null value is not valid.	false
Total 38 Filtered total: 4				

Figure 8-2 Specifying client information as data source custom properties

Flowing client information implicitly

If you do not want to set the client information explicitly at the data source level, or when the same data source is used by many different applications, you can choose to set the `enableClientInformation` property, as shown in Figure 8-3.

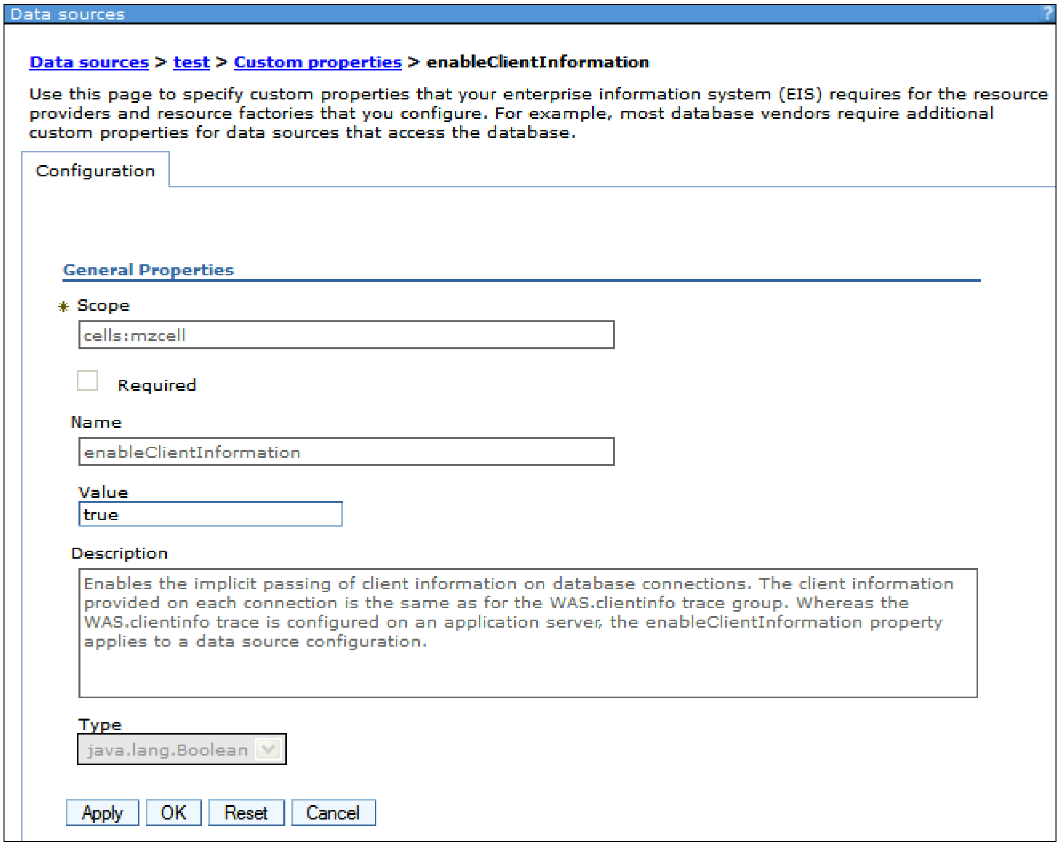


Figure 8-3 Using the enableClientInformation Custom property

For example, during our tests where we used the type 2 driver, we did not specify any specific client information strings. In that case, only the WebSphere Application Server application name (DayTrader-EE6) is passed to DB2 (QWHCEUTX - the user transaction name).

Specifying client information at the application level

Another place to specify the client information is by using the *extended properties* at the application level.

Figure 8-4 on page 369 and Figure 8-5 on page 369 show how to specify this information in more detail by using the Admin console. We use the D0ZG WASTestClientInfo application to demonstrate this feature.

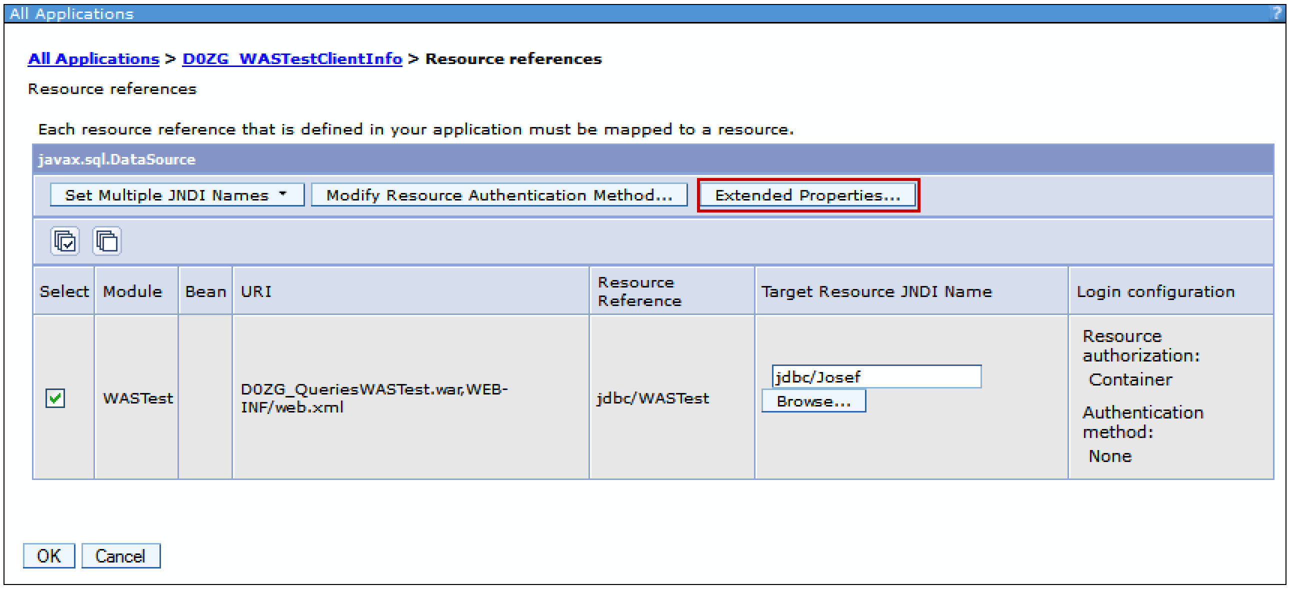


Figure 8-4 Application Resource reference window

In this example, we specified only the `clientApplicationInformation` string.

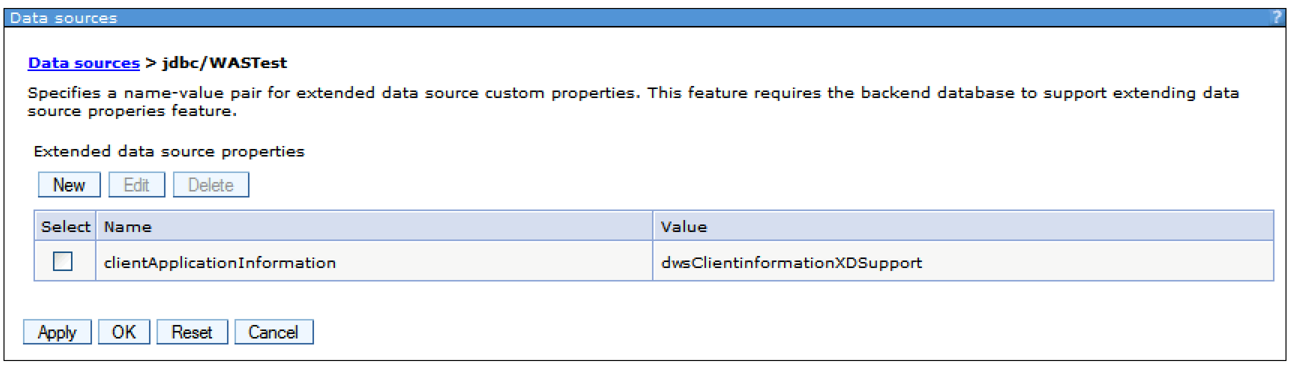


Figure 8-5 Specifying client information as an extended data source property

8.2.2 Using client information strings to classify work in WLM and RMF reporting

You can use the client information when you classify work on the z/OS system. When work must be run on a z/OS system, the work is classified by the z/OS workload manager (WLM) component. A *priority* is assigned to this piece of work, which is done by WLM based on the classification rules that you specify in the WLM policy.

Classifying work when using a type 4 connection

When work comes into DB2 (through the distributed address space) using a Java type 4 connection, an enclave is created and the work is classified by using the WLM classification criteria that is described in the WLM policy.

Then, you list the classification options that are related to the usage of client information when using a type 4 connection:

AI *Accounting information*. This is the value of the DB2 accounting string that is associated with the DDF server thread, as described by the QMDAAINF field in the DB2 DSNDQMDA mapping macro. WLM imposes a maximum length of 143 bytes for accounting information. (The DB2 macros can be found in the `hlq.SDSNMACS` library.)

PC *Process name*. This attribute can be used to classify work by using the application name or the transaction name. The value is defined by the QWHCEUTX field in the DB2 DSNDQWHC mapping macro.

SPM *Subsystem parameter*. This qualifier has a maximum length of 255 bytes. Its content depends on the environment that you run in. When classifying DDF work, the first 16 bytes contain the client's user ID. The next 18 bytes contain the client's workstation name. The remaining 221 bytes are reserved.

If the length of the client's user ID is less than 16 bytes, this attribute uses blanks after the user ID to pad the length. If the length of the client's workstation name is less than 18 bytes, the attribution uses blanks after the workstation name to pad the length.

There are many other classification types that can be used to qualify work. For more information, see the following resources:

- The “Defining Work Qualifiers” section in *z/OS MVS Planning: Workload Management*, SA22-7602-20, found at:

  http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.ieaw100%2Fiea2w1c052.htm

- The “Classification attributes” section in *DB2 10 for z/OS Managing Performance*, SC19-2978, found at:

  http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db2z10.doc.perf/src/tpc/db2z_classificationattributes.htm

The classification rules for the DDF work that we used during this project are shown in Figure 8-6 on page 371.

```
 Subsystem-Type  Xref  Notes  Options  Help
--------------------------------------------------------------------------
                 Modify Rules for the Subsystem Type    Row 19 to 41 of 41
Command ===> ___________________________________________ Scroll ===> CSR

Subsystem Type . : DDF        Fold qualifier names?   N  (Y or N)
Description  . . . DDF Work Requests

Action codes:   A=After     C=Copy        M=Move     I=Insert rule
                B=Before    D=Delete row  R=Repeat   IS=Insert Sub-rule
                                                             More ===>
          --------Qualifier--------              -------Class--------
Action    Type       Name     Start                Service     Report
                                         DEFAULTS: DDFBAT      ________
 ____   1  SI        DOZ*     ___                  DDFDEF      RDOZGDEF
 ____   2    PC        Trade*   ___                DDFONL      RTRADEOZ
 ____   2    PC        dwsClie* ___                DDFONL      RDWSOZ
```

Figure 8-6 Classifying DDF work by using the subsystem and process name

The process name (or application name) is used to qualify the work. Both types of work (Trade* and dwsClie*) use the same service class (DDFONL), but to distinguish between them, we use a separate reporting class for each application:

- The Trade* application uses RTRADE0Z.
- The dwsClie* application uses RDWS0Z.

Classifying work when using a type 2 connection

When using a type 2 connection, the work comes into DB2 through the RRS attach. The work is already classified and the existing work unit is used to run the DB2 part of the work. The classification is done when the transaction starts at the application server, not when it comes into DB2 (as it is for DDF work).

WebSphere Application Server transaction qualification

WLM cannot classify work that is based on the HTTP URL. However, work can be classified in WLM by using *transaction classes*, and WebSphere Application Server offers a configuration option to assign transaction classes to HTTP URLs.

You define the URL to transaction class assignments in a *WebSphere Application Server classification document*, which is a common XML file, as shown in Figure 8-7.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Classification SYSTEM "Classification.dtd" >
<Classification schema_version="1.0">
  <InboundClassification type="http" schema_version="1.0"
        default_transaction_class="WHTTP">
   <http_classification_info uri="/daytrader*" transaction_class="DTRADE">
    </http_classification_info>
<   http_classification_infouri="/wastestClientInfo*"transaction_class="DWS">
    </http_classification_info>
  </InboundClassification>
</Classification>
```

Figure 8-7 WebSphere Application Server classification document wlm.xml

The default transaction class is WHTTP. When the URI uses `/daytrader*`, the transaction is associated with the DTRADE transaction class (and we use transaction class DWS when the URI contains `/wastestClientInfo*`.

The URI information is obtained from the deployment descriptor of the application. To retrieve this information, open the administration console, select the application that you want to classify, and click **View Deployment Descriptor**, as shown in Figure 8-8.

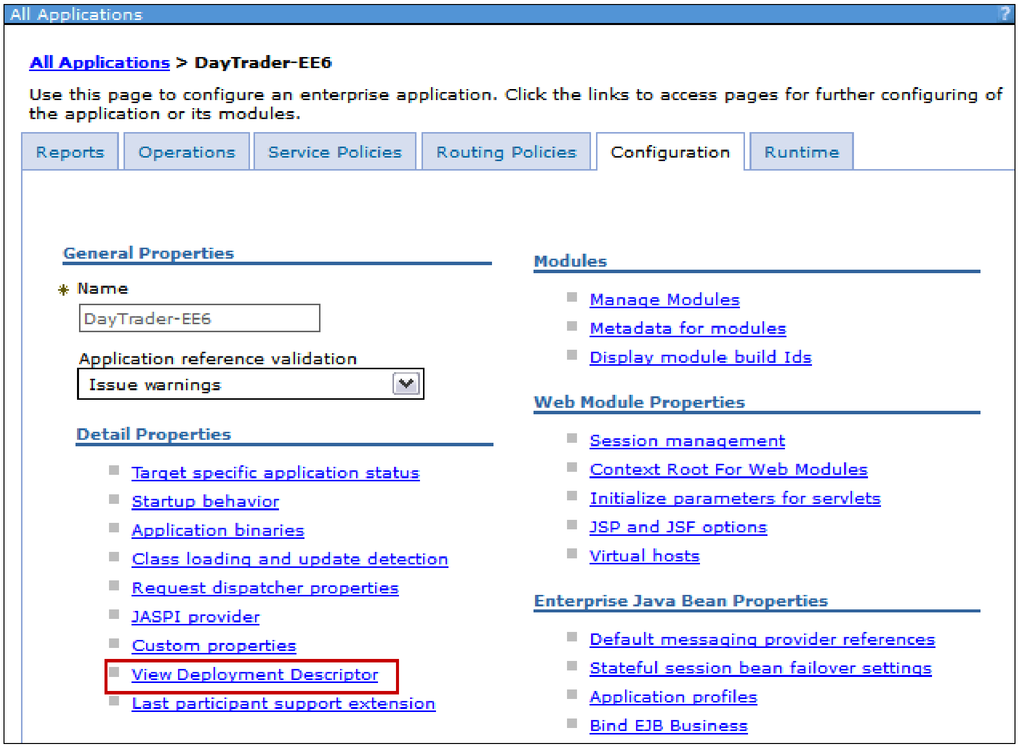


Figure 8-8 Selecting the application's deployment descriptor

The context-root that is shown in Figure 8-9 is used to assign the transaction class.

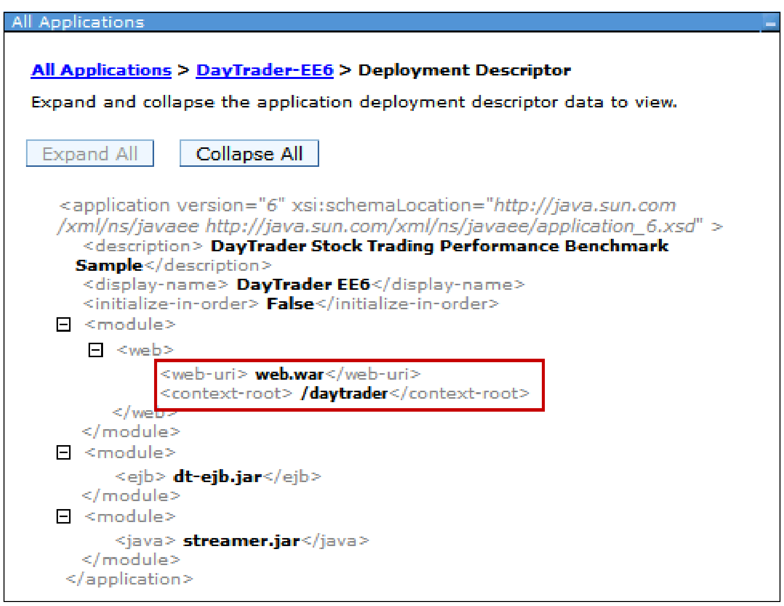


Figure 8-9 DayTrader-EE6 deployment descriptor

Now that you have built the XML file, tell the application server to use this file by setting the `wlm_classification_file` environment variable to the name of our classification file. To do so, navigate to the appropriate WebSphere Application Server console application, click **Environment** → **Manage WebSphere variables**, as shown in Figure 8-10.

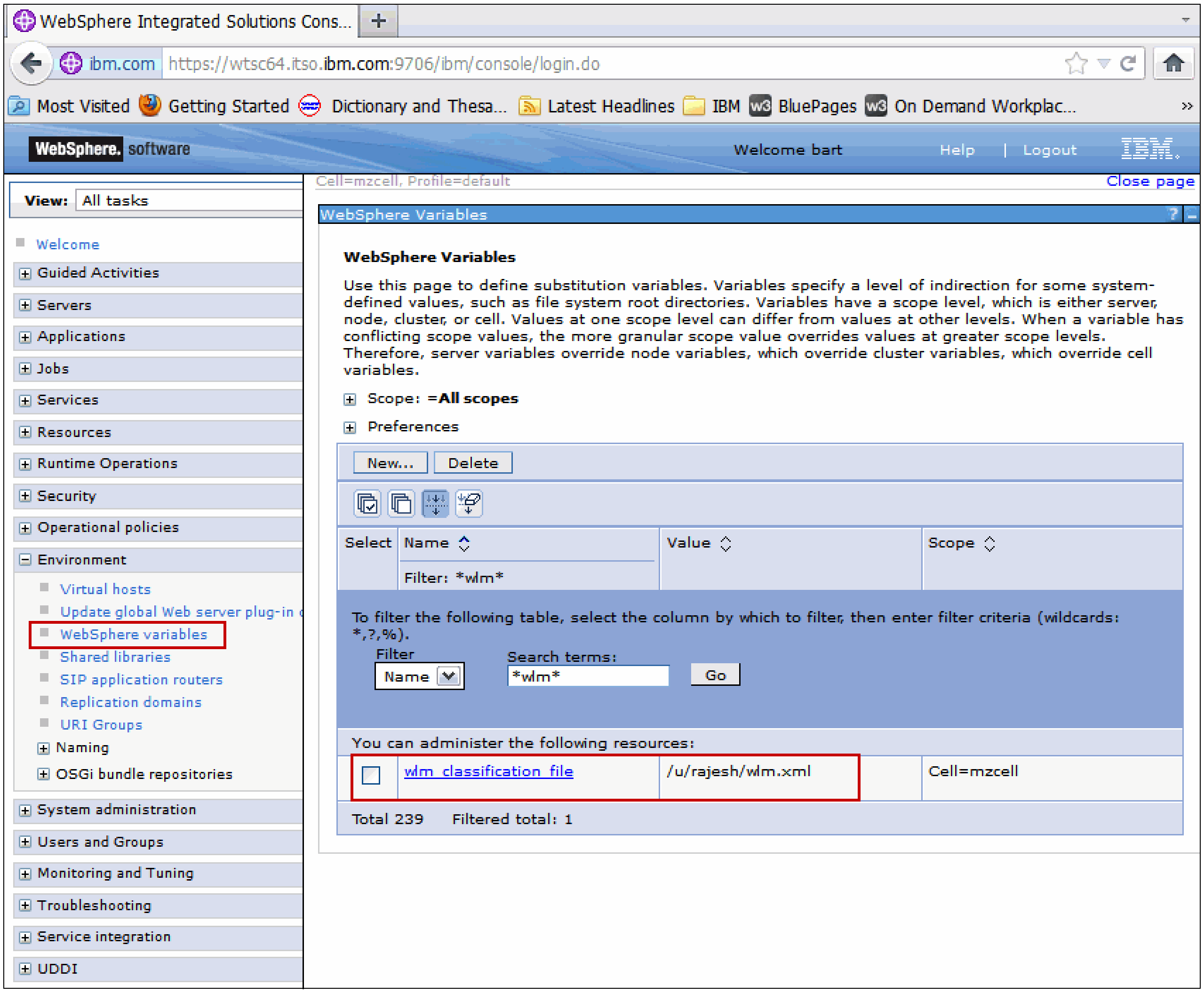


Figure 8-10 Setting the wlm_classification_file variable

> **Note:** Make sure that WebSphere Application Server has the necessary permissions to access the WLM classification file.

During the starting sequence, WebSphere Application Server issues a runtime message to confirm that the current `WLM_CLASSIFICATION_FILE` setting is being used, as shown in Figure 8-11.

```
BBOM0001I wlm_classification_file: /u/rajesh/wlm.xml.
```

Figure 8-11 Current wlm_classification_file that is being used at start

You can also change the classification file, and check the current setting by using console commands, as shown in Figure 8-12. Message BB000211I indicates the success or failure of the **RECLASSIFY** command option.

```
F MZSR014,RECLASSIFY,FILE='/u/rajesh/wlm.xml'

BBOJ0129I: The /u/rajesh/wlm.xml workload classification file was 795
loaded at 2012/08/11 00:22:47.297 (GMT)
BBO00211I MODIFY COMMAND RECLASSIFY,FILE='/u/rajesh/wlm.xml'
COMPLETED SUCCESSFULLY


BBO00211I MODIFY COMMAND RECLASSIFY,FILE='/u/rajesh/wlm.xml' COMPLETED
WITH ERRORS


F MZSR014,DISPLAY,WORK,CLINFO

BBOJ0129I: The /u/rajesh/wlm.xml workload classification file was 798
loaded at 2012/08/11 00:22:47.297 (GMT)
BBO00281I CLASSIFICATION COUNTERS FOR HTTP WORK
BBO00282I CHECKED 0, MATCHED 0, USED 0, COST 3, DESC: HTTP root
BBO00282I CHECKED 0, MATCHED 0, USED 0, COST 2, DESC: HTTP root
BBO00282I CHECKED 0, MATCHED 0, USED 0, COST 3, DESC: HTTP root
BBO00283I FOR HTTP WORK: TOTAL CLASSIFIED 0, WEIGHTED TOTAL COST 0
BBO00188I END OF OUTPUT FOR COMMAND DISPLAY,WORK,CLINFO
```

Figure 8-12 Changing and displaying the workload classification file

MZSR014 is the name of one of our WebSphere Application Server servers.

For more information about the workload classification file, see the following website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.zseries.doc/ae/rrun_wlm_tclass_dtd.html

With the transaction workload classification in place on the WebSphere Application Server side, you can now use the transaction classes in the WLM classification rules, as illustrated in Figure 8-13 on page 375.

```
 Subsystem-Type  Xref  Notes  Options  Help
--------------------------------------------------------------------------
                 Modify Rules for the Subsystem Type     Row 6 to 11 of 11
Command ===> ___________________________________________ Scroll ===> CSR

Subsystem Type . : CB         Fold qualifier names?   N  (Y or N)
Description  . . . WebSphere/Component Broker

Action codes:   A=After     C=Copy        M=Move     I=Insert rule
                B=Before    D=Delete row  R=Repeat   IS=Insert Sub-rule
                                                             More ===>
          --------Qualifier--------               -------Class--------
Action    Type       Name     Start                Service     Report
                                         DEFAULTS: WASONL      RCB
 ____  1  CN         MZSR01   ___                  WASONL      RMZSR01
 ____  2    TC         DTRADE   ___                WASONL      RTRADE
 ____  2    TC         DWS      ___                WASONL      RDWS
```

Figure 8-13 WebSphere work classification using transaction classes

In this example, use the CB subsystem type to classify the WebSphere Application Server work:

CN *Collection name*. This is the logical server name that is defined by using the Component Broker System Management Utility. It represents a set of business objects that are grouped and run in a logical server. This is the *WebSphere Application Server cluster name*.

TC *Transaction class*. This is the name that results from mapping the URI to a name.

DTRADE and DWS are the transaction classes that were assigned through the WLM classification file. When a transaction arrives on the MZSR01 cluster and it is assigned to the DTRADE transaction class, it runs by using the WASONL service class, and it uses the RTRADE RMF reporting class. Using a different reporting class allows you to distinguish between different transactions classes when they use the same service class.

8.2.3 Other techniques to segregate/correlate work

Although usage of client strings is a preferred practice, you can use other techniques to distinguish between different applications or groups of applications that are running in a WebSphere Application Server talking to DB2 for z/OS.

Using separate DB2 collections

When you use JDBC, all applications use the same DB2 packages. They are called SYS* and by default they are bound into a collection called NULLID. If that is the case, you cannot really distinguish between transactions by looking at the DB2 package name.

To get around this problem, you can bind the DB2 JDBC packages into separate collections, one per application or group of applications. To do so, use the **`DB2Binder`** utility and specify **`-collection`** ***`collection-name`***.

You can also use the DB2 **`BIND PACKAGE`** command with the **`COPY(`*`collection-name.package-name`*`)`** keyword. For more information about this command, see 4.3.9, “Bind JDBC packages” on page 165.

On the WebSphere Application Server side, at the data source level, you can use a specific collection for that data source by setting the `currentPackageSet` property, as shown in Figure 8-14. You can also set this property inside your program on the `Connection` or `DataSource` object, but to spare the programmer the effort of specifying this type of information, specify this information at the data source level through the administration console.

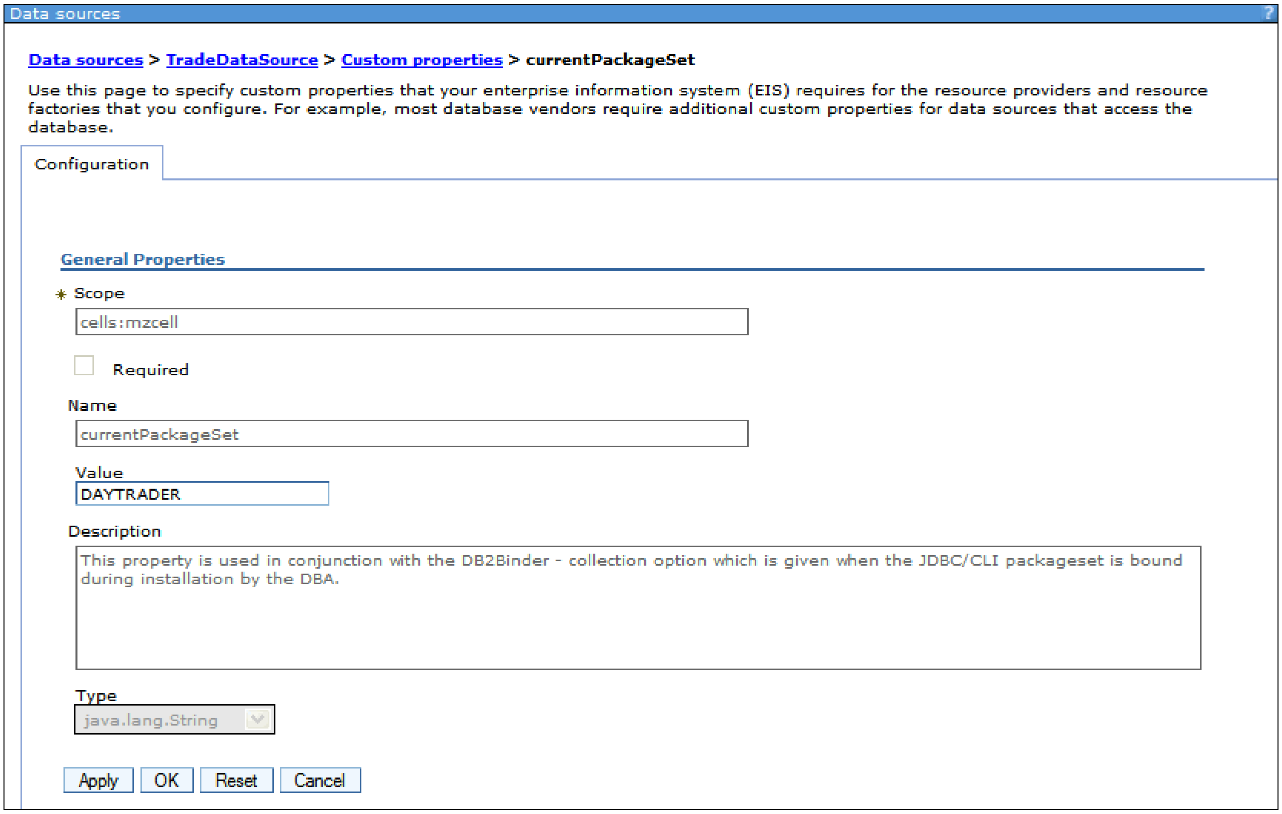


Figure 8-14 Setting currentPackageSet property

Applications that use the TradeDataSource data source now use the SYS* packages from the DAYTRADER collection. When you use the `currentPackageSet` property, all packages that are used by the applications that use this data source must be present in the collection you point to through the `currentPackageSet` property.

Using separate DB2 plan names when using a type 2 connection

If you use a type 2 connection, you do not have to specify a planname when you create a connection to DB2 for z/OS. DB2 uses an *implicit* planname, similar to when you use a type 4 connection. However, you can provide a planname for a type 2 connection. If you create a separate plan for each application (that uses a type 2 connection), you can use the planname to identify the application in much the same way that you do for other types of connections, such as CICS, IMS, and TSO. In this case, create a plan and point it to the standard JDBC packages. For example:

```
BIND PLAN(plntrade) PKLIST(NULLID.*) ..
```

In addition to creating the plan, you must indicate in the data source to use this particular plan by setting the `planName` property, as shown in Figure 8-15 on page 377.

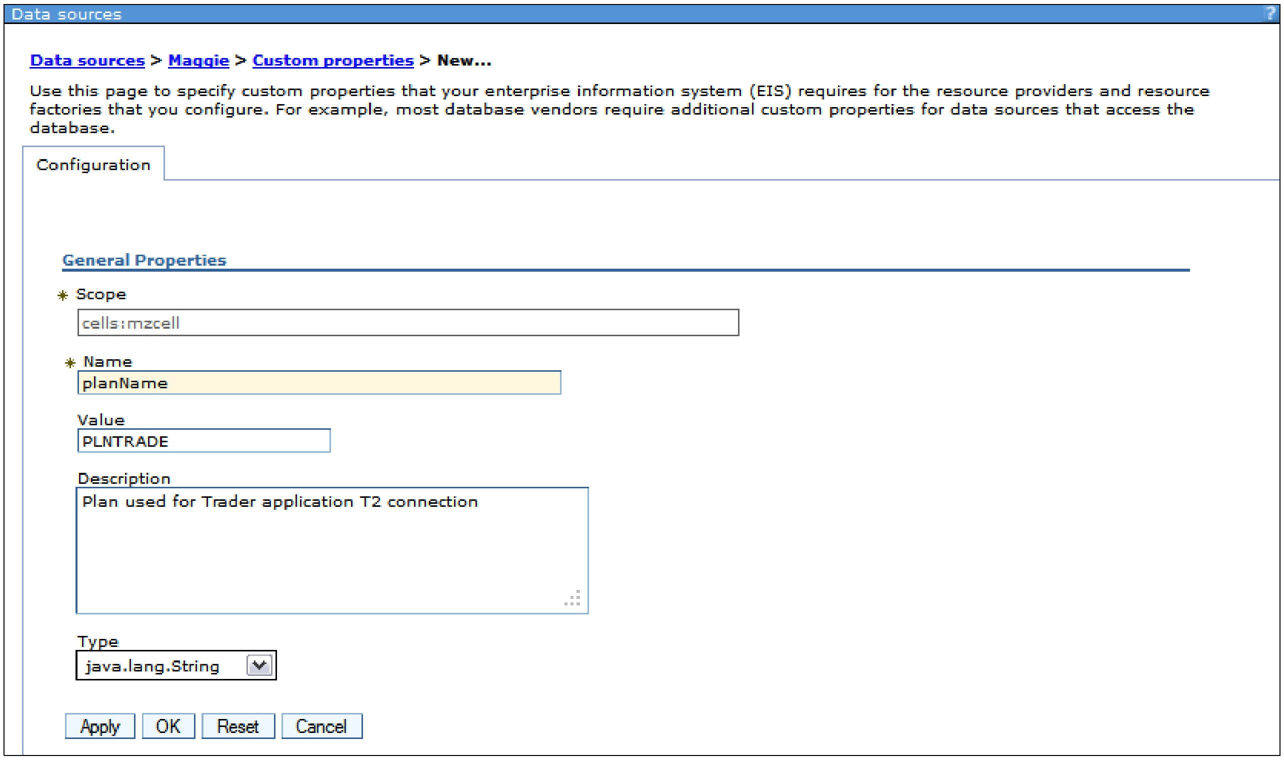


Figure 8-15 Specifying the planName data source property

Now that you have set up a way to correlate WebSphere Application Server, DB2, and RMF information, you can start monitoring our applications.

8.3 Monitoring from WebSphere Application Server

This section describes the different monitoring options and tools you have at your disposal in WebSphere Application Server:

- SMF120 records
- WebSphere Performance Monitoring Infrastructure (PMI)
- Using request metrics

8.3.1 Using SMF 120 records

In z/OS, you can use the system management facilities (SMF) component to gather and record data for evaluating system usage. WebSphere Application Server logs activity data through SMF record type 120. Record 120 includes many subtypes:

- Server Activity record: Subtype 1
- Server Interval record: Subtype 3
- Java Platform, Enterprise Edition Container Activity Record: Subtype 5
- Java Platform, Enterprise Edition Container Interval Record: Subtype 6
- WebContainer Activity record: Subtype 7
- WebContainer Interval record: Subtype 8

- Request Activity record: Subtype 9
- Outbound Request record: Subtype 10

WebSphere Application Server for z/OS Version 7 introduced SMF type 120 subtype 9. It bundles most of the data that is also spread across the other subtypes, and adds additional information, such as how much zAAP processing that the server uses in processing a request. WebSphere Application Server creates one subtype 9 record for every request that the server processes for both external requests (application requests) and internal requests, such as when the controller “talks to” the servant regions.

The other record 120 subtypes are still available, but as subtype 9 combines the information from the other subtypes, we use this information to illustrate the type of information that is available.

Enabling SMF 120 data collection

To collect this information, you must make sure that SMF can write the record type that you want to collect. You can verify this situation by looking at your SMFPRMxx member in PARMLIB or by running **D SMF,O**. A (partial) sample output is shown in Example 8-1.

Example 8-1 D SMF,O output

```
D SMF,O
IEE967I 21.09.51 SMF PARAMETERS 799
        MEMBER = SMFPRM00
        SMFDLEXIT(USER3(IRRADU86)) -- DEFAULT
        SMFDLEXIT(USER2(IRRADU00)) -- DEFAULT
        SMFDPEXIT(USER3(IRRADU86)) -- DEFAULT
        SMFDPEXIT(USER2(IRRADU00)) -- DEFAULT
        EMPTYEXCPSEC(NOSUPPRESS) -- DEFAULT
        MULCFUNC -- DEFAULT
        DSPSIZMAX(2048M) -- DEFAULT
        BUFUSEWARN(25) -- DEFAULT
        BUFSIZMAX(0128M) -- DEFAULT
        MAXEVENTINTRECS(00) -- DEFAULT
        SYNCVAL(00) -- DEFAULT
        DUMPABND(RETRY) -- DEFAULT
        SUBSYS(STC,NOINTERVAL) -- SYS
        SUBSYS(STC,NODETAIL) -- SYS
        SUBSYS(STC,EXITS(IEFUSO)) -- PARMLIB
        SUBSYS(STC,EXITS(IEFUJP)) -- PARMLIB
        SUBSYS(STC,EXITS(IEFUJI)) -- PARMLIB
        SUBSYS(STC,EXITS(IEFACTRT)) -- PARMLIB
        SUBSYS(STC,EXITS(IEFU85)) -- PARMLIB
        SUBSYS(STC,EXITS(IEFU84)) -- PARMLIB
        SUBSYS(STC,EXITS(IEFU83)) -- PARMLIB
        SUBSYS(STC,EXITS(IEFU29)) -- PARMLIB
        SUBSYS(STC,TYPE(0:18,20:98,100:255)) -- PARMLIB
         ....
```

In this case, record types 100 - 255 are enabled, which includes the type 120 record.

By default, WebSphere Application Server does not write any records to SMF. You must activate the writing of these records at the application server level. This can be done in different ways. In this example, we use the administration console interface to enable the SMF recording by clicking **Servers** → **Server Types** → **WebSphere Application Servers**, selecting the server, and clicking **Java and Process Management** → **Process definition** → **Control** → **Environment entries**.

Figure 8-16 shows where to find the Java and Process Management and Process definition options under the Server Infrastructure heading.

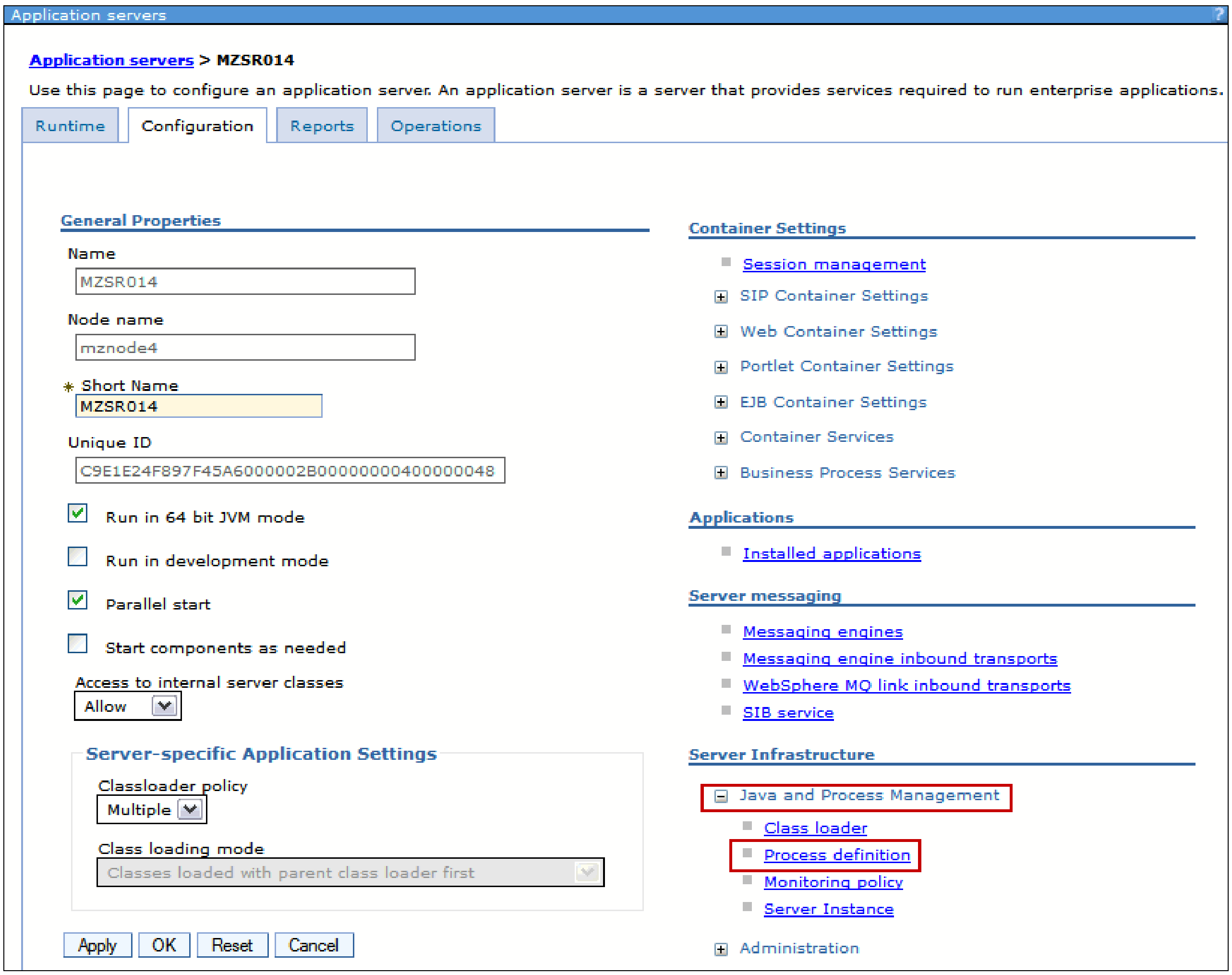


Figure 8-16 Java and Process Management option

The SMF options must be specified at the Control region level. Therefore, you must select the applicable control region and add the new environment variables there, as shown in Figure 8-17.

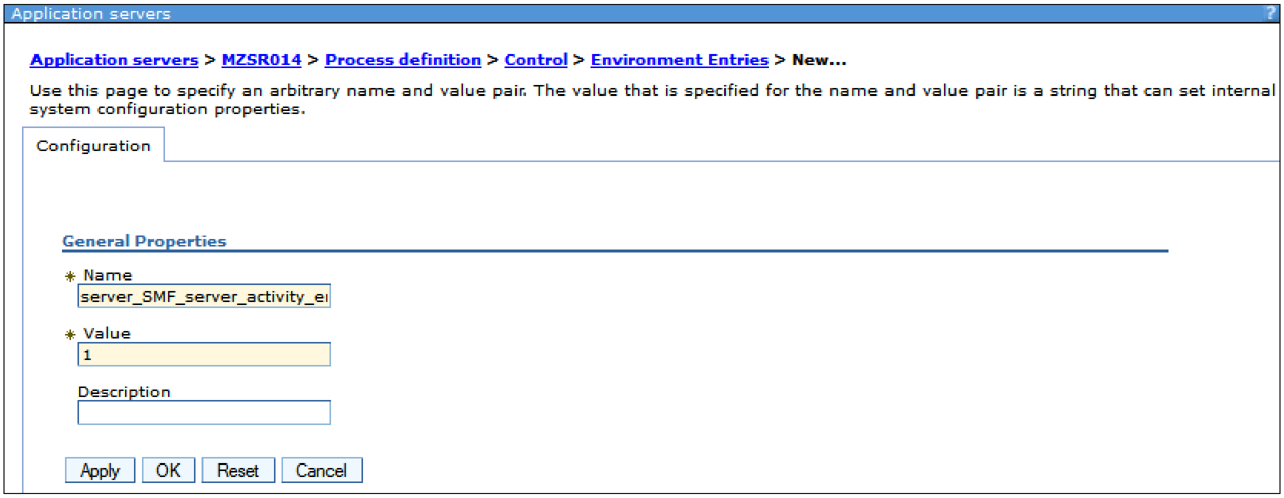


Figure 8-17 Adding an SMF property

We added the following options by using the administration console, which is shown in Figure 8-18, to activate the SMF recording.

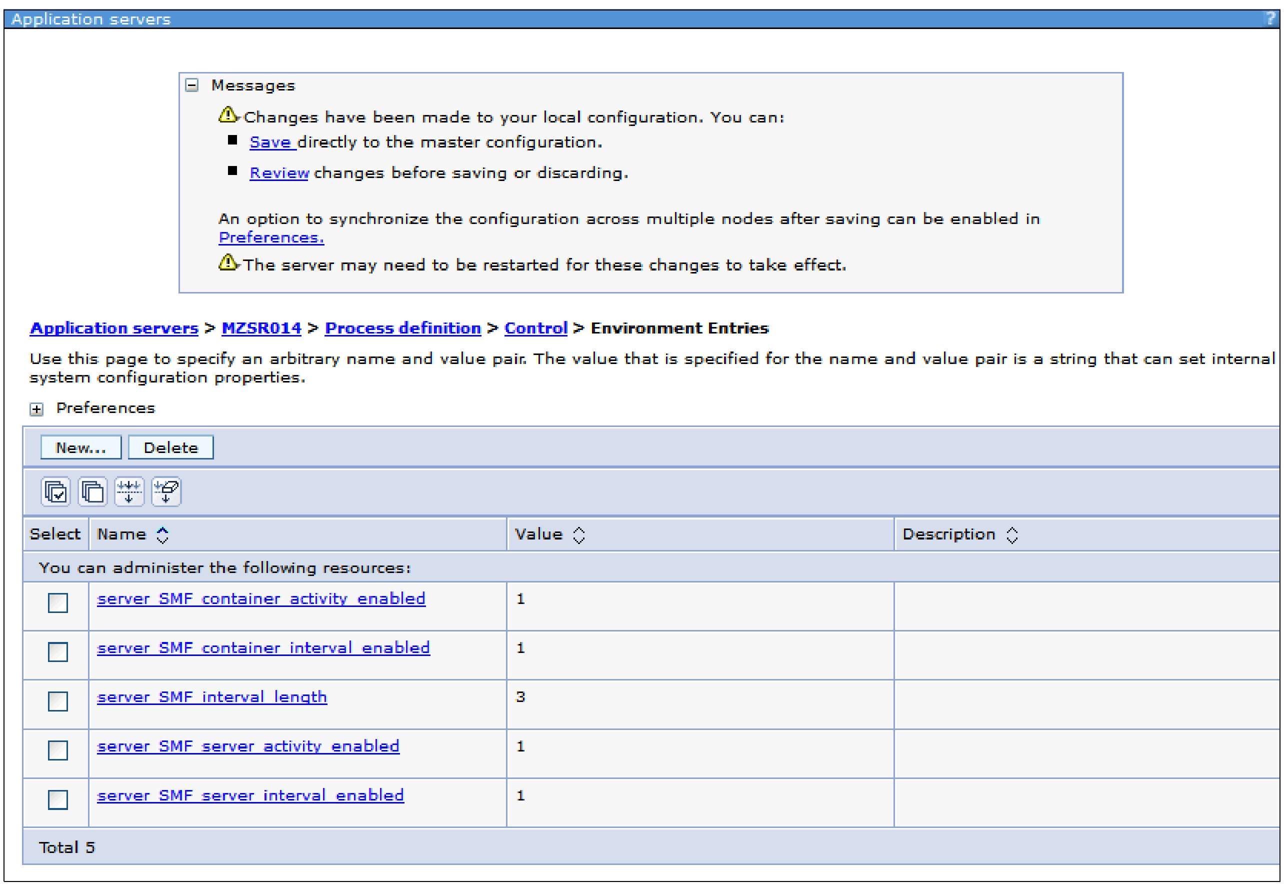


Figure 8-18 SMF recording properties that are set through the administration console

WebContainer SMF recording (SMF 120 subtype 7 and 8) is activated and deactivated along with the activation and deactivation of SMF recording for the Java Platform, Enterprise Edition container (SMF 120 subtype 5 and 6), so there are no specific options to activate subtype 7 and 8.

Here are other properties that you can set (value = 1 to activate) through the administration console:

- **server_SMF_request_activity_enabled** to enable subtype 9

  The following settings add additional information to the subtype 9 record:

  - **server_SMF_request_activity_CPU_detail**
  - **server_SMF_request_activity_timestamps**
  - **server_SMF_request_activity_security**
  - **server_SMF_request_activity_async**
- **server_SMF_outbound_enabled** to enable subtype 10

The subtype 9 record can also be activated by using z/OS console commands, which are illustrated in Example 8-2. MZSR014 is the WebSphere Application Server name. You can also display the current settings that are in effect.

Example 8-2 Using MVS commands to activate SMF 120 type 9 recording

```
F MZSR014,SMF,REQUEST,ON
BBOO0211I MODIFY COMMAND SMF,REQUEST,ON COMPLETED SUCCESSFULLY

F MZSR014,SMF,REQUEST,CPU,ON
BBOO0211I MODIFY COMMAND SMF,REQUEST,CPU,ON COMPLETED SUCCESSFULLY

F MZSR014,SMF,REQUEST,TIMESTAMPS,ON
BBOO0211I MODIFY COMMAND SMF,REQUEST,TIMESTAMPS,ON COMPLETED
SUCCESSFULLY

F MZSR014,SMF,REQUEST,SECURITY,ON
BBOO0211I MODIFY COMMAND SMF,REQUEST,SECURITY,ON COMPLETED
SUCCESSFULLY

F MZSR014,DISPLAY,SMF
BBOO0344I SMF 120-9: FORCED_ON, CPU USAGE: FORCED_ON, TIMESTAMPS:
FORCED_ON, SECURITY INFO: FORCED_ON, ASYNC: OFF
BBOO0345I SMF 120-9: TIME OF LAST WRITE: 2012/08/07 18:25:15.814883,
SUCCESSFUL WRITES: 433, FAILED WRITES: 0
BBOO0346I SMF 120-9: LAST FAILED WRITE TIME: NEVER, RC: 0
BBOO0389I SMF 120-10: OFF
BBOO0387I SMF 120-10: TIME OF LAST WRITE: NEVER, SUCCESSFUL WRITES: 0,
FAILED WRITES: 0
BBOO0388I SMF 120-10: LAST FAILED WRITE TIME: NEVER, RC: 0
BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,SMF
```

Note: The changes that you make to the SMF 120 subtype 9 settings through console commands remain active only until the server is restarted, and changes that are made through the administration console remain after the server is restarted.

Analyzing SMF 120 information

If you want to learn how to format SMF type 120 records, you can download a sample Java application that is called the SMF Browse from the WebSphere Application Server for z/OS website:

https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=zosos390

The documentation for the SMF Browser is available in the browser package.

The WebSphere Application Server information Center also has information that is related to this topic in the “Viewing the output data set” topic. It is available at the following website:

http://www14.software.ibm.com/webapp/wsbroker/redirect?version=phil&product=was-nd-zos&topic=ttrb_SMFviewdata

Another excellent source of information about SMF 120 subtype 9 records is the white paper *Understanding SMF Record Type 120, Subtype 9.* It is available at the following website:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101342

In our example, we used the following commands to generate a summary (SUMPERF) and a detailed (DEFAULT) report of the SMF 120 records that were collected during one of the runs that were using the Trader sample application:

```
java -cp bbomsmfv.jar:batchsmf.jar com.ibm.ws390.sm.smfview.SMF
'INFILE(BART.WAS.TEST1.SMF120)' 'PLUGIN(PERFSUM,/tmp/smf120sum.txt)'
java -cp bbomsmfv.jar:batchsmf.jar com.ibm.ws390.sm.smfview.SMF
'INFILE(BART.WAS.TEST1.SMF120)' 'PLUGIN(DEFAULT,/tmp/smf120detail.txt)'
```

The second parm of the **PLUGIN** option indicates the file that the output is directed to.

The SMF 120 records contain much f information. We describe only the subtype 9 record in a here. Samples of subtypes 1, 3, 7, and 8 for both the summary and detailed output can be found in Appendix E, “SMF 120 records subtypes 1, 3, 7, and 8” on page 545.

Example 8-3 shows the summary (SUMPERF) output by the SMF Browser program for one of the SMF 120.9 (Request Activity) records. It shows the elapsed and CPU time (in microseconds) and the CPU time that was used on a zAAP engine, in case that is available. In this case, the entire request was offloaded to zAAP (CPU and zAAP time are the same). The record also provides information about the time the request came into the application server, when it was queued, dispatched, and ended. The output also indicates which programs ran; in this case, they are all JSPs.

Example 8-3 Subtype 9 summary

```
SMF -Record Time     Server   Bean/WebAppName  Bytes   Bytes  # of  El.Time  CPU_Time(uSec)     Other SMF 120.9
Numbr -Type hh:mm:ss Instance  Method/Servlet  toSvr   frSvr Calls   (msec)   Tot-CPU      zAAP Sections Present
1---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+ ----------------
 694 120.9 19:58:06 MZSR014 STC24171-HTTP  /                          25       584       584
        .9Ts:  2012/08/10 23:58:06.377368 Received
        .9Ts:  2012/08/10 23:58:06.377459 Queued
        .9Ts:  2012/08/10 23:58:06.386165 Dispatched
        .9Ts:  2012/08/10 23:58:06.401038 dispatchComplete
        .9Ts:  2012/08/10 23:58:06.402788 Complete
        .9N ip addr=9.12.6.9 port=24146        832    6176                                         .9Cl/daytrader/ap
         9CPU:Web DayTrader-EE6#web.wa/TradeAppServlet         1        0        29
         9CPU:Web DayTrader-EE6#web.wa//quote.jsp              1        0        61
         9CPU:Web DayTrader-EE6#web.wa//displayQuote.jsp       1        2       214
```

Example 8-4 shows the detailed (DEFAULT) output that is created by the SMF Browser program for the same SMF 120.9 (Request Activity) record that we analyzed in Example 8-3 on page 382. The detailed output contains much information.

One thing that might be of interest is the transaction class that is used by the transaction.

Example 8-4 Subtype 9 detailed output

```
--------------------------------------------------------------------------------
Record#: 694;
  Type: 120; Size: 3624; Date: Fri Aug 10 19:58:06 EDT 2012;
  SystemID: SC64; SubsystemID: WAS; Flag: 94;
  Subtype: 9 (REQUEST ACTIVITY);

  #Subtype Version: 2;
  Index of this record: 1;
  Total number of records: 1;
  record continuation token    * 000000c4 0120a481 -------- -------- *
  #Triplets: 11;
  Triplet #: 1; offsetDec: 204; offsetHex: cc; lengthDec: 76; lengthHex: 4c; count: 1;
  Triplet #: 2; offsetDec: 280; offsetHex: 118; lengthDec: 156; lengthHex: 9c; count: 1;
  Triplet #: 3; offsetDec: 436; offsetHex: 1b4; lengthDec: 68; lengthHex: 44; count: 1;
  Triplet #: 4; offsetDec: 504; offsetHex: 1f8; lengthDec: 736; lengthHex: 2e0; count: 1;
  Triplet #: 5; offsetDec: 1240; offsetHex: 4d8; lengthDec: 132; lengthHex: 84; count: 1;
  Triplet #: 6; offsetDec: 1372; offsetHex: 55c; lengthDec: 188; lengthHex: bc; count: 1;
  Triplet #: 7; offsetDec: 1560; offsetHex: 618; lengthDec: 420; lengthHex: 1a4; count: 3;
  Triplet #: 8; offsetDec: 0; offsetHex: 0; lengthDec: 0; lengthHex: 0; count: 0;
  Triplet #: 9; offsetDec: 1980; offsetHex: 7bc; lengthDec: 1644; lengthHex: 66c; count: 3;
  Triplet #: 10; offsetDec: 0; offsetHex: 0; lengthDec: 0; lengthHex: 0; count: 0;
  Triplet #: 11; offsetDec: 0; offsetHex: 0; lengthDec: 0; lengthHex: 0; count: 0;

  Triplet #: 1; Type: PlatformNeutralSection;
    Server Info Version    : 1;
    Cell Short Name        : MZCELL;
    Node Short Name        : MZNODE4;
    Cluster Short Name     : MZSR01;
    Server Short Name      : MZSR014;
    Server/Controller PID  : 65569;
    WAS Release            : 8;
    WAS Release x of .x.y.z: 5;
    WAS Release y of .x.y.z: 0;
    WAS Release z of .x.y.z: 0;
    Reserved                   * 00000000 00000000 00000000 00000000 *
                               * 00000000 00000000 00000000 00000000 *

  Triplet #: 2; Type: ZosServerInfoSection;
    Server Info Version                : 2;
    System Name (CVTSNAME)             : SC64;
    Sysplex Name                       : SANDBOX;
    Controller Name                    : MZSR014;
    Controller Job ID                  : STC24171;
    Controller STOKEN                   * 000002a4 00000062 -------- -------- *
    Controller ASID (HEX)               * 00a9---- -------- -------- -------- *
    CPU Usage Overflow                 : 0;
   CEEGMTO failed/unavailable         : 1;
   Cluster UUID                        * c9e1e23b fc4d1532 000002b0 00000004 *
                                       * 00000048 -------- -------- -------- *
   Server UUID                         * c9e1e24f 897f45a6 000002b0 00000004 *
                                       * 00000048 -------- -------- -------- *
   Daemon Group Name                  : MZCELL;
   LE GMT Offset (Hours) from CEEGMTO  : 0;
```

```
  LE GMT Offset (Minutes) from CEEGMTO: 0;
  LE GMT Offset (Seconds) from CEEGMTO: 0;
  System GMT Offset from CVTLDTO (HEX) * ffffca5b 17000000 -------- -------- *
  Maintenance Level                   : gm1215.02;
  Reserved                             * 00000000 00000000 00000000 00000000 *
                                       * 00000000 -------- -------- -------- *

Triplet #: 3; Type: PlatformNeutralRequestInfoSection;
  Version                    : 1;
  Dispatch Servant PID (HEX)  * 00010083 -------- -------- -------- *
  Dispatch Task ID            * 25982400 0000003c -------- -------- *
  Dispatch TCB CPU           : 583;
  Completion Minor Code       * 00000000 -------- -------- -------- *
  Reserved                    * 00000000 -------- -------- -------- *
  Request Type               : 2 (HTTP);
  Reserved                    * 00000000 00000000 00000000 00000000 *
                              * 00000000 00000000 00000000 00000000 *

Triplet #: 4; Type: ZosRequestInfoSection;
  Server Info Version           : 2;
  Time Received                  * ca002664 00198ea5 00000000 00000000 *
  Time Queued                    * ca002664 001f3b25 00000000 00000000 *
  Time Dispatched                * ca002664 023f5e74 00000000 00000000 *
  Time Dispatch Complete         * ca002664 05e0ee74 00000000 00000000 *
  Time Complete                  * ca002664 064e4ca3 00000000 00000000 *
  Servant Job Name              : MZSR014S;
  Servant Job ID                : STC24174;
  Servant SToken                 * 000003f0 00000120 -------- -------- *
  Servant ASID (HEX)             * 00fc---- -------- -------- -------- *
  Reserved for alignment         * 0000---- -------- -------- -------- *
  Servant Tcb Address            * 007bdad0 -------- -------- -------- *
  Servant TToken                 * 000003f0 00000120 0000004c 007bdad0 *
  CPU Offload                   : 580;
  Servant Enclave Token          * 000000c4 0120a481 -------- -------- *
  Reserved                       * 00000000 00000000 00000000 00000000 *
                                 * 00000000 00000000 00000000 00000000 *
  Enclave CPU So Far            : 2393980;
  zAAP CPU So Far               : 2393980;
  zAAP Eligible on CP           : 0;
  zIIP on CPU So Far            : 0;
  zIIP Qual Time So Far         : 0;
  zIIP CPU So Far               : 0;
  zAAP Normalization Factor     : 256;
  Enclave Delete CPU            : 2419626;
  Enclave Delete zAAP CPU       : 2393980;
  Enclave Delete zAAP Norm      : 256;
  Reserved                       * 00000000 -------- -------- -------- *
  Enclave Delete zIIP Norm      : 0;
  Enclave Delete zIIP Service   : 0;
  Enclave Delete zAAP Service   : 34;
  Enclave Delete CPU  Service   : 34;
  Enclave Delete Resp Time Ratio : 2;
  Reserved for alignment         * 00000000 00000000 00000000 -------- *
  GTID                           * 00000000 00000000 00000000 00000000 *
                                 * 00000000 00000000 00000000 00000000 *
                                 * 00000000 00000000 00000000 00000000 *
                                 * 00000000 00000000 00000000 00000000 *
                                 * 00000000 00000000 00------ -------- *
  Reserved for alignment         * 000000-- -------- -------- -------- *
  Dispatch Timeout              : 0;
```

```
  Transaction Class              : ;
  Flags                           * 84d00000 -------- -------- -------- *
  Reserved                        * 00000000 00000000 00000000 00000000 *
                                  * 00000000 00000000 00000000 00000000 *
  Classification attributes: ;
  Stalled thread dump action     : 3;
  CPU time used dump action      : 3;
  DPM dump action                : 3;
  Timeout recovery               : 2;
  Dispatch timeout               : 300;
  Queue timeout                  : 297;
  Request timeout                : 180;
  CPU time used limit            : 0;
  DPM interval                   : 0;
  Message Tag                    : ;
  Obtained affinity              : ;
  Routing affinity               : C9E1E24F897F45A6000002B00000000400000048sn6zGpx_39-MGb4qNtoil8h;

Triplet #: 5; Type: TimeStampSection;
  Time Received         : 2012/08/10 23:58:06.377368;
  Time Queued           : 2012/08/10 23:58:06.377459;
  Time Dispatched       : 2012/08/10 23:58:06.386165;
  Time Dispatch Complete: 2012/08/10 23:58:06.401038;
  Time Complete         : 2012/08/10 23:58:06.402788;
  Reserved for alignment     * 0000---- -------- -------- -------- *

Triplet #: 6; Type: NetworkDataSection;
  Version             : 1;
  Bytes Received      : 832;
  Bytes Sent          : 6176;
  Target Port         : 99;
  Origin String Length: 27;
  Origin String       : ip addr=9.12.6.9 port=24146;
  Reserved                   * 00000000 00000000 00000000 00000000 *
                             * 00000000 00000000 00000000 00000000 *

Triplet #: 7; Type: ClassificationDataSection;
  Version    : 1;
  Data Type  : 6 (URI);
  Data Length: 14;
  Data       : /daytrader/app (EBCDIC);

Triplet #: 7; Type: ClassificationDataSection;
  Version    : 1;
  Data Type  : 7 (Target Hostname);
  Data Length: 19;
  Data       : wtsc64.itso.ibm.com (EBCDIC);

Triplet #: 7; Type: ClassificationDataSection;
  Version    : 1;
  Data Type  : 8 (Target Port);
  Data Length: 2;
  Data       : 99 (EBCDIC);

Triplet #: 9; Type: CpuUsageSection;
  Version         : 1;
  Data Type       : 2;
  Request Type    : 2 (Web Container);
  CPU Time        : 29;
  Elapsed Time    : 0;
```

```
  Invocation Count: 1;
  String 1 Length : 21;
  String 1        : 21 (DayTrader-EE6#web.war);
  String 2 Length : 15;
  String 2        : 15 (TradeAppServlet);

Triplet #: 9; Type: CpuUsageSection;
  Version         : 1;
  Data Type       : 2;
  Request Type    : 2 (Web Container);
  CPU Time        : 61;
   Elapsed Time    : 0;
   Invocation Count: 1;
   String 1 Length : 21;
   String 1        : 21 (DayTrader-EE6#web.war);
   String 2 Length : 10;
   String 2        : 10 (/quote.jsp);

 Triplet #: 9; Type: CpuUsageSection;
   Version         : 1;
   Data Type       : 2;
   Request Type    : 2 (Web Container);
   CPU Time        : 214;
   Elapsed Time    : 2;
   Invocation Count: 1;
   String 1 Length : 21;
   String 1        : 21 (DayTrader-EE6#web.war);
   String 2 Length : 17;
   String 2        : 17 (/displayQuote.jsp);

--------------------------------------------------------------------------------
```

8.3.2 WebSphere Application Server Performance Monitoring Infrastructure

A typical web application consists of a web server, application server, and a database. Monitoring and tuning the application server is critical to the overall performance. Performance Monitoring Infrastructure (PMI) is the core monitoring infrastructure for WebSphere Application Server. PMI provides a comprehensive set of data that explains the runtime and application resource behavior. For example, PMI provides database connection pool size, servlet response time, Enterprise JavaBeans (EJB) method response time, Java virtual machine (JVM) garbage collection time, and processor usage.

Using PMI data, performance bottlenecks in the application server can be identified and addressed. For example, one of the PMI statistics in the Java DataBase Connectivity (JDBC) connection pool is the number of statements that are discarded from the prepared statement cache, which we use as an example to illustrate the value of PMI data. This statistic can be used to adjust the prepared statement cache size to minimize the discards and to improve the database query performance.

PMI data can be monitored and analyzed by IBM Tivoli® Performance Viewer, other Tivoli tools, your own applications, or third-party tools. As Tivoli Performance Viewer ships with WebSphere Application Server, we use it to visualize the PMI data in our example.

Java Platform, Enterprise Edition (Java EE) 1.4 includes a Performance Data Framework that is defined as part of JSR-077 (Java Platform, Enterprise Edition Management Specification). This framework specifies the performance data that must be available for various Java EE components. WebSphere Application Server PMI complies with Java EE 1.4 standards by implementing the Java EE 1.4 Performance Data Framework. In addition to providing statistics that are defined in Java EE 1.4, PMI provides additional statistics about the Java EE components, such as servlets and enterprise beans, and WebSphere Application Server -specific components, such as thread pools.

Obtaining PMI data

PMI data covers many performance aspects of web applications. As this publication focuses on web applications that access DB2 for z/OS database resources, we limit the discussion to a few parameters that are related to database access as an illustration of how to obtain and use PMI information.

You activate PMI data collection at the Application Server level. To do so, expand **Monitoring and Tuning** in the left pane of the administration console and click **Performance Monitoring Infrastructure**. Select the application server that you want to collect data for (MZSR014 in our case) and click **Start Monitoring**, as shown in Figure 8-19.

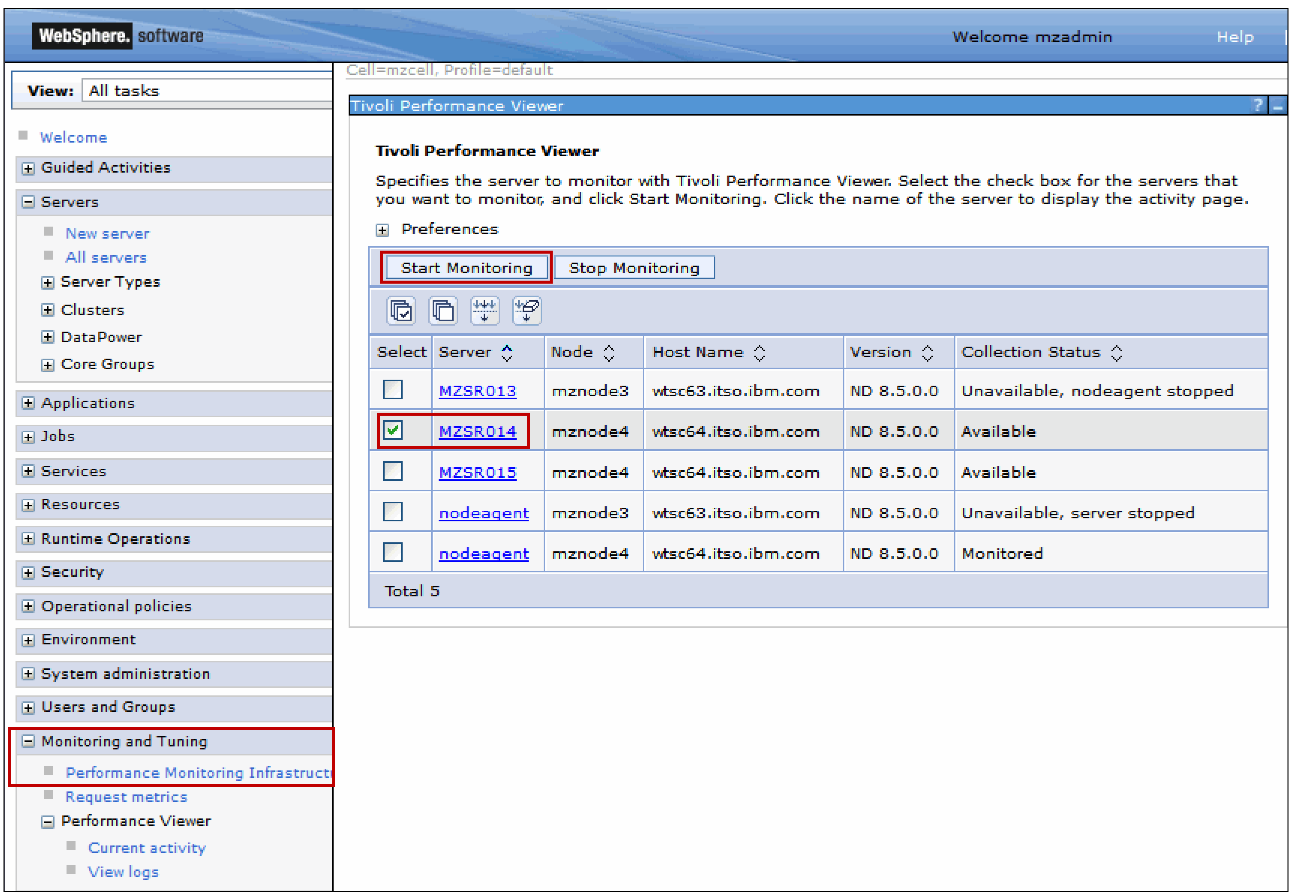


Figure 8-19 Start PMI collection

You receive confirmation that monitoring has started and the server is now in the Monitored status, as shown in Figure 8-20.

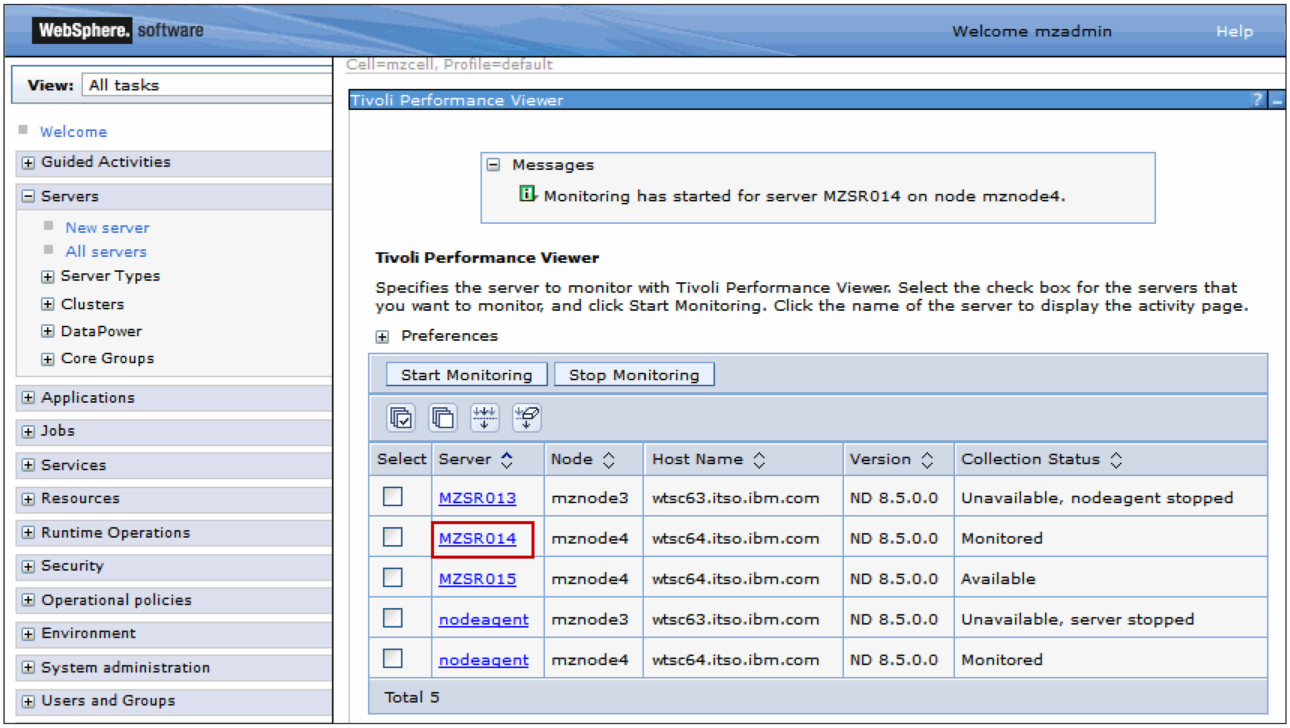


Figure 8-20 PMI collection that is activated for the application server

PMI can collect many different types of data at various levels of detail. To do so, click **Monitoring and Tuning** → **Request Metrics**. We used the standard settings in our example. For more information about the different options and levels of granularity at which PMI data can be collected, see the following website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.express.doc/ae/tprf_pmi_encoll.html

Viewing PMI data

To view PMI data, use the Tivoli Performance Viewer tool that is built into WebSphere Application Server. To use it, expand **Monitoring and Tuning** in the left pane of the administration console and click **Performance Viewer** → **Current activity**. Then, select the server that you want to see the PMI data for. A window similar to Figure 8-21 opens.

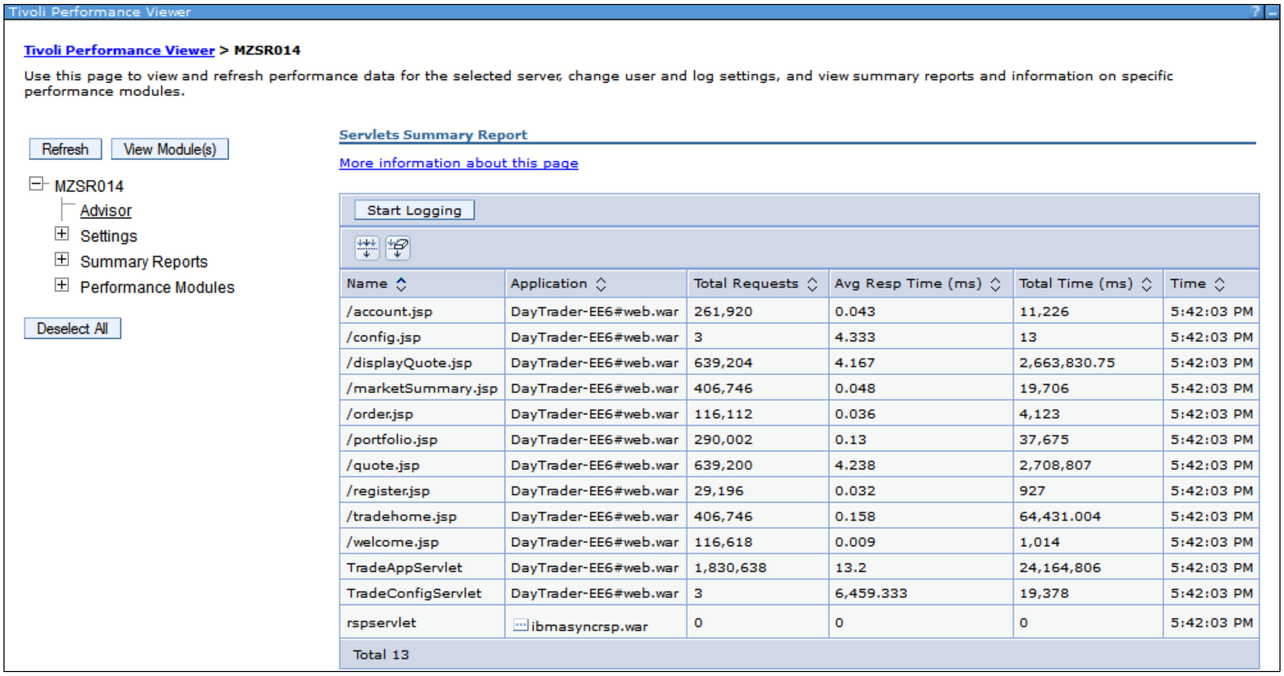


Figure 8-21 Tivoli Performance Viewer - Servlet Summary Report

In this case, the servlet summary report of our DayTrader workload is shown. On the left, you have many options to display different summary reports and look at the different performance modules that visualize the PMI data. On the right, you see the (selected) report, which is the servlet summary report in this example. It shows the name of the servlet, the application it belongs to, and the average response time.

As you are interested in performance aspects that affect database access, look at the PMI data of the connection pool that is associated with our data source (we are using a type 4 connection during this DayTrader run). To do so, expand **Performance modules** → **JDBC Connection Pools** → **JDBC Universal Driver (XA)** and select **jdbc/TradeDataSource**. The result is shown in Figure 8-22. This snapshot is from the time when the workload was increasing. The CreateCount went up quickly from 0 - 50.

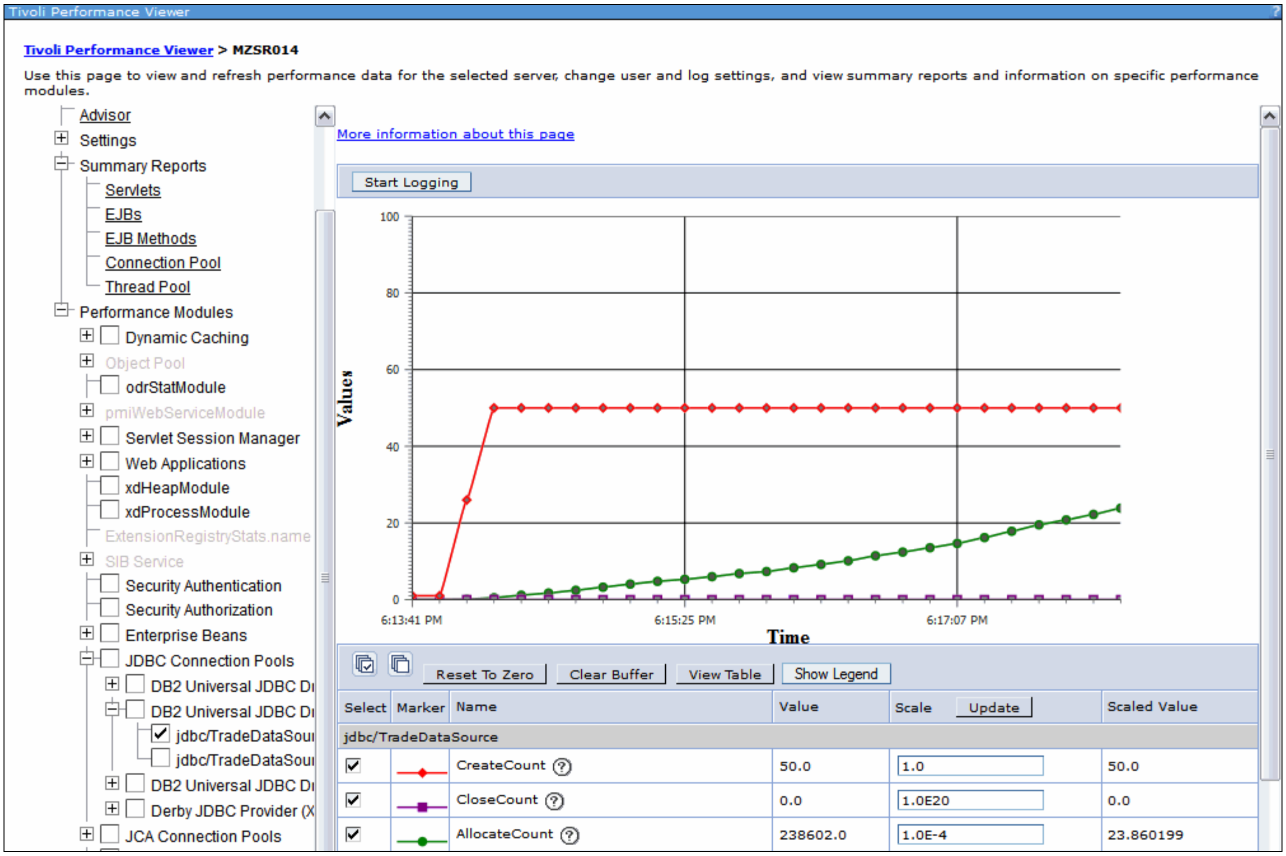


Figure 8-22 JDBC Connection Pool statistics at startup

The AllocateCount continues to go up as more transactions run. Notice that the count does not go beyond 50 connections. You can use the administration console to verify whether 50 is the maximum size of the connection by clicking **Data Sources** → **TradeDataSourceXA** → **Connection pools**, as shown in Figure 8-23.

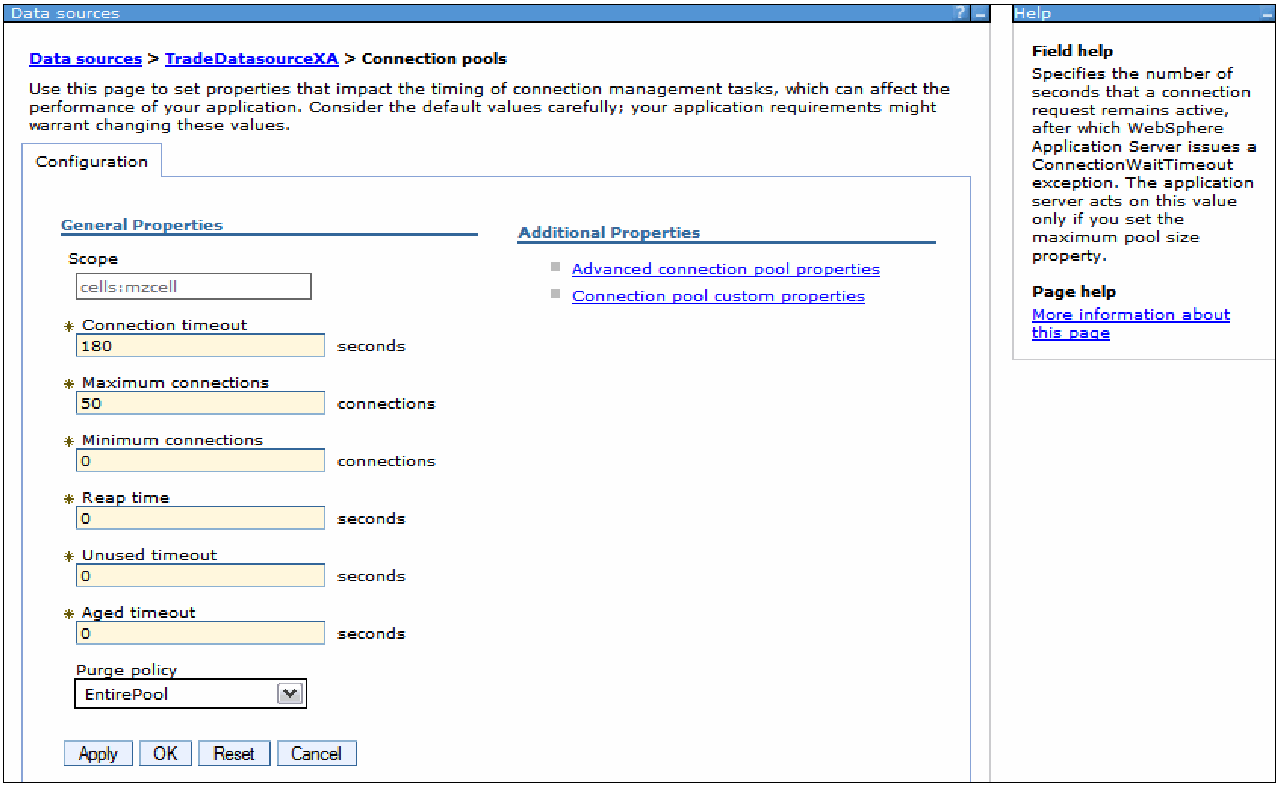


Figure 8-23 Connection pool properties

Using performance advisors

Tivoli Performance Viewer also has some performance advisor modules that are built into it. Select the **Advisor** option in the left pane to activate it. The bottom part of the advisor output window contains a number of alerts and configuration tips, as shown in Figure 8-24.

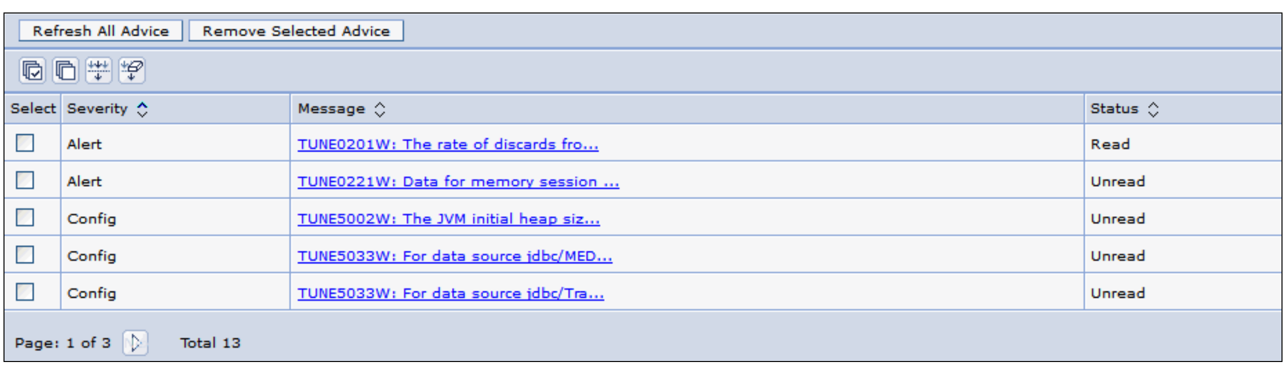


Figure 8-24 Advisor output

Zoom in on the first alert, **TUNE0201:The rate of discard fro..**’, by clicking it. The result is shown in Figure 8-25. The advisor indicates that there are many discards from the WebSphere Application Server statement cache. Creating a prepared statement object is a rather expensive operation, so discarding and creating many prepared statement objects is likely to affect performance. The discard rate is high at 1400/sec.

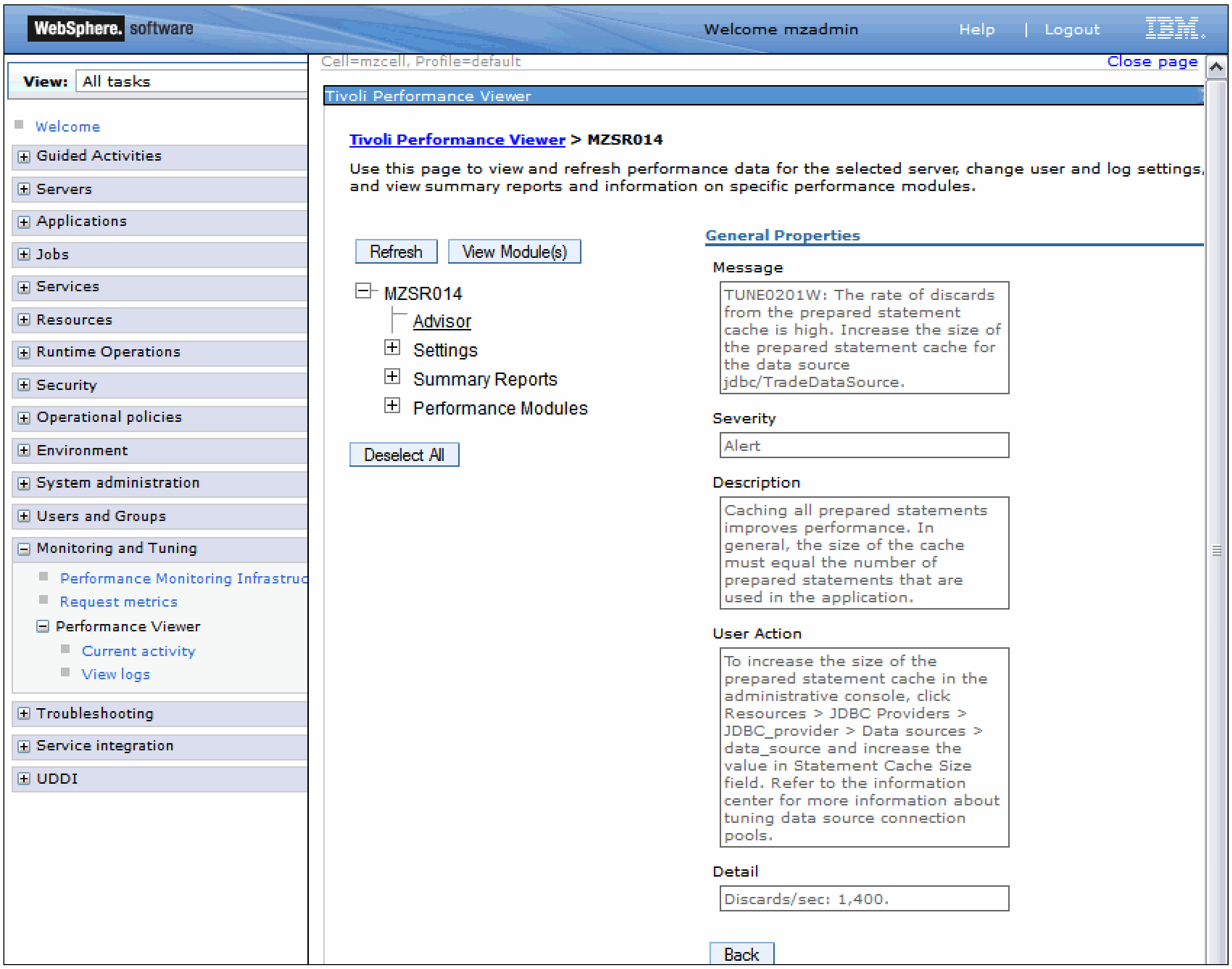


Figure 8-25 Tuning advice TUNE0201W

To verify this discard rate, go to the performance metrics of the connection pool and look for the PrepStmtCacheDiscardCount statistic, as shown in Figure 8-26 on page 393.

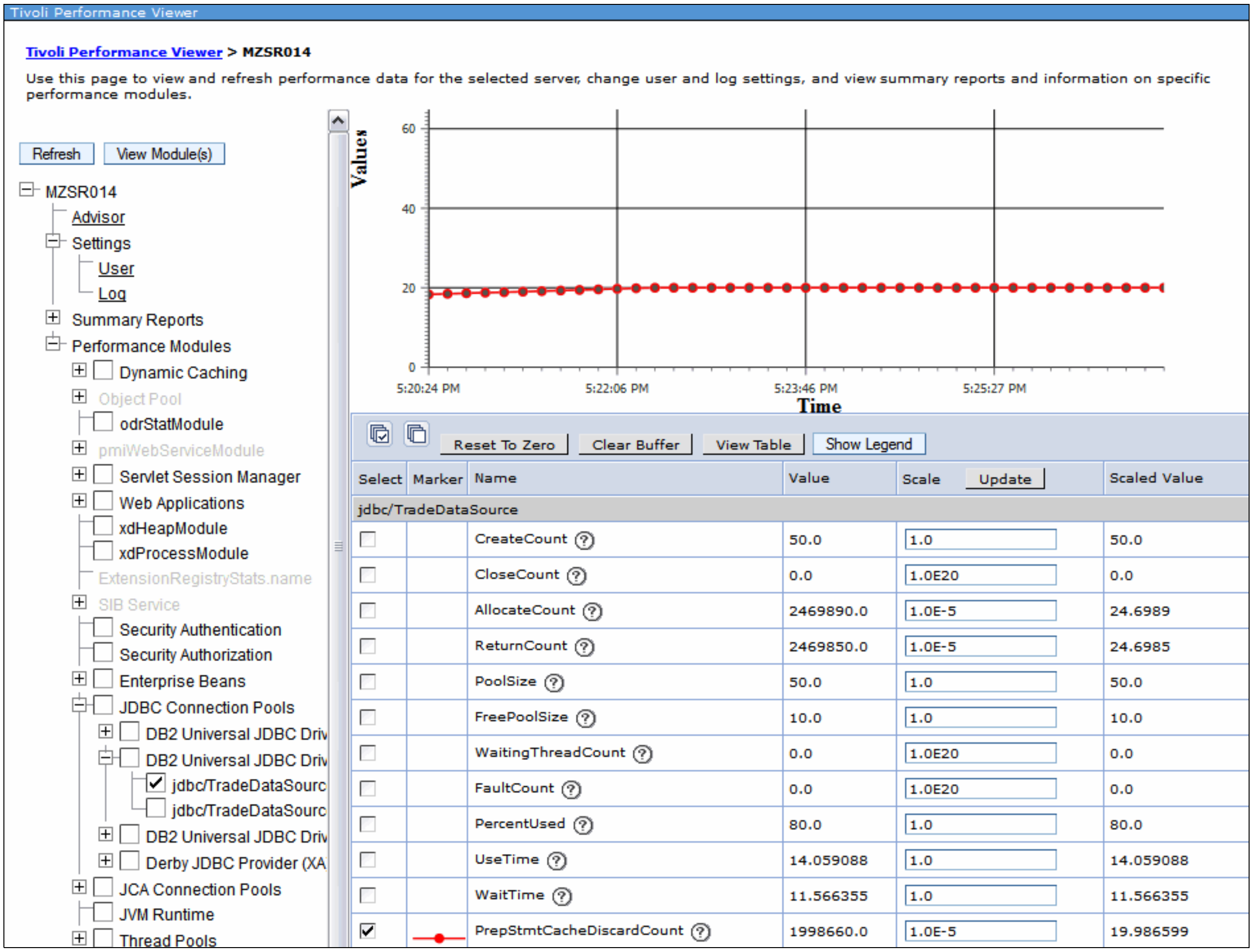


Figure 8-26 Connection pool - PrepStmtCacheDiscardCount

This statistic confirms the alert from the performance advisor. So, what is the current setting for the statement cache size? You can verify the setting by going to the administration console and clicking **Data sources** → **TradeDatasourceXA** → **WebSphere Application Server data source properties**, as shown in Figure 8-27.

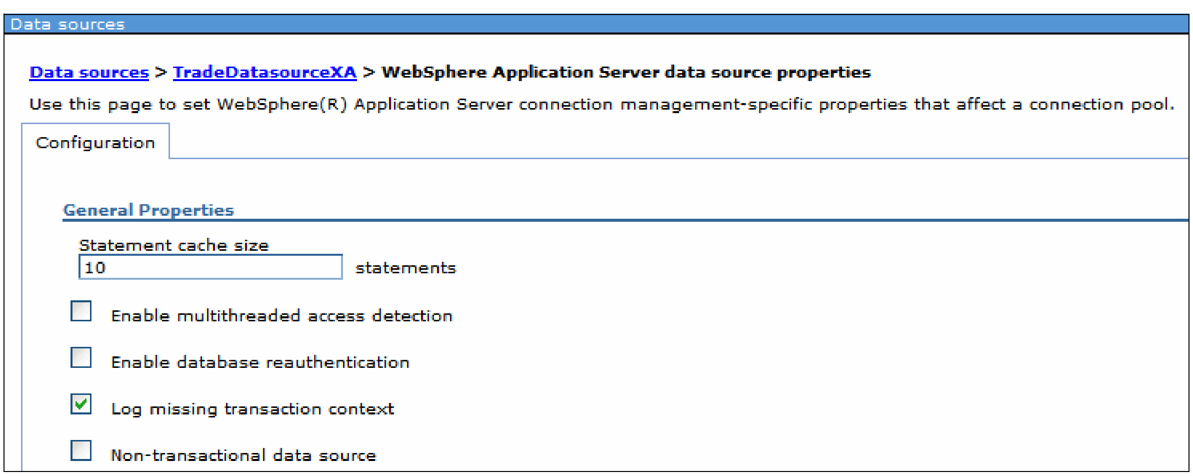


Figure 8-27 Data source statement cache size

A value of 10 is indeed low for this workload. Change it to 60. The result of the change is shown in Figure 8-28.

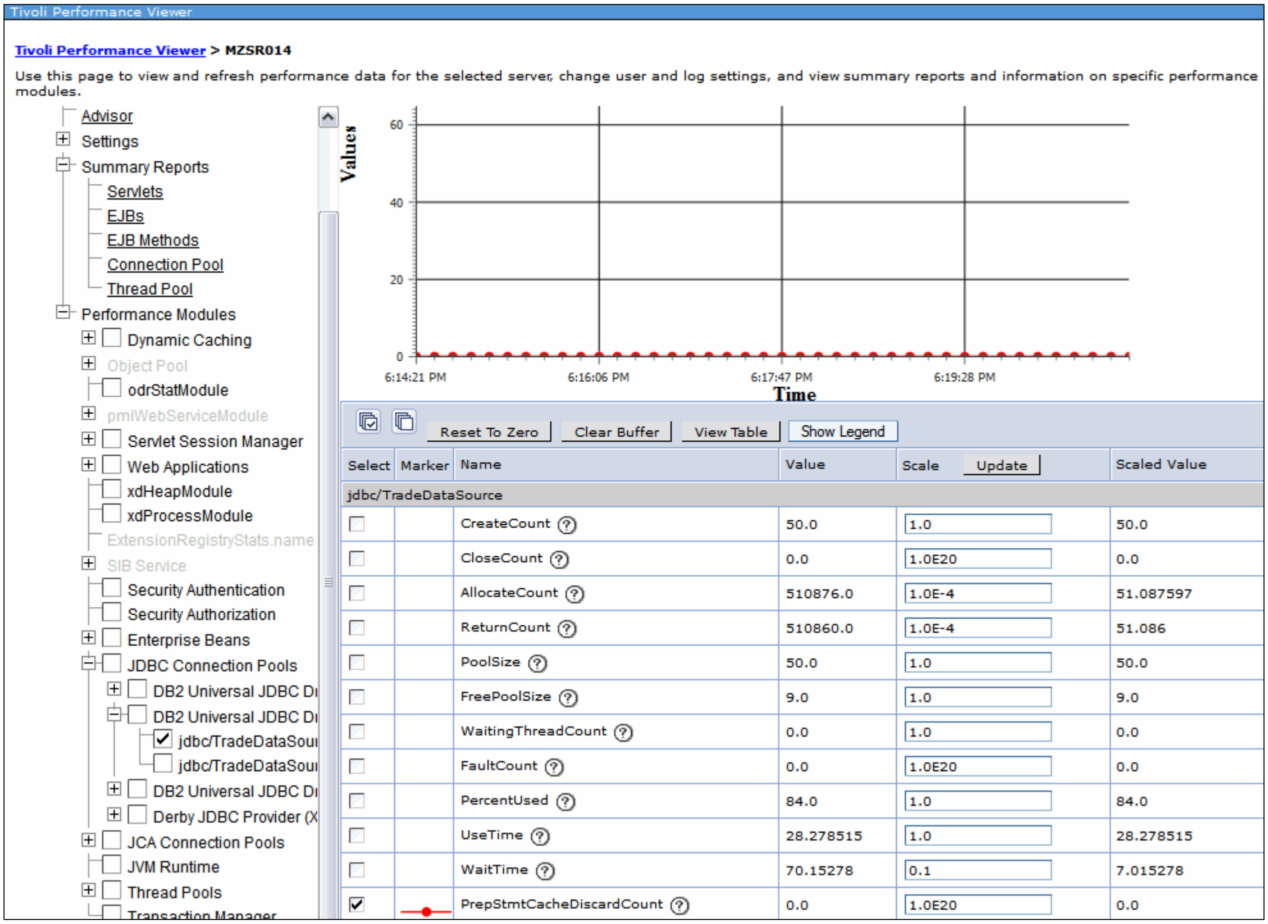


Figure 8-28   PrepStmtCacheDiscardCount after the change

As you can see, the discard count is now down to zero from 1400/sec, which is a great improvement.

This is just one example of how to use PMI and Tivoli Performance Viewer to analyze WebSphere Application Server performance data. For more information about the usage of PMI, see the WebSphere Application Server Information Center PMI topics found at:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.express.doc/ae/cprf_pmidata.html

A good article about WebSphere Application Server performance called “Case study: Tuning WebSphere Application Server V7 and V8 for performance” can be found at:

http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html#sec3c

8.4 Monitoring from the DB2 side

Even after an application moves to production, it is important to keep monitoring the application. Over time, the behavior might change, for example, because the workload increases or the data becomes disorganized. Therefore, it is important to continuously, or at least periodically, check the performance of your applications. Dealing with all DB2 performance aspects is beyond the scope of this book, but this section provides an overview of the information that is available and how to analyze DB2 performance.

This section describes the following topics:

- Which information to gather
- Analyzing DB2 statistics data
- Analyzing DB2 accounting data

8.4.1 Which information to gather

Most installations run with a set of standard DB2 traces that are always active. The information is often used for chargeback purposes, but it contains much information that can be used to check the health of the system. As a preferred practice, have the following traces permanently active on all your DB2 systems:

- DB2 Statistics trace classes
  - 1: System-wide information about the work that is performed by the DB2 system.
  - 3: Information about deadlocks, timeouts, lock escalations, and long running units of work. This trace class is valuable for identifying concurrency problems.
  - 4: DDF exception conditions.
  - 5: Data sharing statistics.

> **Tip:** **`STATIME DSNZPARM`** determines the interval at which DB2 writes out its statistics information for classes 1 and 5. The default value in Version 9 is 5 minutes and 1 minute in Version 10. Use `STATIME=1`. The cost of gathering this information is negligible and it provides valuable information for analyzing performance problems.
>
> In DB2 10, IFCIDs 0001, 0002, 0202, 0217, 0225, and 0230 are no longer controlled by **`STATIME`**. These trace records are written at fixed, one-minute intervals.

- DB2 Accounting trace classes
  - 1: Total ET and CPU time of the thread/plan and many useful counters.
  - 2: In addition to the total time, the time (ET and CPU) spent inside DB2 is collected. This trace class is more expensive (typically around 2.5% processing time for online transactions) than gathering class 1 information. It can have an impact, especially for applications that issue many DB2 requests, such as fetch intensive applications (up to 10% in heavy batch environments).

    However, this information is required to determine whether the time is spent in DB2 or elsewhere, and is valuable. So, unless you have a processor issue, activate accounting class 2 and leave it on. If that is not possible, activate it for one hour each day during a high activity period.

– 3: Activating this trace class adds an additional level of granularity as you can use it to determine when DB2 must wait for something, for example, for a lock to become available, how many times that occurs, and how long you wait for it.

  Gathering accounting class 3 information is cheap, so it is not a problem to have it on always. Only in cases where transactions experience many lock/latch contentions can tracking them can have a noticeable impact. If that is the case, you can disable accounting trace class 3.

– 7: Similar to accounting class 2, information about the ET and CPU time that is used is gathered, but at the package/DBRM level.

– 8: Similar to accounting class 3 information, but at the package/DBRM level.

– 10: This class obtains additional information at the package level about the type of SQL statements being run, locking information, and buffer pool related information.

Typically, accounting classes 1, 2, 3, 7, and 8 are turned on all the time.

The DB2 statistics and accounting information is normally written to SMF. DB2 statistics records use SMF type 100 and DB2 accounting records use SMF type 101 records. Before you send data to SMF, make sure that SMF is enabled so that it can write DB2 trace record types 100 and 101 (and 102, which is used for performance type records). For more information about this topic, see “Enabling SMF 120 data collection” on page 378.

Both traces can be started at DB2 start time through the **SMFSTAT** (statistics) and **SMFACCT** (for accounting) DSNZPARMs.

These traces must be started on each of the members of a DB2 data sharing group to be able to get the complete picture of all the work in the data sharing group.

8.4.2 Creating DB2 accounting records at a transaction boundary

To have a good understanding of the activity that is performed by WebSphere Application Server transactions against the database, it is important to create a DB2 accounting record at the transaction boundary at commit time. As WebSphere Application Server can use either RRS (type 2) or DRDA (type 4) attach, this section briefly explains how to create granular DB2 accounting data for each type.

Ensuring DB2 accounting records when using an RRS (T2)

Typically, DB2 creates an accounting record when a thread ends, or in the case of thread reuse, when a new user performs a sign-on operation. When you use WebSphere Application Server, a thread can be reused many times before it ends or a new user comes in, and a DB2 accounting record is produced. This situation makes it difficult to analyze the performance of such applications, as sometimes a thread runs 10 actual transactions before it produces a DB2 accounting record and other times only 5.

To avoid this situation, you can direct the RRS attach to write a DB2 accounting record at commit time (provided there are no open held cursors). The easiest way to achieve this task from a WebSphere Application Server application is to set the `accountingInterval` custom property on the data source to `COMMIT` by using the administration console, as shown in Figure 8-29 on page 397.

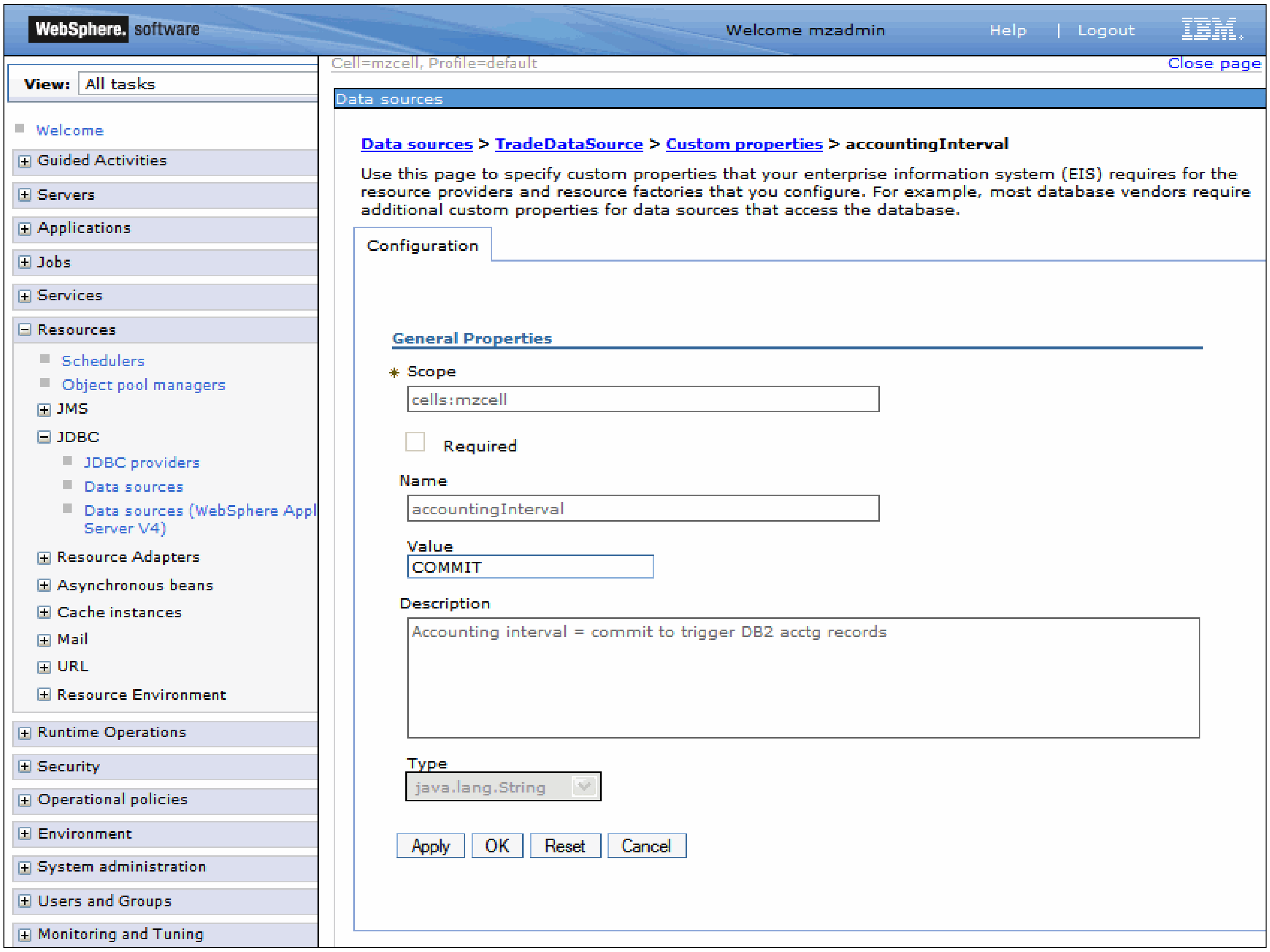


Figure 8-29 Specifying the accountingInterval customer property

If the transaction has no open `WITH HOLD` cursors, each time a commit point is reached (the application issues **`SRRCMIT`** explicitly or implicitly), DB2 cuts an accounting record. If the accounting interval is `COMMIT` and an **`SRRCMIT`** is issued while a held cursor is open, the accounting interval spans that commit and ends at the next valid accounting interval end point (such as the next **`SRRCMIT`** that is issued without open held cursors, application termination, or **`SIGNON`** with a new authorization ID).

Ensuring DB2 accounting records when using DRDA (T4)

To make sure that DB2 cuts an accounting record at the end of each transaction when using type 4 connectivity through DRDA, you do not have to specify a custom property. DB2 cuts an accounting record at commit time for DRDA work when the following conditions are true:

- `DSNZPARM CMTSTAT=INACTIVE`
- At commit time, DB2 is *not* using one or more of the following items:
  - A held cursor.
  - A held LOB locator.
  - A package that is bound with `KEEPDYNAMIC(YES)`. However, if `KEEPDYNAMIC` is the only reason to prevent the thread from being pooled, an accounting record is still created. (A typical example of this are SAP applications.)

– A declared temporary table that is active (the table was not explicitly dropped through the **DROP TABLE** statement or the **ON COMMIT DROP TABLE** clause on the **DECLARE GLOBAL TEMPORARY TABLE** statement).

If these conditions are met, a DB2 accounting record is cut and the WLM enclave is reset.

8.4.3 DB2 rollup accounting

In DB2 8, rollup accounting was introduced for RRS and DRDA workloads. The idea behind rollup accounting is to reduce the number of SMF records that are produced by high volume DB2 workloads. Instead of writing a DB2 accounting record for each transaction, you can use rollup accounting to write an accounting record every *x* transactions, where x is determined by the value of **ACCUMACC DSNZPARM**. With `ACCUMACC=10`, for example, DB2 writes accounting records after 10 transactions complete.

When you use accounting rollup, you can also specify how DB2 aggregates the accounting records by using **ACCUMUID DSNZPARM**. You can look at **ACCUMUID** as an SQL **GROUP BY** specification. There are 18 different settings that you can specify for **ACCUMUID**. For more information, see *DB2 10 for z/OS Installation and Migration Guide*, GC19-2974.

In our example, we use `ACCUMUID=2` during some of our tests. “2” indicates that the aggregation is done by “user application name” or “transaction name”; this is the value of the `clientApplicationInformation` property or **CURRENT CLIENT_APPLNAME** special register value.

So, with `ACCUMACC=10` and `ACCUMUID=2`, if you run 20 transactions named tran_1 and 10 transactions of tran_2, DB2 produces three accounting records; one for all executions of tran_2 and two for the executions of tran_1. The information in such a rollup accounting record is the sum of all the work of these 10 transactions.

The advantage of using rollup accounting is obvious. It can reduce the number of (SMF) accounting records that DB2 produces. The disadvantage of using **ACCUMACC** is that you lose transaction granularity information. With rollup accounting, you can no longer see how each individual transaction performed because all accounting data of *x* transactions is rolled into a single accounting record. This situation can make it difficult to analyze performance problems, especially when the problem occurred only briefly or when using a high **ACCUMACC** value.

Tip: DB2 10 introduced an option to compress records that are written to SMF by using **SMFCOMP=YES DSNZPARM**, which compresses the SMF trace record before it is written to the SMF data set. If you use **ACCUMACC > 1** to reduce the data volume that is produced by DB2 accounting records, you might want to consider switching to **ACCUMACC=NO** and **SMFCOMP=YES** to achieve this task, and keep the transaction level granularity that you give up by using **ACCUMACC >1**.

Another option to reduce the size of the SMF (offloaded) data sets is to use SMS compression on the dataclass (`compaction=YES`) for those data sets.

8.4.4 Analyzing DB2 statistics data

DB2 statistics data provides information about the entire subsystem (or data sharing group). DB2 accounting data provides information about individual transactions (or the number of executions of a transaction where **ACCUMAC > 0 DSNZPARM** is used).

Start by looking at the overall subsystem statistics. This information can be used to check the overall health of the DB2 system.

Note: If this DB2 system runs other work than just transactions coming from WebSphere Application Server, these other transactions are also included in the information of the DB2 statistics record.

We use IBM Tivoli OMEGAMON DB2 Performance Expert on z/OS V5.1.1 batch reporting to look at the DB2 statistics data.

Example 8-5 shows a sample SYSIN to create a DB2 statistics report.

Example 8-5 Create a DB2 statistics report

```
//SYSIN    DD  *
DB2PM
* *********************************
*       GLOBAL PARMS
* *********************************
  GLOBAL
*        Adjust for US East Coast DST
         TIMEZONE (+4)
         FROM(,21:29:00)
           TO(,21:30:01)
*        Include the entire data sharing group
         INCLUDE( GROUP(DBOZG))
***********************************
*       STATISTICS REPORTS
***********************************
 STATISTICS
            REPORT
              LAYOUT (LONG)
EXEC
```

In our example, we want to look at a one-minute interval. (As the DB2 statistics interval is one minute, we could also have used a **STATISTICTS TRACE** report instead).

Statistics highlights

Example 8-6 shows the header of the statistics report. It indicates the interval that we are looking at, the DB2 subsystem, and also provides an idea about the number of threads that were created, and the number of commits that occurred in the interval you are reporting on.

Example 8-6 Statistics report - highlights section

```
LOCATION: DBOZ                  OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1M1)                PAGE: 1-1
      GROUP: DBOZG                            STATISTICS REPORT - LONG                   REQUESTED FROM: ALL      21:29:00.00
     MEMBER: DOZ1                                                                                  TO: DATES    21:30:01.00
  SUBSYSTEM: DOZ1                                                                         INTERVAL FROM: 08/13/12 21:29:00.00
DB2 VERSION: V10                                   SCOPE: MEMBER                                   TO: 08/13/12 21:30:00.12

---- HIGHLIGHTS -------------------------------------------------------------------------------------------------------
INTERVAL START  : 08/13/12 21:29:00.00    SAMPLING START: 08/13/12 21:29:00.00    TOTAL THREADS      :    80.00
INTERVAL END    : 08/13/12 21:30:00.12    SAMPLING END  : 08/13/12 21:30:00.12    TOTAL COMMITS      : 85191.00
INTERVAL ELAPSED:          1:00.120127    OUTAGE ELAPSED:             0.000000    DATA SHARING MEMBER:      N/A
```

The number of commits includes the commits from both local attach work (such as TSO, RRS, and utilities), as well at the commits that are received for DRDA work. The number of threads is the number of create thread operations, which do not include the Database Access Threads (DBATs) associated with DRDA work.

SQL DML and dynamic statement cache sections

To get a high-level idea about the type of SQL work that occurred during the reporting interval, you can use the SQL DML section (see Example 8-7).

Example 8-7 Statistics report - SQL DML section

```
SQL DML                      QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------  --------  -------  -------  -------
SELECT                           0.00     0.00     0.00     0.00
INSERT                        3229.00    53.71    40.36     0.04
  NUMBER OF ROWS              3229.00    53.71    40.36     0.04
UPDATE                       13865.00   230.62   173.31     0.16
  NUMBER OF ROWS             14688.00   244.31   183.60     0.17
MERGE                            0.00     0.00     0.00     0.00
DELETE                         823.00    13.69    10.29     0.01
  NUMBER OF ROWS               823.00    13.69    10.29     0.01

PREPARE                        109.9K  1827.56  1373.41     1.29
DESCRIBE                     27676.00   460.34   345.95     0.32
DESCRIBE TABLE                   0.00     0.00     0.00     0.00
OPEN                         95096.00  1581.77  1188.70     1.12
CLOSE                         1610.00    26.78    20.13     0.02
FETCH                         1610.00    26.78    20.13     0.02
  NUMBER OF ROWS               110.4K  1836.04  1379.79     1.30

TOTAL DML                      253.8K  4221.25  3172.27     2.98
```

As the DayTrader application that we used during our testing is a JDBC application using dynamic SQL, it is important to verify that you have a good hit ratio in the global dynamic statement cache. You can verify that in the DYNAMIC SQL STMT section in the statistics report, as shown in Example 8-8.

Example 8-8 Statistics report - dynamic SQL statements section

```
DYNAMIC SQL STMT             QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------  --------  -------  -------  -------
PREPARE REQUESTS               109.9K  1827.56  1373.41     1.29
  FULL PREPARES                  7.00     0.12     0.09     0.00
  SHORT PREPARES               109.9K  1827.42  1373.31     1.29
GLOBAL CACHE HIT RATIO (%)      99.99      N/A      N/A      N/A
```

Almost all prepares result in a short prepare, which means that the statement was found in the global dynamic statement cache, which results in a high cache hit ratio of 99.99%.

The trade workload uses parameter markers, which increases the chance of finding a matching statement in the dynamic statement cache.

For more information about this topic, see 4.3.2, “Enabling DB2 dynamic statement cache” on page 141 and 6.2, “Dynamic SQL” on page 299.

Subsystem services and DDF and DRDA location sections

The SUBSYSTEM SERVICES section (Example 8-9), and the DRDA REMOTE LOCS section (Example 8-10) can be used to quickly determine whether the bulk of the work is using a type 2 or a type 4 connection.

Using a type 4 connection

Example 8-9 shows the SUBSYSTEM SERVICES section when using a type 2 connection.

Example 8-9 Statistics report - subsystem services section

```
SUBSYSTEM SERVICES           QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------  --------  -------  -------  -------
IDENTIFY                        20.00     0.33     0.25     0.00
CREATE THREAD                   80.00     1.33     1.00     0.00
SIGNON                           0.00     0.00     0.00     0.00
TERMINATE                      100.00     1.66     1.25     0.00
ROLLBACK                         0.00     0.00     0.00     0.00

COMMIT PHASE 1                   0.00     0.00     0.00     0.00
COMMIT PHASE 2                   0.00     0.00     0.00     0.00
READ ONLY COMMIT                 1.00     0.02     0.01     0.00
UNITS OF RECOVERY INDOUBT        0.00     0.00     0.00     0.00
UNITS OF REC.INDBT RESOLVED      0.00     0.00     0.00     0.00
SYNCHS(SINGLE PHASE COMMIT)     80.00     1.33     1.00     0.00
QUEUED AT CREATE THREAD          0.00     0.00     0.00     0.00
SUBSYSTEM ALLIED MEMORY EOT      0.00     0.00     0.00     0.00
SUBSYSTEM ALLIED MEMORY EOM      0.00     0.00     0.00     0.00
SYSTEM EVENT CHECKPOINT          0.00     0.00     0.00     0.00

HIGH WATER MARK IDBACK           9.00     0.15     0.11     0.00
HIGH WATER MARK IDFORE           2.00     0.03     0.02     0.00
HIGH WATER MARK CTHREAD          9.00     0.15     0.11     0.00
```

When you use a type 2 connection, you expect to see high numbers in the various commit counters in the SUBSYSTEM SERVICES section, and when you use type 4, the commits show up in the DRDA REMOTE LOCS section of the DB2 statistics report. It is clear from the reports that this workload was using a type 4 connection, as almost all commit requests are in the (SINGLE PHASE) COMMITS bucket in the DRDA REMOTE LOCS section.

When the number of active (allied) threads exceeds **CTHREAD DSNZPARM** value, new create thread requests are queued. When this situation occurs, the QUEUED AT CREATE THREAD counter is incremented. Typically, you want to see a zero value in this field. However, it is possible that you hit **CTHREAD** when there is a significant spike in the workload, or when things slow down for some reason. In those cases, it is better to queue the threads, or even deny them, than to let them start processing. Using a large **CTHREAD** value allows much work to start, but when the system is flooded, adding more work makes things worse. Therefore, queuing work at create thread time, or even outside DB2 (in the application server), is better than leaving the gates wide open (using a high **CTHREAD** value) and adding more work to a system that is already under stress.

Example 8-10 Statistics report - DRDA remote locations section

```
DRDA REMOTE LOCS                 SENT  RECEIVED
---------------------------  --------  --------
TRANSACTIONS                      N/A       N/A
CONVERSATIONS                    0.00  12447.00
```

```
CONVERSATIONS QUEUED              0.00
CONVERSATIONS DEALLOCATED         0.00

SQL STATEMENTS                    0.00    412.6K
(SINGLE PHASE) COMMITS            0.00  85110.00
(SINGLE PHASE) ROLLBACKS          0.00      0.00
ROWS                            110.4K      0.00
MESSAGES                        814.8K    814.8K
BYTES                           105.3M    135.9M
BLOCKS                          188.6K      0.00
MESSAGES IN BUFFER                 N/A
CONT->LIM.BLOCK FETCH SWTCH        N/A
STATEMENTS BOUND AT SERVER         N/A

PREPARE REQUEST                    N/A       N/A
LAST AGENT REQUEST                 N/A       N/A
TWO PHASE COMMIT REQUEST           N/A       N/A
TWO PHASE BACKOUT REQUEST          N/A       N/A
FORGET RESPONSES                   N/A       N/A
COMMIT RESPONSES                   N/A       N/A
BACKOUT RESPONSES                  N/A       N/A
THREAD INDOUBT-REM.L.COORD.       0.00
COMMITS DONE-REM.LOC.COORD.        N/A
BACKOUTS DONE-REM.L.COORD.         N/A
```

Because this is a workload that is using a type 4 connection, the work enters DB2 through the DDF address space, in which case it is interesting to have a look at the GLOBAL DDF ACTIVITY section as well, as shown in Example 8-11 on page 403.

Well-behaved transactions run a number of SQL statements and issue a commit. Then, the DBAT (that represents the thread in DB2) and the connection that is tied to the transport in the Data Server driver are disconnected from each other. The connection goes inactive (waiting for the next request from the application server to arrive) and the DBAT is put into a pool (so it can be reused by other connections that must run SQL statements. These types of connections (that can go inactive at commit) are also called *type 2 inactive connections*, and the DBATs are called *pooled DBATs*.

The **CMTSTAT** subsystem parameter controls whether threads are made active or inactive after they successfully commit or roll back and hold no cursors. A thread can become inactive only if it holds no cursors, has no temporary tables that are defined, and runs no statements from the dynamic statement cache.

Note: Type 2 (inactive) connections have nothing to do with the type of Java driver that is used by the application. On the contrary, DB2 type 2 (inactive) connections are always associated with work entering DB2 through DRDA and always use a Java type 4 driver.

For more information about setting active/inactive connections, see *DB2 10 for z/OS Installation and Migration Guide*, GC19-2974.

When a connection wants to process an SQL request and enters the DB2 server, the request is put on a queue to allow a DBAT to be selected from the pool, or created, to process the request.

The ACC QU INACT CONNS (TYPE 2) counter indicates how many of these inactive connections were put on this queue during the interval that you are looking at. It is a good indicator of the amount of DRDA work that goes through the system.

Typically, a connection is only on that queue for a short time. Since Version 10, DB2 provides information about the MIN/MAX and AVG QUEUE TIME in case you suspect that there is a problem with connections not being able to obtain a DBAT quickly.

If the maximum number of DBATs is reached (**MAXDBAT DSNZPARM**), new requests are queued and the DBAT/CONN QUEUED-MAX ACTIVE counter is incremented. Similar to the QUEUED AT CREATE THREAD counter, you want to have a zero value in this field under normal conditions. But as indicated above, it is often better to queue requests outside DB2 (and have a non-zero value in this field) than to let all the work into DB2 and get stuck during DB2 processing.

Well-behaved transactions commit regularly and allow the connection to become inactive and the DBAT to be pooled so other transactions (connections) can reuse a pooled DBAT. The number of times a pooled DBAT is reused can be found in the DISCON (POOL) DBATS REUSED counter.

There are conditions that do not let a connection go inactive. To optimize resource usage, you want to make sure that these conditions do not apply to your applications. For more information, see Chapter 10, "Managing DB2 threads", in *DB2 10 for z/OS Managing Performance*, SC19-2978.

Example 8-11 show that 110.0K requests came in during this interval, and DB2 was able to reuse a DBAT 110.0K times, which is optimal reuse.

Example 8-11 Statistics report - global DDF activity section

```
GLOBAL DDF ACTIVITY           QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------   --------  -------  -------  -------
DBAT/CONN QUEUED-MAX ACTIVE       0.00     0.00     0.00      N/A
CONN REJECTED-MAX CONNECTED       0.00     0.00     0.00      N/A
CONN CLOSED - MAX QUEUED          0.00     0.00     0.00      N/A

COLD START CONNECTIONS            0.00     0.00     0.00     0.00
WARM START CONNECTIONS            0.00     0.00     0.00     0.00
RESYNCHRONIZATION ATTEMPTED       0.00     0.00     0.00     0.00
RESYNCHRONIZATION SUCCEEDED       0.00     0.00     0.00     0.00

CUR TYPE 1 INACTIVE DBATS         0.00      N/A      N/A      N/A
HWM TYPE 1 INACTIVE DBATS         2.00      N/A      N/A      N/A
TYPE 1 CONNECTIONS TERMINAT       0.00     0.00      N/A      N/A

CUR INACTIVE CONNS (TYPE 2)       4.00      N/A      N/A      N/A
HWM INACTIVE CONNS (TYPE 2)      20.00      N/A      N/A      N/A
ACC QU INACT CONNS (TYPE 2)     110.0K  1829.89      N/A      N/A
CUR QU INACT CONNS (TYPE 2)       2.00      N/A      N/A      N/A
  MIN QUEUE TIME              0.000008      N/A      N/A      N/A
  MAX QUEUE TIME              0.116363      N/A      N/A      N/A
  AVG QUEUE TIME              0.000078      N/A      N/A      N/A
HWM QU INACT CONNS (TYPE 2)      15.00      N/A      N/A      N/A

CUR ACTIVE AND DISCON DBATS      20.00      N/A      N/A      N/A
HWM ACTIVE AND DISCON DBATS      29.00      N/A      N/A      N/A
HWM TOTL REMOTE CONNECTIONS      22.00      N/A      N/A      N/A
```

```
CUR DISCON DBATS NOT IN USE     14.00      N/A      N/A      N/A
HWM DISCON DBATS NOT IN USE     29.00      N/A      N/A      N/A
DBATS CREATED                    3.00      N/A      N/A      N/A
DISCON (POOL) DBATS REUSED     110.0K      N/A      N/A      N/A

CUR ACTIVE DBATS-BND DEALLC      0.00      N/A      N/A      N/A
HWM ACTIVE DBATS-BND DEALLC      0.00      N/A      N/A      N/A
```

Using a type 2 connection

Example 8-12 shows the subsystem services section when you use a type 2 connection.

This information is from a different test run and the time interval is much larger in this case, so you cannot compare this data with the data from the type 4 run. In addition, because a type 2 connection uses a local RRS attach, all work is directed to a single DB2 member, which is the same member that runs the WebSphere Application Server. If there is a need to spread the work across multiple members, which is typically the case in a data sharing environment, you can run a WebSphere Application Server on the other LPAR and use the HTTP server to "spray" the work between the two application servers. For this example, the number of members or the way the workload is distributed is not important.

Example 8-12 Statistics report - subsystem services - T2

```
SUBSYSTEM SERVICES           QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------  --------  -------  -------  -------
IDENTIFY                        58.00     0.11     1.02     0.00
CREATE THREAD                   57.00     0.11     1.00     0.00
SIGNON                          57.00     0.11     1.00     0.00
TERMINATE                        1.00     0.00     0.02     0.00
ROLLBACK                         0.00     0.00     0.00     0.00

COMMIT PHASE 1                   0.00     0.00     0.00     0.00
COMMIT PHASE 2                 147.4K   272.92  2585.53     0.11
READ ONLY COMMIT              1142.3K  2115.34    20.0K     0.89
UNITS OF RECOVERY INDOUBT        0.00     0.00     0.00     0.00
UNITS OF REC.INDBT RESOLVED      0.00     0.00     0.00     0.00
SYNCHS(SINGLE PHASE COMMIT)      0.00     0.00     0.00     0.00
QUEUED AT CREATE THREAD          0.00     0.00     0.00     0.00
SUBSYSTEM ALLIED MEMORY EOT      0.00     0.00     0.00     0.00
SUBSYSTEM ALLIED MEMORY EOM      0.00     0.00     0.00     0.00
SYSTEM EVENT CHECKPOINT          5.00     0.01     0.09     0.00

HIGH WATER MARK IDBACK          59.00     0.11     1.04     0.00
HIGH WATER MARK IDFORE           2.00     0.00     0.04     0.00
HIGH WATER MARK CTHREAD         59.00     0.11     1.04     0.00
```

Note the high number of COMMIT PHASE 2 and READ ONLY COMMIT requests. Transactions show only COMMIT PHASE 2 because DB2 is the only subsystem that is involved in the processing and no global transaction is defined.

When you use a type 2 connection, applications coming from WebSphere Application Server come into DB2 through RRS, and this type of RRS connections counts towards the **IDBACK DSNZPARM** (which limits the number of background connections). If the high water mark (HWM) gets close to the **IDBACK**[2] **DSNZPARM** value, you might have to increase the **DSNZPARM** value.

For information about **IDBACK** and other DSNZPARMs, see 4.3.1, “DB2 connectivity installation parameters” on page 138.

Example 8-13 shows the GLOBAL DDF ACTIVITY section from the type 2 run. There is almost no DDF activity (ACC QU INACT CONNS (TYPE 2) is low), which is expected if the entire workload is using a Java type 2 connection (through RRS).

Example 8-13 Statistics report - global DDF activity section - T2

```
GLOBAL DDF ACTIVITY          QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------  --------  -------  -------  -------
DBAT/CONN QUEUED-MAX ACTIVE      0.00     0.00     0.00      N/A
CONN REJECTED-MAX CONNECTED      0.00     0.00     0.00      N/A
CONN CLOSED - MAX QUEUED         0.00     0.00     0.00      N/A

COLD START CONNECTIONS           0.00     0.00     0.00     0.00
WARM START CONNECTIONS           0.00     0.00     0.00     0.00
RESYNCHRONIZATION ATTEMPTED      0.00     0.00     0.00     0.00
RESYNCHRONIZATION SUCCEEDED      0.00     0.00     0.00     0.00

CUR TYPE 1 INACTIVE DBATS        0.00      N/A      N/A      N/A
HWM TYPE 1 INACTIVE DBATS        2.00      N/A      N/A      N/A
TYPE 1 CONNECTIONS TERMINAT      0.00     0.00      N/A      N/A

CUR INACTIVE CONNS (TYPE 2)      0.00      N/A      N/A      N/A
HWM INACTIVE CONNS (TYPE 2)     50.00      N/A      N/A      N/A
ACC QU INACT CONNS (TYPE 2)      5.00     0.01      N/A      N/A
CUR QU INACT CONNS (TYPE 2)      0.00      N/A      N/A      N/A
  MIN QUEUE TIME             0.000002      N/A      N/A      N/A
  MAX QUEUE TIME             0.000010      N/A      N/A      N/A
  AVG QUEUE TIME             0.000005      N/A      N/A      N/A
HWM QU INACT CONNS (TYPE 2)     37.00      N/A      N/A      N/A

CUR ACTIVE AND DISCON DBATS      0.22      N/A      N/A      N/A
HWM ACTIVE AND DISCON DBATS     88.00      N/A      N/A      N/A
HWM TOTL REMOTE CONNECTIONS     78.00      N/A      N/A      N/A

CUR DISCON DBATS NOT IN USE      0.22      N/A      N/A      N/A
HWM DISCON DBATS NOT IN USE     88.00      N/A      N/A      N/A
DBATS CREATED                    1.00      N/A      N/A      N/A
DISCON (POOL) DBATS REUSED       5.00      N/A      N/A      N/A

CUR ACTIVE DBATS-BND DEALLC      0.00      N/A      N/A      N/A
HWM ACTIVE DBATS-BND DEALLC      0.00      N/A      N/A      N/A
```

[2] The **IDBACK** subsystem parameter determines the maximum number of concurrent connections that can be identified to DB2 from batch.

Locking and data sharing locking sections

WebSphere Application Server applications and Java applications in general have issues with locking. This locking occurs mostly because these applications are not always designed with a high degree of concurrency in mind, and also because the DB2 for z/OS locking techniques differ in some areas from other database management systems, which can sometimes result in a different locking behavior when deploying the application on a System z platform compared to other platforms and or DBMSs.

As with all information in the DB2 statistics report, the locking sections contain information about the locking activity for the entire subsystem. In most cases, it is more interesting to look at the locking information for individual applications, so when you look at the DB2 statistics information, you typically want to make sure that only the overall locking behavior and activity are fine. Example 8-14 shows the locking and data sharing locking section from one of the runs we ran during the project that produced this book.

To perform this type of high-level check, look at the following counters:

- Suspensions: When an application requests a lock and that lock cannot be granted because an incompatible lock exists for the resource, the lock request is suspended. You want to compare the number of suspensions against the number of lock requests. If a significant percentage is suspended, applications do not scale, and the locking behavior must be investigated in more detail.
- Timeouts: When a lock request is suspended for longer that the timeout value that is specified in **IRLMRWT DSNZPARM**, it is timed out and a resource unavailable error is returned to the application. Timeouts are disruptive for applications, so they should be investigated in more detail. When statistics trace class 3 is active, DB2 writes an IFCID 196 trace record when a timeout occurs. This trace record includes the type of lock that is being requested, the resource that is being locked, and the holder of the lock. A zero (or low) value for the number of timeouts is appropriate.
- Deadlocks: Locks can be a requested in a way the tran A is waiting for a lock that is held by tran B, and tran B is waiting for a lock that is held by tran A. This is called a *deadlock* situation. IRLM is able to detect these deadlocks and stop one of the trans (known as the *victim*) and let the other one continue. When statistics trace class 3 is active, DB2 writes an IFCID 172 trace record when a deadlock occurs. This trace record includes the types of locks that are being requested, the resources that are involved, and the holders and waiters of the lock. Deadlocks can also be disruptive for the system throughput, so it is important to understand and minimize them. A zero (or low) value is recommended.
- Lock escalation: When a transaction acquires more locks on a table space (part) than the amount specified on the LOCKMAX for the table space (part), the individual row or page locks are replaced by a gross lock on the table space (part). Depending on whether the application is reading or updating, the lock is escalated to S or X. As for deadlocks and timeouts, statistics trace class 3 creates an IFCID 337 trace record each time lock escalation occurs. The information in this record includes the object that is undergoing lock escalation at that time.

For more information about locking, see *DB2 9 for z/OS: Resource Serialization and Concurrency Control*, SG24-4725.

Example 8-14 Statistics report - locking and data sharing locking sections

```
LOCKING ACTIVITY            QUANTITY  /SECOND  /THREAD  /COMMIT
--------------------------  --------  -------  -------  -------
SUSPENSIONS (ALL)            4043.00    67.25    50.54     0.05
SUSPENSIONS (LOCK ONLY)       530.00     8.82     6.63     0.01
SUSPENSIONS (IRLM LATCH)     3448.00    57.35    43.10     0.04
SUSPENSIONS (OTHER)            65.00     1.08     0.81     0.00
```

```
TIMEOUTS                         0.00     0.00     0.00     0.00
DEADLOCKS                        0.00     0.00     0.00     0.00

LOCK REQUESTS                  590.5K  9822.30  7381.47     6.93
UNLOCK REQUESTS                132.7K  2207.78  1659.15     1.56
QUERY REQUESTS                  22.00     0.37     0.27     0.00
CHANGE REQUESTS              27952.00   464.94   349.40     0.33
OTHER REQUESTS                   0.00     0.00     0.00     0.00

LOCK ESCALATION (SHARED)         0.00     0.00     0.00     0.00
LOCK ESCALATION (EXCLUSIVE)      0.00     0.00     0.00     0.00

DRAIN REQUESTS                   0.00     0.00     0.00     0.00
DRAIN REQUESTS FAILED            0.00     0.00     0.00     0.00
CLAIM REQUESTS                 305.3K  5077.72  3815.91     3.58
CLAIM REQUESTS FAILED            0.00     0.00     0.00     0.00

DATA SHARING LOCKING         QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------  --------  -------  -------  -------
GLOBAL CONTENTION RATE (%)       1.21
FALSE CONTENTION RATE (%)        0.68
P/L-LOCKS XES RATE (%)          41.45

LOCK REQUESTS (P-LOCKS)      31956.00   531.54   399.45     0.38
UNLOCK REQUESTS (P-LOCKS)    31100.00   517.30   388.75     0.37
CHANGE REQUESTS (P-LOCKS)        3.00     0.05     0.04     0.00

SYNCH.XES - LOCK REQUESTS      258.0K  4291.54  3225.10     3.03
SYNCH.XES - CHANGE REQUESTS  19856.00   330.27   248.20     0.23
SYNCH.XES - UNLOCK REQUESTS    273.6K  4551.25  3420.27     3.21
BACKGROUND.XES -CHILD LOCKS      1.00     0.02     0.01     0.00
ASYNCH.XES -CONVERTED LOCKS  19311.00   321.21   241.39     0.23

SUSPENDS - IRLM GLOBAL CONT   3107.00    51.68    38.84     0.04
SUSPENDS - XES GLOBAL CONT.      0.00     0.00     0.00     0.00
SUSPENDS - FALSE CONT. MBR    3912.00    65.07    48.90     0.05
SUSPENDS - FALSE CONT. LPAR       N/A      N/A      N/A      N/A
REJECTED - XES                   5.00     0.08     0.06     0.00
INCOMPATIBLE RETAINED LOCK       0.00     0.00     0.00     0.00

NOTIFY MESSAGES SENT            43.00     0.72     0.54     0.00
NOTIFY MESSAGES RECEIVED        80.00     1.33     1.00     0.00
P-LOCK/NOTIFY EXITS ENGINES    500.00      N/A      N/A      N/A
P-LCK/NFY EX.ENGINE UNAVAIL      0.00     0.00     0.00     0.00

PSET/PART P-LCK NEGOTIATION      0.00     0.00     0.00     0.00
PAGE P-LOCK NEGOTIATION       2041.00    33.95    25.51     0.02
OTHER P-LOCK NEGOTIATION         1.00     0.02     0.01     0.00
P-LOCK CHANGE DURING NEG.     2042.00    33.97    25.52     0.02
```

The data sharing locking section is also important:

- IRLM global lock suspension: This is similar to local lock suspension, but the lock holder is on another member of the data sharing group. As global IRLM suspensions are more expensive than local lock suspensions, it is important to keep the IRLM global suspension to a minimum.
- XES global contention: XES (the z/OS lock manager) does not use as many lock states as IRLM. Therefore, it is possible that XES thinks there is lock contention on a resource, but when IRLM check its lock information for this resource, there is no lock contention. This is called *XES contention*. With the introduction of locking protocol 2 in DB2 8, XES contention is low. Therefore, if you see a significant value, it warrants further investigation.
- False contention: This is when two different resources hash to the same entry in the hash table in the lock structure. In this case, there is no contention because after a closer look, it turns out that different resources are locking. This typically happens when the lock structure is not large enough, increasing the likelihood that different resources hash to the same entry.

IBM OMEGAMON XE for DB2 PE on z/OS calculates the *GLOBAL CONTENTION RATE* for you. Try to keep it below 3 - 5%. It also calculates the *FALSE CONTENTION RATE*. False contention should be less than 1 - 3% of the total number of IRLM requests sent to XES.

Buffer pool section

Another set of important statistics to see whether the system is performing well can be found in the buffer pool (BP) section of the DB2 statistics report. DB2 provides this type of information for each of the buffer pools that are used by the different applications in the system. It is a common practice to use different buffer pools for different purposes. Typically, different pools are used for different types of objects, table spaces in one (or more) pools, indexes in others, or by the type of access against the object, that is, random versus sequential.

Another way to assign buffer pools is to dedicate certain pools to specific applications. Typically, the buffer pool assignments are a mixture of all the above. IBM OMEGAMON XE for DB2 PE on z/OS reports on each of the DB2 buffer pools that were used in a specific DB2 statistics interval, but it also rolls up all the BP activity into a single TOTAL section, as shown in Example 8-15. If you want to get an idea about the overall BP activity, this is the place to start your analysis. (The information for the individual BPs is identical to the information in the TOTAL section).

Example 8-15 Statistics report - buffer pool section

```
TOTAL  READ OPERATIONS        QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------  --------  -------  -------  -------
BPOOL HIT RATIO (%)             99.94
BPOOL HIT RATIO (%) SEQU          N/C
BPOOL HIT RATIO (%) RANDOM      99.96
GETPAGE REQUEST                519.0K  8632.22  6487.13     6.09
  COND. REQUEST FAILED           0.00     0.00     0.00     0.00
GETPAGE REQUEST-SEQUENTIAL       0.00     0.00     0.00     0.00
  COND. REQ-SEQU FAILED          0.00     0.00     0.00     0.00
GETPAGE REQUEST-RANDOM         519.0K  8632.22  6487.13     6.09
  COND. REQ-RANDOM FAILED        0.00     0.00     0.00     0.00

SYNCHRONOUS READS              210.00     3.49     2.63     0.00
SYNCHRON. READS-SEQUENTIAL       0.00     0.00     0.00     0.00
SYNCHRON. READS-RANDOM         210.00     3.49     2.63     0.00
GETPAGE PER SYN.READ-RANDOM   2471.29
```

```
SEQUENTIAL PREFETCH REQUEST      0.00      0.00      0.00      0.00
SEQUENTIAL PREFETCH READS        0.00      0.00      0.00      0.00
PAGES READ VIA SEQ.PREFETCH      0.00      0.00      0.00      0.00
S.PRF.PAGES READ/S.PRF.READ       N/C
LIST PREFETCH REQUESTS           5.00      0.08      0.06      0.00
LIST PREFETCH READS              0.00      0.00      0.00      0.00
PAGES READ VIA LIST PREFTCH      0.00      0.00      0.00      0.00
L.PRF.PAGES READ/L.PRF.READ       N/C
DYNAMIC PREFETCH REQUESTED    1166.00     19.39     14.57      0.01
DYNAMIC PREFETCH READS          23.00      0.38      0.29      0.00
PAGES READ VIA DYN.PREFETCH     91.00      1.51      1.14      0.00
D.PRF.PAGES READ/D.PRF.READ      3.96
PREF.DISABLED-NO BUFFER          0.00      0.00      0.00      0.00
PREF.DISABLED-NO READ ENG        0.00      0.00      0.00      0.00
PAGE-INS REQUIRED FOR READ     224.00      3.73      2.80      0.00

TOTAL  SORT/MERGE            QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------  --------  -------  -------  -------
MAX WORKFILES CONCURR. USED      0.00       N/A       N/A       N/A
MERGE PASSES REQUESTED           0.00      0.00      0.00      0.00
MERGE PASS DEGRADED-LOW BUF      0.00      0.00      0.00      0.00
WORKFILE REQ.REJCTD-LOW BUF      0.00      0.00      0.00      0.00
WORKFILE REQ-ALL MERGE PASS      0.00      0.00      0.00      0.00
WORKFILE NOT CREATED-NO BUF      0.00      0.00      0.00      0.00
WORKFILE PRF NOT SCHEDULED       0.00      0.00      0.00      0.00

TOTAL  WRITE OPERATIONS      QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------  --------  -------  -------  -------
BUFFER UPDATES               46107.00    766.91    576.34      0.54
PAGES WRITTEN                    0.00      0.00      0.00      0.00
BUFF.UPDATES/PAGES WRITTEN        N/C

SYNCHRONOUS WRITES               0.00      0.00      0.00      0.00
ASYNCHRONOUS WRITES              0.00      0.00      0.00      0.00
PAGES WRITTEN PER WRITE I/O       N/C
PAGES WRTN FOR CASTOUT I/O     554.00      9.21      6.92      0.01
NUMBER OF CASTOUT I/O           13.00      0.22      0.16      0.00

HORIZ.DEF.WRITE THRESHOLD        0.00      0.00      0.00      0.00
VERTI.DEF.WRITE THRESHOLD        0.00      0.00      0.00      0.00
DM THRESHOLD                     0.00      0.00      0.00      0.00
WRITE ENGINE NOT AVAILABLE        N/A       N/A       N/A       N/A
PAGE-INS REQUIRED FOR WRITE      0.00      0.00      0.00      0.00
```

The BP information is made up of three sections: read activity, sort activity, merge activity, and write activity. In the read activity, you want to check the following items:

- GETPAGE REQUEST: This is the number of times an SQL statement had to request a page from the DB2 buffer manager component. When an SQL statement must retrieve a row, that row lives on a page, and that page lives in the DB2 buffer pool (or on disk, or in the group buffer pool (GBP) in a data sharing system). So to obtain the row, there is request to the DB2 buffer manager component to get the page (getpage). The amount of getpage activity is a good measure for the amount of work that DB2 is performing in a certain interval.

- SYNCHRONOUS READS: When the DB2 buffer manager finds that the page is not in the buffer pool (or GBP), it must read the page from disk. When DB2 reads a single page from disk, and the application waits for this page to be brought into the BP; this is a *synchronous read* operation.

  Even though disk I/O has improved dramatically over the last couple of years, it is still orders of magnitude slower than retrieving a page that is in the BP. Therefore, reducing the number of SYNC I/Os has a positive effect on the transaction response time. So, it is important to verify the amount of I/O activity in the system. (You also have BP information at the application (accounting) level to show the BP activity of individual users, plans, and applications.)

- Prefetch reads: Besides synchronous I/Os, DB2 also has a number of I/O mechanisms where it anticipates I/Os and tries to bring in pages from disk before the application requests the page. These are called *asynchronous I/Os*, as the application does not wait for the I/O, and the I/Os are performed in the background by a DB2 system task.

  There are three types of asynchronous I/Os:

  – SEQUENTIAL PREFETCH: The DB2 optimizer expects that data is processed sequentially, for example, when you must scan the entire table to retrieve the result set, and DB2 reads a number of pages ahead of time (typically 32, but it varies depending on the buffer pool size). The usage of sequential prefetch was reduced since DB2 9 in favor of dynamic prefetch.

  – LIST PREFETCH: This type of asynchronous I/O is used when the optimizer decides to use an access path that retrieves a number of record IDs (RIDs) (a page number and an ID map entry on that page) and used the page numbers to retrieve a number of pages in a single I/O operation. All access paths using list prefetch use this type of I/O when retrieving pages from disk.

  – DYNAMIC PREFETCH: This type of prefetch is activated dynamically when DB2 detects that data is being accessed in a sequential manner, and to anticipate subsequent getpages, it prefetches a number of pages. So, even when the optimizer does not decide to use a table space scan, when the application at run time appears to be going through the data in a sequential manner, DB2 triggers dynamic prefetch, bringing in pages that are likely to be needed.

  For each type of prefetch, there are three types of counters:

  – ... REQUESTS: This number represents the number of times this type of prefetch was requested.

  – ... READS; This number represents the number of times that this type of prefetch triggered a prefetch engine to read in pages from disk. When all pages are in the BP, there is no need to perform an I/O operation. In this case, the counter for the number of requests is incremented, but the counter for the number of reads is not.

  – PAGES READ VIA...: This is the number of pages that were brought in by prefetch operations. IBM OMEGAMON XE for DB2 PE conveniently calculates the average number of pages that were read from disk by each prefetch operation in the ... PAGES READ/ ...READ field. In Example 8-15 on page 408, D.PRF.PAGES READ/D.PRF.READ is 3.96. Considering that the prefetch quantity is 32 in our example, fewer than four pages had to be read from disk; the rest were already in the BP.

If you want to look at a single number for the BP performance, people often use the *BP hit ratio*. BP hit ratio gives you some idea about whether pages being requested by the applications are in the BP. In our example, the hit ratio is 99.94%, which is high and not a typical number, especially considering this is the total of all buffer pools in the system. Another measure of how your buffer pools are doing is to calculate the *number of getpages per sync. I/O* (calculated by IBM OMEGAMON XE for DB2 PE as GETPAGE PER SYN.READ-RANDOM). As the application is waiting for sync. I/Os to complete, the higher this ratio, the better. In our case, DB2 must perform a sync I/O only every 2471.29 getpage requests.

Here are a few other counters to monitor:

- PREF.DISABLED-NO BUFFER: This indicates the number of times DB2 was unable to perform a prefetch operation because there were no buffers available for the prefetch operation. This can occur because the sequential prefetch buffer pool threshold is reached (more than 90% of the pages in the BP are not available to be reused) or the VPSEQT buffer pool threshold is set to zero (effectively disabling all prefetch for that buffer pool). Hitting the BP sequential threshold is not good, so you might want to increase the BP size, or adjust the BP write thresholds to trigger writing back to disk sooner.
- PREF.DISABLED-NO READ ENG: This indicates that DB2 ran out of prefetch engines. There are 600 prefetch engines in DB2 and all of them were in use at some point. This happens when many queries run that use DB2 parallelism, and each of the parallel tasks is triggering prefetch. As you cannot adjust the number of prefetch engines, it is better to spread out the queries that are using all these engines (run fewer queries in parallel).
- PAGE-INS REQUIRED FOR READ: When DB2 must perform a read I/O operation, its page frame is fixed in real storage (to make sure that the frame is not stolen when I/O is performed in to that frame). DB2 checks to see whether the page frame that holds the buffer in the BP must be paged in from disk before doing I/O (of the data) into that page frame. If the frame was on auxiliary storage, this counter is incremented. So, if that is the case, DB2 suffers a double I/O penalty, first when the paging I/O brings back the frame from auxiliary storage, and then another I/O to bring the actual data or index page from disk into the DB2 buffer pool. When you see a high number in this field, it is best to reevaluate the DB2 real storage requirements. Maybe you must add more real storage to the LPAR. Another way to avoid paging for DB2 buffer pool it to use the **`PGFIX(YES)`** buffer pool option. This option fixes all the DB2 pages for that buffer pool in real storage so they are not paged out to auxiliary storage. However, make sure that there is enough real storage available for the LPAR, or fixing the DB2 buffer pools might increase paging activity for other workloads in the system.

Have these three counters show a low or zero value. If that is not the case, you must investigate and at least understand why they are non-zero if you cannot remedy the problem.

If there is DB2 sort activity that is triggered by SQL activity, such as by **`ORDER BY`** or **`GROUP BY`**, the SORT/MERGE section shows non-zero values. It is important to make sure the workfile requests that degraded or were rejected are kept to a minimum.

The last section of the buffer pool report contains information about DB2 write operations. This is where DB2 must write the updated pages in the buffer pool or group buffer pool back disk. Most write activity is asynchronous to the application, meaning applications typically do not have to wait for write I/Os to complete. However, it is a preferred practice to verify this section in the DB2 statistics report to make sure that there are no issues with the write performance.

Here are some typical counters to verify:

- BUFFER UPDATES; This represents the number of rows that were inserted, updated, or deleted during the DB2 statistics interval. If multiple rows are updated on the same page, the counter is incremented multiple times, once for each row.
- PAGES WRITTEN: This is the number of pages from the local buffer pool that was written back to disk.
- SYNCHRONOUS WRITES: In some situations, DB2 can perform a synchronous write operation. This is a write that the application must wait for until it completes. A high number here means that many applications are waiting for a write I/O to complete and that is not a good thing. Therefore, if you see a high number here, you must investigate what is the trigger for them and see whether you can reduce them.
- ASYNCHRONOUS WRITES: This is the normal type of write operation. When the buffer pool hits certain thresholds (see below), it triggers an asynchronous write operation to externalize updated pages in the buffer pool back to disk. When a transaction reaches a commit point, DB2 writes to the log and forces the log buffers to disk to harden the fact that the transaction has reached a transaction boundary, but the updated pages are not written back to disk at that point. In case the system fails before the data pages are written back to disk (so with updates only in the buffer pool), DB2 can still recover because the information is on the DB2 log.

  The counter above is zero because all the objects in the test run are group buffer pool dependent, so the writes to disk occur from the group buffer pool, not the local buffer pool.
- PAGES WRTN FOR CASTOUT I/O: When you use DB2 data sharing, at commit the log buffers are written to disk but the updated pages are also written to the group buffer pool when the object is group buffer pool dependent. This ensures data coherency between the different members of the data sharing group. Later, the changed pages in the group buffer pool must be written back to disk. This is called *castout processing*. This counter represents the pages that are written back to disk by the castout operation on this member.
- NUMBER OF CASTOUT I/O: This number represents the number of castout I/O operations, as opposed to the number of pages that are written by castout operations.

There are a number of buffer pool thresholds that indicate to DB2 that it is time to write updated pages back from the virtual pool to disk. Here are two of these thresholds:

- HORIZ.DEF.WRITE THRESHOLD: When the number of unavailable pages reaches the DWQT buffer pool threshold (set to 30% by default, but you can change the value by running the `-ALTER BUFFERPOOL` command), deferred (asynchronous) writes are triggered.
- VERTI.DEF.WRITE THRESHOLD: This is the same as the deferred write threshold, but at the page set level. When the number of updated pages for a data set reaches the VDWQT, deferred (asynchronous) writes begin for that data set. The default is 5%, but you can change the value by running the `-ALTER BUFFERPOOL` command. The VDWQT value can either be a percentage (of the BP size) or the number of changed pages that changed before the asynchronous writes are triggered.

For the workload that produced the BP numbers above, these values are zero. This is because all objects are GBP-dependent and the updated pages (few per transaction) are written to the GBP at commit time, and are written back to disk through castout I/O operations. As such, the DWQT and VDQWT are not reached.

As with the BP read operation section, there are a few counters that should have a zero or low value:

- DM THRESHOLD: This counter indicates that the number of unavailable pages in the BP reached 95% or more, which is the data manager threshold. Normally, DB2 accesses the page in the virtual buffer pool once for each page, no matter how many rows are retrieved or updated on that page. If the threshold is exceeded, a getpage request is done for each row instead of each page. If more than one row is retrieved or updated in a page, more than one getpage and release page request is performed on that page. This is a bad thing, but it is an autonomic mechanism that is built into DB2 to slow things down, giving the write engines time to write some of the unavailable pages back to disk.
- WRITE ENGINE NOT AVAILABLE: This counter is the number of times that a write engine was not available. This field is no longer populated by DB2, as it was too common to hit the maximum of 300 engines running at the same time. Under normal circumstances, it is not a problem when you hit this limit, as applications normally do not wait for write I/Os to complete.
- PAGE-INS REQUIRED FOR WRITE: This counter is similar to the page-in counter for read operations. Before doing a write I/O from a buffer, the frame is fixed in real storage. When DB2 detects the frame in auxiliary storage before the write I/O starts, it increments the counter.

For more information about buffer pool tuning, see *DB2 9 for z/OS: Buffer Pool Monitoring and Tuning*, REDP-4604.

Group buffer pool section

When you are running a DB2 data sharing system, and some of the objects are group buffer pool dependent (GBP-dependent), the DB2 statistics report also has a group buffer pool section, as shown in Example 8-16.

Example 8-16 Statistics report - group buffer pool section

```
GROUP TOTAL                    QUANTITY  /SECOND  /THREAD  /COMMIT
-----------------------------  --------  -------  -------  -------
GROUP BP R/W RATIO (%)            46.13      N/A      N/A      N/A
GBP-DEPENDENT GETPAGES           517.7K  8611.01  6471.19     6.08
SYN.READ(XI)-DATA RETURNED     13273.00   220.77   165.91     0.16
SYN.READ(XI)-NO DATA RETURN      169.00     2.81     2.11     0.00
SYN.READ(NF)-DATA RETURNED       111.00     1.85     1.39     0.00
SYN.READ(NF)-NO DATA RETURN       41.00     0.68     0.51     0.00
UNREGISTER PAGE                    0.00     0.00     0.00     0.00

CLEAN PAGES SYNC.WRITTEN           0.00     0.00     0.00     0.00
CLEAN PAGES ASYNC.WRTN             0.00     0.00     0.00     0.00
REG.PAGE LIST (RPL) REQUEST       73.00     1.21     0.91     0.00
NUMBER OF PAGES RETR.FROM GBP     55.00     0.91     0.69     0.00
PAGES CASTOUT                    554.00     9.21     6.92     0.01
UNLOCK CASTOUT                    13.00     0.22     0.16     0.00

READ CASTOUT CLASS                17.00     0.28     0.21     0.00
READ DIRECTORY INFO                0.00     0.00     0.00     0.00
READ STORAGE STATISTICS           47.00     0.78     0.59     0.00
REGISTER PAGE                    173.00     2.88     2.16     0.00
DELETE NAME                        0.00     0.00     0.00     0.00
ASYNCH GBP REQUESTS            15997.00   266.08   199.96     0.19
EXPLICIT X-INVALIDATIONS           0.00     0.00     0.00     0.00
```

```
CASTOUT CLASS THRESHOLD          17.00     0.28     0.21     0.00
GROUP BP CASTOUT THRESHOLD        0.00     0.00     0.00     0.00
GBP CHECKPOINTS TRIGGERED         0.00     0.00     0.00     0.00
WRITE FAILED-NO STORAGE           0.00     0.00     0.00     0.00

WRITE TO SEC-GBP FAILED           0.00     0.00     0.00     0.00
DELETE NAME LIST SEC-GBP          0.00     0.00     0.00     0.00
DELETE NAME FROM SEC-GBP          0.00     0.00     0.00     0.00
UNLOCK CASTOUT STATS SEC-GBP      0.00     0.00     0.00     0.00
ASYNCH SEC-GBP REQUESTS           0.00     0.00     0.00     0.00

GROUP TOTAL  CONTINUED   QUANTITY  /SECOND  /THREAD  /COMMIT
-----------------------  --------  -------  -------  -------
WRITE AND REGISTER       21843.00   363.32   273.04     0.26
WRITE AND REGISTER MULT   3124.00    51.96    39.05     0.04
CHANGED PGS SYNC.WRTN    28879.00   480.35   360.99     0.34
CHANGED PGS ASYNC.WRTN     257.00     4.27     3.21     0.00
PAGES WRITE & REG MULT    7293.00   121.31    91.16     0.09
READ FOR CASTOUT             1.00     0.02     0.01     0.00
READ FOR CASTOUT MULT       83.00     1.38     1.04     0.00

PAGE P-LOCK LOCK REQ     31760.00   528.28   397.00     0.37
 SPACE MAP PAGES          1807.00    30.06    22.59     0.02
 DATA PAGES              14583.00   242.56   182.29     0.17
 INDEX LEAF PAGES        15370.00   255.65   192.13     0.18

PAGE P-LOCK UNLOCK REQ   29010.00   482.53   362.63     0.34

PAGE P-LOCK LOCK SUSP     3469.00    57.70    43.36     0.04
 SPACE MAP PAGES           272.00     4.52     3.40     0.00
 DATA PAGES               1442.00    23.99    18.02     0.02
 INDEX LEAF PAGES         1755.00    29.19    21.94     0.02

PAGE P-LOCK LOCK NEG      2041.00    33.95    25.51     0.02
 SPACE MAP PAGES           113.00     1.88     1.41     0.00
 DATA PAGES                942.00    15.67    11.77     0.01
 INDEX LEAF PAGES          986.00    16.40    12.32     0.01
```

The main purpose of the group buffer pool in a DB2 data sharing environment is to ensure buffer coherency, that is, all members of the data sharing group can obtain the correct (most recent) version of the page that they want to process. To achieve this task, updated pages are written to the group buffer pool at commit time, and if these pages are present in the local buffer pool of other members, these pages are cross-invalidated (XI). The next time such a member requests that page, it sees the XI flag and it retrieves a new (latest) copy of the pages either from the group buffer pool or from disk.

To reduce this type of data sharing impact (refreshing XI-pages) to a minimum, make sure that if such a refresh operation must occur that the requested page is available in the group buffer pool. A page can be retrieved from the group buffer pool at microsecond speed, and reading from disk typically takes milliseconds.

To see whether DB2 can accomplish this task, check the group buffer pool section:

- SYN.READ(XI)-DATA RETURNED: This is the number of times that this DB2 member found an XI page in its local buffer pool, went to the GBP, and found the page in the GBP.
- SYN.READ(XI)-NO DATA RETURN: This is the number of times that this DB2 member found an XI page in its local buffer pool, went to the GBP, and did not find the page in the GBP.

In general, keep the Sync.Read(XI) miss ratio below 10% by using the following formulas:

- TOTAL SYN.READ(XI) = SYN.READ(XI)-DATA RETURNED + SYN.READ(XI)-NO DATA RETURN
- Sync.Read(XI) miss ratio = SYN.READ(XI)-NO DATA RETURN / TOTAL SYN.READ(XI)

In our example, the ration is 169/(169 + 13273) = 1.25%, which is good.

Typically, only changed pages are written to the GBP (**GBPCACHE(CHANGED)** is the default). However, you can use the group buffer pool as an auxiliary storage level for unchanged pages as well if you specify **GBPCACHE(ALL)**. This way, unchanged pages are also written to the GBP. When DB2 does not find a page in the local BP, it checks the GBP first before going to disk (GBP retrieval is much faster than I/O from disk). If the page is in the GBP, it is reused from there instead of reading from disk.

To verify the efficiency of this extra caching level, you can use the SYN.READ(NF)-DATA RETURNED and SYN.READ(NF)-NO DATA RETURN GBP statistics. They are similar to the XI information, but they represent the time that DB2 was not able to find the page in the local BP, and went to the GBP, and was either successful or unsuccessful in finding the page in the GBP. Our tests used the default **GBPCACHE(CHANGED)** setting, and the local BP hit ratio that we described in the (local) BP section above, is high, so there is little benefit in using **GBPCACHE(ALL)** for this application.

The process to write changed pages from the GBP back to disk is called castout processing. As for local BP, castout processing is also triggered by a number of thresholds:

- CASTOUT CLASS THRESHOLD: This is the number of times the group buffer pool castout was initiated because the group buffer pool class castout threshold was exceeded. This is similar to the VDWQT threshold at the page set level for local buffer pools. Queues inside the GBP are not by page set like local BP, but are based on a class. When the number of changed pages exceeds the percentage that is specified by CLASST, castout for that class is triggered. The default is 5%.
- GROUP BP CASTOUT THRESHOLD: This is the number of times group buffer pool castout was initiated because the group buffer pool castout threshold was exceeded. This threshold is similar to the DWQT threshold for local buffer pools. When the number of changed pages exceeds the threshold, castout is triggered. The default GBPOOL threshold is 30%.
- GBP CHECKPOINTS TRIGGERED: This is the number of times group buffer pool castout was initiated because a group buffer pool checkpoint was initiated (the default is every four minutes). This is also similar to DB2 system checkpoint, which triggers asynchronous writes for all updated pages in the local BP. You see only a non-zero value on the member if that member is the GBP structure owner. It is the structure owner that is responsible for GBP checkpoint processing.

In our test case, the castout is driven by the CLASST threshold, which is expected behavior. There are not enough different changed pages in the GBP to trigger the GBPOOL threshold.

As with the local BP, there are a few counters where a zero (or low) value is good:

- WRITE FAILED-NO STORAGE: This occurs when a process must write to the GBP but is unable to because the GBP is full. This is a bad condition. Applications must write all their changed pages to the GBP at commit time (for objects that are GBP-dependent). If they cannot do so, these transactions are suspended (in commit processing). DB2 does not fail these transactions immediately, but waits a few seconds and tries to write to the GBP again. If the write keeps failing, it is added to the logical page list (LPL) requiring recovery.

  If the problem is not because of a momentary surge in activity, you need either to decrease the group buffer pool castout thresholds, to trigger castout earlier, or to increase the number of data entries in the group buffer pool through either increasing the total size of the group buffer pool, or adjust the ratio of directory entries to data entries in favor of data entries.

  > **Tip:** You can also use the `ALLOWAUTOALT(YES)` option to allow XES to dynamically adjust your DB2 GBP. This feature is not intended to adjust GBP settings to deal with sudden spikes of activity, but it designed to adjust the settings when the workload changes gradually over time.
  >
  > For more information, see “Auto Alter capabilities” in *DB2 10 for z/OS Data Sharing: Planning and Administration*, SC19-2973 and “Identifying the coupling facility structures”, found at:
  >
  > http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.ieaf100%2Ficfstr.htm

- WRITE TO SEC-GBP FAILED: This is similar to the previous counter, but applies to writes to the secondary group buffer pool.

As page P-lock processing is handled by the DB2 buffer manager component, the DB2 group buffer pool statistics also contain valuable information about the page P-lock activity at the BP level (the example above shows only the total for all GBPs, but the DB2 statistics report contains this type of information for each group buffer pool.

The DB2 statistics record contains information about the page P-lock activity in terms of the number of requests, the number of suspensions, and the number of negotiations. It also distinguishes between page P-locks for space map pages, data pages (when using row level locking), and page P-locks for index leaf pages. A higher number of page P-lock requests means that DB2 must do some additional processing of these transactions, which typically translates into more processor and increased elapsed time. Also, acquiring a page P-lock is less expensive than a suspension for a page P-lock, which in turn is less expensive than negotiating a page P-lock. (Unlike L-locks or transaction locks, P-locks can be negotiated between members, but it is an expensive process, typically requiring forced writes to the active log data set and synchronous writes to the group buffer pool.

CPU information

The CPU Times section of the DB2 statistics report, which is shown in Example 8-17 on page 417, provides information about the amount of processing that is used by the different DB2 system address spaces:

- SYSTEM SERVICES ADDRESS SPACE (ssidMSTR)
- DATABASE SERVICES ADDRESS SPAC (ssidDBM1)
- IRLM
- DDF ADDRESS SPACE (ssidDIST)

Example 8-17 Statistics report - CPU Times section

```
CPU TIMES                          TCB TIME      PREEMPT SRB   NONPREEMPT SRB       TOTAL TIME  PREEMPT IIP SRB        /COMMIT
-------------------------------  ---------------  ---------------  ---------------  ---------------  ---------------  --------------
SYSTEM SERVICES ADDRESS SPACE      0.032263         0.468393         0.011236         0.511892              N/A         0.000006
DATABASE SERVICES ADDRESS SPACE    0.011671         0.270898         0.075548         0.358117         0.011764         0.000004
IRLM                               0.000014         0.000000         1.351361         1.351376              N/A         0.000016
DDF ADDRESS SPACE                  3.492625        36.759432         0.721285        40.973342        22.948061         0.000481

TOTAL                              3.536572        37.498723         2.159430        43.194726        22.959825         0.000507
```

The CPU time that is reported here is the amount of processing that is used by DB2 to perform system-related activity, on behalf of the applications that are running SQL requests. For example, when DB2 must access a table space the first time, the data set is not open yet. Therefore, the application is suspended, and DB2 switches to a task control block (TCB) in the DBM 1 address space. This TCB performs the allocation and physical open of the data set. The processing to perform the allocation and physical open is charged to the TCB processing time of the DBM1 address space. After the data set open is done, the application is resumed and continues processing.

The WLM-managed SP address spaces are not considered system address spaces. The processing that is used by those spaces is reported in the DB2 nested activity section of the DB2 accounting reports for the different applications.

Note: The processing time that is reported by the DDF (ssidDIST) address space includes both the processing time that is used by system tasks running in this address space, but it also includes the processing time that is used by all the database access threads (DBATs) in the system. DBATs run as pre-emptible Service Request Blocks (SRBs) in the DIST address space, so all the work that is done by remote connections running SQL statements against a DB2 for z/OS system shows up in the DB2 accounting records, and in the CPU Times section in the DB2 statistics report, under PREEMPT SRB time.

This section separates the CPU Time into the following types:

- TCB Time: This is the amount of processing time that is used by work that runs using a task control block as a dispatchable unit of work.
- PREEMPT SRB: This is the amount of processing time that is used by work that runs by using a pre-emptible service request block as a dispatchable unit of work.
- NONPREEMPT SRB: This is similar to PREEMPT SRB, but this type of dispatchable unit must voluntarily relinquish control, but the other types can be interrupted at any time by the MVS dispatcher. DB2 has few of these types of SRB. IRLM still uses this type of dispatchable unit, but IRLM requests are short, must run to completion without being interrupted, and must be serviced with a high priority.
- TOTAL TIME: This is the sum of TCB Time, PREEMPT SRB, and NONPREEMPT SRB.
- PREEMPT IIP SRB: This is the amount of processing that DB2 ran on a specialty engine, such as zIIP or zAAP. It is *not* included in the other CPU Time fields, as users are not charged for the use of the zIIP or zAAP engine.

The workload in Example 8-17 is a distributed workload, and the majority of the processing time is PREEMPT SRB time in the DIST address space. Note the considerable amount of zIIP offload for the DDF work, 22.959825/ (22.959825 + 37.498723)=38.4%.

When you use **ZOSMETRICS=YES DSNZPARM**, at each statistics interval, DB2 communicates with RMF and obtains some useful information about the LPAR that the DB2 system is running on, as shown in Example 8-18. This information includes the following items:

- CP LPAR: The number of CPs on the LPAR.
- CPU UTILIZATION LPAR: The total CPU usage of the LPAR at the time that the information is collected.
- CPU UTILIZATION DB2: The CPU usage of all the DB2 system address spaces. These are all address spaces starting with ssid, where ssid is the DB2 subsystem name. In our example, the DB2 CPU usage is higher than the sum of DB2 MSTR + DB2 DBM1. This is because our workload is a distributed workload, so the CPU that is used by the ssidDIST address space is also included in this number.
- REAL STORAGE LPAR: The amount of real storage on the LPAR.
- USED REAL STORAGE DB2: The amount of real storage that is used by DB2.
- USED VIRTUAL STOR DB2: The amount of virtual storage that is used by DB2, as seen by RMF. This is equal to the amount of real storage, so nothing pages out to auxiliary storage.

Example 8-18 Statistics report - RMF CPU and storage metrics

```
CPU AND STORAGE METRICS                 QUANTITY
---------------------------  ------------------
CP LPAR                                     6.00
CPU UTILIZATION LPAR                       45.00
CPU UTILIZATION DB2                         5.00
CPU UTILIZATION DB2 MSTR                    0.00
CPU UTILIZATION DB2 DBM1                    1.00

UNREFERENCED INTERVAL COUNT             65535.00

REAL STORAGE LPAR      (MB)              8191.00
FREE REAL STORAGE LPAR (MB)              1359.00
USED REAL STORAGE DB2  (MB)               327.00

VIRTUAL STORAGE LPAR   (MB)             70991.00
FREE VIRTUAL STOR LPAR (MB)             65007.00
USED VIRTUAL STOR DB2  (MB)               327.00
```

There are many more sections in the DB2 statistics record with information about how the subsystem is doing. Describing all of them is beyond the scope of this book. For more information, see *DB2 10 for z/OS Managing Performance*, SC19-2978 and *Tivoli OMEGAMON XE for DB2 on z/OS Report Reference*, SH12-6963.

8.4.5 Analyzing DB2 accounting data

Besides information about the entire DB2 subsystem that is contained in the DB2 statistics trace records, DB2 can also collect information about the performance and work that is performed by individual transactions. This information is in the DB2 accounting trace records, IFCID 3 for plan level information, and IFCID 239 for package level information.

Example 8-19 on page 419 shows a sample SYSIN that creates a DB2 accounting report using the IBM OMEGAMON XE for DB2 PE on z/OS batch reporting facility.

Example 8-19 Create a DB2 accounting report

```
//SYSIN    DD  *
DB2PM
* *********************************
*       GLOBAL PARMS
* *********************************
  GLOBAL
*        Adjust for US East Coast DST
         TIMEZONE (+4)
         FROM(,21:29:00)
           TO(,21:30:01)
*        Include the entire group
         INCLUDE( GROUP(DB0ZG))
 * ***********************************
 *       ACCOUNTING REPORTS
 * ***********************************
  ACCOUNTING
              REPORT
                       DDNAME(ACTRCDD1)
                       LAYOUT(LONG)
                       ORDER(TRANSACT)
EXEC
```

The statements in Example 8-19 generate an IBM OMEGAMON XE for DB2 PE on z/OS accounting report for all members of data sharing group DB0ZG (which includes two members named D0Z1 and D0Z2). The information is grouped by DB2 transaction name (**ORDER(TRANSACT)**). The report covers only a single minute. This report is congruent with Example 8-18 on page 418.

As this is an accounting report, the averages are based on the number of occurrences (the number of transactions that qualify for the filter criteria in that interval). This is different from an accounting trace report, which shows individual transactions (so there are no averages).

As with the statistics report, the accounting report consists of many sections. As an example, we use the sections of the transaction that is called ‘TraderClientApplication’. For more information about how we set the client information strings for this test, see “Specifying client information at the data source level” on page 366.

Identification and highlights

Example 8-21 on page 420 shows the identification section, elapsed time distribution, and class 2 time distribution section of the DB2 accounting report for the TraderClientApplication transaction.

Example 8-20 Accounting report - identification elapsed time and class 2 time distribution

```
LOCATION: DB0Z                  OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1M1)                PAGE: 1-4
      GROUP: DB0ZG                           ACCOUNTING REPORT - LONG                  REQUESTED FROM: ALL      21:29:00.00
     MEMBER: D0Z1                                                                                 TO: DATES    21:30:01.00
  SUBSYSTEM: D0Z1                                 ORDER: TRANSACT                       INTERVAL FROM: 08/13/12 21:29:00.37
DB2 VERSION: V10                                  SCOPE: MEMBER                                   TO: 08/13/12 21:30:00.99

TRANSACT: TraderClientApplication

ELAPSED TIME DISTRIBUTION                                           CLASS 2 TIME DISTRIBUTION
----------------------------------------------------------------    ----------------------------------------------------------------
APPL   |==============================> 62%                         CPU    |==========> 21%
DB2    |==========> 20%                                             SECPU  |============> 24%
SUSP   |=========> 18%                                              NOTACC |====> 9%
                                                                    SUSP   |======================> 47%
```

Example 8-20 on page 419 provides a quick overview of where the majority of the time is spent; in the application, in DB2 (and when in DB2, whether the transaction is mainly using CPU), on a general CP or specialty engine, or is suspended in DB2 waiting for some known event to complete. In this case, most of the time in DB2 is spent being suspended. This does not necessarily mean that there is a problem with the transaction. It can be normal that the transaction is lightweight and the overall transaction response time is good and well within the service levels.

As we are looking at the overall performance of the TraderClientApplication here, we use an accounting report, which means that most counters are averages, which are based on the number of transactions that are run during the reporting interval. The denominator to calculate these averages is #OCCURRENCES, which can be found in the highlights section of the accounting report, as shown in Example 8-21.

Example 8-21 Accounting report - highlights

```
HIGHLIGHTS
--------------------------
#OCCURRENCES    :    86521
#ALLIEDS        :        0
#ALLIEDS DISTRIB:        0
#DBATS          :    86521
#DBATS DISTRIB. :        0
#NO PROGRAM DATA:        0
#NORMAL TERMINAT:    86521
#DDFRRSAF ROLLUP:        0
#ABNORMAL TERMIN:        0
#CP/X PARALLEL. :        0
#IO PARALLELISM :        0
#INCREMENT. BIND:        0
#COMMITS        :    86522
#ROLLBACKS      :        1
#SVPT REQUESTS  :        0
#SVPT RELEASE   :        0
#SVPT ROLLBACK  :        0
MAX SQL CASC LVL:        0
UPDATE/COMMIT   :     0.21
SYNCH I/O AVG.  : 0.000680
```

The example indicates that #DBATS is 86521, so we are clearly dealing with a type 4 connection. We are not using accounting rollup (#DDFRRSAF ROLLUP is zero). It is a preferred practice to check #COMMITS versus #ROLLBACKS to make sure that the vast majority of the work is succeeding (and committing), and not constantly failing (rolling back its unit of work).

The highlights section calculates the SYNCH I/O AVG. seen by DB2, which provides you with an easy way to do a smoke test and see whether average synchronous I/O times are correct. In this case, they are 680 microseconds.

The normal termination section, which is shown in Example 8-22 on page 421, gives you a quick idea about what triggered the accounting record. This is from the same accounting report in Example 8-21 that uses type 4 connectivity, so expect to see a high number of TYPE2 INACTIVE threads (where DB2 separates the DBAT and the connection at commit time, by pooling the DBAT and making the connection inactive).

Example 8-22 Accounting report - normal termination

```
NORMAL TERM.      AVERAGE     TOTAL
---------------  --------  --------
NEW USER             0.00         1
DEALLOCATION         0.00         1
APPL.PROGR. END      0.00         0
RESIGNON             0.00         0
DBAT INACTIVE        0.00         0
TYPE2 INACTIVE       1.00     86519
RRS COMMIT           0.00         0

END USER THRESH      0.00         0
BLOCK STOR THR       0.00         0
STALENESS THR        0.00         0
```

SQL DML that is performed by the application

When monitoring transaction performance, it is important to have information about the transaction profile. An important part of the transaction profile is the number and type of SQL statements that are issued by the transaction against the database engine. This information is in the SQL DML section of the DB2 accounting report, as shown in Example 8-23.

Example 8-23 Accounting report - SQL DML

```
SQL DML   AVERAGE    TOTAL
-------- -------- --------
SELECT       0.00        0
INSERT       0.04     3274
 ROWS        0.04     3274
UPDATE       0.16    14073
 ROWS        0.17    14911
MERGE        0.00        0
DELETE       0.01      838
 ROWS        0.01      838

DESCRIBE     0.32    28103
DESC.TBL     0.00        0
PREPARE      1.29   111659
OPEN         1.12    96651
FETCH        0.02     1634
 ROWS        1.30   112137
CLOSE        0.02     1634

DML-ALL      2.98   257866
```

On average, the TraderClientApplication application runs 2.98 SQL statements, typically **PREPARE**, **OPEN**, and a **FETCH** statements. The number of FETCH operations is only 0.02, but the number of ROWS fetched is 1.30. This is the result of a DB2 10 performance enhancement for DRDA, where **OPEN** and **FETCH** are combined in a single operation, but as a result, the **FETCH** operation is not counted. However, the number of rows fetched (1.30) accurately reflects the number of rows that were retrieved. The transaction also occasionally performs an **INSERT** or **UPDATE** operation.

As the application is using dynamic SQL (because of the number of **PREPARE** operations), it is important to verify whether these **PREPARE** operations are able to use the global dynamic statement cache or not, that is, whether they are performing a short prepare (FOUND IN CACHE) or a full prepare (NOT FOUND IN CACHE). To verify this situation, see the DYNAMIC SQL STMT section in the accounting report, as shown in Example 8-24.

Example 8-24 Accounting report - DYNAMIC SQL STMT

```
DYNAMIC SQL STMT       AVERAGE     TOTAL
--------------------  --------  --------
REOPTIMIZATION            0.00         0
NOT FOUND IN CACHE        0.00         7
FOUND IN CACHE            1.29    111652
IMPLICIT PREPARES         0.00         0
PREPARES AVOIDED          0.00         0
CACHE_LIMIT_EXCEEDED      0.00         0
PREP_STMT_PURGED          0.00         0
CSWL - STMTS PARSED       0.00         0
CSWL - LITS REPLACED      0.00         0
CSWL - MATCHES FOUND      0.00         0
CSWL - DUPLS CREATED      0.00         0
```

In our case, we have a good cache hit ratio. The Trader application uses a limited number of SQL statements and they typically use parameter markers. When you use parameter markers, DB2 uses a '?' at prepare time, and provides the actual value of the parameter marker at execution time. So, the only difference between the SQL statements is the value that is provided at run time. The actual SQL statement text (using the '?') is used by DB2 to determine whether the statement is in the cache. Using a parameter marker instead of a literal value increases the chance of finding a statement cache match, allowing DB2 to reuse the cached statement.

Locking information

Another important part of the transaction profile is the locking behavior of the transaction. The locking section and data sharing locking section is shown in Example 8-25. We already looked at some of the fields when we described the DB2 statistics report in "Locking and data sharing locking sections" on page 406. For more information about local and global suspensions, timeouts, deadlocks, and lock escalations, see that section. The only difference between the accounting and statistics fields is that the accounting information applies to a specific transaction/application, but the statistics data applies to all the transactions in the DB2 subsystem that ran during the reporting time frame.

Example 8-25 Accounting report - LOCKING and DATA SHARING

```
LOCKING                AVERAGE    TOTAL
--------------------- -------- --------
TIMEOUTS                  0.00        0
DEADLOCKS                 0.00        0
ESCAL.(SHARED)            0.00        0
ESCAL.(EXCLUS)            0.00        0
MAX PG/ROW LOCKS HELD     1.89      100
LOCK REQUEST              6.56   567488
UNLOCK REQUEST            1.21   104367
QUERY REQUEST             0.00        0
CHANGE REQUEST            0.33    28384
OTHER REQUEST             0.00        0
TOTAL SUSPENSIONS         0.05     4063
```

```
 LOCK SUSPENSIONS        0.01      530
 IRLM LATCH SUSPENS.     0.04     3531
 OTHER SUSPENS.          0.00        2

DATA SHARING          AVERAGE     TOTAL
------------------  --------  --------
GLOBAL CONT RATE(%)      1.91       N/A
FALSE CONT RATE(%)       1.07       N/A
P/L-LOCKS XES(%)        47.70       N/A
LOCK REQ   - PLOCKS      0.38     32490
UNLOCK REQ - PLOCKS      0.34     29613
CHANGE REQ - PLOCKS      0.00         3
LOCK REQ   - XES         3.31    286161
UNLOCK REQ - XES         0.45     38937
CHANGE REQ - XES         0.26     22095
SUSPENDS   - IRLM        0.04      3140
SUSPENDS   - XES         0.00         0
CONVERSIONS- XES         0.22     19317
FALSE CONTENTIONS        0.05      4006
INCOMPATIBLE LOCKS       0.00         0
NOTIFY MSGS SENT         0.00         2
```

LOCK REQUESTS

This counter represents the number of **(L-)LOCK** requests that were sent to IRLM. It is important to reduce the number of locks as much as possible, as acquiring locks uses processing time, but might also prevent other people from accessing the same resource if they must access the page/row in a way that is not compatible with the lock that your transaction is holding. This can be achieved by application design and the usage of DB2 lock avoidance techniques.

UNLOCK REQUESTS

This counter represents the number of **(L-)UNLOCK** requests that were sent to IRLM. A single UNLOCK request can release many locks in a single operation. For example, at commit time, DB2 issues an **UNLOCK ANY** request to release all locks that are no longer required. If a program that is using isolation level CS is fetching from a read only cursor, DB2 tries to avoid taking locks as much as possible (lock avoidance). However, DB2 might have to acquire locks as it fetches rows from the cursor. If a lock was acquired and the cursor moves off the row/page, that lock is released by an **UNLOCK** request.

So, DB2 issues **UNLOCK** requests mainly for this type of cursor fetching (unlocking a row/page at a time) or at commit (unlocking them all).

> **Tip:** To assess the effectiveness of the DB2 data lock avoidance techniques at a high level, you can calculate **#UNLOCK/COMMIT**. If that value is greater than 5, DB2 lock avoidance is not effective. In that case, you might want to ensure that the application uses the **ISOLATION(CS)** and **CURRENTDATA(NO) BIND** options.

In Example 8-25 on page 422, UNLOCK/COMMIT is 1.21 (avg #unlocks, as the #commits is almost identical to the #occurrences here), which is a good value. This is a high-level check. If, for example, the application is light and is doing only a few fetches, even if lock avoidance is not working at all, the ratio is still low. Therefore, it is always important to check the overall transaction profile and not blindly apply any rules of thumb.

For more information about lock avoidance, see *DB2 9 for z/OS: Resource Serialization and Concurrency Control*, SG24-4725.

MAX PG/ROW LOCKS HELD

There is an interesting piece of information in the accounting locking section that is not available in the statistics record. This is the MAX PG/ROW LOCKS HELD field. It is a useful indicator of the application's commit frequency. The counter applies to low level locks only (page or row, LOB and XML).

> **Tip:** In general, try to issue a `COMMIT` frequently enough to keep the average MAX PG/ROW LOCKS HELD below 100.

The AVERAGE value that is shown in the accounting REPORT is the average of MAX PG/ROW LOCKS HELD of all the accounting records that qualify for the report. The TOTAL is for the maximum of all MAX PG/ROWS LOCKS HELD, that is, the "high water mark" of all accounting records that qualify for the report. For example, if transaction A has a MAX PG/ROWS LOCKS HELD value of 10, and transaction B has a MAX PG/ROWS LOCKS HELD value of 20, then an accounting report that includes these two transactions has AVERAGE (average of maximum) of 15, and TOTAL (high water mark) of 20.

SYNCH.XES - LOCK REQUEST

In a data sharing environment, locking gets an additional dimension as some of the locks might be sent to the coupling facility to ensure transaction integrity and data coherency. This requires extra CPU cycles, so checking the locking profile and the number of lock requests that must be sent to XES is even more important in a data sharing environment than in a non-data sharing environment.

The SYNCH.XES - LOCK REQUEST counter represents the total number of lock requests that have been synchronously sent to XES. This number includes both P-locks (page set and page) and L-lock requests propagated to z/OS XES synchronously. This number is not incremented if the request is suspended during processing (either because of some type of global contention, or because the XES heuristic algorithm dedicated to convert the request from sync to async. The latter are included in the CONVERSIONS- XES counter.

Buffer pool information

The last piece of information that is important to have about the transaction profile is the (local and group) buffer pool information, which is shown in Example 8-26.

As with the locking information, we already looked at most of the information when we described these sections in the DB2 statistics report in "Buffer pool section" on page 408 and "Group buffer pool section" on page 413. The only difference between the accounting and statistics fields is that the accounting information applies to a specific transaction/application, but the statistics data applies to all the transactions in the DB2 subsystem that ran during the reporting time frame.

Example 8-26 shows only the totals for all buffer pools that are used by the transaction (TOTAL BPOOL). The accounting report contains the same information for each of the buffer pools that were accessed by the transaction.

Example 8-26 Accounting report - buffer pool and group buffer pool

```
TOTAL  BPOOL ACTIVITY   AVERAGE     TOTAL
---------------------  --------  --------
BPOOL HIT RATIO (%)       99.95       N/A
GETPAGES                   6.08    526390
```

```
GETPAGES-FAILED           0.00        0
BUFFER UPDATES            0.54    46784
SYNCHRONOUS WRITE         0.00        0
SYNCHRONOUS READ          0.00      221
SEQ. PREFETCH REQS        0.00        0
LIST PREFETCH REQS        0.00        5
DYN. PREFETCH REQS        0.01     1193
PAGES READ ASYNCHR.       0.00       56

GROUP TOTAL            AVERAGE    TOTAL
---------------------  --------  --------
GBP-DEPEND GETPAGES       6.08   526223
READ(XI)-DATA RETUR       0.15    13393
READ(XI)-NO DATA RT       0.00      180
READ(NF)-DATA RETUR       0.00      111
READ(NF)-NO DATA RT       0.00       41
PREFETCH PAGES READ       0.00       55
CLEAN PAGES WRITTEN       0.00        0
UNREGISTER PAGE           0.00        0
ASYNCH GBP REQUESTS       0.17    15073
EXPLICIT X-INVALID        0.00        0
ASYNCH SEC-GBP REQ        0.00        0
PG P-LOCK LOCK REQ        0.37    32294
 SPACE MAP PAGES          0.02     1862
 DATA PAGES               0.17    14832
 INDEX LEAF PAGES         0.18    15600
PG P-LOCK UNLOCK REQ      0.34    29495
PG P-LOCK LOCK SUSP       0.04     3502
 SPACE MAP PAGES          0.00      277
 DATA PAGES               0.02     1459
 INDEX LEAF PAGES         0.02     1766
WRITE AND REGISTER        0.23    20201
WRITE & REGISTER MULT     0.04     3181
CHANGED PAGES WRITTEN     0.32    27385
```

From an application profile point of view, the following information is typically used.

GETPAGE REQUEST

This is the number of times an SQL statement must request a page from the DB2 buffer manager component. When an SQL statement must retrieve a row, that row lives on a page, and that page lives in the DB2 buffer pool (or on disk, or in the group buffer pool (GBP) in a data sharing system). So, to obtain the row, there is request to the DB2 buffer manager component to get the page (getpage). The amount of getpage activity is a good measure for the amount of work that DB2 is performing to satisfy the SQL requests from the application.

BUFFER UPDATES

This is the number of times a buffer update occurs. This field is incremented every time that a page is updated and is ready to be written to DASD/GBP. DB2 typically increments the counter for each row that is changed (inserted, updated, or deleted). For example, if an application updates two rows on the same page, you are likely to see GETPAGE REQUEST 1, BUFFER UPDATES 2 (provided no additional getpages were required to retrieve the page or any index updates were needed).

If one of the SQL statements triggers workfile activity, for example, to perform ORDER BY, GROUP BY, or a sort during merge scan join processing, this is considered buffer update activity (these workfiles are created and inserted into the buffer pool), even though the actual SQL statements might read just data.

SYNCHRONOUS READS

When the DB2 buffer manager finds that the page is not in the buffer pool (or GBP), it has to read the page from disk. When DB2 reads a single page from disk, and the application waits for this page to be brought into the BP, this is called a *synchronous read* operation. Even though disk I/O has improved dramatically over the last couple of years, it is still orders of magnitude slower than retrieving a page that is already in the BP. Therefore, reducing the number of SYNC I/Os has a positive effect on the transaction response time. So, it is important to verify the amount I/O activity that an application must perform. A high number of sync I/O requests can be a sign of a poor access path (when combined with a high number of getpage requests) or a sign of a buffer pools that is performing poorly.

PAGES READ ASYNCHR.

This is the number of pages that were brought into the buffer pool by prefetch engines on behalf of this application. A high number of asynchronous pages can be a sequential process. This might be normal, for example, during a batch process that must work its way through the entire customer database, but could also be a sign of a process that is touching many more pages than it should (and these pages were brought in by a prefetch engine).

GBP PG P-LOCK activity

When looking at the group buffer pool from an application point of view, it is important to check the page P-lock activity, as this introduces extra impact. The group buffer pool section also has the breakdown (per group buffer pool) between data page P-locks (when using row level locking), space map P-locks, and page P-lock requests for index leaf pages. For example, we use P-locks for DATA PAGES for our example, which indicates the usage of row level locking (which is indeed the case for the Trader application. Switching the Trader application to use RLL virtually eliminates all the deadlocks that existed. Given that deadlocks are much more disruptive than the impact of page P-locks, this is certainly a good tradeoff.

Response time reporting for DRDA (T4) connections

When you analyze application elapsed and CPU time, it is important to understand where the time is being spent. DB2 distinguishes between accounting class 1, class 2, and class 3 times, as shown in Figure 8-30 on page 427 and Figure 8-31 on page 427.

The thread activity time is the entire time that the thread (application/transaction) was active. This time is identical to the DB2 class 1 elapsed time. This covers the time between the first SQL statement (that triggers the DB2 thread creation or reuse) until the thread ends or commits depending on the type of attachment you use. This is described in more detail in 8.4.2, “Creating DB2 accounting records at a transaction boundary” on page 396.

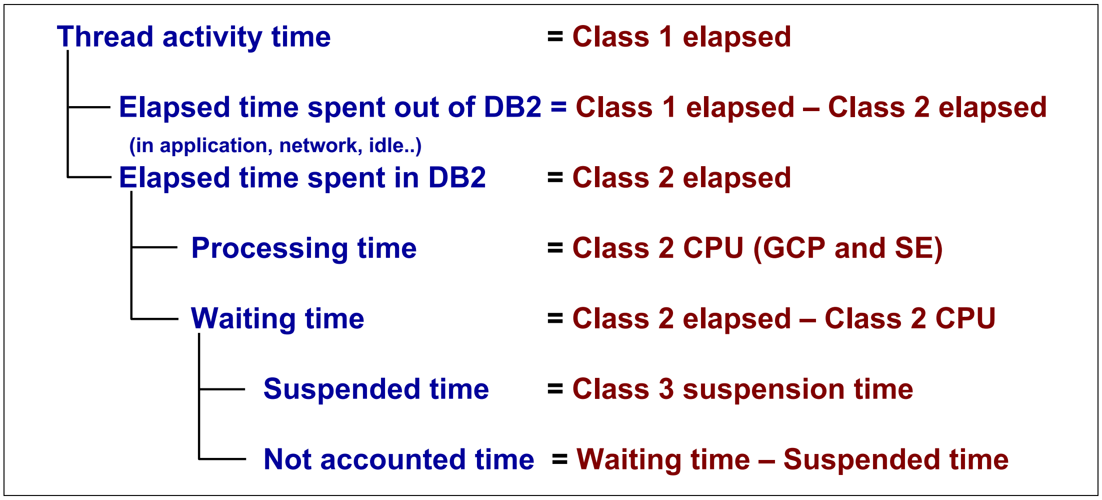


Figure 8-30 Thread activity time reporting

The accounting class 2 time is the time that the transaction spends inside the DB2 database engine. When the thread is running an SQL statement, it can either be using CPU time, recorded as CLASS 2 CPU time (when one general-purpose engine) or SE CPU time (when running on a specialty engine), or waiting for something. This waiting time is divided in to class 3 wait time, which is a wait that DB2 is aware of and can account for, like waiting for a lock that is held by another tran in an incompatible state, and waits where we know were in DB2 but it is not one of the classes 3 wait times that DB2 can report on. This latter time is also known as *not accounted time*, and is typically a small portion of the class 2 elapsed time.

Figure 8-31 shows the life of a transaction and how the work inside and outside of DB2 is reported on, as accounting class 1, class 2, and class 3 time.

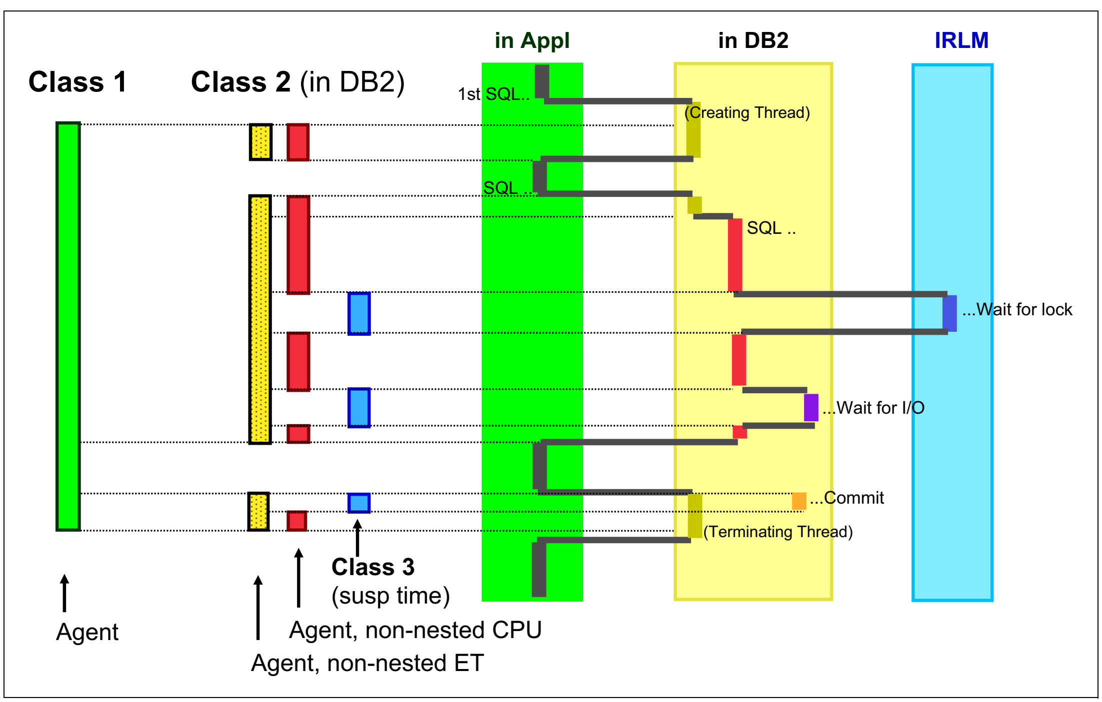


Figure 8-31 Accounting class1, 2, and 3 time reporting

Example 8-27 shows how the transaction is reported on in an OMPE accounting report. DB2 accounting information distinguishes between non-nested (activity on the main thread) and nested time (activity by stored procedures, UDFs, and triggers). This allows you to determine how much time is spent doing stored procedure work, for example, but this is beyond the scope of this publication. This section focuses on the non-nested activity, as the Trader application does not use any stored procedures, UDFs, or triggers.

Example 8-27 Accounting report -Class 1, 2, and 3 times

```
AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)
------------  ----------  ----------  ----------
ELAPSED TIME    0.002269    0.000853         N/P
 NONNESTED      0.002269    0.000853         N/A
 STORED PROC    0.000000    0.000000         N/A
 UDF            0.000000    0.000000         N/A
 TRIGGER        0.000000    0.000000         N/A

CP CPU TIME     0.000201    0.000178         N/P
 AGENT          0.000201    0.000178         N/A
  NONNESTED     0.000201    0.000178         N/P
  STORED PRC    0.000000    0.000000         N/A
  UDF           0.000000    0.000000         N/A
  TRIGGER       0.000000    0.000000         N/A
 PAR.TASKS      0.000000    0.000000         N/A

 SECP CPU       0.000000         N/A         N/A

SE CPU TIME     0.000234    0.000201         N/A
 NONNESTED      0.000234    0.000201         N/A
 STORED PROC    0.000000    0.000000         N/A
 UDF            0.000000    0.000000         N/A
 TRIGGER        0.000000    0.000000         N/A

 PAR.TASKS      0.000000    0.000000         N/A

SUSPEND TIME    0.000000    0.000399         N/A
 AGENT               N/A    0.000399         N/A
 PAR.TASKS           N/A    0.000000         N/A
 STORED PROC    0.000000         N/A         N/A
 UDF            0.000000         N/A         N/A

NOT ACCOUNT.         N/A    0.000075         N/A
DB2 ENT/EXIT         N/A        7.96         N/A
EN/EX-STPROC         N/A        0.00         N/A
EN/EX-UDF            N/A        0.00         N/A
DCAPT.DESCR.         N/A         N/A         N/P
LOG EXTRACT.         N/A         N/A         N/P


CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT
--------------------  ------------  --------
LOCK/LATCH(DB2+IRLM)      0.000036      0.07
 IRLM LOCK+LATCH          0.000033      0.05
 DB2 LATCH                0.000003      0.02
SYNCHRON. I/O             0.000082      0.12
 DATABASE I/O             0.000001      0.00
```

```
 LOG WRITE I/O            0.000081      0.12
OTHER READ I/O            0.000012      0.01
OTHER WRTE I/O            0.000001      0.00
SER.TASK SWTCH            0.000001      0.00
 UPDATE COMMIT            0.000000      0.00
 OPEN/CLOSE               0.000001      0.00
 SYSLGRNG REC             0.000000      0.00
 EXT/DEL/DEF              0.000000      0.00
 OTHER SERVICE            0.000000      0.00
ARC.LOG(QUIES)            0.000000      0.00
LOG READ                  0.000000      0.00
DRAIN LOCK                0.000000      0.00
CLAIM RELEASE             0.000000      0.00
PAGE LATCH                0.000060      0.02
NOTIFY MSGS               0.000000      0.00
GLOBAL CONTENTION         0.000164      0.31
COMMIT PH1 WRITE I/O      0.000000      0.00
ASYNCH CF REQUESTS        0.000044      0.19
TCP/IP LOB XML            0.000000      0.00
TOTAL CLASS 3             0.000399      0.71
```

When the transaction is using a type 4 connection, as is the case here, the work comes in over the network in to the DB2 DIST address space. As the class 1 time is the total time that the thread is active, it also includes the time that the transaction is spending in the application server and in the network. As the class 2 time records the time doing SQL-related activity, the time that we spend in the DDF address space not performing SQL activity is also included in the class 1 time, but it a small amount of time and processing.

Both class 1 and class 2 record both the elapsed and CPU time. For the CPU time, DB2 distinguishes between CPU time that is used on a general-purpose engine (CP CPU time) and time on a specialty engine (zIIP or zAAP, that is, SE CPU time). SE CPU time is not included in the CP CPU time.

The class 2 suspend time is the sum of all the CLASS 3 wait counters that DB2 tracks.

The class 3 suspensions section records the time and number of events that the transaction (on average, as this an accounting report, not an accounting trace) was suspended for each of the suspensions that DB2 tracks.

In this case, the transaction response time is excellent: 2.269 milliseconds.

When you use a type 4 connection, you can calculate *the time in DB2*:

Class 2 non-nested ET + (SP + UDF + trigger Class 1 ET) + non-nested (Class 1 CP CPU + Class 1 SE CPU - Class 2 CP CPU - Class 2 SE CPU)

In our example, this is 0.000853 + (0 + 0 +0) + (0.000201 + 0.000234 - 0.000178 - 0.000201) = 0.000909, or 0.909 milliseconds.

The *time outside DB2* can be calculated:

Total Class 1 ET - time in DB2 (that we previously calculated)

In our example this is 0.002269 - 0.000909 = 0.001360 or 1.36 milliseconds.

This time includes the time that the thread is performing work on the WebSphere Application Server, and the time that is spent in the network. When the thread is reused and does not cut an accounting record at commit, it also includes the time that the thread is idle (waiting for new work to arrive).

When using a type 4 connections, the accounting record also includes a distributed activity section, as shown in Example 8-28, from a DRDA requester with IP address 9.12.6.9 (our WebSphere Application Server).

Example 8-28 Accounting report - Distributed activity

```
---- DISTRIBUTED ACTIVITY --------------------------------------------------------------------------------------------
REQUESTER          : ::9.12.6.9          #COMMIT(1) RECEIVED:   86522    MESSAGES SENT    :      9.57    ROWS SENT    :    1.30
PRODUCT ID         : JDBC DRIVER         #ROLLBK(1) RECEIVED:       0    MESSAGES RECEIVED:      9.57    BLOCKS SENT  :    2.22
METHOD             : DRDA PROTOCOL       SQL RECEIVED      :     4.85    BYTES SENT       :   1237.17    #DDF ACCESSES:   86521
CONV.INITIATED     :               0.15                                  BYTES RECEIVED   :   1596.58    #RLUP THREADS:   86521
                                                                         #THREADS INDOUBT :         0

#COMMIT(2) RECEIVED:                N/A  TRANSACTIONS RECV. :     N/A    #PREPARE RECEIVED:       N/A    MSG.IN BUFFER:     N/A
#BCKOUT(2) RECEIVED:                N/A  #COMMIT(2) RES.SENT:     N/A    #LAST AGENT RECV.:       N/A    #FORGET SENT :     N/A
#COMMIT(2) PERFORM.:                N/A  #BACKOUT(2)RES.SENT:     N/A
                                         #BACKOUT(2)PERFORM.:     N/A
```

In an OMPE accounting report, most of the fields are averages (based on the number of occurrences), but some of the fields contain the total for all occurrences that are included in the report (the fields in bold). This section indicates how much DRDA traffic occurred in terms of the number of SQL statements, bytes, blocks, and messages. When using blocking that rows are put into blocks, which are then sent out in messages. As these are short running transactions with only a small amount of data being passed, there is little blocking activity.

When DB2 accounting trace class 7, 8, or 10 is active, DB2 also produces accounting information at the program or package level, as shown in Example 8-29.

Example 8-29 Accounting report - Package level information

```
LOCATION: DB0Z                     OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1M1)                PAGE: 1-9
      GROUP: DB0ZG                               ACCOUNTING REPORT - LONG                  REQUESTED FROM: ALL      21:29:00.00
     MEMBER: D0Z1                                                                                      TO: DATES    21:30:01.00
  SUBSYSTEM: D0Z1                                     ORDER: TRANSACT                       INTERVAL FROM: 08/13/12 21:29:00.37
DB2 VERSION: V10                                      SCOPE: MEMBER                                    TO: 08/13/12 21:30:00.99

TRANSACT: TraderClientApplication

SYSLN300            VALUE                SYSLN300                  TIMES    SYSLN300             AVERAGE TIME  AVG.EV    TIME/EVENT
------------------  ------------------   ------------------  ------------   ------------------   ------------  ------  ------------
TYPE                PACKAGE              ELAP-CL7 TIME-AVG       0.000744   LOCK/LATCH               0.000034    0.06      0.000593
                                         CP CPU TIME             0.000142    IRLM LOCK+LATCH         0.000032    0.04      0.000741
LOCATION            DB0Z                   AGENT                 0.000142    DB2 LATCH               0.000003    0.02      0.000177
COLLECTION ID       JDBCNOHDBAT            PAR.TASKS             0.000000   SYNCHRONOUS I/O          0.000082    0.12      0.000680
PROGRAM NAME        SYSLN300             SE CPU TIME             0.000157   OTHER READ I/O           0.000012    0.01      0.001411
                                         SUSPENSION-CL8          0.000353   OTHER WRITE I/O          0.000001    0.00      0.000482
ACTIVITY TYPE       NONNESTED              AGENT                 0.000353   SERV.TASK SWITCH         0.000001    0.00      0.006281
ACTIVITY NAME       'BLANK'                PAR.TASKS             0.000000   ARCH.LOG(QUIESCE)        0.000000    0.00           N/C
SCHEMA NAME         'BLANK'              NOT ACCOUNTED           0.000091   ARCHIVE LOG READ         0.000000    0.00           N/C
SUCC AUTH CHECK                  86521   AVG.DB2 ENTRY/EXIT          7.96   DRAIN LOCK               0.000000    0.00      0.000172
OCCURRENCES                      86521   DB2 ENTRY/EXIT            688782   CLAIM RELEASE            0.000000    0.00           N/C
NBR OF ALLOCATIONS               86521                                      PAGE LATCH               0.000060    0.02      0.002916
SQL STMT - AVERAGE                2.98   CP CPU SU                   8.11   NOTIFY MESSAGES          0.000000    0.00      0.000565
SQL STMT - TOTAL                257866     AGENT                     8.11   GLOBAL CONTENTION        0.000164    0.31      0.000535
NBR RLUP THREADS                 86521     PAR.TASKS                 0.00   TCP/IP LOB XML           0.000000    0.00           N/C
                                         SE CPU SU                   9.18   TOTAL CL8 SUSPENS.       0.000353    0.52      0.000685
```

As the Trader workload is using JDBC, all the work that is performed by the application runs under the standard JDBC packages, in this case, SYSLN300. (We bound the package into a special collection called JDBCNOHDBAT - JDBC_No_High_performance_DBAT, so the regular packages do not use **RELEASE(DEALLOCATE)**).

At the package level, DB2 also records the ET and CPU time (GCP and SE) that was spent in the package (Class 7 time), and a large subset of the Class 3 suspension counters is also available at the package level (Class 8 time). When Class 10 is active, there is additional information about the SQL, locking, and buffer pool activity, but we did not record this information during this test.

Package level information is not that useful when using JDBC, as all work runs under the same set of packages. In this case, you must rely on using the client strings to trigger correct segregation of the JDBC work.

When you use SQLJ, package level information can be helpful. In the SQLJ case, the SQL statements run as static SQL and each application binds its own set of packages, and allows for a more granular view at the package level of the application's database access pattern.

Response time reporting for RRS (type 2) connections

Although the accounting information for type 2 connections is 98% the same as when using a type 4 connection, there are a few differences that are worth mentioning.

Example 8-30 shows the accounting class 1, 2, and 3 information when using a type 2 (RRS) connection to access DB2 for z/OS. The application is identical to what we used before (TraderClientApplication); we only changed from a type 4 to type 2 connection in the data source. However, the number of users being simulated was different, so you should not be comparing the type 2 and the type 4 run, as the application profile is different.

Example 8-30 Accounting report - Class 1, 2, and 3 times for T2

```
LOCATION: DBOZ                   OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1M1)                 PAGE: 1-7
      GROUP: DBOZG                              ACCOUNTING REPORT - LONG                REQUESTED FROM: NOT SPECIFIED
     MEMBER: DOZ2                                                                                   TO: NOT SPECIFIED
  SUBSYSTEM: DOZ2                                    ORDER: TRANSACT                     INTERVAL FROM: 08/14/12 22:39:13.46
DB2 VERSION: V10                                     SCOPE: MEMBER                                  TO: 08/14/12 22:48:25.51

TRANSACT: TraderClientApplication

ELAPSED TIME DISTRIBUTION                                         CLASS 2 TIME DISTRIBUTION
----------------------------------------------------------------  ----------------------------------------------------------------
APPL   |======================> 45%                               CPU     |=> 2%
DB2    |=======> 15%                                              SECPU   |
SUSP   |====================> 40%                                 NOTACC  |============> 25%
                                                                  SUSP    |====================================> 73%

AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)   CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT    HIGHLIGHTS
------------  ----------  ----------  ----------   --------------------  ------------  --------    --------------------------
ELAPSED TIME    0.018236    0.010092         N/P   LOCK/LATCH(DB2+IRLM)      0.005214      0.70    #OCCURRENCES    :  1304797
 NONNESTED      0.018236    0.010092         N/A    IRLM LOCK+LATCH          0.001948      0.26    #ALLIEDS        :  1304797
 STORED PROC    0.000000    0.000000         N/A    DB2 LATCH                0.003266      0.44    #ALLIEDS DISTRIB:        0
 UDF            0.000000    0.000000         N/A   SYNCHRON. I/O             0.000001      0.00    #DBATS          :        0
 TRIGGER        0.000000    0.000000         N/A    DATABASE I/O             0.000000      0.00    #DBATS DISTRIB. :        0
                                                    LOG WRITE I/O            0.000001      0.00    #NO PROGRAM DATA:       57
CP CPU TIME     0.000243    0.000197         N/P   OTHER READ I/O            0.000000      0.00    #NORMAL TERMINAT:        0
 AGENT          0.000243    0.000197         N/A   OTHER WRTE I/O            0.000000      0.00    #DDFRRSAF ROLLUP:   130474
  NONNESTED     0.000243    0.000197         N/P   SER.TASK SWTCH            0.000817      0.11    #ABNORMAL TERMIN:       57
  STORED PRC    0.000000    0.000000         N/A    UPDATE COMMIT            0.000817      0.11    #CP/X PARALLEL. :        0
  UDF           0.000000    0.000000         N/A    OPEN/CLOSE               0.000000      0.00    #IO PARALLELISM :        0
  TRIGGER       0.000000    0.000000         N/A    SYSLGRNG REC             0.000000      0.00    #INCREMENT. BIND:        0
 PAR.TASKS      0.000000    0.000000         N/A    EXT/DEL/DEF              0.000000      0.00    #COMMITS        :  1304734
                                                    OTHER SERVICE            0.000000      0.00    #ROLLBACKS      :        0
 SECP CPU       0.000000         N/A         N/A   ARC.LOG(QUIES)            0.000000      0.00    #SVPT REQUESTS  :        0
                                                   LOG READ                  0.000000      0.00    #SVPT RELEASE   :        0
SE CPU TIME     0.000050    0.000041         N/A   DRAIN LOCK                0.000000      0.00    #SVPT ROLLBACK  :        0
 NONNESTED      0.000050    0.000041         N/A   CLAIM RELEASE             0.000000      0.00    MAX SQL CASC LVL:        0
 STORED PROC    0.000000    0.000000         N/A   PAGE LATCH                0.001033      0.08    UPDATE/COMMIT   :     0.21
 UDF            0.000000    0.000000         N/A   NOTIFY MSGS               0.000000      0.00    SYNCH I/O AVG.  : 0.000613
 TRIGGER        0.000000    0.000000         N/A   GLOBAL CONTENTION         0.000292      0.05
                                                   COMMIT PH1 WRITE I/O      0.000000      0.00
 PAR.TASKS      0.000000    0.000000         N/A   ASYNCH CF REQUESTS        0.000000      0.00
                                                   TCP/IP LOB XML            0.000000      0.00
SUSPEND TIME    0.000000    0.007358         N/A   TOTAL CLASS 3             0.007358      0.95
 AGENT               N/A    0.007358         N/A
 PAR.TASKS           N/A    0.000000         N/A
 STORED PROC    0.000000         N/A         N/A
```

```
 UDF            0.000000         N/A         N/A

NOT ACCOUNT.         N/A    0.002497         N/A
DB2 ENT/EXIT         N/A        0.00         N/A
EN/EX-STPROC         N/A        0.00         N/A
EN/EX-UDF            N/A        0.00         N/A
DCAPT.DESCR.         N/A         N/A         N/P
LOG EXTRACT.         N/A         N/A         N/P
```

Now we are looking at an RRS (T2) connection, as indicated by a non-zero number in the #ALLIEDS field (in the highlights section). #DDFRRSAF ROLLUP is non-zero, which indicates that during this run we used rollup accounting (ACCUMACC=10).

This is also confirmed by the non-zero value in the END USER THRESH field (accounting record that is written because the ACCUMACC value was reached for the ACCUMUID aggregation field) in the Normal Term section, as shown in Example 8-31.

Example 8-31 Accounting report - Normal Term

```
NORMAL TERM.      AVERAGE     TOTAL
---------------  --------  --------
NEW USER             0.00         0
DEALLOCATION         0.00         0
APPL.PROGR. END      0.00         0
RESIGNON             0.00         0
DBAT INACTIVE        0.00         0
TYPE2 INACTIVE       0.00         0
RRS COMMIT           0.00         0

END USER THRESH      0.10    130474
BLOCK STOR THR       0.00         0
STALENESS THR        0.00         0
```

To calculate the *time in DB2* for local applications, use the following formula:

Class 2 non-nested ET + (SP + UDF + trigger Class 1 ET)

In our example, this is 0.010092 + (0 + 0 +0) = 0.010092 or 10.092 milliseconds.

The *time outside DB2* can be calculated:

Total Class 1 ET - time in DB2 (that we previously calculated)

In our example, this is 0.018236 - 0.010092 = 0.08144 or 8.144 milliseconds.

When using a local attach such as RRS, the *CPU time spent in the application* can be calculated:

Non-nested (Class 1 CP CPU + Class 1 SE CPU) - non-nested (Class 2 CP CPU + Class 2 SE CPU)

In our case:

(0.000243 + 0.000050) - (0.000197 + 0.000041) = 0.000055 or 55 microseconds

Example 8-32 shows the accounting package level information (when accounting class 7 and 8 are active). Type 2 JDBC and type 4 JDBC use the same packages (SYSLN300, in our case).

Example 8-32 Accounting report -Package information

```
SYSLN300           VALUE              SYSLN300                   TIMES    SYSLN300            AVERAGE TIME  AVG.EV    TIME/EVENT
------------------ ------------------ ------------------  ------------    ------------------  ------------  ------  ------------
TYPE               PACKAGE            ELAP-CL7 TIME-AVG       0.008323    LOCK/LATCH              0.004466    0.60      0.007495
                                      CP CPU TIME             0.000137     IRLM LOCK+LATCH        0.001541    0.20      0.007521
LOCATION           DB0Z                 AGENT                 0.000137     DB2 LATCH              0.002926    0.39      0.007482
COLLECTION ID      JDBCNOHDBAT          PAR.TASKS             0.000000    SYNCHRONOUS I/O         0.000001    0.00      0.000597
PROGRAM NAME       SYSLN300           SE CPU TIME             0.000024    OTHER READ I/O          0.000000    0.00           N/C
                                      SUSPENSION-CL8          0.006609    OTHER WRITE I/O         0.000000    0.00      0.010107
ACTIVITY TYPE      NONNESTED            AGENT                 0.006609    SERV.TASK SWITCH        0.000817    0.11      0.007200
ACTIVITY NAME      'BLANK'              PAR.TASKS             0.000000    ARCH.LOG(QUIESCE)       0.000000    0.00           N/C
SCHEMA NAME        'BLANK'            NOT ACCOUNTED           0.001552    ARCHIVE LOG READ        0.000000    0.00           N/C
SUCC AUTH CHECK                    0  AVG.DB2 ENTRY/EXIT           N/P    DRAIN LOCK              0.000000    0.00           N/C
OCCURRENCES                  1304740  DB2 ENTRY/EXIT               N/P    CLAIM RELEASE           0.000000    0.00           N/C
NBR OF ALLOCATIONS           1304733                                      PAGE LATCH              0.001033    0.08      0.013138
SQL STMT - AVERAGE              3.81  CP CPU SU                   7.95    NOTIFY MESSAGES         0.000000    0.00      0.002867
SQL STMT - TOTAL             4967798    AGENT                     7.95    GLOBAL CONTENTION       0.000292    0.00      4.050997
NBR RLUP THREADS             1304733    PAR.TASKS                 0.00    TCP/IP LOB XML          0.000000    0.00           N/C
                                      SE CPU SU                   1.41    TOTAL CL8 SUSPENS.      0.006609    0.79      0.008372
```

8.4.6 Monitoring threads and connections by using profiles

After we implement and activate the active thread profile monitoring for the DayTrader-EE6 application (for more information, see 4.3.17, “Using DB2 profiles” on page 180), we run the application to see whether the number of seven active threads was exceeded. During workload testing, we notice the messages that are shown in Figure 8-32 several times during workload execution.

```
DSNT772I 1 -D0Z1 DSNLQDIS A MONITOR PROFILE WARNING 352
CONDITION OCCURRED
31802 TIME(S) 2
IN PROFILE ID=1 3
WITH PROFILE FILTERING SCOPE=CLIENT_APPLNAME 4
WITH REASON=00E30505 5
DSNT772I 1 -D0Z1 DSNLQDIS A MONITOR PROFILE WARNING 177
CONDITION OCCURRED
12306 TIME(S) 2
IN PROFILE ID=1 3
WITH PROFILE FILTERING SCOPE=CLIENT_APPLNAME 4
WITH REASON=00E30505 5
```

Figure 8-32 Message DSNT772I

Where:

1. We scheduled the workload to run on both data sharing members, so we observed DSNT772I messages that are issued by both data sharing members.
2. *nnnnn* `TIME(S)` shows the number of times that the warning threshold was exceeded since the last DSNT772I message was issued. As observed in the syslog, DB2 issued the DSNT772I message every 5 minutes. Using the interval duration of 300 seconds, you can use these values to calculate the number of active threads that are needed per second:
   - D0Z1 = 31802 times
     31802 / 300 = 106.66 + 7 = 113.66 active threads per second
   - D072 = 12306 times
     12306 / 300 = 41.02 + 7 = 48.02 active threads per second

3. `PROFILE ID` uniquely identifies the ID of the monitoring profile.
4. The filtering scope was for `CLIENT_APPLNAME`.
5. A warning occurred because the number of concurrent active threads exceeded the warning setting for the **MONITOR THREADS** keyword in the monitor profile of `PROFILE ID 1` for the filtering scopes `CLIENT_APPLNAME`.

Our DB2 setup collects statistics trace class 4 to include IFCID 402 information about DB2 profile warning and exception conditions. Using the IFCID 402 information, we create the OMPE record trace report that is shown in Figure 8-33 to obtain more information.

```
 LOCATION:          OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1M1)       PAGE: 1-1
       GROUP: DB0ZG                    RECORD TRACE - LONG                    REQUESTED FROM: NOT SPECIFIED
      MEMB                                                                                TO: NOT SPECIFIED
   SUBSYST                                                                       ACTUAL FROM: 08/13/12 21:08:0
 DB2 VERSION: V                                                                    PAGE DATE: 08/13/12
OPRIMAUTH CONNECT   INSTANCE      END_USER      WS_NAME                        TRANSACT
 ORIGAUTH CORRNAME  CONNTYPE      RECORD TIME    DESTNO ACE IFC  DESCRIPTION   DATA
 PLANNAME CORRNMBR                TCB CPU TIME              ID
 -------- -------- ----------- ----------------- ------ --- --- -------------- ---------------------------------
N/P      CA037CD25FEA N/P             N/P                          N/P
N/P      'BLANK'     21:08:00.15927885 132857   1 402 SYSTEM PROFILE NETWORKID:  DOZ1      LUNAME:  DOZ1 LUWSEQ:
N/P                  N/P                            MONITORING STA

|-------------------------------------------------------------------------------------------------------------------
|PROFILE ID                               :         1   (THR = THREAD, EXC = EXCEPTIOTSH=THRESHOLD)
|ACCUMULATED COUNTER OF ...
|THR EXC TSH EXCEEDED                     :         0   THR QUEUED/SUSP WHEN EXC TSH WAS EXCEEDED    : 0
|REQUEST FAILED WHEN THR EXC TSH WAS EXCEEDED :     0   THR WARNING TSH BEING EXCEEDED               :1019250
|CONNECTION EXC TSH BEING EXCEEDED        :         0   CONNECTION WARN TSH BEING EXCEEDED           : 0
|IDLE THR EXC TSH BEING EXCEEDED          :         0   IDLE THR WARN TSH BEING EXCEEDED             : 0
|-------------------------------------------------------------------------------------------------------------------
```

Figure 8-33 IFCID 402 record trace report

Additional information

For more information about the topics that are covered in this section, see the setup in 4.3.19, “Configure thread monitoring for the DayTrader-EE6 application” on page 187. For more information about managing and implementing DB2 profile monitoring, see Chapter 45, “Using profiles to monitor and optimize performance”, in *DB2 10 for z/OS, Managing Performance*, SC19-2978.

8.5 Using the performance database

Appendix D, “IBM OMEGAMON XE for DB2 performance database” on page 527 explains how to implement and populate the OMPE performance database (PDB) tables with DB2 statistics and accounting trace information. This section provides an example of an SQL table UDF that is used to encapsulate the PDB query logic from the SQL user. The UDF receives the following input parameters:

- **`clientApplicationInformation`**

  For the DayTrader application, the **`clientApplicationInformation`** data source custom property was set to the value of `TraderClientApplication`.

- Connection type
  - RRS for JDBC type 2 connections
  - DRDA for JDBC type 4 connections

Using the UDF is simple and straightforward. For example, to query the aggregated PDB accounting tables for JDBC type 2 connections that are collected on 14th August 2012, run the query that is shown in Example 8-33.

Example 8-33 .Using the SQL table UDF to query JDBC type 2 accounting information

```
select * from
 table(accounting('TraderClientApplication','RRS')) a
 where substr("DateTime",1,10) = '2012-08-14'
 order by "DateTime"     ;
```

8.5.1 Querying aggregated JDBC type 2 accounting information

Using the UDF, we ran the query that is shown in Figure 8-34 on page 436 to obtain interval aggregated information about our JDBC type 2 workload execution. For each interval, the query returns aggregations on elapsed time, total and DB2 related processor and zIIP usage, number of commits, SQL DML, locks, get page requests, and row statistics on insert, update, and delete activities.

```
 select
      "DateTime"
    , "Elapsed"
    , "TotCPU"
    , "TotzIIP"
    , DB2CPU
    , "DB2zIIP"
    , "Commit"
    , SQL
    , "Locks"
    , "RowsFetched"
    , "RowsInserted"
    , "RowsUpdated"
    , "RowsDeleted"
    , "GetPage"
   from
   table(accounting('TraderClientApplication','RRS')) a
where substr("DateTime",1,10) = '2012-08-14'
   order by "DateTime"     ;
---------+---------+---------+---------+---------+---------+---------+-------
DateTime          Elapsed  TotCPU  TotzIIP   DB2CPU  DB2zIIP  Commit     SQL
---------+---------+---------+---------+---------+---------+---------+-------
2012-08-14-22.39    11.96    4.98     3.40     3.40     2.25    8970   39290
2012-08-14-22.41  2417.46   38.09     7.03    30.28     5.64  123299  164779
2012-08-14-22.42  3375.88   54.40     9.43    43.95     7.69  187120  248216
2012-08-14-22.43  3227.19   55.64     9.14    45.16     7.53  191309  252611
2012-08-14-22.44  3570.88   56.14     9.02    45.67     7.42  192869  255372
2012-08-14-22.45  3299.26   56.33     9.17    45.87     7.58  193129  254434
2012-08-14-22.46  3281.19   56.79     9.02    46.27     7.45  196599  259235
2012-08-14-22.47  3242.99   56.69     8.92    46.15     7.36  196339  259321
2012-08-14-22.48   250.33    4.40      .66     3.59      .54   15100   19722
-+---------+---------+---------+---------+---------
  Locks  Rows    Rows     Rows     Rows    GetPage
         Fetched Inserted Updated  Deleted


-+---------+---------+---------+---------+---------
  92588  22366     6971    9945        0    171534
 814550  62241     4646   22235     1152    757692
1223042  41686     7165   30923     1877   1149004
1248040  45798     7359   31095     1801   1184893
1261375  48539     7371   31672     1887   1203983
1258972  50600     7364   31069     1794   1197565
1282236  56418     7477   31693     1839   1221505
1281040  56666     7425   31869     1925   1218127
  97848  19435      562    2391      136     92090
DSNE610I NUMBER OF ROWS DISPLAYED IS 9
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

Figure 8-34 PDB query JDBC type 2 aggregated accounting data

8.5.2 Querying aggregated JDBC type 4 accounting information

Using the UDF, we ran the query that is shown in Figure 8-35 to obtain interval aggregated information about our JDBC type 4 workload execution. For each interval, the query returns aggregations of elapsed time, total and DB2 related processor and zIIP usage, number of commits, SQL DML, locks, get page requests, and row statistics on insert, update, and delete activities.

```
select
"DateTime" , "Elapsed" , "TotCPU" , "TotzIIP" , DB2CPU , "DB2zIIP" , "Commit"
, SQL , "Locks" , "RowsFetched" , "RowsInserted" , "RowsUpdated" ,
"RowsDeleted" , "GetPage"
   from table(accounting('TraderClientApplication','DRDA')) a
  where substr("DateTime",1,10) = '2012-08-17' order by "DateTime"   ;
---------+---------+---------+---------+---------+---------+-------
DateTime         Elapsed TotCPU TotzIIP DB2CPU DB2zIIP Commit  SQL

---------+---------+---------+---------+---------+---------+-------
2012-08-17-22.59  288.46  23.56   12.14  20.75   10.59  55235 13185
2012-08-17-22.59  291.66  27.38   14.90  23.69   12.76  64120 26333
2012-08-17-23.00  433.34  49.51   26.76  43.21   23.11 123548 27323
2012-08-17-23.00  423.71  48.04   24.41  42.37   21.32 114517 24805
2012-08-17-23.01  309.15  47.00   23.71  41.51   20.69 112304 24123
2012-08-17-23.01  375.32  51.78   28.08  45.12   24.23 131529 28668
2012-08-17-23.02  796.84  94.92   52.00  81.67   44.18 265297 58104
2012-08-17-23.02  141.11  20.66   10.31  18.35    9.06  46233 10033
2012-08-17-23.03 1101.42 124.68   68.45 106.66   57.73 360722 79375
2012-08-17-23.04  660.42  79.00   43.25  67.71   36.55 226388 50443
2012-08-17-23.04  125.28  27.55   13.98  23.90   11.95  76366 17038
2012-08-17-23.05  637.05  78.89   43.05  67.65   36.41 228661 49864
2012-08-17-23.05  161.45  28.07   14.36  24.35   12.28  78467 16906
--+---------+---------+---------+---------+-------
 Locks   Rows    Rows     Rows    Rows    GetPage
        Fetched Inserted Updated Deleted
--+---------+---------+---------+---------+-------
  367481  72629    2067    10083     541   818516
  452411  91083    6059    17562     579   818516
  813130 159809    4691    20264    1185  1469024
  746300 148772    4415    18197    1061  1469024
  730570 145980    4188    17805    1035  1523288
  867468 170043    4969    21239    1256  1523288
 1737859 343606   10085    42965    2515  1957605
  301810  60545    1729     7414     443  1957605
 2361915 467835   13735    58712    3479  2269742
 1482703 290534    8826    37282    2172  1910393
  497295  99019    2956    12587     740  1910393
 1495054 295893    8654    36847    2182  1926461
  508469 100866    2983    12480     707  1926461
```

Figure 8-35 PDB query JDBC type 4 aggregated accounting data

8.5.3 Using RTS to identify DB2 tables that are involved in DML operations

The query result in Figure 8-35 provides information about the number of rows fetched, inserted, updated, and deleted without telling you which tables these activities were performed on. All that we know is that the DayTrader-EE6 application accesses tables belonging to the DBTR8074 database.

Currently, we have no answers to the following questions:

- Which tables were involved in SQL DML update, delete, insert, and fetch operations?
- How many rows were inserted, updated, or deleted for each table?
- If you extrapolate the SQL DML information, how large are the tables going to be in one month, six months, and one year?

Answering these questions is important to identify tables that need extra care when it comes to planning disk capacity, identifying tables for partitioning, and identifying tables that need extra care when it comes to planning Runstats, Reorg, and Backup. For example, you might use the RTS information to identify tables that are volatile or in need of extra reorg utility executions. If you identify tables that start small in size and are going to be huge, you might want to provide for table partitioning and you might want to talk with application developers to determine whether data partitioning secondary indexes (DPSI) are a good choice.

Let us focus on the query output that is shown in Figure 8-35 on page 437, in which we obtain information about the rows that are inserted, updated, and deleted during the workload execution that is performed on 17 August. Before and right after workload execution, we saved the RTS tables using the process that is described in 4.3.24, "DB2 real time statistics" on page 198. We then ran the query that is shown in Figure 8-36 on page 439 to determine the number of rows that were inserted, updated, and deleted for each of the tables that are accessed during workload execution.

```
WITH
Q1 AS
(  SELECT
   DBNAME,NAME,PARTITION,
   NACTIVE,NPAGES,REORGINSERTS,REORGDELETES,REORGUPDATES,
   REORGMASSDELETE,TOTALROWS,SNAPSHOTTS
   FROM TABLESPACESTATS
   WHERE SNAPSHOTTS = '2012-08-17-22.57.57.673670'
     AND DBNAME = 'DBTR8074'
   ORDER BY DBNAME, NAME, SNAPSHOTTS),
Q2 AS
(  SELECT
   DBNAME,NAME,PARTITION,
   NACTIVE,NPAGES,REORGINSERTS,REORGDELETES,REORGUPDATES,
   REORGMASSDELETE,TOTALROWS,SNAPSHOTTS
   FROM TABLESPACESTATS
   WHERE SNAPSHOTTS = '2012-08-17-23.08.49.191718'
     AND DBNAME = 'DBTR8074'
   ORDER BY DBNAME, NAME, SNAPSHOTTS)
SELECT
SUBSTR(Q1.DBNAME,1,8)                        AS DBNAME,
SUBSTR(Q1.NAME  ,1,8)                        AS NAME,
Q1.PARTITION,
Q2.TOTALROWS          - Q1.TOTALROWS         AS #ROWS,
Q2.REORGINSERTS       - Q1.REORGINSERTS      AS INSERTS  ,
Q2.REORGDELETES       - Q1.REORGDELETES      AS DELETES  ,
Q2.REORGUPDATES       - Q1.REORGUPDATES      AS UPDATES  ,
Q2.REORGMASSDELETE    - Q1.REORGMASSDELETE   AS MASSDELETE
FROM Q1,Q2
WHERE
(Q1.DBNAME,Q1.NAME,Q1.PARTITION) = (Q2.DBNAME,Q2.NAME,Q2.PARTITION)
---------+---------+---------+---------+---------+---------+---------+---
DBNAME    NAME      PARTITION   #ROWS INSERTS DELETES  UPDATES MASSDELETE
---------+---------+---------+---------+---------+---------+---------+---
DBTR8074  TSACCEJB          0   11489   11489       0   133129          1
DBTR8074  TSACPREJ          0   11489   11489       0    21603          1
DBTR8074  TSHLDEJB          0    2476   24037   21561    21561          1
DBTR8074  TSKEYGEN          0       3       3       0       83          0
DBTR8074  TSORDEJB          0   45598   45598       0   158305          1
DBTR8074  TSQUOEJB          0    1000    1000       0    45580          1
DSNE610I NUMBER OF ROWS DISPLAYED IS 6
```

Figure 8-36 Using RTS to determine workload-related table changes

The query output that is shown in Figure 8-36 shows table spaces, their SQL insert, update, and delete activities, and the number of rows that are stored in each table space upon workload completion. Because we store only one table per table space, we can relate the name of the table that was involved in the SQL DML operation to the table space that the table belongs to.

Querying performance indicator information

Obtaining the total sum for elapsed and CPU times, the total number of SQL, commits, locks, and getpage requests does not always provide an indication of good or bad application performance. To assess your application test results, you must perform an application target performance comparison. This process compares target performance indicators with corresponding performance indicators of your workload execution. To assist you in performing a target performance comparison, the SQL table UDF that we introduce in 8.5, “Using the performance database” on page 434 provides the following key performance indicators (KPIs):

- AVG-Time: Average elapsed time per commit
- AVG-CPU: Average CPU usage per commit
- Time/SQL: Average elapsed time per SQL
- CPU/SQL: Average CPU usage per SQL
- AVG-SQL: Average number of SQL per commit
- LOCK/Tran: Average number of lock requests per commit
- LOCK/SQL: Average number of locks per SQL
- GETP/Tran: Average number of getpage requests per commit
- GETP/SQL: Average number of getpage requests per SQL

In our workload scenario, we use these performance indicators to compare the DayTrader-EE6 JDBC type 2 with the DayTrader-EE6 JDBC type 4 workload execution.

JDBC type 2

The performance indicators of the DayTrader-EE6 JDBC type 2 are shown in Figure 8-37.

```
 select
   "DateTime"
  ,"AVG-Time"
  ,"AVG-CPU"
  ,"Time/SQL"
  ,"CPU/SQL"
  ,"AVG-SQL"
  ,"LOCK/Tran"
  ,"LOCK/SQL"
  ,"GETP/Tran"
  ,"GETP/SQL"
   from table(accounting('TraderClientApplication','RRS')) a
   where substr("DateTime",1,10) = '2012-08-14'
   order by "DateTime"     ;
---------+---------+---------+---------+---------+---------+-
DateTime          AVG-Time AVG-CPU  Time/SQL CPU/SQL  AVG-SQL
---------+---------+---------+---------+---------+---------+-
2012-08-14-22.39   .001333 .000555  .000304 .000126 4.380156
2012-08-14-22.41   .019606 .000308  .014670 .000231 1.336417
2012-08-14-22.42   .018041 .000290  .013600 .000219 1.326507
2012-08-14-22.43   .016868 .000290  .012775 .000220 1.320434
2012-08-14-22.44   .018514 .000291  .013983 .000219 1.324069
2012-08-14-22.45   .017083 .000291  .012967 .000221 1.317430
2012-08-14-22.46   .016689 .000288  .012657 .000219 1.318597
2012-08-14-22.47   .016517 .000288  .012505 .000218 1.320781
2012-08-14-22.48   .016578 .000291  .012692 .000223 1.306092

--------+---------+---------+---------+
LOCK/Tran LOCK/SQL GETP/Tran GETP/SQL
--------+---------+---------+---------+
10.321962 2.356528 19.123076 4.365843
 6.606298 4.943287  6.145159 4.598231
 6.536137 4.927329  6.140466 4.629048
 6.523686 4.940560  6.193608 4.690583
 6.540060 4.939362  6.242491 4.714624
 6.518813 4.948128  6.200855 4.706780
 6.522088 4.946230  6.213180 4.711960
 6.524633 4.939977  6.204202 4.697371
 6.480000 4.961362  6.098675 4.669404
```

Figure 8-37 Performance indicators JDBC type 2

JDBC type 4

The performance indicators of the DayTrader-EE6 JDBC type 2 are shown in Figure 8-38.

```
 select
   "DateTime"
  ,"AVG-Time"
  ,"AVG-CPU"
  ,"Time/SQL"
  ,"CPU/SQL"
  ,"AVG-SQL"
  ,"LOCK/Tran"
  ,"LOCK/SQL"
  ,"GETP/Tran"
  ,"GETP/SQL"
   from
   table(accounting('TraderClientApplication','DRDA')) a
   where substr("DateTime",1,10) = '2012-08-17'
   order by "DateTime"     ;
---------+---------+---------+---------+---------+---------+
DateTime         AVG-Time AVG-CPU Time/SQL CPU/SQL  AVG-SQL
---------+---------+---------+---------+---------+---------+
2012-08-17-22.58  .002664 .000604  .001339 .000303 1.989921
2012-08-17-22.59  .005222 .000426  .021877 .001786  .238707
2012-08-17-22.59  .004548 .000427  .011075 .001039  .410683
2012-08-17-23.00  .003507 .000400  .015859 .001812  .221152
2012-08-17-23.00  .003699 .000419  .017081 .001936  .216605
2012-08-17-23.01  .002752 .000418  .012815 .001948  .214800
2012-08-17-23.01  .002853 .000393  .013091 .001806  .217959
2012-08-17-23.02  .003003 .000357  .013714 .001633  .219014
2012-08-17-23.02  .003052 .000446  .014064 .002059  .217009
2012-08-17-23.03  .003053 .000345  .013876 .001570  .220044
2012-08-17-23.04  .002917 .000348  .013092 .001566  .222816
2012-08-17-23.04  .001640 .000360  .007352 .001616  .223109
2012-08-17-23.05  .002786 .000345  .012775 .001582  .218069
2012-08-17-23.05  .002057 .000357  .009549 .001660  .215453
2012-08-17-23.06  .003343 .000345  .015042 .001555  .222288
---------+---------+---------+---------+
LOCK/Tran  LOCK/SQL GETP/Tran  GETP/SQL
---------+---------+---------+---------+
 9.476892  4.762445 17.073746   8.580111
 6.653046 27.871141 14.818792  62.079332
 7.055692 17.180382 12.765377  31.083279
 6.581490 29.759909 11.890309  53.765106
 6.516936 30.086676 12.827999  59.222898
 6.505289 30.285204 13.563969  63.146706
 6.595260 30.259104 11.581385  53.135482
 6.550616 29.909455  7.378918  33.691398
 6.528021 30.081730 42.342158 195.116615
 6.547743 29.756409  6.292219  28.595174
 6.549388 29.393632  8.438578  37.872311
 6.511994 29.187404 25.016276 112.125425
 6.538299 29.982632  8.424965  38.634305
 6.480036 30.076245 24.551225 113.951319
 6.552779 29.478718  6.300677  28.344596
```

Figure 8-38 Performance indicators JDBC type 4

Conclusion

When we compare the JDBC type 2 with the JDBC type 4 performance indicators. We notice a ratio of less than 1 SQL per commit for the JDBC type 4 workload. This is caused by the data source custom property `AUTOCOMMIT=ON`, which causes the JDBC driver to issue an SQL commit for each SQL statement. When we compare the JDBC type 2 with the JDBC type 4 performance indicators, we notice higher resource usage for CPU, number of locks, and number of get page requests per SQL.

8.6 Monitoring from the z/OS side with RMF

Now, we look at the monitoring that you can perform on the z/OS side. You can use an online monitor to look at the current system performance, but here we focus on how to look at the historical performance information by using RMF monitor postprocessor reports. There are many different reports that you can look at, but here we focus on the workload activity report.

In 8.2.2, "Using client information strings to classify work in WLM and RMF reporting" on page 369, we set up our WLM service classes and, for monitoring purposes, reporting classes:

- RTRADE as the reporting class for the Trader application inside the WebSphere Application Server
- RTRADE0Z as the reporting class for the DDF enclaves that run the DB2 work when the Trader application is using a type 4 connection.

Example 8-34 shows a sample JCL that you can use to run the RMF post processor. The first step sorts the data, which is especially important when you use multiple input data sets, for example, when combining data from multiple systems. The second step generates the reports. In this case, we use the following JCL:

```
SYSRPTS(WLMGL(SCLASS,RCLASS,SCPER,RCPER,SYSNAM(SC64)))
```

This JCL creates a sysplex-wide *workload activity report*. We look at only one of the systems, SC64 (SYSNAM(SC64)). The report has information about the different service classes (SCLASS), report classes (RCLASS), and within the service class, the individual service class periods (SCPER), and the individual reporting class periods (RCPER) within each reporting class.

For more information about the different post-processor reporting option, see Chapter 17. "Long-term reporting with the Postprocessor z/OS RMF", in *z/OS V1R13 Resource Measurement Facility (RMF) User's Guide*, SC33-7990.

Example 8-34 JCL that is used to create the postprocessor workload activity report

```
//BAT4RMF  JOB (999,POK),'BART JOB',CLASS=A,MSGCLASS=T,
// NOTIFY=&SYSUID,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63
//RMFSORT  EXEC PGM=SORT
//SORTIN   DD DISP=SHR,DSN=DB2SMF.WASRB.SC64.T4.SMFRMF
//SORTOUT  DD   DISP=(NEW,PASS),UNIT=(SYSDA,5),SPACE=(CYL,(800,800))
//SORTWK01 DD   DISP=(NEW,DELETE),UNIT=SYSDA,SPACE=(CYL,(100,200))
//SORTWK02 DD   DISP=(NEW,DELETE),UNIT=SYSDA,SPACE=(CYL,(100,200))
//SORTWK03 DD   DISP=(NEW,DELETE),UNIT=SYSDA,SPACE=(CYL,(100,200))
//SORTWK04 DD   DISP=(NEW,DELETE),UNIT=SYSDA,SPACE=(CYL,(100,200))
//SORTWK05 DD   DISP=(NEW,DELETE),UNIT=SYSDA,SPACE=(CYL,(100,200))
//SYSPRINT DD   SYSOUT=*
```

```
//SYSOUT   DD   SYSOUT=*
//SYSIN    DD   *
  SORT FIELDS=(11,4,CH,A,7,4,CH,A),EQUALS
  MODS E15=(ERBPPE15,36000,,N),E35=(ERBPPE35,3000,,N)
//SYSOUT   DD   SYSOUT=*
//SYSPRINT DD   SYSOUT=*
//RMFPOST  EXEC PGM=ERBRMFPP
//*
//*
//MFPINPUT DD   DISP=(OLD,DELETE),DSN=*.RMFSORT.SORTOUT
//MFPMSGDS DD   SYSOUT=*
//SYSOUT   DD   SYSOUT=*
//PPXSRPTS DD   SYSOUT=*
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD *
  SYSRPTS(WLMGL(SCLASS,RCLASS,SCPER,RCPER,SYSNAM(SC64)))
  SYSOUT(T)
/*
```

8.6.1 Workload activity when using a type 4 connection

When the applications running in WebSphere Application Server on z/OS use type 4 connectivity to DB2 for z/OS, the work consists of two pieces. The first part is the work that is done by application inside the application server, and the second part is the DB2 work. As the work comes into DB2 through DDF, it has its own enclaves (inside the DDF address space) to represent and account for the DB2 part of the work.

Workload activity that is reported for the DB2 work in the RTRADE0Z reporting class

Example 8-35 shows the *workload activity report class period* report for the RTRADE0Z reporting class for period 1. The DDFONL service class that is used by this reporting class uses two periods. The period 2 part is shown in Example 8-36 on page 445. As an example, we picked a one minute interval that stated at 21.29.01. (We reduced the RMF interval to 1 minute to have more granularity in the reports.)

In this one minute interval, DB2 completed 43968 transactions in period 1, or 732.8 transactions per second by running, on average, 6.91 threads (enclaves) in parallel.

Example 8-35 Workload activity - reporting class RTRADE0Z period 1

```
                                                  W O R K L O A D   A C T I V I T Y
                                                                                                                       PAGE  72
       z/OS V1R13            SYSPLEX SANDBOX           DATE 08/13/2012          INTERVAL 01.00.000   MODE = GOAL
                         CONVERTED TO z/OS V1R13 RMF   TIME 21.29.01

                                        POLICY ACTIVATION DATE/TIME 08/10/2012 19.52.00

------------------------------------------------------------------------------------------------------------ REPORT CLASS PERIODS

REPORT BY: POLICY=WLMPOL                          REPORT CLASS=RTRADEOZ                           PERIOD=1
                                                  HOMOGENEOUS: GOAL DERIVED FROM SERVICE CLASS DDFONL

-TRANSACTIONS-  TRANS-TIME HHH.MM.SS.TTT  --DASD I/O--  ---SERVICE---  SERVICE TIME  ---APPL %---  --PROMOTED--  ----STORAGE----
AVG       6.91  ACTUAL                 9  SSCHRT   3.9  IOC         0  CPU   24.780  CP     20.60  BLK    0.000  AVG        0.00
MPL       6.91  EXECUTION              9  RESP     0.0  CPU     1447K  SRB    0.000  AAPCP   0.00  ENQ    0.000  TOTAL      0.00
ENDED    43968  QUEUED                 0  CONN     0.0  MSO         0  RCT    0.000  IIPCP   0.07  CRM    0.000  SHARED     0.00
END/S   732.80  R/S AFFIN              0  DISC     0.0  SRB         0  IIT    0.000                LCK    0.020
#SWAPS       0  INELIGIBLE             0  Q+PEND   0.0  TOT     1447K  HST    0.000  AAP     0.00  SUP    0.000  -PAGE-IN RATES-
EXCTD        0  CONVERSION             0  IOSQ     0.0  /SEC    24117  AAP    0.000  IIP    20.69                SINGLE      0.0
AVG ENC   6.91  STD DEV               29                               IIP   12.417                              BLOCK       0.0
REM ENC   0.00                                           ABSRPTN  3491                                             SHARED      0.0
```

```
MS ENC    0.00                                     TRX SERV 3491                                       HSP        0.0

GOAL: RESPONSE TIME 000.00.01.000 FOR  90%

        RESPONSE TIME EX   PERF  AVG   --EXEC USING%--  -------------- EXEC DELAYS % -----------  -USING%-  --- DELAY % ---    %
SYSTEM      ACTUAL%    VEL% INDX ADRSP  CPU AAP IIP I/O  TOT CPU IIP                              CRY CNT  UNK IDL CRY CNT  QUI

SC64           100      4.9  0.5   6.4  1.5 0.0 1.2 0.0   52  52 0.7                              0.0 0.0   45 0.0 0.0 0.0  0.0

                                           ----------RESPONSE TIME DISTRIBUTION----------
   ----TIME----    --NUMBER OF TRANSACTIONS--    -------PERCENT-------  0    10   20   30   40   50   60   70   80   90   100
   HH.MM.SS.TTT    CUM TOTAL        IN BUCKET    CUM TOTAL   IN BUCKET  |....|....|....|....|....|....|....|....|....|....|
<  00.00.00.500        43939            43939          100         100  >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<= 00.00.00.600        43941                2          100         0.0  >
<= 00.00.00.700        43955               14          100         0.0  >
<= 00.00.00.800        43957                2          100         0.0  >
<= 00.00.00.900        43966                9          100         0.0  >
<= 00.00.01.000        43968                2          100         0.0  >
<= 00.00.01.100        43968                0          100         0.0  >
<= 00.00.01.200        43968                0          100         0.0  >
<= 00.00.01.300        43968                0          100         0.0  >
<= 00.00.01.400        43968                0          100         0.0  >
<= 00.00.01.500        43968                0          100         0.0  >
<= 00.00.02.000        43968                0          100         0.0  >
<= 00.00.04.000        43968                0          100         0.0  >
>  00.00.04.000        43968                0          100         0.0  >
```

The number of CPU seconds that were needed to do these 43968 transactions is 24.78. The SERVICE CPU time includes the CPU time on zAAP and zIIP (12.417). So, we used 12.363 seconds on a general CP in this case.

The workload activity report also gives an indication, in percent of an engine, that this reporting or service class (period) used. This information can be found in the APPL% column. In this case, it is 20.6% of a general engine, or one-fifth of an engine. It also used 20.69% of a zIIP engine. In this example, the zIIP time is *not* included in the CP%.

> **Tip:** The service time CPU includes the CPU seconds that are used on a zIIP or zAAP engine. The CP percentage in the APPL% column does *not* include the zIIP and zAAP processing.

A workload activity reporting or service class *period* report also indicates whether the WLM goal for the service class period is met. If the performance index (PI) is less than one, which is the case here, the goal is exceeded. When the PI =1, you meet the goal, and when the PI > 1, WLM could not achieve the goal that is specified in the policy.

After the performance index in a period report, there is also a *response time distribution* section. In this run, 43939 transactions out of 43968 completed in less than or equal to 0.5 seconds, so you are exceeding the response time goal of 000.00.01.000 second for 90% of the transactions.

Example 8-36 shows the period 2 information for the RTRADE0Z reporting class in this one minute interval. Only 24 transactions finished in period two.

Example 8-36 Workload activity - reporting class RTRADE0Z period 2

```
                                                 W O R K L O A D   A C T I V I T Y
                                                                                                                      PAGE  73
      z/OS V1R13            SYSPLEX SANDBOX            DATE 08/13/2012          INTERVAL 01.00.000   MODE = GOAL
                        CONVERTED TO z/OS V1R13 RMF    TIME 21.29.01

                                          POLICY ACTIVATION DATE/TIME 08/10/2012 19.52.00

REPORT BY: POLICY=WLMPOL                            REPORT CLASS=RTRADEOZ                                PERIOD=2
                                                    HOMOGENEOUS: GOAL DERIVED FROM SERVICE CLASS DDFONL

-TRANSACTIONS-  TRANS-TIME HHH.MM.SS.TTT  --DASD I/O--  ---SERVICE---   SERVICE TIME  ---APPL %---  --PROMOTED--  ----STORAGE----
AVG       0.03  ACTUAL               156  SSCHRT   0.1  IOC         0   CPU    0.014  CP      0.02  BLK    0.000  AVG        0.00
MPL       0.03  EXECUTION            156  RESP     0.0  CPU       804   SRB    0.000  AAPCP   0.00  ENQ    0.000  TOTAL      0.00
ENDED       24  QUEUED                 0  CONN     0.0  MSO         0   RCT    0.000  IIPCP   0.00  CRM    0.000  SHARED     0.00
```

```
END/S     0.40  R/S AFFIN            0  DISC     0.0  SRB        0  IIT    0.000              LCK    0.000
#SWAPS       0  INELIGIBLE           0  Q+PEND   0.0  TOT      804  HST    0.000  AAP    0.00  SUP    0.000  -PAGE-IN RATES-
EXCTD        0  CONVERSION           0  IOSQ     0.0  /SEC      13  AAP    0.000  IIP    0.00                SINGLE      0.0
AVG ENC   0.03  STD DEV            255                             IIP    0.003                              BLOCK       0.0
REM ENC   0.00                                        ABSRPTN   500                                          SHARED      0.0
MS ENC    0.00                                        TRX SERV  500                                          HSP         0.0

GOAL: EXECUTION VELOCITY 40.0%     VELOCITY MIGRATION:   I/O MGMT   0.0%     INIT MGMT  0.0%

         RESPONSE TIME EX   PERF  AVG   --EXEC USING%--  -------------- EXEC DELAYS % -----------  -USING%-  --- DELAY % ---    %
SYSTEM                VEL% INDX ADRSP  CPU AAP IIP I/O  TOT CPU                                   CRY CNT  UNK IDL CRY CNT  QUI

SC64        --N/A--     0.0  0.0   0.0  0.0 0.0 0.0 0.0   50  50                                   0.0 0.0   50 0.0 0.0 0.0  0.0

                                    ----------RESPONSE TIME DISTRIBUTIONS----------
SYSTEM: SC64 ----INTERVAL: 00.01.00.000 ---MRT CHANGES:  0 ---
   ----TIME----  -NUMBER OF TRANSACTIONS-  ------PERCENT------
   HH.MM.SS.TTT  CUM TOTAL      IN BUCKET  CUM TOTAL IN BUCKET
<  00.00.00.030          9              9       37.5      37.5
<= 00.00.00.036         14              5       58.3      20.8
<= 00.00.00.042         16              2       66.7       8.3
<= 00.00.00.048         17              1       70.8       4.2
<= 00.00.00.054         17              0       70.8       0.0
<= 00.00.00.061         18              1       75.0       4.2
<= 00.00.00.067         18              0       75.0       0.0
<= 00.00.00.073         18              0       75.0       0.0
<= 00.00.00.079         18              0       75.0       0.0
<= 00.00.00.085         18              0       75.0       0.0
<= 00.00.00.091         18              0       75.0       0.0
<= 00.00.00.122         18              0       75.0       0.0
<= 00.00.00.244         19              1       79.2       4.2
>  00.00.00.244         24              5        100      20.8
```

After looking at the different periods, look at all the transactions in the reporting class, with both periods combined, as shown in Example 8-37. The total number of transactions is 43992 (or 43968 in period 1 + 24 in period 2). As almost all the transactions that are completed in period 1, so the report for the entire report class is similar to the report of period 1. However, the report class report does not have a performance index or response time distribution information. That information is available only at the period report level.

Example 8-37 Workload activity - reporting class RTRADE0Z total

```
                                              W O R K L O A D   A C T I V I T Y
                                                                                                                        PAGE  74
      z/OS V1R13             SYSPLEX SANDBOX           DATE 08/13/2012          INTERVAL 01.00.000   MODE = GOAL
                         CONVERTED TO z/OS V1R13 RMF   TIME 21.29.01

                                       POLICY ACTIVATION DATE/TIME 08/10/2012 19.52.00

------------------------------------------------------------------------------------------------------------ REPORT CLASS(ES)

REPORT BY: POLICY=WLMPOL                          REPORT CLASS=RTRADE0Z
                                                  DESCRIPTION =DDF  DAY TRADER

-TRANSACTIONS-  TRANS-TIME HHH.MM.SS.TTT  --DASD I/O--  ---SERVICE---  SERVICE TIME  ---APPL %---  --PROMOTED--  ----STORAGE----
AVG       6.93  ACTUAL                 9  SSCHRT   4.0  IOC         0  CPU   24.794  CP     20.62  BLK    0.000  AVG         0.00
MPL       6.93  EXECUTION              9  RESP     0.0  CPU     1448K  SRB    0.000  AAPCP   0.00  ENQ    0.000  TOTAL       0.00
ENDED    43992  QUEUED                 0  CONN     0.0  MSO         0  RCT    0.000  IIPCP   0.07  CRM    0.000  SHARED      0.00
END/S   733.20  R/S AFFIN              0  DISC     0.0  SRB         0  IIT    0.000                LCK    0.020
#SWAPS       0  INELIGIBLE             0  Q+PEND   0.0  TOT     1448K  HST    0.000  AAP     0.00  SUP    0.000  -PAGE-IN RATES-
EXCTD        0  CONVERSION             0  IOSQ     0.0  /SEC    24130  AAP    0.000  IIP    20.70                SINGLE      0.0
AVG ENC   6.93  STD DEV               29                               IIP   12.420                              BLOCK       0.0
REM ENC   0.00                                          ABSRPTN  3480                                            SHARED      0.0
MS ENC    0.00                                          TRX SERV 3480                                            HSP         0.0
```

Workload activity that is reported for the WebSphere Application Server work in the RTRADE reporting class

The section “Workload activity that is reported for the DB2 work in the RTRADE0Z reporting class” on page 444 described the workload activity report for the DDF side of the work, that is, the part of the transaction where it runs SQL requests in DB2 for z/OS.

When you use a type 4 connection, the part of the transaction that is running inside the WebSphere Application Server is represented by a different enclave. It is classified by the Subsystem Type CB in the WLM classification panels. (WebSphere Application Server uses Subsystem Type CB for enclave work.)

In Figure 8-13 on page 375, we use the Transaction Class to select the service class (WASONL) and reporting class (RTRADE) that applies to the trader application. The workload activity report for SC64 (that runs our WebSphere Application Server) for period 1 of the RTRADE reporting class is shown in Example 8-38.

Example 8-38 Workload activity - reporting class RTRADE period 1

```
                                                 W O R K L O A D   A C T I V I T Y
                                                                                                                       PAGE  70
       z/OS V1R13             SYSPLEX SANDBOX           DATE 08/13/2012          INTERVAL 01.00.000   MODE = GOAL
                          CONVERTED TO z/OS V1R13 RMF   TIME 21.29.01

                                        POLICY ACTIVATION DATE/TIME 08/10/2012 19.52.00

------------------------------------------------------------------------------------------------------------- REPORT CLASS PERIODS

REPORT BY: POLICY=WLMPOL                         REPORT CLASS=RTRADE                                       PERIOD=1
                                                 HOMOGENEOUS: GOAL DERIVED FROM SERVICE CLASS WASONL

-TRANSACTIONS-  TRANS-TIME HHH.MM.SS.TTT  --DASD I/O--  ---SERVICE---   SERVICE TIME  ---APPL %---  --PROMOTED--  ----STORAGE----
AVG      33.02  ACTUAL                50  SSCHRT   0.0  IOC         0   CPU   60.113  CP     41.19  BLK    0.000  AVG        0.00
MPL      33.02  EXECUTION             49  RESP     0.0  CPU     3510K   SRB    0.000  AAPCP  32.28  ENQ    0.000  TOTAL      0.00
ENDED    39697  QUEUED                 0  CONN     0.0  MSO         0   RCT    0.000  IIPCP   0.00  CRM    0.000  SHARED     0.00
END/S   661.62  R/S AFFIN              0  DISC     0.0  SRB         0   IIT    0.000                LCK    0.000
#SWAPS       0  INELIGIBLE             0  Q+PEND   0.0  TOT     3510K   HST    0.000  AAP    58.99  SUP    0.000  -PAGE-IN RATES-
EXCTD        0  CONVERSION             0  IOSQ     0.0  /SEC    58504   AAP   35.396  IIP     0.00                SINGLE      0.0
AVG ENC  33.02  STD DEV               68                                IIP    0.000                              BLOCK       0.0
REM ENC   0.00                                          ABSRPTN  1772                                             SHARED      0.0
MS ENC    0.00                                          TRX SERV 1772                                             HSP         0.0

          RESP  -------------------------------- STATE SAMPLES BREAKDOWN (%) -------------------------------  ------STATE------
SUB    P  TIME  --ACTIVE-- READY IDLE  -----------------------------WAITING FOR-----------------------------  SWITCHED SAMPL(%)
TYPE       (%)   SUB  APPL                                                                                    LOCAL SYSPL REMOT
CB    BTE  0.0   0.0   0.0   0.0  0.0                                                                           0.0   0.0   0.0
CB    EXE  101  99.3   0.7   0.0  0.0                                                                           0.0   0.0   0.0

GOAL: RESPONSE TIME 000.00.01.000 FOR  90%

         RESPONSE TIME EX   PERF  AVG   --EXEC USING%--  -------------- EXEC DELAYS % -----------  -USING%-  --- DELAY % ---    %
SYSTEM      ACTUAL%    VEL% INDX ADRSP  CPU AAP IIP I/O  TOT AAP CPU  Q                             CRY CNT  UNK IDL CRY CNT  QUI
                                                                     MPL
SC64           100     13.5  0.5  31.0  0.3 0.9 0.0 0.0  7.8 7.5 0.2 0.1                            0.0 0.0   91 0.0 0.0 0.0  0.0

                                       ----------RESPONSE TIME DISTRIBUTION----------
   ----TIME----    --NUMBER OF TRANSACTIONS--    -------PERCENT-------  0    10   20   30   40   50   60   70   80   90   100
   HH.MM.SS.TTT    CUM TOTAL        IN BUCKET    CUM TOTAL   IN BUCKET  |....|....|....|....|....|....|....|....|....|....|
<  00.00.00.500        39595            39595          100         100  >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<= 00.00.00.600        39637               42          100         0.1  >
<= 00.00.00.700        39652               15          100         0.0  >
<= 00.00.00.800        39666               14          100         0.0  >
<= 00.00.00.900        39678               12          100         0.0  >
<= 00.00.01.000        39684                6          100         0.0  >
<= 00.00.01.100        39687                3          100         0.0  >
<= 00.00.01.200        39689                2          100         0.0  >
<= 00.00.01.300        39689                0          100         0.0  >
<= 00.00.01.400        39689                0          100         0.0  >
<= 00.00.01.500        39689                0          100         0.0  >
<= 00.00.02.000        39697                8          100         0.0  >
<= 00.00.04.000        39697                0          100         0.0  >
>  00.00.04.000        39697                0          100         0.0  >

REPORT BY: POLICY=WLMPOL                         REPORT CLASS=RTRADE                                       PERIOD=2

----------        ALL DATA ZERO        ----------
```

It contains information similar to the workload activity report of the DDF work that was described in “Workload activity that is reported for the DB2 work in the RTRADE0Z reporting class” on page 444. However, consider the State Samples Breakdown section. State samples are collected on an ongoing basis and reported as a percentage of average transaction response time and have two phases:

- BTE phase: The begin-to-end phase applies to requests that are handled by the application control region (ACR).
- EXE phase: The execution phase applies to requests that are handled by the servant regions.

The performance is good; the sampled state did not show any delays where the work is waiting.

For more information about WLM Delay Monitoring, go to the following website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.zseries.doc%2Fae%2Frprf_wlmdm.html

8.6.2 Workload activity when using a type 2 connection

When the applications that are running in WebSphere Application Server on z/OS use type 2 connectivity to DB2 for z/OS, the work is reported as one piece (compared to two pieces in the case of type 4). The enclave now includes both the work in WebSphere Application Server and the work in DB2 (when running SQL statements).

Workload activity that is reported for the RTRADE reporting class

To show some additional functionality of the RMF post-processor, we use a *duration report*. A duration report does not report on individual RMF intervals, but allows multiple intervals to be grouped.

> **Note:** There are some caveats when using duration reports. For more information, go to:
>
> http://publib.boulder.ibm.com/infocenter/zos/v1r13/topic/com.ibm.zos.r13.erbb200/dintv.htm

Example 8-39 shows the **SYSIN DD** statements that are used to create a workload activity report for the 9 minute time frame 22:39 - 22:48.

Example 8-39 Duration report SYSIN

```
//SYSIN    DD *
  SUMMARY(TOT,INT)
  DATE(08142012,08142012)
  RTOD(2239,2248)
  DINTV(0009)
  REPORTS(CPU)
   SYSRPTS(WLMGL(SCLASS,RCLASS,SCPER,RCPER))
  SYSOUT(T)
/*
```

Example 8-40 shows the *workload activity report class period* report for the RTRADE reporting class for period 1. The report looks similar to the ones we looked at before. When using a type 2 connection, all the work that is done by the WebSphere Application Server application, including the SQL activity, is done under the enclave that is created by the WebSphere Application Server control region. As a result, the appl % CP is much higher than in the type 4 case, which is expected, as the DB2 work is also included now. The state samples breakdown shows a small percentage of delay for TYP8, which is the J2C Resource manager delay when you call a J2C connector to resource managers, such as DB2.

Example 8-40 Workload activity - reporting class RTRADE period 1

```
                                                W O R K L O A D   A C T I V I T Y
                                                                                                                  PAGE  72
      z/OS V1R13           SYSPLEX SANDBOX          START 08/14/2012-22.39.00 INTERVAL 000.08.59   MODE = GOAL
                      CONVERTED TO z/OS V1R13 RMF     END   08/14/2012-22.47.59

                                         POLICY ACTIVATION DATE/TIME 08/10/2012 19.52.00

------------------------------------------------------------------------------------------------------------ REPORT CLASS PERIODS

REPORT BY: POLICY=WLMPOL                         REPORT CLASS=RTRADE                                   PERIOD=1
                                                 HOMOGENEOUS: GOAL DERIVED FROM SERVICE CLASS WASONL

-TRANSACTIONS-  TRANS-TIME HHH.MM.SS.TTT  --DASD I/O--  ---SERVICE---   SERVICE TIME  ---APPL %---  --PROMOTED--  ----STORAGE----
AVG      27.23  ACTUAL                39  SSCHRT   0.2  IOC          0   CPU  554.785  CP      80.93  BLK    0.000  AVG        0.00
MPL      27.23  EXECUTION             37  RESP     0.2  CPU     32396K  SRB    0.000  AAPCP   0.62  ENQ    0.000  TOTAL      0.00
ENDED   392496  QUEUED                 1  CONN     0.1  MSO          0  RCT    0.000  IIPCP   0.00  CRM    0.000  SHARED     0.00
END/S   726.85  R/S AFFIN              0  DISC     0.0  SRB          0  IIT    0.000                LCK    0.008
#SWAPS       0  INELIGIBLE             0  Q+PEND   0.1  TOT     32396K  HST    0.000  AAP     21.81  SUP    0.000  -PAGE-IN RATES-
EXCTD        0  CONVERSION             0  IOSQ     0.0  /SEC     59994  AAP  117.756  IIP      0.00                SINGLE       0.0
AVG ENC  27.23  STD DEV               37                                 IIP    0.000                              BLOCK        0.0
REM ENC   0.00                                          ABSRPTN  2203                                              SHARED       0.0
MS ENC    0.00                                          TRX SERV 2203                                              HSP          0.0

          RESP  -------------------------------- STATE SAMPLES BREAKDOWN (%) -------------------------------  ------STATE------
SUB    P  TIME  --ACTIVE-- READY IDLE  -----------------------------WAITING FOR----------------------------  SWITCHED SAMPL(%)
TYPE       (%)   SUB  APPL             TYP8                                                                  LOCAL SYSPL REMOT
CB    BTE  0.0   0.0   0.0   0.0  0.0   0.0                                                                    0.0   0.0   0.0
CB    EXE 95.6  99.0   0.5   0.0  0.0   0.4                                                                    0.0   0.0   0.0

CB  : TYP8 - CONNECTOR

GOAL: RESPONSE TIME 000.00.01.000 FOR  90%

         RESPONSE TIME EX   PERF  AVG   --EXEC USING%--  -------------- EXEC DELAYS % -----------  -USING%-  --- DELAY % ---    %
SYSTEM      ACTUAL%    VEL% INDX ADRSP  CPU AAP IIP I/O  TOT CPU AAP   Q                            CRY CNT  UNK IDL CRY CNT  QUI
                                                                      MPL
SC64           100      2.2  0.5  28.4  1.5 0.1 0.0 0.0   70  69 0.6 0.5                            0.0 0.0   28 0.0 0.0 0.0  0.0

                                        ----------RESPONSE TIME DISTRIBUTION----------
   ----TIME----    --NUMBER OF TRANSACTIONS--    -------PERCENT-------  0    10   20   30   40   50   60   70   80   90   100
   HH.MM.SS.TTT    CUM TOTAL        IN BUCKET    CUM TOTAL   IN BUCKET  |....|....|....|....|....|....|....|....|....|....|
<  00.00.00.500         392K             392K          100         100  >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<= 00.00.00.600         392K               34          100         0.0  >
<= 00.00.00.700         392K                9          100         0.0  >
<= 00.00.00.800         392K                7          100         0.0  >
<= 00.00.00.900         392K                6          100         0.0  >
<= 00.00.01.000         392K                4          100         0.0  >
<= 00.00.01.100         392K                7          100         0.0  >
<= 00.00.01.200         392K               12          100         0.0  >
<= 00.00.01.300         392K                5          100         0.0  >
<= 00.00.01.400         392K               24          100         0.0  >
<= 00.00.01.500         392K               18          100         0.0  >
<= 00.00.02.000         392K               12          100         0.0  >
<= 00.00.04.000         392K                7          100         0.0  >
>  00.00.04.000         392K                0          100         0.0  >
```

Example 8-41 shows the similar *workload activity report class period* report for the RTRADE reporting class for period 2. Only 48 transactions ended in this period.

Example 8-41 Workload activity - reporting class RTRADE period 2

```
                                                W O R K L O A D   A C T I V I T Y
                                                                                                                  PAGE  73
      z/OS V1R13           SYSPLEX SANDBOX          START 08/14/2012-22.39.00 INTERVAL 000.08.59   MODE = GOAL
                      CONVERTED TO z/OS V1R13 RMF     END   08/14/2012-22.47.59
```

```
                                        POLICY ACTIVATION DATE/TIME 08/10/2012 19.52.00

REPORT BY: POLICY=WLMPOL                          REPORT CLASS=RTRADE                               PERIOD=2
                                                  HOMOGENEOUS: GOAL DERIVED FROM SERVICE CLASS WASONL

-TRANSACTIONS-  TRANS-TIME HHH.MM.SS.TTT  --DASD I/O--  ---SERVICE---   SERVICE TIME  ---APPL %---  --PROMOTED--  ----STORAGE----
AVG       0.05  ACTUAL                633  SSCHRT   0.1  IOC          0  CPU    5.884  CP      0.35  BLK    0.000  AVG        0.00
MPL       0.05  EXECUTION             631  RESP     0.2  CPU     343611  SRB    0.000  AAPCP   0.05  ENQ    0.000  TOTAL      0.00
ENDED       48  QUEUED                  2  CONN     0.2  MSO          0  RCT    0.000  IIPCP   0.00  CRM    0.000  SHARED     0.00
END/S     0.09  R/S AFFIN               0  DISC     0.0  SRB          0  IIT    0.000                LCK    0.012
#SWAPS       0  INELIGIBLE              0  Q+PEND   0.1  TOT     343611  HST    0.000  AAP     0.74  SUP    0.000  -PAGE-IN RATES-
EXCTD        0  CONVERSION              0  IOSQ     0.0  /SEC       636  AAP    3.982  IIP     0.00                SINGLE      0.0
AVG ENC   0.05  STD DEV             2.233                                IIP    0.000                              BLOCK       0.0
REM ENC   0.00                                         ABSRPTN    13K                                              SHARED      0.0
MS ENC    0.00                                         TRX SERV   13K                                              HSP         0.0

GOAL: EXECUTION VELOCITY 40.0%     VELOCITY MIGRATION:   I/O MGMT  31.2%     INIT MGMT 31.2%

         RESPONSE TIME EX   PERF  AVG   --EXEC USING%--  -------------- EXEC DELAYS % -----------  -USING%-  --- DELAY % ---    %
SYSTEM                VEL% INDX ADRSP  CPU AAP IIP I/O  TOT AAP CPU                                CRY CNT  UNK IDL CRY CNT  QUI

SC64       --N/A--    31.2  1.3   0.0  7.3  15 0.0 0.0   49  46 2.8                                0.0 0.0   29 0.0 0.0 0.0  0.0

                                        ----------RESPONSE TIME DISTRIBUTIONS----------
SYSTEM: SC64 ----INTERVAL: 00.08.59.998 ---MRT CHANGES:  0 ---
   ----TIME----  -NUMBER OF TRANSACTIONS-  ------PERCENT------
   HH.MM.SS.TTT  CUM TOTAL      IN BUCKET  CUM TOTAL IN BUCKET
<  00.00.01.193         45             45       93.8      93.8
<= 00.00.01.432         45              0       93.8       0.0
<= 00.00.01.670         45              0       93.8       0.0
<= 00.00.01.909         45              0       93.8       0.0
<= 00.00.02.148         45              0       93.8       0.0
<= 00.00.02.387         45              0       93.8       0.0
<= 00.00.02.625         45              0       93.8       0.0
<= 00.00.02.864         45              0       93.8       0.0
<= 00.00.03.103         45              0       93.8       0.0
<= 00.00.03.341         45              0       93.8       0.0
<= 00.00.03.580         45              0       93.8       0.0
<= 00.00.04.774         45              0       93.8       0.0
<= 00.00.09.548         47              2       97.9       4.2
>  00.00.09.548         48              1        100       2.1
```

Example 8-42 shows the *workload activity* report for the RTRADE (looking at the overall DRDA activity of 392544 transactions).

Example 8-42 Workload activity - reporting class for trade (DRDA)

```
                                             W O R K L O A D   A C T I V I T Y
                                                                                                                        PAGE  74
      z/OS V1R13             SYSPLEX SANDBOX           START 08/14/2012-22.39.00 INTERVAL 000.08.59   MODE = GOAL
                        CONVERTED TO z/OS V1R13 RMF     END   08/14/2012-22.47.59

                                        POLICY ACTIVATION DATE/TIME 08/10/2012 19.52.00

------------------------------------------------------------------------------------------------------------ REPORT CLASS(ES)

REPORT BY: POLICY=WLMPOL                          REPORT CLASS=RTRADE
                                                  DESCRIPTION =report class for trade (drda)

-TRANSACTIONS-  TRANS-TIME HHH.MM.SS.TTT  --DASD I/O--  ---SERVICE---   SERVICE TIME  ---APPL %---  --PROMOTED--  ----STORAGE----
AVG      27.28  ACTUAL                 39  SSCHRT   0.2  IOC          0  CPU  560.670  CP     81.28  BLK    0.000  AVG        0.00
MPL      27.28  EXECUTION              37  RESP     0.2  CPU     32740K  SRB    0.000  AAPCP   0.67  ENQ    0.000  TOTAL      0.00
ENDED   392544  QUEUED                  1  CONN     0.1  MSO          0  RCT    0.000  IIPCP   0.00  CRM    0.000  SHARED     0.00
END/S   726.94  R/S AFFIN               0  DISC     0.0  SRB          0  IIT    0.000                LCK    0.020
#SWAPS       0  INELIGIBLE              0  Q+PEND   0.1  TOT     32740K  HST    0.000  AAP    22.54  SUP    0.000  -PAGE-IN RATES-
EXCTD        0  CONVERSION              0  IOSQ     0.0  /SEC     60630  AAP  121.738  IIP     0.00                SINGLE      0.0
AVG ENC  27.28  STD DEV                45                                IIP    0.000                              BLOCK       0.0
REM ENC   0.00                                         ABSRPTN  2223                                               SHARED      0.0
MS ENC    0.00                                         TRX SERV 2223                                               HSP         0.0
```

9

Error handling and problem determination

When you run enterprise applications in a production environment, you must understand how the system behaves under failure conditions, what is needed to capture the information, and how to determine and document the error conditions.

This chapter covers the following topics:

- Error handling
- Correlating DB2 and WebSphere Application Server information
- Common tools for problem determination
- Typical problem scenario: Deadlock

9.1 Error handling

Java programs use try / catch constructs for exception processing. This section introduces typical JDBC methods for database and JDBC error handling.

9.1.1 Basic error message

When a JDBC or SQLJ program has an error in the driver or the database, an object of type SQLException is passed to each catch clause. Then, you can use the methods of the java.sql.SQLException class to obtain the error information:

- **`getErrorCode()`**: Returns the `SQLCODE`.
- **`getNextException()`**: Returns the next Exception object in the exception chain.
- **`getSQLState()`**: Returns the `SQLSTATE`. `SQLSTATE` is a five-character string, which makes it easier for applications to check the error category. The structure of the `SQLSTATE` values is the same for all IBM relational database products.

The IBM Data Server Driver for JDBC and SQLJ does not throw an exception for warning messages, but it accumulates warnings when SQL statements return positive SQLCODEs, and when SQL statements return a zero SQLCODEs with a non-zero `SQLSTATE`. You can use methods from the `java.sql.SQLWarning` class to handle them:

- **`getWarnings()`**: Returns the `SQLCODE`.
- **`getNextWarning()`**: Returns the next `SQLWARN` in the chain.
- **`getSQLState()`**: Returns the `SQLSTATE`.

The object of the `SQLException` or `SQLWarning` class can call the following methods under the `java.lang.Throwable` class. They provide additional information.

- **`getMessage()`**: Returns the description of the error or warning.
- **`printStackTrace()`**: Prints the current exception or throwable and its backtrace to a standard error stream.

Example 9-1 shows how to print a warning, `SQLCODE`, error message, and stack trace.

Example 9-1 Example of processing an SQLWarning and SQLError

```
String url = "jdbc:db2://d0zg.itso.ibm.com:39000/DB0Z:" +
      "retrieveMessagesFromServerOnGetMessage=true;";
Connection con;
Statement stmt1;
ResultSet rs1;
int retCode;
try
{
  Class.forName("com.ibm.db2.jcc.DB2Driver");
  con = DriverManager.getConnection(url, "user", "pw");
  stmt1 = con.createStatement();

  retCode = stmt1.executeUpdate("UPDATE DSN81010.ACT SET ACTDESC = 'TEST' WHERE" +
      " ACTNO = 4321");
  SQLWarning sqlwarn = stmt1.getWarnings();
  System.out.println ("Warning description: " + sqlwarn.getMessage());
  System.out.println ("SQLSTATE: " + sqlwarn.getSQLState());
```

```
  System.out.println ("Warning code: " + sqlwarn.getErrorCode());

  rs1 = stmt1.executeQuery("SELECT ACTDESC FROM DSN81010.ACT " +
      " WHERE ACTNO = 4321A");
  while (rs1.next()) {
  system.out.println("Active Description Is "+rs1.getString(1));
  }
  rs1.close();

  con.commit();
  con.close();
}

catch(SQLException qex)
{
   System.err.println ("SQLException information");
   System.err.println ("Error msg: " + qex.getMessage());
   System.err.println ("SQLSTATE: " + qex.getSQLState());
   System.err.println ("Error code: " + qex.getErrorCode());
   qex.printStackTrace();
}
```

Example 9-2 lists the output with the warning messages, error messages, and stack trace.

Example 9-2 The output of warning, error, and stack trace

```
Warning description: ROW NOT FOUND FOR FETCH, UPDATE, OR DELETE, OR THE RESULT OF
A QUERY IS AN EMPTY TABLE. SQLCODE=100, SQLSTATE=02000, DRIVER=3.58.104
SQLSTATE: 02000
Warning code: 100
SQLException information
Error msg: 4321A IS AN INVALID NUMERIC CONSTANT. SQLCODE=-103, SQLSTATE=42604,
DRIVER=3.58.104
SQLSTATE: 42604
Error code: -103
com.ibm.db2.jcc.am.uo: 4321A IS AN INVALID NUMERIC CONSTANT. SQLCODE=-103,
SQLSTATE=42604, DRIVER=3.58.104
   at com.ibm.db2.jcc.am.ed.a(ed.java:676)
   at com.ibm.db2.jcc.am.ed.a(ed.java:60)
   at com.ibm.db2.jcc.am.ed.a(ed.java:127)
   at com.ibm.db2.jcc.am.wm.c(wm.java:2517)
.....
   at com.ibm.db2.jcc.am.wm.a(wm.java:645)
   at com.ibm.db2.jcc.am.wm.executeQuery(wm.java:629)
   at WAS_DB2.BasicError.main(BasicError.java:45)
```

9.1.2 SQLCA formatting

The general `SQLException` and `SQLWarning` classes do not provide an interface to retrieve the DB2 SQLCA, which is the DB2 data structure that contains detailed information about the running of an SQL statement. Suppose that you receive an `SQLException` that says that you tried to insert a null value into a NOT NULL column; you can get information about which column had the problem through the SQLCA.

The IBM Data Server Driver for JDBC and SQLJ provides the com.ibm.db2.jcc.DB2Diagnosable class, which extends the SQLException class. If the JDBC driver detects an error, DB2Diagnosable gives you the same information as the standard SQLException class. However, if the database server detects the error, DB2Diagnosable adds the following methods, which give you additional information about the error:

- **printTrace**(): Prints diagnostic information.
- **getThrowable()**: Returns a java.lang.Throwable object that caused the SQLException, or null, if no such object exists.
- **getSqlca()**: Returns an DB2Sqlca object with the following information:
  - An SQL error code
  - The SQLERRMC values
  - The SQLERRP value
  - The SQLERRD values
  - The SQLWARN values
  - The SQLSTATE

The meaning of each field in SQLCA depends on the specific error. The most interesting part of the SQLCA is a string that is called SQLERRM, which contains several error tokens, which are separated by the character 0xFF. The **DB2Sqlca.getSqlErrmcTokens()** method tokenizes this string for you.

For more information about what the individual error tokens mean for a given SQLCODE, see *DB2 10 for z/OS, DB2 codes*, found at:

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=%2Fcom.ibm.db2z10.doc.codes%2Fsrc%2Fcodes%2Fdb2z_codes.htm

Look up the error text in the description of the SQLCODE. The tokens appear sequentially in the SQLERRM in the order that they appear in the message text.

Here are the basic steps to format SQLCA:

1. If the SQLException is an instance of DB2Diagnosable, cast the object to a DB2Diagnosable object.
2. Optional: Run the **DB2Diagnosable.printTrace** method to write all SQLException information to a java.io.PrintWriter object.
3. Run the **DB2Diagnosable.getThrowable** method to determine whether an underlying java.lang.Throwable caused the SQLException.
4. Run the **DB2Diagnosable.getSqlca** method to retrieve the DB2Sqlca object.
5. Run the **DB2Sqlca.getSqlCode** method to retrieve an SQL error code value.
6. Run the **DB2Sqlca.getSqlState** method to retrieve the SQLSTATE value.
7. Run the **DB2Sqlca.getSqlErrmc** method to retrieve a string that contains all SQLERRMC values, or run the **DB2Sqlca.getSqlErrmcTokens** method to retrieve the SQLERRMC values in an array.
8. Run the **DB2Sqlca.getSqlErrp** method to retrieve the SQLERRP value.
9. Run the **DB2Sqlca.getSqlErrd** method to retrieve the SQLERRD values in an array.
10. Run the **DB2Sqlca.getSqlWarn** method to retrieve the SQLWARN values in an array.

The code in Example 9-3 demonstrates how to obtain SQLCA from an `SQLException`.

Example 9-3 Processing SQLException and format SQLCA

```
......
try {
......
   rs1 = stmt1.executeQuery("SELECT ACTDESC FROM DSN81010.ACT " +
      " WHERE ACTNO = 4321A");
   while (rs1.next()) {
   system.out.println("Active Description Is "+rs1.getString(1));
   }
   rs1.close();
   con.commit();
   con.close();
}
catch(SQLException sqle)
{
   if (sqle instanceof DB2Diagnosable) {
      com.ibm.db2.jcc.DB2Diagnosable diagnosable =
        (com.ibm.db2.jcc.DB2Diagnosable)sqle;

     DB2Sqlca sqlca = diagnosable.getSqlca(); // Get DB2Sqlca object
      if (sqlca != null) {         // Check that DB2Sqlca is not null
         int sqlCode = sqlca.getSqlCode(); // Get the SQL error code
         String sqlErrmc = sqlca.getSqlErrmc(); // Get the entire SQLERRMC
         String[] sqlErrmcTokens = sqlca.getSqlErrmcTokens();
                           // Retrieve individual SQLERRMC tokens
         String sqlErrp = sqlca.getSqlErrp();  // Get the SQLERRP
         int[] sqlErrd = sqlca.getSqlErrd();  // Get SQLERRD fields
         char[] sqlWarn = sqlca.getSqlWarn();  // Get SQLWARN fields
         String sqlState = sqlca.getSqlState();// Get SQLSTATE

         System.err.println ("--------------- SQLCA ---------------");
         System.err.println ("Error code: " + sqlCode);
         System.err.println ("SQLSTATE: " + sqlState);
         System.err.println ("SQLERRMC: " + sqlErrmc);
         if (sqlErrmcTokens != null) {
            for (int i=0; i< sqlErrmcTokens.length; i++) {
            System.err.println (" token " + i + ": " + sqlErrmcTokens[i]);
            }
         }
      System.err.println ( "SQLERRP: " + sqlErrp );
      System.err.println (
      "SQLERRD(1): " + sqlErrd[0] + "\n" +
      "SQLERRD(2): " + sqlErrd[1] + "\n" +
      "SQLERRD(3): " + sqlErrd[2] + "\n" +
      "SQLERRD(4): " + sqlErrd[3] + "\n" +
      "SQLERRD(5): " + sqlErrd[4] + "\n" +
      "SQLERRD(6): " + sqlErrd[5] );
      System.err.println (
         "SQLWARN1: " + sqlWarn[0] + "\n" +
         "SQLWARN2: " + sqlWarn[1] + "\n" +
         "SQLWARN3: " + sqlWarn[2] + "\n" +
         "SQLWARN4: " + sqlWarn[3] + "\n" +
         "SQLWARN5: " + sqlWarn[4] + "\n" +
```

```
            "SQLWARN6: " + sqlWarn[5] + "\n" +
            "SQLWARN7: " + sqlWarn[6] + "\n" +
            "SQLWARN8: " + sqlWarn[7] + "\n" +
            "SQLWARN9: " + sqlWarn[8] + "\n" +
            "SQLWARNA: " + sqlWarn[9] );
        }
    }
    else {
            System.err.println ("SQLException information");
            System.err.println ("Error msg: " + sqle.getMessage());
            System.err.println ("SQLSTATE: " + sqle.getSQLState());
            System.err.println ("Error code: " + sqle.getErrorCode());
    }
}
```

The output of the program is shown in Example 9-4. `SQLCODE -103` is returned because of an invalid numeric constant "4321A" (as the number contains the letter "A"). `SQLERRP` contains the DB2 module name that issues the `SQLCODE`. `SQLERRD(5)` indicates the starting position of the invalid constant in the SQL statement, byte 48 in this case. This can be helpful for syntax checking of complicated SQL statements.

Example 9-4 The output of JDBC program SQLCA formatting

```
--------------- SQLCA ---------------
Error code: -103
SQLSTATE: 42604
SQLERRMC: 4321A
  token 0: 4321A
SQLERRP: DSNHLEX
SQLERRD(1): 12
SQLERRD(2): 0
SQLERRD(3): 0
SQLERRD(4): -1
SQLERRD(5): 48
SQLERRD(6): 803
SQLWARN1:
SQLWARN2:
......
SQLWARNA:
```

9.1.3 Multiple SQL error handling

In some scenarios, DB2 can return multiple SQLCODEs in succession, so your application should handle each of these SQLCODEs correctly.

For example, `deferPrepares`, a property of the IBM Data Server Driver for JDBC and SQLJ, allows the **PREPARE** and **EXECUTE** statements to be sent across the network as a single message, to reduce network processing:

```
....
PreparedStatement pst = con.prepareStatement("SELECT C1 FROM T1");
pst.executeQuery();
....
```

If table T1 does not exist, SQLCODEs -204, -516, and -514 are returned by the server in succession. You must use **getNextException** to handle each SQLCODE accordingly.

Example 9-5 provides an example of how to code this type of error handling.

Example 9-5 Handling chained exceptions

```
......
catch(SQLException qex)
{
   System.err.println ("SQLException information");
   while(qex!=null) {
      System.err.println ("Error msg: " + qex.getMessage());
      System.err.println ("SQLSTATE: " + qex.getSQLState());
      System.err.println ("Error code: " + qex.getErrorCode());
      qex.printStackTrace();
      ...... //SQLCODE handle logic
      qex = qex.getNextException();
   }
}
```

9.2 Correlating DB2 and WebSphere Application Server information

For more information about this topic, see 4.1.3, “WLM configuration” on page 106, 5.3.4, “Configuring a subsystem ID on the data source” on page 238, and 8.2.1, “Using client information strings for correlating data” on page 363.

9.3 Common tools for problem determination

DB2 for z/OS and WebSphere Application Server contain many tools and service aids to assist you when a problem occurs. The more you know about these tools and service aids, the easier it is for you to diagnose problems and send data to IBM. This section provides an overview of the commonly used tools and where to find more information about each of them.

9.3.1 Application log

Application programs can write their own logging to track what they are doing. Logs are most often used when debugging an application. However, they are also used for audit purposes. This type of logging is an excellent way to understand what the application is doing, but it typically requires changes to the application to activate the traces, or add additional trace entries for certain areas that are experiencing a problem.

As it is not always possible to go in and make changes to the application, we focus more on the trace capabilities that do not require any changes to the applications themselves.

9.3.2 IBM Data Server Driver for JDBC and SQLJ trace

There are many different types of traces that can help you determine the cause of a problem, including traces at the application server level, the database engine, and the Java driver level. This section explains how you can use the IBM Data Server Driver for JDBC and SQLJ trace (JCC trace) to focus on application, driver, and database problems, or at least collect enough information that allows you to determine where the actual problem area is.

There are many different ways to accomplish tracing. The option that you choose depends on whether you want to activate the tracing outside your application code or within the application, and how granular the trace must be (one application, all applications using a data source, or all applications within the application server).

This section describes a few options, but the focus is on activating tracing outside the application code in a WebSphere Application Server environment.

JCC trace in the application

The way that you activate the JCC trace and the level of detail that the trace can collect depends on the interface that you use to activate the tracing.

Using the DataSource interface

If you use the `DataSource` interface to connect to a data source, you can use the following methods to start the JCC trace:

- Run the **`DB2BaseDataSource.setTraceLevel`** method to set the type of tracing that you need. The default trace level is `TRACE_ALL`. For more information about the different JCC trace levels that you can specify, see “TraceLevels” on page 465.
- Run the **`DB2BaseDataSource.setJccLogWriter`** method to specify the trace destination and turn on the trace. For more information about how to specify a trace destination, see “Other trace-related properties” on page 466.

Another option that you can use when using the `DataSource` interface is to run the **`javax.sql.DataSource.setLogWriter`** method to turn on the trace. However, when you use this method, **`TRACE_ALL`** is the only available trace level.

Using the DriverManager interface

If you use the `DriverManager` interface to connect to a data source, you can complete the following steps to start the JCC trace:

1. Run the **`DriverManager.getConnection`** method with the `traceLevel` property set in the **`info`** parameter or **`url`** parameter with the type of tracing that you want to activate. The default trace level is `TRACE_ALL`. For more information about how to specify more than one type of tracing, see “TraceLevels” on page 465.
2. Run the **`DriverManager.setLogWriter`** method to specify the trace destination and turn on the trace.

After a connection is established, you can turn off or on the trace, change the trace destination, or change the trace level by running the **`DB2Connection.setJccLogWriter`** method (**`DB2Connection.setJCCLogWriter(java.io.PrintWriter logWriter, int traceLevel)`**).

To turn off the trace, set the `logWriter` value to null. The `logWriter` property is an object of type `java.io.PrintWriter`. If your application cannot handle `java.io.PrintWriter` objects, you can use the `traceFile` property to specify the destination of the trace output. To use the `traceFile` property, set the `logWriter` property to null, and set the `traceFile` property to the name of the file to which the driver writes the trace data. This file and the directory in which it is must be writable. If the file exists, the driver overwrites it.

Another option when you use the `DriverManager` interface it to specify the `traceFile` and `traceLevel` properties as part of the URL when you load the driver. For example:

```
String url = "jdbc:db2://d0zg.itso.ibm.com:39000/DB0Z" +
":traceFile=/u/jcctrace;" +
"traceLevel=" +
com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS + ";";
```

Using the DB2TraceManager methods

You can also use the **`DB2TraceManager`** methods. The **`DB2TraceManager`** class can suspend and resume tracing of any type of log writer.

All of the methods that are mentioned above require you to change the application code to activate the JCC trace or at least preinstall code to be able to trace these events. This is often not an option, either because it requires program changes that are typically subject to change management control and testing procedures, but it is also possible that you are dealing with a packaged application that you bought and you do not have the source program to add the trace points to be able to activate the JCC trace. Therefore, it is a preferred practice to activate the JCC trace outside of the application code itself, either at the application server level or the data source level.

JCC tracing at the data source level

An easy way to activate the JCC trace outside the application code is by specifying the properties at the data source level as a custom property. In this example, which is shown in Figure 9-1 we specify the `traceLevel`, the `traceDirectory`, and the `traceFile` properties.

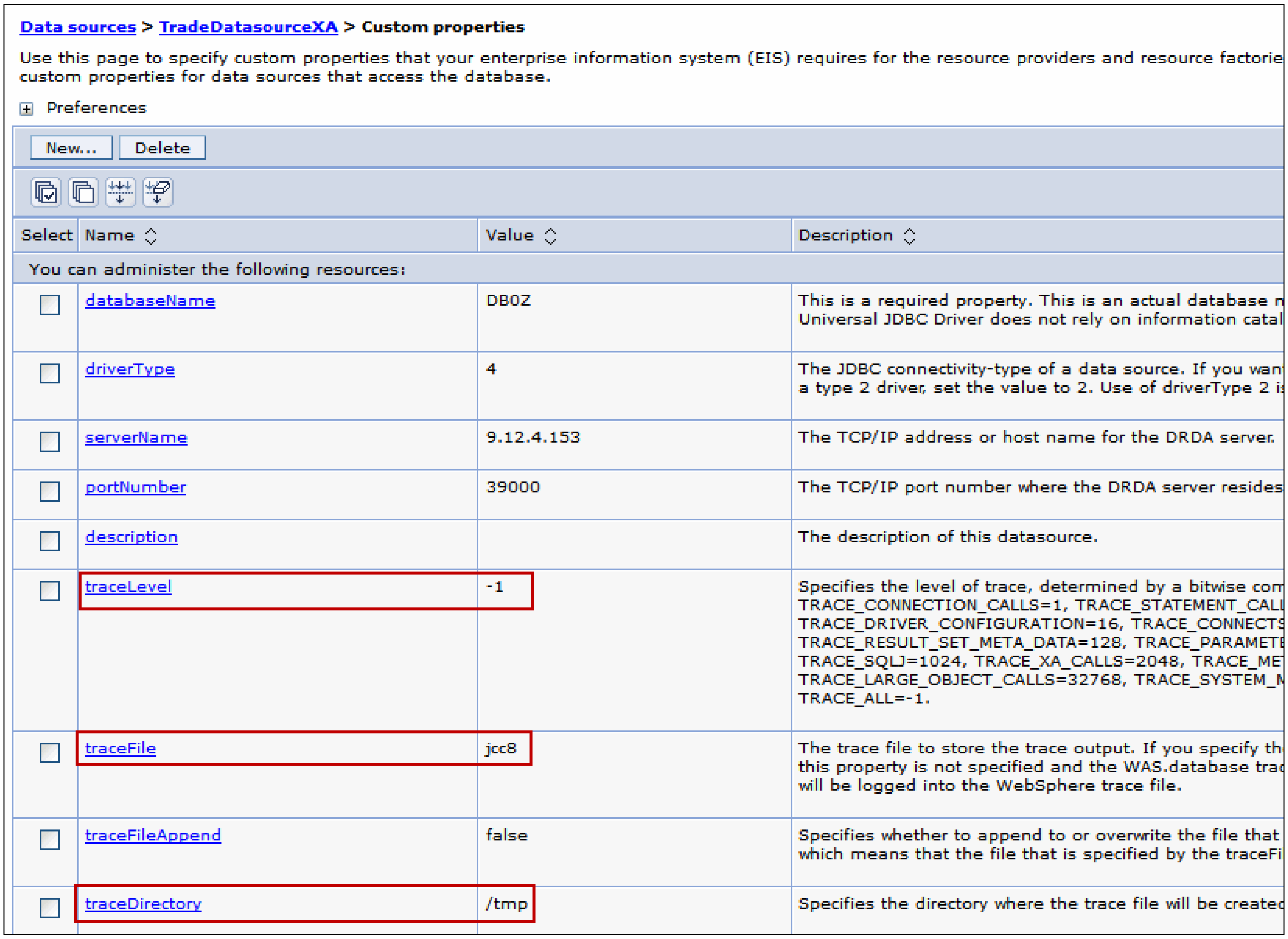


Figure 9-1 Specifying JCC trace parameters at the data source level

The advantage of using the data source custom properties to activate the JCC trace is that the application does not have to be changed. However, the disadvantage of this method is that the application server must be stopped and started to activate these settings. This action might not be practical in a production environment.

Combining JCC and WebSphere Application Server tracing

If WebSphere Application Server is involved and JCC traces are also required, you can also turn on WebSphere tracing instead of specifying the JCC trace file at the data source level.

Perform the actions that are described in the following sections to take a combined WebSphere and IBM Data Server Driver for JDBC and SQLJ trace (JCC- Java Universal Driver) trace. You can use the WebSphere Application Server administration console if the data source is managed by WebSphere Application Server.

Specifying the JCC traceLevel property

Combine the JCC and WebSphere trace so you can specify the JCC trace level at the data source level.

Using the administration console, click **JDBC** → **Data sources**, select your data source, click **Custom properties**, and specify the traceLevel. In our example, we use 131072, which is the TRACE_SYSTEM_MONITOR, as shown in Figure 9-2. You can specify any valid trace level that you want. For more information about the different trace levels, see “TraceLevels” on page 465.

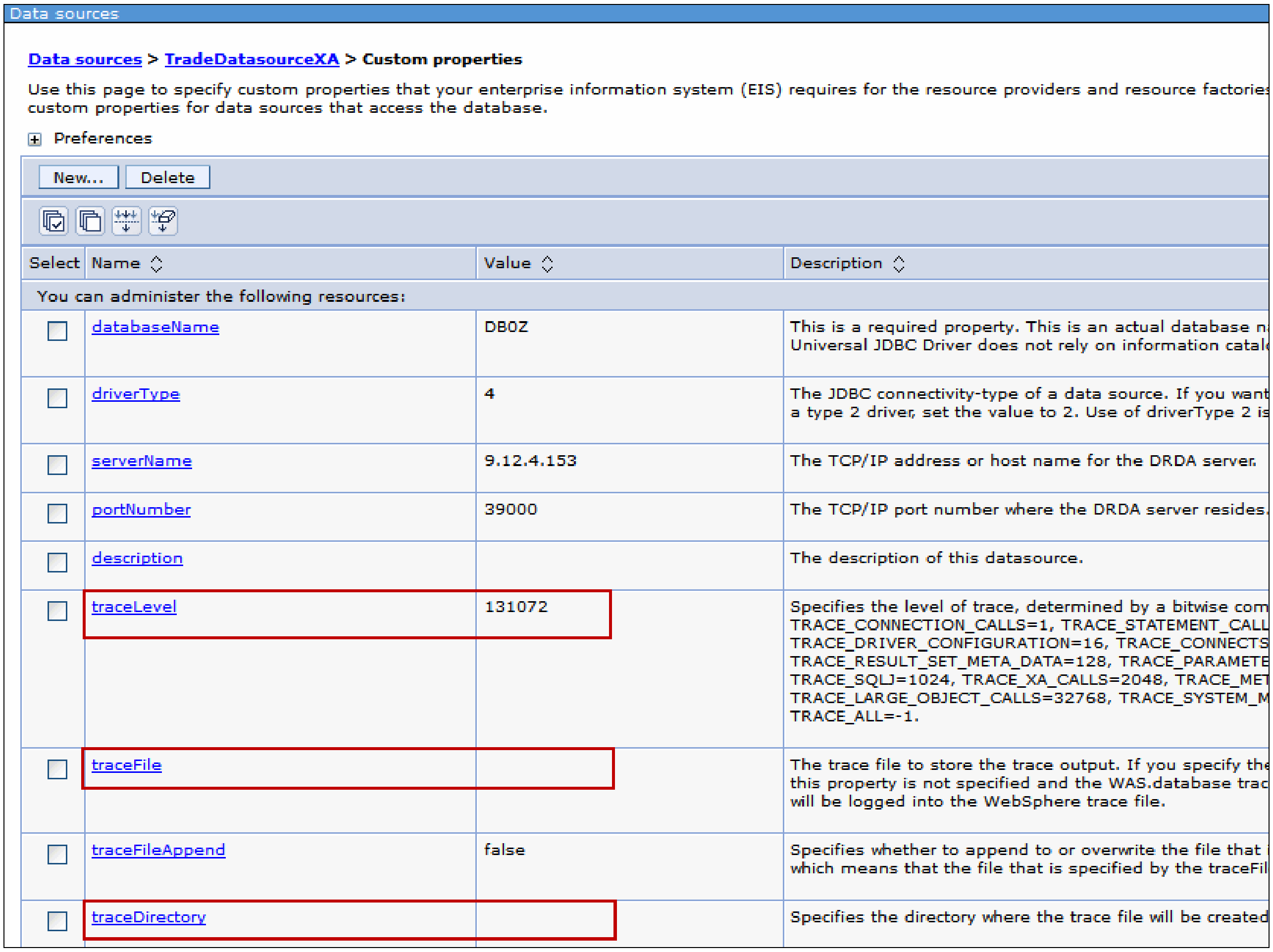


Figure 9-2 Specify only traceLevel at the data source level

We do not specify the `traceFile` and `traceDirectory` properties, which allows the JCC trace to be automatically embedded in the WebSphere Application Server trace (the SYSOUT DD-card of the servant region when you use WebSphere Application Server on z/OS).

Turning on the WebSphere Application Server trace

You can enable WebSphere Application Server traces by going to the administration console and clicking **Troubleshooting** → **Logging and Tracing**, selecting your application server, and clicking **Change log detail levels**. Click the **Runtime** tab to activate dynamically the trace and specify the traces that you want to activate. As shown in Figure 9-3, we use a detailed trace and specified the following string:

```
*=info:WAS.j2c=all:RRA=all:WAS.database=all:Transaction=all
```

If you want to make this trace permanent, select the **Save runtime changes to configuration as well** check box, but you typically want to activate the trace only for a short time, so we did not select the check box.

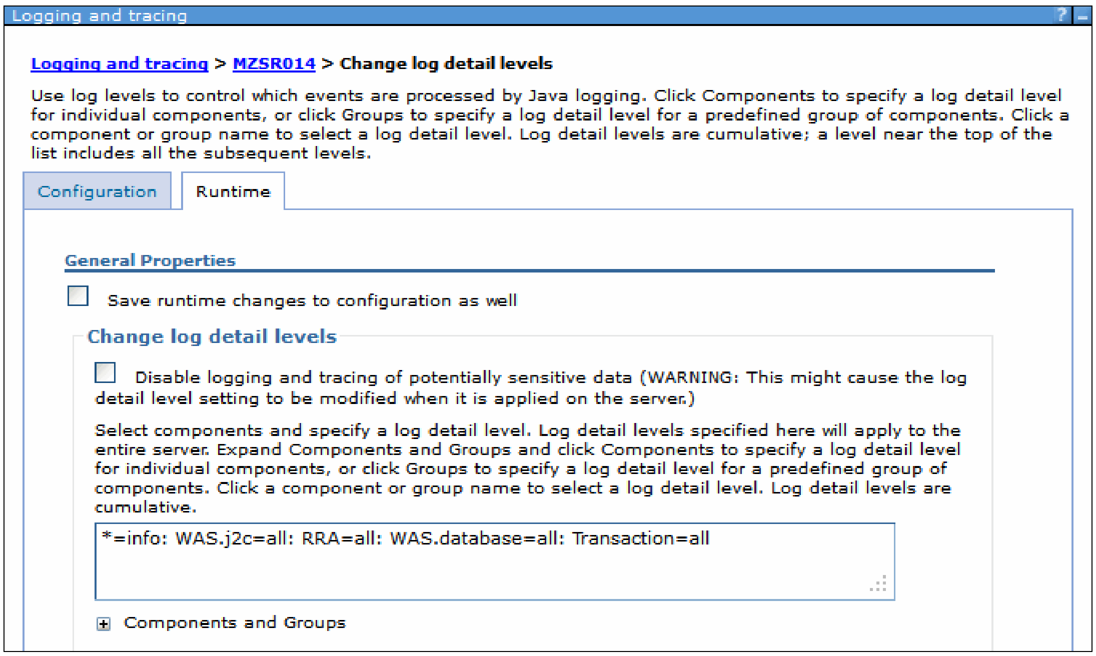


Figure 9-3 Set the log detail level in WebSphere Application Server

When the changes are saved, the trace is activated dynamically. There is no need to stop and start the application server.

You can verify whether the changed trace options were picked up by checking the servant's SYSOUT information. There should be a message similar to the following one:

```
Trace: 2012/11/20 22:41:31.769 02 t=7B74F8 c=UNK key=P8 tag= (13007004)
  SourceId: com.ibm.ejs.ras.ManagerAdmin
  ExtendedMessage: BBOO0222I: TRAS0018I: The trace state has changed. The new
    trace state is *=info:WAS.j2c=all:RRA=all:WAS.database=all:Transaction=all.
```

From then on, the WebSphere Application Server and JCC trace are active. The trace log (in SYSOUT) combines the output of the WebSphere Application Server trace with the JCC trace, as shown in Example 9-6.

Example 9-6 Combined WebSphere and JCC trace to SYSOUT DD statement

```
Trace: 2012/11/20 22:42:34.222 02 t=7B7718 c=UNK key=P8 tag= (13007004)
 SourceId: com.ibm.ws.rsadapter.spi.WSRdbManagedConnectionImpl.enforceAutoCommit
  ExtendedMessage: Entry; false
Trace: 2012/11/20 22:42:34.222 02 t=7B7718 c=UNK key=P8 tag= (13007004)
 SourceId: com.ibm.ws.rsadapter.spi.WSRdbManagedConnectionImpl.enforceAutoCommit
```

```
 ExtendedMessage: Exit
Trace: 2012/11/20 22:42:34.222 02 t=7B7718 c=UNK key=P8 tag= (13007004)
 SourceId: com.ibm.ws.rsadapter.spi.WSRdbManagedConnectionImpl
 ExtendedMessage: No Matching Prepared Statement found in cache
Trace: 2012/11/20 22:42:34.223 02 t=7B7718 c=UNK key=P8 tag= (13007004)
 SourceId: com.ibm.ws.db2.logwriter
 ExtendedMessage: Ýjcc¨ÝSystemMonitor:start¨
Trace: 2012/11/20 22:42:34.223 02 t=7B7718 c=UNK key=P8 tag= (13007004)
 SourceId: com.ibm.ws.db2.logwriter
  ExtendedMessage: Ýjcc¨ÝTime:2012-11-20-22:42:34.223¨ÝThread:WebSphere WLM Dispatch Thread t=007b7718¨
    ÝConnection@370f35ff¨prepareStatement (select * from orderejb o where o.orderstatus = 'closed' AND
    o.account_accountid = (select a.accountid from accountejb a where a.profile_userid = ?), 1003, 1007)
    called
Trace: 2012/11/20 22:42:34.234 02 t=7B7718 c=UNK key=P8 tag= (13007004)
 SourceId: com.ibm.ws.db2.logwriter
  ExtendedMessage: Ýjcc¨ÝTime:2012-11-20-22:42:34.234¨ÝThread:WebSphere WLM Dispatch Thread t=007b7718¨
    ÝConnection@370f35ff¨prepareStatement () returned com.ibm.db2.jcc.t4.k@ec3e3073
Trace: 2012/11/20 22:42:34.234 02 t=7B7718 c=UNK key=P8 tag= (13007004)
 SourceId: com.ibm.ws.db2.logwriter
  ExtendedMessage: Ýjcc¨ÝThread:WebSphere WLM Dispatch Thread t=007b7718¨ÝSystemMonitor:stop¨ core:
    10.774125ms | network: 0.0ms | server: 0.0ms
Trace: 2012/11/20 22:42:34.240 02 t=7B7718 c=UNK key=P8 tag= (13007004)
 SourceId: com.ibm.ws.rsadapter.jdbc.WSJccPreparedStatement.<init>
  ExtendedMessage: Entry; com.ibm.db2.jcc.t4.k@ec3e3073,
    com.ibm.ws.rsadapter.jdbc.WSJccSQLJPDQConnection@e7857dfb, DEFAULT CURSOR HOLDABILITY VALUE (0), PSTMT:
    select * from orderejb o where o.orderstatus = 'closed' AND o.account_accountid = (select a.accountid
    from accountejb a where a.profile_userid = ?) 1003 1007 0 0 4
Trace: 2012/11/20 22:42:34.240 02 t=7B7718 c=UNK key=P8 tag= (13007004)
 SourceId: com.ibm.ws.rsadapter.jdbc.WSJccPreparedStatement
 ExtendedMessage: current fetchSize is 0
Trace: 2012/11/20 22:42:34.240 02 t=7B7718 c=UNK key=P8 tag= (13007004)
 SourceId: com.ibm.ws.rsadapter.jdbc.WSJccPreparedStatement.<init>
 ExtendedMessage: Exit; com.ibm.ws.rsadapter.jdbc.WSJccPreparedStatement@aaca8816
```

The entries in bold that are marked by [jcc] (not shown correctly because of code page differences) are from the JCC trace. The others are written as part of the WebSphere Application Server traces that were activated as well.

WebSphere Application Server traces can be verbose, so try to limit the type of tracing to a minimum and trace only the events in which you are interested.

Specifying the JCC trace at the driver configuration properties level

Another (final) way to activate the JCC trace is through the IBM Data Server Driver for JDBC and SQLJ configuration properties file. This properties file applies to the entire JVM, so for WebSphere Application Server on z/OS, this is the complete servant region.

The major advantage of activating the JCC trace in the configuration properties file is that changes to the settings are automatically picked up without stopping and starting the application server.

To use the driver configuration properties file, you must point the WebSphere Application Server to it. At the administration console, click **Application servers**, select ***your application server***, and click **Process definition** → **Servant** → **Java Virtual Machine** → **Custom properties**, as shown in Figure 9-4. You can add the `db2.jcc.propertiesFile` property and point it to the location where the properties file is.

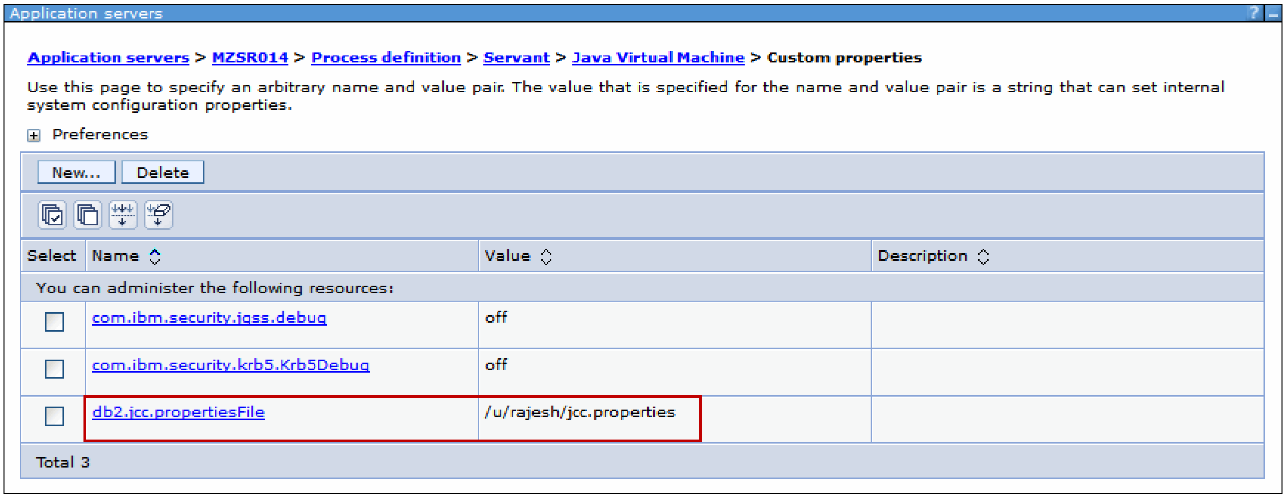


Figure 9-4   Specifying db2.jcc.propertiesFile as a custom property

We use `/u/rajesh/jcc.properties` in our example. The content of the file is shown in Example 9-7. The lines that start with a ‘#’ are comments lines. In our case, we specified only parameters that are related to tracing, but you can specify other driver-wide properties as well.

Example 9-7   jcc.properties file

```
db2.jcc.tracePollingInterval=10
db2.jcc.tracePolling=true
db2.jcc.override.traceDirectory=/tmp
db2.jcc.override.traceFile=jcc6
db2.jcc.override.traceLevel=0
#db2.jcc.override.traceLevel=-1
#db2.jcc.override.traceLevel=131072
```

For a complete list of driver properties settings, see the IBM Data Server Driver for JDBC and SQLJ configuration properties topic in the Information Center at the following website:

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.imjcc10.doc.updates/src/tpc/imjcc_r0052075.htm

The IBM Data Server Driver for JDBC and SQLJ configuration properties have a driver-wide scope. If there is a corresponding `Connection` or `DataSource` property that is specified, those properties typically override the setting in the properties file. For example, `db2.jcc.traceLevel` is a configuration file property, and `traceLevel` is the equivalent `Connection` or `DataSource` property setting and it overrides the configuration file property setting. So in this case, the configuration property provides a default value for the `Connection` or `DataSource` property.

The `db2.jcc.override.traceLevel` configuration property also maps to the `traceLevel` `Connection` or `DataSource` property, but here the configuration file property setting overrides the `Connection` or `DataSource` property value. Using the `*.override.*` ‘flavor’ of the configuration property allows you to take control of the property setting at the driver level.

Note the `db2.jcc.tracePolling=true` setting. This indicates to the driver that it must check for possible changes in the properties file and `db2.jcc.tracePollingInterval=10` directs the driver to perform this check every 10 seconds.

The use of the override feature together with regular polling allows you to dynamically activate/de-activate the JCC trace for the JVM.

Tip: When you direct the trace to a directory, make sure that you have write authority for that directory. Otherwise, you might think the trace is not active, but the driver is unable to write the data to the specified directory.

You can also set up circular logging when you use a type 4 connection by using the `db2.jcc.traceOption=1` setting. Combined with the `db2.jcc.traceFileSize` and `db2.jcc.traceFileCount` properties, you dedicate a number of trace files, each of a certain size. When all the trace files reach the maximum size, the first file is reused and the existing data is overwritten, which can be useful when you must trace a situation where you are not sure when the problem will occur. So, you set up circular tracing and activate the trace, and when the problem occurs, you turn off the trace immediately, which gives you a good chance to capture the problem in the trace without using large trace files (some of the JCC trace options are verbose).

TraceLevels

Table 9-1 shows the different trace levels that are available with the IBM Data Server Driver for JDBC and SQLJ.

Table 9-1 IBM Data Server Driver for JDBC and SQLJ trace levels

TraceLevel	Trace value in hex	Trace value in decimal
TRACE_NONE	(X'00')	(0)
TRACE_CONNECTION_CALLS	(X'01')	(1)
TRACE_STATEMENT_CALLS	(X'02')	(2)
TRACE_RESULT_SET_CALLS	(X'04')	(4)
TRACE_DRIVER_CONFIGURATION	(X'10')	(16)
TRACE_CONNECTS	(X'20')	(32)
TRACE_DRDA_FLOWS	(X'40')	(64)
TRACE_RESULT_SET_META_DATA	(X'80')	(128)
TRACE_PARAMETER_META_DATA	(X'100')	(256)
TRACE_DIAGNOSTICS	(X'200')	(512)
TRACE_SQLJ	(X'400')	(1024)
TRACE_META_CALLS	(X'2000')	(8192)
TRACE_DATASOURCE_CALLS	(X'4000')	(16384)
TRACE_LARGE_OBJECT_CALLS	(X'8000')	(32768)
TRACE_T2ZOS (*)	(X'10000')	(65536)
TRACE_SYSTEM_MONITOR	(X'20000')	(131072)
TRACE_TRACEPOINTS	(X'40000')	(262144)

TraceLevel	Trace value in hex	Trace value in decimal
TRACE_ALL	(X'FFFFFFFF')	(-1)
(*) JCC type 2 Driver for z/OS only		

If you want to combine multiple trace levels, you can use OR to combine the values.

Suppose you want to trace the following items:

```
(TRACE_CONNECTION_CALLS |
TRACE_STATEMENT_CALLS |
TRACE_RESULT_SET_CALLS |
TRACE_DRIVER_CONFIGURATION |
TRACE_CONNECTS |
TRACE_DIAGNOSTICS)
```

The traceLevel should be set to the sum of the integer values of these constants:

1 + 2 + 4 + 16 + 32 + 512 = 567

So, you specify the following string:

```
jdbc:db2://localhost:50000/sample:traceDirectory=\tmp;traceFile=jcctrace.log;trace
FileAppend=false;traceLevel=567;
```

Other trace-related properties

Besides the `traceLevel` property, there are many other properties that indicate how and where trace data is produced. We list the most important ones here:

- `traceDirectory` specifies the directory into which trace information is written. The data type of this property is `String`.

  When `traceDirectory` is specified, trace information for multiple connections on the same `DataSource` is written to multiple files.

  When `traceDirectory` is specified, a connection is traced to a file named `traceFile_`*`origin_n`*, where:

  – *n* is the nth connection for a data source.

  – *origin* indicates the origin of the log writer that is in use. Possible values of *origin* are:

cpds	The log writer for a `DB2ConnectionPoolDataSource` object
driver	The log writer for a `DB2Driver` object
global	The log writer for a `DB2TraceManager` object
sds	The log writer for a `DB2SimpleDataSource` object
xads	The log writer for a `DB2XADataSource` object

  If the `traceFile` property is also specified, the `traceDirectory` value is not used.

- `traceFile` specifies the name of a file into which the IBM Data Server Driver for JDBC and SQLJ writes trace information. The data type of this property is `String`. The `traceFile` property is an alternative to the `logWriter` property for directing the output trace stream to a file.

- `traceFileAppend` specifies whether to append to or overwrite the file that is specified by the `traceFile` property. The data type of this property is `boolean`. The default is `false`, which means that the file that is specified by the `traceFile` property is overwritten.

- `traceFileCount` specifies the maximum number of trace files for circular tracing.

- `traceFileSize` specifies the maximum size of each trace file for circular tracing.
- `traceOption` specifies the way in which trace data is collected. The data type of this property is `int`. Here are possible values:

`DB2BaseDataSource.NOT_SET` (0)	Specifies that a single trace file is generated, and that there is no limit to the size of the file. This is the default. If the value of `traceOption` is `NOT_SET`, the `traceFileSize` and `traceFileCount` properties are ignored.
`DB2BaseDataSource.TRACE_OPTION_CIRCULAR` (1)	Specifies that the IBM Data Server Driver for JDBC and SQLJ does circular tracing.

A sample TRACE_ALL JCC trace

To give you some idea about the amount of information that is available in the JCC trace, here are a few snippets from a detailed (`TRACE_ALL`) trace. A slightly longer version is available in Appendix F, "Sample IBM Data Server Driver for JDBC and SQLJ trace" on page 555.

General trace entry layout

First, look at the general layout of a trace entry, for example:

```
[jcc][Time:2012-11-16-21:49:08.222][Thread:WebSphere WLM Dispatch Thread
          t=007bd580][PreparedStatement@41d88590]executeQuery () called
```

Each line is prefixed with a string [jcc]. After the prefix are one or more tokens in [ ]:

- [t2] or [t4] when the trace entry is specific for the driver type (N/A here)
- Timestamp (in GMT) (`[Time:2012-11-16-21:49:08.222]`).
- Thread name (`[Thread:WebSphere WLM Dispatch Thread t=007bd580]`)
- Object name that is associated with the trace entry (Connection, Statement, ResultSet, ...) (`[PreparedStatement@41d88590]`).
- Tracepoint number when applicable (N/A here).
- The rest of the line is the method name that is called or returned with arguments or the return value (**`executeQuery () called`**).

Begin-end event tracing

The JCC trace typically records begin and end events, such as '**`before - after execution`**', **'`Systemonitor:start -stop`'**, as shown in Example 9-8, which makes it easy to understand the flow of the SQL statement and programs.

Example 9-8 JCC trace excerpt

```
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-21:49:08.222][Thread:WebSphere WLM Dispatch Thread
t=007bd580][PreparedStatement@41d88590]executeQuery () called
[jcc][t4] [time:2012-11-16-21:49:08.222][Thread:WebSphere WLM Dispatch Thread
        t=007bd580][tracepoint:431]Before Executing, AutoCommit=false RLSCONV=242
[jcc][t4] [time:2012-11-16-21:49:08.222][Thread:WebSphere WLM Dispatch Thread
        t=007bd580][tracepoint:815]====== connected to primary server = true
...
...
[jcc][t4] [time:2012-11-16-21:49:08.225][Thread:WebSphere WLM Dispatch Thread
t=007bd580][tracepoint:431]After Executing, AutoCommit=false RLSCONV=240
```

```
[jcc][Time:2012-11-16-21:49:08.225][Thread:WebSphere WLM Dispatch Thread
t=007bd580][PreparedStatement@41d88590]executeQuery () returned
com.ibm.db2.jcc.t4.i@9a783140
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core:
3.0797499999999998ms | network: 0.921125ms | server: 0.37ms [STMT@1104709008]
```

Sections for the different traceLevel settings

The different JCC trace levels that you can specify typically add their own sections to the trace that can be easily identified. The **TRACE_CONNECTS** option for example adds a BEGIN/END TRACE_CONNECTS entry to the trace and some additional lines that describe the connection, as shown in Example 9-9.

Example 9-9 TRACE_CONNECT entries

```
[jcc][Connection@13361385] BEGIN TRACE_CONNECTS
[jcc][Connection@13361385] Successfully connected to server
jdbc:db2://9.12.4.153:39000/DB0Z
[jcc][Connection@13361385] User: rajesh
[jcc][Connection@13361385] Database product name: DB2
[jcc][Connection@13361385] Database product version: DSN10015
[jcc][Connection@13361385] Driver name: IBM DB2 JDBC Universal Driver Architecture
[jcc][Connection@13361385] Driver version: 3.64.82
[jcc][Connection@13361385] DB2 Application Correlator: ::9.12.6.9.65123.CA7B405C24DB.0000
[jcc][Connection@13361385] END TRACE_CONNECTS
```

Tracing the DRDA flow

When you use Type 4 connectivity, you can also see the DRDA flows and buffers being passed between the application server (driver sends buffers) and the database server (driver receives buffers from the database server). Example 9-10 shows a sample flow for a **SELECT** statement being prepared and run, and the result coming back from the database server. The output was edited to shorten it. The complete JCC trace of this transaction can be found in Appendix F, “Sample IBM Data Server Driver for JDBC and SQLJ trace” on page 555

Example 9-10 DRDA flow

```
[jcc][t4]   DRDA manager levels: { SQLAM=10, AGENT=10, CMNTCPIP=5, RDB=8, SECMGR=9,
XAMGR=7, SYNCPTMGR=0, RSYNCMGR=0 }
...
[jcc][t4]        SEND BUFFER: PRPSQLSTT            (ASCII)          (EBCDIC)
[jcc][t4]
[jcc][t4]        SEND BUFFER: SQLATTR              (ASCII)          (EBCDIC)
[jcc][t4]
[jcc][t4]        SEND BUFFER: SQLSTT               (ASCII)          (EBCDIC)
[jcc][t4]
[jcc][t4]        SEND BUFFER: OPNQRY               (ASCII)          (EBCDIC)
[jcc][t4]
[jcc][t4]        SEND BUFFER: SQLDTA               (ASCII)          (EBCDIC)
[jcc][t4] [time:2012-11-16-21:49:08.224][Thread:WebSphere WLM Dispatch Thread
t=007bd580][tracepoint:101]Request flushed.
[jcc][t4] [time:2012-11-16-21:49:08.224][Thread:WebSphere WLM Dispatch Thread
t=007bd580][tracepoint:102]Reply to be filled.
[jcc][t4][time:2012-11-16-21:49:08.225][Thread:WebSphere WLM Dispatch Thread
t=007bd580][tracepoint:2][Reply.fill]
[jcc][t4]        RECEIVE BUFFER: SQLCARD           (ASCII)          (EBCDIC)
[jcc][t4]
[jcc][t4]        RECEIVE BUFFER: MONITORRD         (ASCII)          (EBCDIC)
[jcc][t4]
[jcc][t4]        RECEIVE BUFFER: OPNQRYRM          (ASCII)          (EBCDIC)
```

```
[jcc][t4]
[jcc][t4]       RECEIVE BUFFER: QRYDSC           (ASCII)          (EBCDIC)
[jcc][t4]
[jcc][t4]       RECEIVE BUFFER: QRYDTA           (ASCII)          (EBCDIC)
[jcc][t4]
[jcc][t4]       RECEIVE BUFFER: ENDQRYRM         (ASCII)          (EBCDIC)
[jcc][t4]
[jcc][t4]       RECEIVE BUFFER: SQLCARD          (ASCII)          (EBCDIC)
[jcc][t4]
[jcc][t4]       RECEIVE BUFFER: MONITORRD        (ASCII)          (EBCDIC)
[jcc][t4]
```

DB2 correlation information

The JCC trace also provides information that allows you to correlate the JCC trace data with other data on the DB2 for z/OS side, such as accounting and performance traces. Example 9-11 shows the set methods that are invoked to set the client string and the values that they are set to.

In addition, in a number of places, the JCC trace also provides the *instance number* and the *commit sequence number*. They are part of the *Logical Unit of Work ID (LUWID)* that should uniquely define a transaction. In versions before DB2 10, you often see multiple transactions using the same LUWID (when they had not been making changes to the database). However, starting with DB2 10, the LUWID's commit sequence number should be incremented each time.

Example 9-11 DB2 correlation information

```
[jcc][Time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread
t=007bd580][Connection@13361385]getDB2Correlator () returned ::9.12.6.9.65123.CA7B405C24DB
[jcc][Connection@13361385] BEGIN TRACE_CONNECTS
[jcc][Connection@13361385] Successfully connected to server
jdbc:db2://9.12.4.153:39000/DB0Z
[jcc][Connection@13361385] User: rajesh
[jcc][Connection@13361385] Database product name: DB2
[jcc][Connection@13361385] Database product version: DSN10015
[jcc][Connection@13361385] Driver name: IBM DB2 JDBC Universal Driver Architecture
[jcc][Connection@13361385] Driver version: 3.64.82
[jcc][Connection@13361385] DB2 Application Correlator: ::9.12.6.9.65123.CA7B405C24DB.0000
[jcc][Connection@13361385] END TRACE_CONNECTS
...
[jcc][Time:2012-11-16-21:49:08.224][Thread:WebSphere WLM Dispatch Thread t=007bd580]
     [Connection@13361385]setDB2ClientUser (TraderClientUser) called
[jcc][Time:2012-11-16-21:49:08.224][Thread:WebSphere WLM Dispatch Thread t=007bd580
   ] [Connection@13361385]setDB2ClientWorkstation (TraderClientWorkstation) called
[jcc][Time:2012-11-16-21:49:08.224][Thread:WebSphere WLM Dispatch Thread t=007bd580]
     [Connection@13361385]setDB2ClientApplicationInformation
                          (TraderClientApplicationInformation) called
[jcc][Time:2012-11-16-21:49:08.224][Thread:WebSphere WLM Dispatch Thread t=007bd580]
     [Connection@13361385]setDB2ClientAccountingInformation
                          (TraderClientAccountingInformation) called
...
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread
t=007bd580][Connection@13361385]commit () called
[jcc][t4][time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread
t=007bd580][tracepoint:1][Request.flush]
[jcc][t4]       SEND BUFFER: RDBCMM              (ASCII)           (EBCDIC)
[jcc][t4]       0 1 2 3 4 5 6 7   8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
[jcc][t4] 0000  000FD00100010009  200E0005119FF2    ........ ......   ..}...........2
```

```
[jcc][t4]
...
...
[jcc][Connection@13361385] DB2 LUWID: ::9.12.6.9.65123.CA7B405C24DB.0007
[jcc][t4] [time:2012-11-16-21:49:08.233][Thread:WebSphere WLM Dispatch Thread
t=007bd580][tracepoint:300]currXACallInfoOffset : 0commit
```

In our case, the DB2 correlator is CA7B405C24DB.0007. To verify this transaction, go back to the DB2 accounting data and find the matching accounting record for this transaction.

To find the matching trace data (accounting, performance traces) on the DB2 for z/OS side, consider the following items:

- If your system is using a clock that is taking leap seconds into account, you might see a 25 second discrepancy between the times in the JCC trace and the times in the DB2 trace records. (At the time of writing, the number of leap seconds in effect is 25.) At the time of the commit of this transaction, the JCC trace shows the following information:

  [jcc][t4] [time:2012-11-16-21:49:**08.**233]

  The time stamp in the DB2 accounting record shows the following information:

  ACCT TSTAMP: 11/16/12 21:49:**33.**23
- When you are matching the LUWID, you can use the commit sequence number from the JCC trace to match with DB2 performance trace records. However, when you look for the corresponding accounting record, you must use the commit sequence number from the JCC trace and add one to it, so in our example, 0007 + 1 = 8.

The corresponding DB2 accounting record is shown in Example 9-12.

Example 9-12 DB2 accounting record that matches the JCC trace

```
LOCATION: DB0Z                  OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1M1)                 PAGE: 1-55
      GROUP: DB0ZG                             ACCOUNTING TRACE - LONG                    REQUESTED FROM: ALL      21:49:00.00
     MEMBER: D0Z1                                                                                    TO: DATES    23:59:59.99
  SUBSYSTEM: D0Z1                                                                            ACTUAL FROM: 11/16/12 21:49:12.97
DB2 VERSION: V10

---- IDENTIFICATION --------------------------------------------------------------------------------------------------------
ACCT TSTAMP: 11/16/12 21:49:33.23    PLANNAME: TraderCl       WLM SCL: DDFONL         CICS NET: N/A
BEGIN TIME : 11/16/12 21:49:33.22    PROD TYP: JDBC DRIVER                            CICS LUN: N/A
END TIME   : 11/16/12 21:49:33.23    PROD VER: V3 R64M0       LUW NET: G90C0609       CICS INS: N/A
REQUESTER  : ::9.12.6.9              CORRNAME: db2jcc_a       LUW LUN: FE63
MAINPACK   : SYSLN300                CORRNMBR: ppli           LUW INS: CA7B405C24DB   ENDUSER : TraderClientUser
PRIMAUTH   : RAJESH                  CONNTYPE: DRDA           LUW SEQ:            8   TRANSACT: TraderClientApplicationInformati
ORIGAUTH   : RAJESH                  CONNECT : SERVER                                 WSNAME  : TraderClientWorkst

ELAPSED TIME DISTRIBUTION                                          CLASS 2 TIME DISTRIBUTION
----------------------------------------------------------------   ----------------------------------------------------------------
APPL   |=================> 37%                                      CPU     |======> 13%
DB2    |====> 8%                                                   SECPU   |
SUSP   |==========================> 55%                            NOTACC  |
                                                                   SUSP    |==========================================> 87%

TIMES/EVENTS  APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)   CLASS 3 SUSPENSIONS   ELAPSED TIME    EVENTS    HIGHLIGHTS
------------  ----------  ----------  ----------   --------------------  ------------  --------    --------------------------
ELAPSED TIME    0.008414    0.005296         N/P   LOCK/LATCH(DB2+IRLM)      0.000000         0    THREAD TYPE   : DBAT
 NONNESTED      0.008414    0.005296         N/A    IRLM LOCK+LATCH          0.000000         0    TERM.CONDITION: NORMAL
 STORED PROC    0.000000    0.000000         N/A    DB2 LATCH                0.000000         0    INVOKE REASON : TYP2 INACT
 UDF            0.000000    0.000000         N/A   SYNCHRON. I/O             0.000416         1    PARALLELISM   : NO
 TRIGGER        0.000000    0.000000         N/A    DATABASE I/O             0.000000         0    QUANTITY      :          0
                                                    LOG WRITE I/O            0.000416         1    COMMITS       :          1
CP CPU TIME     0.000811    0.000694         N/P   OTHER READ I/O            0.000000         0    ROLLBACK      :          0
 AGENT          0.000811    0.000694         N/A   OTHER WRTE I/O            0.000000         0    SVPT REQUESTS :          0
  NONNESTED     0.000811    0.000694         N/P   SER.TASK SWTCH            0.000000         0    SVPT RELEASE  :          0
  STORED PRC    0.000000    0.000000         N/A    UPDATE COMMIT            0.000000         0    SVPT ROLLBACK :          0
  UDF           0.000000    0.000000         N/A    OPEN/CLOSE               0.000000         0    INCREM.BINDS  :          0
  TRIGGER       0.000000    0.000000         N/A    SYSLGRNG REC             0.000000         0    UPDATE/COMMIT :       1.00
 PAR.TASKS      0.000000    0.000000         N/A    EXT/DEL/DEF              0.000000         0    SYNCH I/O AVG.:   0.000416
                                                    OTHER SERVICE            0.000000         0    PROGRAMS      :          1
 SECP CPU       0.000000         N/A         N/A   ARC.LOG(QUIES)            0.000000         0    MAX CASCADE   :          0
                                                   LOG READ                  0.000000         0
```

```
SE CPU TIME      0.000000    0.000000         N/A   DRAIN LOCK               0.000000        0
 NONNESTED       0.000000    0.000000         N/A   CLAIM RELEASE            0.000000        0
 STORED PROC     0.000000    0.000000         N/A   PAGE LATCH               0.000000        0
 UDF             0.000000    0.000000         N/A   NOTIFY MSGS              0.000000        0
 TRIGGER         0.000000    0.000000         N/A   GLOBAL CONTENTION        0.003075        4
                                                    COMMIT PH1 WRITE I/O     0.000000        0
 PAR.TASKS       0.000000    0.000000         N/A   ASYNCH CF REQUESTS       0.001105        2
                                                    TCP/IP LOB XML           0.000000        0
SUSPEND TIME     0.000000    0.004596         N/A   TOTAL CLASS 3            0.004596        7
 AGENT                N/A    0.004596         N/A
 PAR.TASKS            N/A    0.000000         N/A
 STORED PROC     0.000000         N/A         N/A
 UDF             0.000000         N/A         N/A

NOT ACCOUNT.          N/A    0.000006         N/A
DB2 ENT/EXIT          N/A          10         N/A
EN/EX-STPROC          N/A           0         N/A
EN/EX-UDF             N/A           0         N/A
DCAPT.DESCR.          N/A         N/A         N/P
LOG EXTRACT.          N/A         N/A         N/P
```

DB2SystemMonitor

To help you isolate performance problems with your Java-DB2 applications, the IBM Data Server Driver for JDBC and SQLJ provides a proprietary API (`DB2SystemMonitor` class) to enable application monitoring.

The driver collects the timing information that is shown in Figure 9-5.

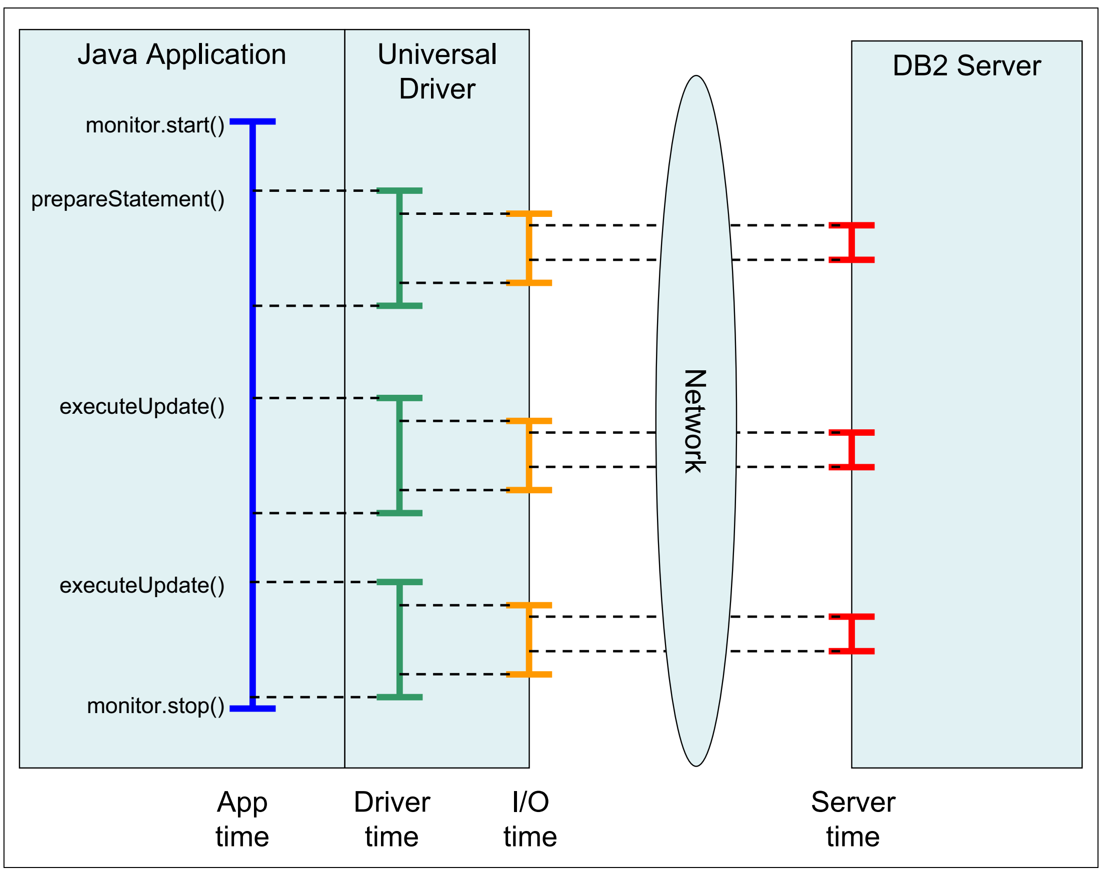


Figure 9-5 DB2SytemMonitor information

- Server time (the time that is spent in DB2 itself)
- Network I/O time (the time that used to flow the DRDA protocol stream across the network)
- Core driver time (the time that is spent in the driver, which includes network I/O time and server time)
- Application time (the time between the **start()** and **stop()** calls)

There are two methods to obtain this information:

- Use the `DB2SystemMonitor` interface
- Use the `TRACE_SYSTEM_MONITOR` trace level

To collect system monitoring data by using the `DB2SystemMonitor` interface, complete the following steps:

1. Run the **DB2Connection.getDB2SystemMonitor** method to create a `DB2SystemMonitor` object.
2. Run the **DB2SystemMonitor.enable** method to enable the `DB2SystemMonitor` object for the connection.
3. Run the **DB2SystemMonitor.start** method to start system monitoring.
4. When the activity that is to be monitored is complete, run **DB2SystemMonitor.stop** to stop system monitoring.
5. Lastly, run the following methods to retrieve the elapsed time data:
   - `DB2SystemMonitor.getCoreDriverTimeMicros`
   - `DB2SystemMonitor.getNetworkIOTimeMicros`
   - `DB2SystemMonitor.getServerTimeMicros`
   - `DB2SystemMonitor.getApplicationTimeMillis`

Note: Starting with Version 3.63 or Version 4.13, the The *server time* that is returned by `DB2SystemMonitor.getServerTimeMicros` now includes commit and rollback time. (This was not the case before these driver levels.)

Using the `DB2SystemMonitor` interface allows you to trace specific areas of your application that you are interested in, but it also requires that you change you application code to incorporate these calls.

An easier, yet not as specific, way is to use set the `TRACE_SYSTEM_MONITOR` trace level, either at the connection, data source, or JVM level. This method allows you to obtain this information without making any changes to the application; simply starting this trace level is enough.

To show the information that can be obtained this way, we used the following settings in the `jcc.properties` file.

```
db2.jcc.override.traceLevel=131072
db2.jcc.override.traceDirectory=/tmp
db2.jcc.override.traceFile=jcc6
db2.jcc.tracePollingInterval=10
db2.jcc.tracePolling=true
```

131072 = x’20000’ is equal to the `TRACE_SYSTEM_MONITOR` trace level.

We ran a few simple servlets from the DayTrader workload and captured the trace file (`jcc6_global_9`). An (edited) excerpt is shown in Example 9-13.

Example 9-13 JCC trace with SystemMonitor active

```
[jcc]
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-19:55:13.627][Thread:WebSphere WLM Dispatch Thread
t=007bd580][Connection@50fc259c]setAutoCommit (false) called
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 0.022ms | network: 0.0ms
                                                                           | server: 0.0ms
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-19:55:13.627][Thread:WebSphere WLM Dispatch Thread
t=007bd580][Connection@50fc259c]prepareStatement (select * from orderejb o where o.orderstatus = 'closed'
AND o.account_accountid = (select a.accountid from accountejb a where a.profile_userid = ?), 1003, 1007)
called
[jcc][Time:2012-11-16-19:55:13.627][Thread:WebSphere WLM Dispatch Thread
t=007bd580][Connection@50fc259c]prepareStatement () returned com.ibm.db2.jcc.t4.k@5fe94b19
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 0.14775ms | network: 0.0ms
                                                                           | server: 0.0ms
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-19:55:13.627][Thread:WebSphere WLM Dispatch Thread
t=007bd580][PreparedStatement@5fe94b19]executeQuery () called
[jcc][Time:2012-11-16-19:55:13.627][Thread:WebSphere WLM Dispatch Thread
t=007bd580][Connection@50fc259c]isClosed () returned false
[jcc][Time:2012-11-16-19:55:13.627][Thread:WebSphere WLM Dispatch Thread
t=007bd580][T4XAResource@79204cc2]makeEntryCurrent(new,old) (0, 0) called
[jcc][Time:2012-11-16-19:55:13.627][Thread:WebSphere WLM Dispatch Thread
t=007bd580][Connection@50fc259c]setDB2ClientUser (TraderClientUser) called
[jcc][Time:2012-11-16-19:55:13.627][Thread:WebSphere WLM Dispatch Thread
t=007bd580][Connection@50fc259c]setDB2ClientWorkstation (TraderClientWorkstation) called
[jcc][Time:2012-11-16-19:55:13.627][Thread:WebSphere WLM Dispatch Thread
t=007bd580][Connection@50fc259c]setDB2ClientApplicationInformation (TraderClientApplicationInformation)
called
[jcc][Time:2012-11-16-19:55:13.628][Thread:WebSphere WLM Dispatch Thread
t=007bd580][Connection@50fc259c]setDB2ClientAccountingInformation (TraderClientAccountingInformation)
called
[jcc][Time:2012-11-16-19:55:13.629][Thread:WebSphere WLM Dispatch Thread
t=007bd580][PreparedStatement@5fe94b19]executeQuery () returned com.ibm.db2.jcc.t4.i@8edf7393
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 2.292ms
                                                 | network: 1.211875ms | server: 0.867ms [STMT@1609124633]
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-19:55:13.629][Thread:WebSphere WLM Dispatch Thread
t=007bd580][ResultSet@8edf7393]next () called
[jcc][Time:2012-11-16-19:55:13.630][Thread:WebSphere WLM Dispatch Thread
t=007bd580][ResultSet@8edf7393]next () returned true
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 0.04924999999999995ms |
network: 0.0ms | server: 0.0ms [STMT@1609124633]
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-19:55:13.630][Thread:WebSphere WLM Dispatch Thread
t=007bd580][ResultSet@8edf7393]getString (3) called
[jcc][Time:2012-11-16-19:55:13.630][Thread:WebSphere WLM Dispatch Thread
t=007bd580][ResultSet@8edf7393]getString () returned buy
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 0.039125ms | network:
0.0ms | server: 0.0ms [STMT@1609124633]
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-19:55:13.630][Thread:WebSphere WLM Dispatch Thread
t=007bd580][ResultSet@8edf7393]getString (4) called
[jcc][Time:2012-11-16-19:55:13.630][Thread:WebSphere WLM Dispatch Thread
t=007bd580][ResultSet@8edf7393]getString () returned closed
```

```
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 0.027999999999999997ms |
network: 0.0ms | server: 0.0ms [STMT@1609124633]
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-19:55:13.630][Thread:WebSphere WLM Dispatch Thread
t=007bd580][ResultSet@8edf7393]getString (10) called
[jcc][Time:2012-11-16-19:55:13.630][Thread:WebSphere WLM Dispatch Thread
t=007bd580][ResultSet@8edf7393]getString () returned s:168
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 0.034374999999999996ms |
network: 0.0ms | server: 0.0ms [STMT@1609124633]
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-19:55:13.630][Thread:WebSphere WLM Dispatch Thread
t=007bd580][Connection@50fc259c]prepareStatement (update orderejb set orderstatus = ?, completiondate = ?
where orderid = ?, 1003, 1007) called
[jcc][Time:2012-11-16-19:55:13.630][Thread:WebSphere WLM Dispatch Thread
t=007bd580][Connection@50fc259c]prepareStatement () returned com.ibm.db2.jcc.t4.k@a34f491
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 0.095625ms | network:
0.0ms | server: 0.0ms
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-19:55:13.630][Thread:WebSphere WLM Dispatch Thread
t=007bd580][PreparedStatement@a34f491]executeUpdate () called
[jcc][Time:2012-11-16-19:55:13.631][Thread:WebSphere WLM Dispatch Thread
t=007bd580][PreparedStatement@a34f491]executeUpdate () returned 1
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 0.9506249999999999ms |
network: 0.726625ms | server: 0.502ms [STMT@171242641]
...
```

Each invocation starts with a `[jcc][SystemMonitor:start]` entry and ends with an `[SystemMonitor:stop]` entry followed by the information of each of the components (core driver, the network, and the database server):

```
core: 0.022ms | network: 0.0ms | server: 0.0ms
```

This entry did not result to a call to the database, as the server time (and network time; this example uses type 4 connectivity) are zero. It indicates which method was called (**setAutoCommit (false)**, in this case).

You can get a good idea about what the application is requesting from the database and how long it took to perform actions in each of the components.

For the `executeQuery()` invocation in Example 9-13 on page 473, the request was sent to the database, and the call spent `core: 2.292ms | network: 1.211875ms | server: 0.867ms`, or 1.080125 (2.292 - 1.211875) ms in the driver, 0.344875(1.211875-0.867) ms in the network, and 0.867 ms in the database engine. In this case, these are reasonable numbers. However, when you run into a problem situation, this is an easy way to find calls that took a long time to complete and immediately see whether the time was spent in the driver, the network, or the database engine.

9.3.3 DB2 commands

Besides traces, there are also many DB2 for z/OS commands that can be helpful when you are analyzing problems. Here are the two most used commands:

- **DISPLAY DATABASE**
- **DISPLAY THREAD**

This section introduces the general usage of these commands. For more information, see *DB2 10 for z/OS Command Reference*, SC19-2972.

DISPLAY DATABASE command

The **-DISPLAY DATABASE** command can show information about the status and usage of the DB2 database objects (table spaces, partitions, indexes, and index partitions) in that database. You cannot display the status of a particular table, only the table space it is in.

The **-DISPLAY DATABASE** command has a number of options that allow you to display different types of information about the database. Here are the keywords that are most often used when dealing with concurrency issues:

- **USE:** You can use this option to quickly check whether a certain transaction, job, and so on is accessing (holding locks or waiting for them on) the displayed object. The command output shows information, such as the connection-IDs, correlation-IDs, authorization IDs, LUW-ID, and location of any threads accessing the local database.
- **RESTRICT:** This option lists the objects that are in a restricted status, which typically prevents an application from accessing the object. When the system is not performing well and is generating many DSNT500I (resource unavailable) messages, make sure that no objects are in a restricted state that can prevent transactions, batch jobs, or utilities from accessing the table or index spaces.
- **CLAIMERS:** This option lists the claims on objects whose status is displayed, and information that allows you to identify who acquired the claim, such as the LUW-ID and location of any remote threads accessing the local database, and the connection-IDs, correlation-IDs, and authorization IDs, as well as the agent token number for the claim, if the claim is local. You can then match the token with the output of the **-DIS THREAD** command to obtain more information about the thread.
- **LOCKS:** This option provides you with information about the parent transaction locks (L-locks) for objects whose statuses are displayed, the drain locks for a resource that is held by running jobs, and the page set or partition physical locks (P-locks) for a resource. It also presents thread identification information, so you can match it with the output of the **-DIS THREAD** command.

DISPLAY THREAD command

The **-DISPLAY THREAD** command can be helpful in identifying users that are active in the system. For distributed threads, you also have the information about the workstation name, the client user ID, and the application name. Here are some of the commonly used options:

- **SCOPE:** The default value is LOCAL. If you want to check all the threads in a DB2 data sharing group, use **SCOPE(GROUP)** instead.
- **TYPE:** Indicates the type of thread that you want to display, such as ACTIVE, INDOUBT, INACTIVE, and SYSTEM. For example, when you use a two-phase commit protocol, if DB2 or a transaction manager has a problem and cannot not automatically resolve the indoubt status with the commit coordinator, you can use the **INDOUBT** option to display thread information and then recover it manually.
- **LUWID:** Displays information about the distributed threads that have the specified LUWID. You can also specify a thread token here, which is a 1 - 6 digit decimal number that appears after the equal sign in all DB2 messages that display a LUWID.

In Example 9-14, we can see both local and distributed threads in the whole data sharing group.

Example 9-14 -DIS THREAD(*) SCOPE(GROUP) output

```
DSNV401I  -DOZ2 DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -DOZ2 ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
TSO      T  *     3 DB2R6        DB2R6             00B1 12157
```

```
 V437-WORKSTATION=TSO, USERID=DB2R6,
      APPLICATION NAME=DB2R6
RRSAF    T    12638 D0Z2ADMT_DMN D0Z2ADMT ?RRSAF   009C     2
 V437-WORKSTATION=RRSAF, USERID=D0Z2ADMT,
      APPLICATION NAME=D0Z2ADMT_DMN
RRSAF    T      165 D0Z2ADMT_II  D0Z2ADMT ?RRSAF   009C     7
 V437-WORKSTATION=RRSAF, USERID=D0Z2ADMT,
      APPLICATION NAME=D0Z2ADMT_II
DISPLAY ACTIVE REPORT COMPLETE
DSNV473I  -D0Z2 ACTIVE THREADS FOUND FOR MEMBER: D0Z1
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
RRSAF    T    25304 D0Z1ADMT_DMN D0Z1ADMT ?RRSAF   0095     2
 V437-WORKSTATION=RRSAF, USERID=D0Z1ADMT,
      APPLICATION NAME=D0Z1ADMT_DMN
RRSAF    T      225 D0Z1ADMT_II  D0Z1ADMT ?RRSAF   0095     7
 V437-WORKSTATION=RRSAF, USERID=D0Z1ADMT,
      APPLICATION NAME=D0Z1ADMT_II
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I  -D0Z2 DSNVDT '-DIS THREAD' NORMAL COMPLETION
```

The command is issued on D0Z2, so those threads are displayed first. (The thread token is also displayed). Next are the threads on the other members, D0Z1 in this case.

Identifying a pending thread with the DISPLAY command

As described in “DISPLAY DATABASE command” on page 475, the **-DISPLAY DATABASE** command provides some information about a thread, including the thread token. You can use the thread token from the **-DISPLAY DATABASE** command output to match it with the thread token from the **-DISPLAY THREAD** output.

For example, when a thread is accessing an object, you cannot perform an **ALTER** or **DROP** operation against the object. If the SQL accessing the object is a dynamic SQL statement, an `SQLCODE -904` with a reason code 00E70081 is issued by DB2, as shown in Example 9-15. This is a common issue when an application does not **COMMIT** in a timely manner.

Example 9-15 Resource unavailable at ALTER TABLE time

```
ALTER TABLE DB2R6.ACT ALTER ACTDESC SET DATA TYPE VARCHAR(30);
---------+---------+---------+---------+---------+---------+---------+---------+
DSNT408I SQLCODE = -904, ERROR:  UNSUCCESSFUL EXECUTION CAUSED BY AN
         UNAVAILABLE RESOURCE. REASON 00E70081, TYPE OF RESOURCE 00000A00, AND
         RESOURCE NAME DB2R6.ACT
DSNT418I SQLSTATE   = 57011 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXIDMH SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = 15 0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'0000000F'  X'00000000'  X'00000000'  X'FFFFFFFF'
         X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION

00E70081 Explanation:
A DROP or ALTER statement was issued but the object cannot be dropped or altered.
The object is referenced by a prepared dynamic SQL statement that is currently
stored in the prepared statement cache and is in use by an application.
```

You can use the resource name from the message, to check who is accessing table DB2R6.ACT by running **-DISPLAY DATABASE(DSN00023) SPACENAM(ACT) USE/CLAIMERS**. The output of the command is shown in Example 9-16.

Example 9-16 -DIS DB(DSN00023) SP(ACT) USE output

```
DSNT360I -D0Z2 ***********************************
DSNT361I  -D0Z2 *  DISPLAY DATABASE SUMMARY
                *    GLOBAL USE
DSNT360I  -D0Z2 ***********************************
DSNT362I  -D0Z2     DATABASE = DSN00023  STATUS = RW
                   DBD LENGTH = 4028
DSNT397I  -D0Z2
NAME     TYPE PART  STATUS            CONNID   CORRID       USERID
-------- ---- ----- ----------------- -------- ------------ --------
ACT      TS    0001 RW                SERVER   db2jcc_appli DB2R6
 G97B8F7B.FEF9.CA4D6EA21017=140295 ACCESSING DATA FOR
 ::9.123.143.123
   -                   MEMBER NAME D0Z1
ACT      TS
******* DISPLAY OF DATABASE DSN00023 ENDED      **********************
DSN9022I  -D0Z2 DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
```

We can see one thread is accessing the table. Its LUWID is `G97B8F7B.FEF9.CA4D6EA21017` and the token is `140295`, which can be used to narrow down the scope of the display thread command, as shown in Example 9-17.

Example 9-17 -DIS THREAD(*) SCOPE(GROUP) LUWID(140295) output

```
DSNV401I  -D0Z2 DISPLAY THREAD REPORT FOLLOWS -
DSNV419I  -D0Z2 NO CONNECTIONS FOUND
DSNV473I  -D0Z2 ACTIVE THREADS FOUND FOR MEMBER: D0Z1
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
SERVER   RA *     4 db2jcc_appli DB2R6    DISTSERV 0084 140295
 V437-WORKSTATION=IBM-M0666QA2EQE, USERID=DB2R6,
      APPLICATION NAME=db2jcc_application
 V445-G97B8F7B.FEF9.CA4D6EA21017=140295 ACCESSING DATA FOR
  ::9.123.143.123
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I  -D0Z2 DSNVDT '-DIS THREAD' NORMAL COMPLETION
```

In this case, we must go and talk to user DB2R6 who is using workstation IBM-M0666QA2AQE to see what the `db2jcc_application` is doing that results in the resource unavailable message.

9.4 Typical problem scenario: Deadlock

Many performance-related problems are caused by lock contention, especially in a WebSphere Application Server or Java environment, as Java applications do not always know the implications that their application design has on the database server.

This section describes how to analyze a DB2 deadlock problem. It is also applicable to other types of concurrency problems, such as timeouts or long suspension times.

9.4.1 Analyzing the Servant log and DB2 MSTR job log

When an application that is running in WebSphere Application Server gets into a deadlock situation, for example, the application receives an `SQLCODE -913/-911`, which typically shows up in the WebSphere Application Server servant log. Example 9-18 shows a deadlock incident in server MZSR014.

Example 9-18 WebSphere Application Server Servant log

```
SDSF OUTPUT DISPLAY MZSR014S STC05379 DSID   103 LINE 601     COLUMNS 55- 134
 COMMAND INPUT ===>                                            SCROLL ===> CSR

BossLog: { 0283} 2012/10/10 09:26:33.986 03 SYSTEM=SC64 CELL=MZCELL NODE=MZNODE4
CLUSTER=MZSR01 SERVER=MZSR014  PID=0X0400A1 TID=0X257679000000004F t=7BAE88 c=UNK
./bbgrjtr.cpp+733 tag= ... Error: TradeDirect:getAccountProfileData -- error
getting profile data com.ibm.db2.jcc.am.SqlException: DB2 SQL Error: SQLCODE=-913,
SQLSTATE=57033, SQLERRMC=00C90088;00000302;DBTR8074.TSACPREJ.X'000003',
DRIVER=3.64.82 org.apache.geronimo.samples.daytrader.util.Log.error
```

When this condition occurs, there is also in information that is written to the DB2 MSTR STC's JOBLOG, and to the SYSLOG.

There should be a DSNT375I or DSNT376I message, which is accompanied by DSNT501I messages, as shown in Example 9-19. (Reason code 00C90088 indicates a deadlock, but 00C9008E indicates a timeout condition.)

Example 9-19 Message in DB2 MSTR JOBLOG

```
SDSF OUTPUT DISPLAY DOZ2MSTR STC03395  DSID    2 LINE  CHARS '05.41.29' FOUND
 COMMAND INPUT ===>                                       SCROLL ===> CSR
05:26:33.98 STC03395  DSNT375I  -DOZ2 PLAN=DISTSERV WITH 773
   773          CORRELATION-ID=db2jcc_appli
   773          CONNECTION-ID=SERVER
   773          LUW-ID=G90C048E.C25C.CA4C1542DE25=12053
   773
   773   THREAD-INFO=RAJESH:TraderClientWorkst:TraderClientUser:TraderClientAp
   773   plicationInformati:STATIC:583:*:*
   773          IS DEADLOCKED WITH PLAN=DISTSERV WITH
   773          CORRELATION-ID=db2jcc_appli
   773          CONNECTION-ID=SERVER
   773          LUW-ID=G90C0609.C36D.CA4C155D1D64=140244
   773
   773   THREAD-INFO=RAJESH:TraderClientWorkst:TraderClientUser:TraderClientAp
   773   plicationInformati:STATIC:3521:*:*
   773          ON MEMBER DOZ1

05:26:33.98 STC03395 00000090  DSNT501I  -DOZ2 DSNILMCL RESOURCE UNAVAILABLE 774
   774            CORRELATION-ID=db2jcc_appli
   774            CONNECTION-ID=SERVER
   774            LUW-ID=G90C048E.C25C.CA4C1542DE25=673975
   774            REASON 00C90088
   774            TYPE 00000302
   774            NAME DBTR8074.TSACPREJ.X'000003'
```

Tip: The WebSphere Application Server Servant log uses GMT time, while the DB2 MSTR uses local time.

DSNT375I and DSNT376I messages contain information about the threads that are involved in the deadlock or timeout. `MEMBER` is the name of the DB2 member where the thread is executing. `THREAD-INFO` is presented in a colon-delimited list that contains the following segments.

- The primary authorization ID that is associated with the thread.
- The name of the user's workstation.
- The ID of the user.
- The name of the application.
- The statement type for the previously run statement: dynamic or static.
- The statement identifier for the currently executing statement, if available. The statement identifier can be used to identify the particular SQL statement.
- The name of the role that is associated with the thread.
- The correlation token that can be used to correlate work at the remote system with work that is performed at the DB2 subsystem.

A DSNT501I message indicates the resource name, type, and reason code.

For more information about the message, see *DB2 10 for z/OS Messages*, GC19-2979.

9.4.2 Analyzing the deadlock trace record

Generally, a DB2 system is running with the recommended statistics traces active all the time. Those traces include IFCID 172 (deadlock) and IFCID 196 (timeout) trace records. You can use an OMPE lockout trace to format these trace records by running the following command:

```
DB2PM LOCKING
     TRACE    LEVEL(LOCKOUT)
EXEC
```

The output is shown in Example 9-20.

Example 9-20 Deadlock lockout trace

```
PLANNAME CONNECT               RELATED TIMESTAMP EVENT    TYPE      NAME                    EVENT SPECIFIC DATA
------------------------------ ----------------- -------- --------- ----------------------- ----------------------------------------
RAJESH   db2jcc_a DRDA         09:26:58.98205191 DEADLOCK                                   COUNTER =1445K     WAITERS =    2
RAJESH   ppli     CA4C1542DE25 N/P                                                          TSTAMP  =10/10/12 09:26:58.98
DISTSERV SERVER                                           DATAPAGE  DB  =DBTR8074           HASH    =X'00062403'
ENDUSER :TraderClientUser                                           OB  =10                 ---------- BLOCKER is HOLDER -----------
WSNAME  :TraderClientWorkst                                         PAGE=X'000003'          LUW=G90C0609.C36D.CA4C155D1D64
TRANSACT:TraderClientApplicationInformati                                                   MEMBER  =DOZ1      CONNECT =SERVER
                                                                                            PLANNAME=DISTSERV  CORRID  =db2jcc_appli
                                                                                            DURATION=MANUAL    PRIMAUTH=RAJESH
                                                                                            STATE   =S         STMTINFO=STATIC
                                                                                            ENDUSER =TraderClientUser
                                                                                            WSNAME  =TraderClientWorkst
                                                                                            TRANSAC=TraderClientApplicationInformati
                                                                                            PROGNAME=SYSLN300
                                                                                            COLLID  =NULLID
                                                                                            LOCATION=N/P
                                                                                            CONTOKEN=X'5359534C564C3031'
                                                                                            STMTID  =X'0000000000000DC1'
                                                                                            ---------------- WAITER -------*VICTIM*-
                                                                                            LUW=G90C048E.C25C.CA4C1542DE25
                                                                                            MEMBER  =DOZ2      CONNECT =SERVER
```

```
                                                                                                 PLANNAME=DISTSERV  CORRID  =db2jcc_appli
                                                                                                 DURATION=COMMIT    PRIMAUTH=RAJESH
                                                                                                 REQUEST =CHANGE    WORTH   =   17
                                                                                                 STATE   =X         STMTINFO=STATIC
                                                                                                 ENDUSER =TraderClientUser
                                                                                                 WSNAME  =TraderClientWorkst
                                                                                                 TRANSAC=TraderClientApplicationInformati
                                                                                                 PROGNAME=SYSLN300
                                                                                                 COLLID  =NULLID
                                                                                                 LOCATION=N/P
                                                                                                 CONTOKEN=X'5359534C564C3031'
                                                                                                 STMTID  =X'0000000000000247'
RAJESH   db2jcc_a DRDA                                 DATAPAGE  DB  =DBTR8074                  HASH    =X'0002E403'
RAJESH   ppli     CA4C1542DE25                                   OB  =25                        ---------- BLOCKER is HOLDER --*VICTIM*-
DISTSERV SERVER                                                  PAGE=X'000003'                 LUW=G90C048E.C25C.CA4C1542DE25
ENDUSER :TraderClientUser                                                                        MEMBER  =D0Z2      CONNECT =SERVER
WSNAME  :TraderClientWorkst                                                                      PLANNAME=DISTSERV  CORRID  =db2jcc_appli
TRANSACT:TraderClientApplicationInformati                                                        DURATION=MANUAL    PRIMAUTH=RAJESH
                                                                                                 STATE   =S         STMTINFO=DYNAMIC
                                                                                                 ENDUSER =TraderClientUser
                                                                                                 WSNAME  =TraderClientWorkst
                                                                                                 TRANSAC=TraderClientApplicationInformati
                                                                                                 PROGNAME=SYSLN300
                                                                                                 COLLID  =NULLID
                                                                                                 LOCATION=N/P
                                                                                                 CONTOKEN=X'5359534C564C3031'
                                                                                                 STMTID  =X'0000000000000247'
                                                                                                 ---------------- WAITER ----------------
                                                                                                 LUW=G90C0609.C36D.CA4C155D1D64
                                                                                                 MEMBER  =D0Z1      CONNECT =SERVER
                                                                                                 PLANNAME=DISTSERV  CORRID  =db2jcc_appli
                                                                                                 DURATION=COMMIT    PRIMAUTH=RAJESH
                                                                                                 REQUEST =CHANGE    WORTH   =   18
                                                                                                 STATE   =X         STMTINFO=DYNAMIC
                                                                                                 ENDUSER =TraderClientUser
                                                                                                 WSNAME  =TraderClientWorkst
                                                                                                 TRANSAC=TraderClientApplicationInformati
                                                                                                 PROGNAME=SYSLN300
                                                                                                 COLLID  =NULLID
                                                                                                 LOCATION=N/P
                                                                                                 CONTOKEN=X'5359534C564C3031'
                                                                                                 STMTID  =X'0000000000000DC3'
```

The deadlock record provides more detailed information than is shown in the DB2 MSTR log or SYSLOG.

In this particular case, the following transactions and resources are involved.

- On member D0Z1, thread A (LUW=G90C0609.C36D.CA4C155D1D64) of application "TraderClientApplicationInformati" holdsan S-LOCK on page x'03' on table space DBTR8074.TSACPREJ, and it is waiting for X-LOCK on page x'03' of table space DBTR8074.TSACCEJB.
- On member D0Z2, thread B (LUW=G90C048E.C25C.CA4C1542DE25) from application "TraderClientApplicationInformati" holds an S-LOCK on page x'03' of table space DBTR8074.TSACCEJB, and is waiting for an X-LOCK on page x'03' of table space DBTR8074.TSACPREJ.

Because S-LOCK and X-LOCK are not compatible, these two threads get into a deadlock. The victim of the deadlock is thread B (LUW=G90C048E.C25C.CA4C1542DE25) on D0Z2. The survivor of the deadlock in on D0Z1 is thread A (G90C0609.C36D.CA4C155D1D64).

Note: The time stamp of the deadlock record is **:26:58, while the time stamp of the DSNT375I and SQLCODE -913 is **:26:33, which is a 25 second difference.

This machine is connected to an external timer facility that is configured to use leap seconds (the delta between UTC (Coordinated Universal Time) and UT1 (mean solar time - observed Earth rotation). The DB2 trace records use an STCK time stamp, which is not adjusted for leap seconds. The job log messages use the local time (including the 25 leap seconds).

Unfortunately, OMPE does not allow you to use a `TIMEZONE` that includes seconds (only hours and minutes).

9.4.3 Identifying the application and SQL statements

You can then use the `THREAD-INFO` or `ENDUSER, TRANSAC` information from the DSNT375I message or the deadlock trace record to identify applications that are incurring this deadlock condition.

In addition, the STMTID information that was introduced in DB2 10 indicates the SQL statements that resulted in the deadlock condition.

According to deadlock trace record information from Example 9-20 on page 479, you can conclude the following information:

- On member D0Z1:
  - STMTID x’0DC1' (3521) is holding an S-LOCK on page x'03' of table space DBTR8074.TSACPREJ.
  - STMTID x’0DC3' (3523) is waiting for an X-LOCK on page x'03' of table space DBTR8074.TSACCEJB.
- On member D0Z2:
  - STMTID x’0247' (583) is holding an S-LOCK on page x'03' of table space DBTR8074.TSACCEJB.
  - This statement is waiting for an X-LOCK on page x'03' of table space DBTR8074.TSACPREJ.

The SQL statements are dynamic SQL in our case. Because the dynamic statement cache is enabled, you can use the **`EXPLAIN STMTCACHE`** statement on all members of the data sharing system to extract information from the dynamic statement cache and insert the information into the `DSN_STATEMENT_CACHE_TABLE` table.

Example 9-21 shows how to identity the SQL statements that are involved in our deadlock using `STMT_ID` and `GROUP_MEMBER`.

Example 9-21 Identity deadlocked dynamic SQL from DSN_STATEMENT_CACHE_TABLE

```
SELECT  STMT_ID, HEX(STMT_ID) AS HEX_STMT_ID
       ,GROUP_MEMBER, PRIMAUTH
       ,BIND_ISO, STMT_TEXT
FROM DSN_STATEMENT_CACHE_TABLE
  WHERE (STMT_ID = 3521 AND GROUP_MEMBER = 'D0Z1')
     OR (STMT_ID = 583  AND GROUP_MEMBER = 'D0Z2')
     OR (STMT_ID = 3523 AND GROUP_MEMBER = 'D0Z1')
```

```
---------+---------+---------+---------+---------+---------+--+---------+--
    STMT_ID  HEX_STMT_ID  GROUP_MEMBER             PRIMAUTH     BIND_ISO
---------+---------+---------+---------+---------+---------+--+---------+--
       3521  00000DC1     DOZ1                     RAJESH       RS
       3523  00000DC3     DOZ1                     RAJESH       RS
        583  00000247     DOZ2                     RAJESH       RS
-------+---------+---------+---------+---------+---------+---------+---------+---------+--------
STMT_TEXT
-------+---------+---------+---------+---------+---------+---------+---------+---------+--------
select * from accountprofileejb ap where ap.userid =
        (select profile_userid from accountejb a where a.profile_userid=?)
update accountejb set lastLogin=?, logincount=logincount+1 where profile_userid=?
update accountprofileejb set passwd = ?, fullname = ?, address = ?, email = ?, creditcard = ?
        where userid = (select profile_userid from accountejb a where a.profile_userid=?)
```

The objects that are involved in the deadlock are using page level locking. With the high transaction rate and the rather small number of pages that are involved, page level locking is locking too much data, resulting in this deadlock condition. Changing to row level locking solves the problem.

9.4.4 Getting more information from the record trace

If you want to check the sequence in which the locks are requested and the SQL statements are run by the thread, the DSNT375I message and deadlock record provide the time that the deadlock occurred and the instance numbers of the transactions that are involved. The instance number is part of the LUW-ID that uniquely identifies a transaction, and it can be used as a filter criteria by OMPE:

```
GLOBAL
   TIMEZONE(+4)
   FROM(,05:24:00)
   TO(,05:27:00)
   INCLUDE (INSTANCE(CA4C155D1D64,CA4C1542DE25))
RECTRACE TRACE
   LEVEL(LONG)
EXEC
```

In this report, you can find the deadlock record and work your way back to the start of the SQL statement of each of the transactions and SQL statements that are involved, and then move forward in the trace again to determine in which sequence the locks were acquired, and how that led to the locking problem that you are trying to resolve.

For more information about tracing, see *DB2 9 for z/OS: Resource Serialization and Concurrency Control*, SG24-4725.

A

DB2 administrative task scheduler

In Chapter 4, “DB2 infrastructure setup” on page 99, which describes the DB2 infrastructure of the scenario for this book, we used the administrative task scheduler (ADMT) to trigger batch jobs in the event any of the DB2 members started or stopped. We furthermore used ADMT for autonomic statistics monitoring to trigger RUNSTATS utility executions on objects that have no or outdated statistics. Autonomic statistics monitoring tasks are executable on any member of the data sharing group.

This appendix provides information about the implementation tasks that we performed to put the initial infrastructure in place and then describes the steps that we took to add batch jobs and regular autonomic statistic monitoring tasks to the ADMT task list.

This appendix describes the installation and use of the DB2 administrative task scheduler by detailing these activities in the following sections:

- Implementation
- Administrative scheduler operation
- Using ADMT for DB2STOP, DB2START, and statistics monitoring
- Additional information

A.1 Implementation

The ADMT infrastructure that we implemented in our DB2 data sharing environment is illustrated in Figure A-1.

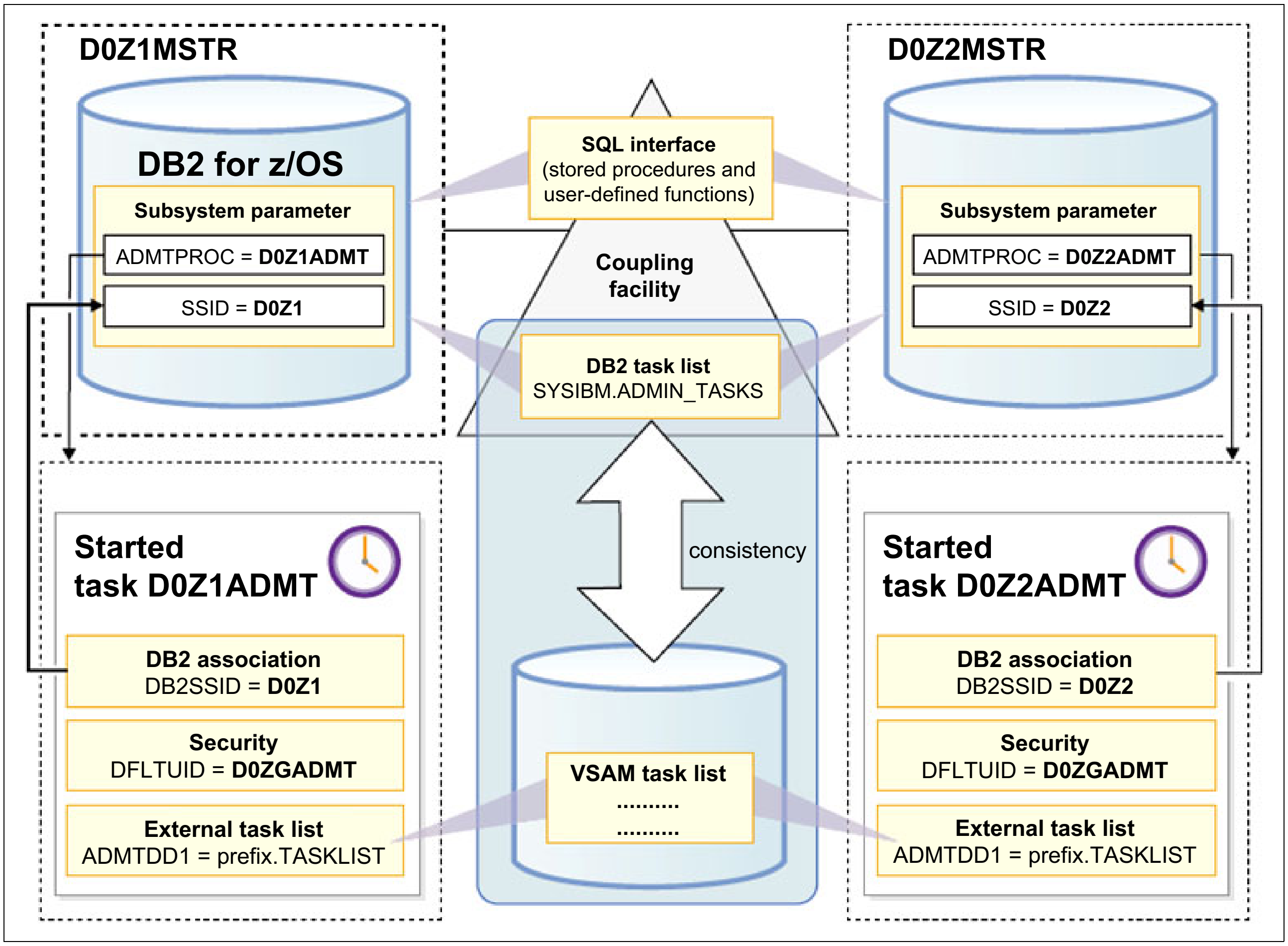


Figure A-1 ADMT data sharing overview

The illustration that is shown in Figure A-2 on page 485 provides an overview of the administration scheduler installation jobs and outlines the implementation tasks.

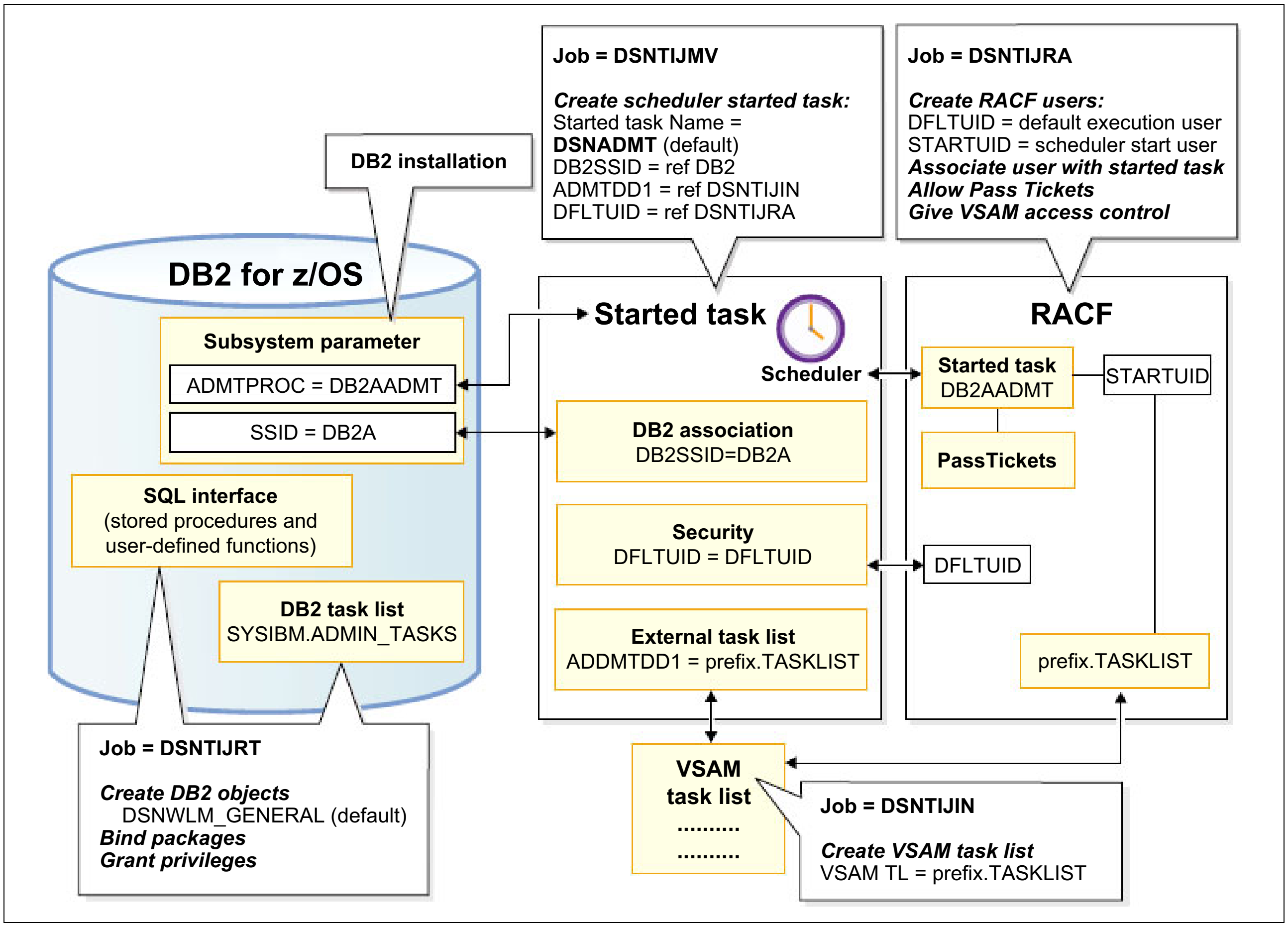


Figure A-2 Overview admin scheduler installation

A.1.1 Installing the DSNTIJMV job

DSNTIJMV creates a template of the administrative task scheduler (ADMT) JCL. You use this template to customize the JCL that you need to run ADMT in your environment. In our example, we installed the JCL procedure that is shown in Example A-1, once for each DB2 member.

Example A-1 ADMT STC JCL

```
//*********************************************************************
//*        JCL FOR PROCEDURE FOR THE STARTUP OF
//*        THE DB2 ADMINISTRATIVE SCHEDULER ADDRESS SPACE.
//*
//*        INSTALLATION MAY CHANGE PROGRAM LIBRARY
//*        NAMES IN STEPLIB DD STATEMENT TO THE
//*        LIBRARY IN WHICH DB2 MODULES ARE
//*        LOADED USING THE PROCEDURE VARIABLE:
//*            LIB
//*
//*        Before using this proc
//*        - Locate and review the settings for the following
//*          parameters:
//*          - DB2SSID: The name of this DB2 subsystem
//*          - DFLTUID: The default ID used by Administrative Scheduler
```

```
//*                     to execute its tasks.  Must differ from
//*                     the ID used to start this address space
//*          - TRACE  : Whether to activate tracing for the Admin-
//*                     istrative Scheduler (OFF or ON, default is OFF)
//*
//*        Following optional parameters of DSNADMT0 may be added:
//*          - MAXTHD:  The maximum number of threads that can execute
//*                     scheduled tasks concurrently. Default is 99
//*          - ERRFREQ: Interval in minutes between the display of
//*                     two successive identical error messages to
//*                     the console. Default is 1
//*          - STOPONDB2STOP: stops the Administrative Scheduler when
//*                           DB2 comes down. No value needed
//*
//*********************************************************************
//D0Z1ADMT PROC LIB='DB0ZT.SDSNLOAD',
//          DB2SSID=D0Z1,
//          DFLTUID=D0ZGADMT,
//          TRACE=ON,
//          MAXTHD=10,
//          MAXHIST=10
//STARTADM EXEC PGM=DSNADMT0,DYNAMNBR=100,REGION=0K,
//          PARM=('DB2SSID=&DB2SSID',
//          ' DFLTUID=&DFLTUID',
//          ' TRACE=&TRACE',
//          ' MAXTHD=&MAXTHD',
//          ' MAXHIST=&MAXHIST')
//STEPLIB  DD DISP=SHR,DSN=DB0ZT.SDSNEXIT
//         DD DISP=SHR,DSN=&LIB
//ADMTDD1  DD DISP=SHR,DSN=DB0ZD.
```

We configured the STC JCL for the D0Z2ADMT started task by using the JCL template that is shown in Example A-1 on page 485, which has a procedure name of D0Z2ADMT and has the DB2SSID JCL parameter set to D0Z2.

A.1.2 Installing the DSNTIJIN job

DSNTIJIN defines the VSAM cluster for the ADMT task list data set that is used across all instances of the administrative task scheduler of our data sharing group. We ran the **DEFINE CLUSTER** command that is shown in Example A-2 to create the VSAM cluster that is used by ADMT STCs D0Z1ADMT and D0Z2ADMT.

Example A-2 ADMT TASKLIST data set - DEFINE CLUSTER

```
DEFINE CLUSTER                              -
       ( NAME(DB0ZD.TASKLIST)             -
         KILOBYTES(40000 40)                  -
         RECORDSIZE(8120 8120)              -
         CISZ(8192)                         -
         NUMBERED                           -
         SHAREOPTIONS(4 3) )                -
     DATA                                   -
       ( NAME(DB0ZD.TASKLIST.DATA) -
                             )
```

A.1.3 Installing the DSNTIJRA job

DSNTIJRA performs the following security-related tasks in RACF.

Defining RACF user IDs

DSNTIJRA defines one user ID for each ADMT started task and one default user ID that is shared across our ADMTs for triggering tasks that we defined in ADMT. The user IDs that we created are shown in the RACF commands that are illustrated in Example A-3.

Example A-3 Create ADMT user IDs

```
/* STC user IDs */
AU   DOZ1ADMT +
     DATA('DEFAULT EXECUTION UID') +
     NAME('DB2 ADMIN SCHEDULER EXECUTION UID') +
     OMVS( UID(0) SHARED PROGRAM(/bin/sh ) HOME(/u/d0z1admt)) +
     DFLTGRP(DB2) +
     OWNER(DB2)
AU   DOZ2ADMT +
     DATA('DEFAULT EXECUTION UID') +
     OMVS( UID(0) SHARED PROGRAM(/bin/sh ) HOME(/u/d0z2admt)) +
     NAME('DB2 ADMIN SCHEDULER EXECUTION UID') +
     DFLTGRP(DB2) +
     OWNER(DB2)
/* ADMT STC default user that is used in STC JCL DFLTUID parm */
AU   DOZGADMT +
     DATA('DEFAULT EXECUTION UID') +
     OMVS( UID(0) SHARED PROGRAM(/bin/sh ) HOME(/u/d0zgadmt)) +
     NAME('DB2 ADMIN SCHEDULER EXECUTION UID') +
     DFLTGRP(DB2) +
     OWNER(DB2)
```

Associating STC user IDs with ADMT STCs

We used the RACF commands that are shown in Example A-4 to associate the STC users that we defined in Example A-3 with their corresponding STC names. We additionally connected each user to RACF group SYS1, as we used that RACF group as a default group for the ADMT STCs. If we had not connected the ADMT users to that group, the ADMT STCs would not have been associated with their user IDs, as defined in the RACF commands of Example A-4.

Example A-4 RACF started class for ADMT

```
CO DOZ1ADMT GROUP(SYS1)
/* associate user DOZ1ADMT with STC name DOZ1ADMT */
RDEF STARTED DOZ1ADMT.* +
     STDATA(USER(DOZ1ADMT) GROUP(SYS1))
CO DOZ2ADMT GROUP(SYS1)
/* associate user DOZ2ADMT with STC name DOZ2ADMT */
RDEF STARTED DOZ2ADMT.* +
     STDATA(USER(DOZ2ADMT) GROUP(SYS1))
SETR REFRESH GENCMD(*) GENERIC(*) RACLIST(STARTED)
```

RACF program control

We used the RACF commands that are shown in Example A-5 to define RACF program control for the ADMT programs.

Example A-5 RACF program control for ADMT

```
SETROPTS WHEN(PROGRAM)
RDEFINE PROGRAM DSNADMT0 +
                ADDMEM('DB0ZT.SDSNLOAD'//NOPADCHK) +
                UACC(READ)
RDEFINE PROGRAM DSNARRS  +
                ADDMEM('DB0ZT.SDSNLOAD'//NOPADCHK) +
                UACC(READ)
RDEFINE PROGRAM DSN3ID00 +
                ADDMEM('DB0ZT.SDSNLOAD'//NOPADCHK) +
                UACC(READ)
SETROPTS WHEN(PROGRAM) REFRESH
```

RACF passtickets for ADMT started tasks

We used the RACF commands that are shown in Example A-6 to allow for RACF passtickets to be used by the ADMT STCs.

Example A-6 RACF passtickets for ADMT STCs

```
/* Activate RACF class PTKTDATA if not yet activated*/
SETROPTS CLASSACT(PTKTDATA)
SETROPTS RACLIST(PTKTDATA)
SETROPTS GENERIC(PTKTDATA) GENCMD(PTKTDATA)
/* set up BPX.DAEMON.HFSCTL FACILITY class if not yet configured */
RDEFINE FACILITY BPX.DAEMON.HFSCTL UACC(NONE)
/* permit ADMT STC uses to read BPX.DAEMON.HFSCTL */
PERMIT BPX.DAEMON.HFSCTL CL(FACILITY) ID(DOZ1ADMT) ACCESS(READ)
PERMIT BPX.DAEMON.HFSCTL CL(FACILITY) ID(DOZ2ADMT) ACCESS(READ)
/* set up BPX.SERVER FACILITY class if not yet configured */
RDEFINE FACILITY BPX.SERVER UACC(NONE)
/* permit ADMT STC users to read BPX.SERVER       */
PERMIT BPX.SERVER       CL(FACILITY) ID(DOZ1ADMT) ACCESS(READ)
PERMIT BPX.SERVER       CL(FACILITY) ID(DOZ2ADMT) ACCESS(READ)
/* set up the BPX.DAEMON FACILITY class if not yet configured */
RDEFINE FACILITY BPX.DAEMON UACC(NONE)
/* permit ADMT STC users to read BPX.DAEMON       */
PERMIT BPX.DAEMON       CL(FACILITY) ID(DOZ1ADMT) ACCESS(READ)
PERMIT BPX.DAEMON       CL(FACILITY) ID(DOZ2ADMT) ACCESS(READ)
/* Define  PTKTDATA profiles STC procedures DOZ1ADMT, DOZ2ADMT */
RDEFINE PTKTDATA IRRPTAUTH.DOZ1ADMT.* UACC(NONE)
RDEFINE PTKTDATA IRRPTAUTH.DOZ2ADMT.* UACC(NONE)
RDEFINE PTKTDATA DOZ1ADMT +
        SSIGNON(KEYMASKED(CACD4AD6D79ECA71)) +
        UACC(NONE) APPLDATA('NO REPLAY PROTECTION')
RDEFINE PTKTDATA DOZ2ADMT +
        SSIGNON(KEYMASKED(CACD4AD6D79ECA71)) +
        UACC(NONE) APPLDATA('NO REPLAY PROTECTION')
/* permit ADMT STC users to access PTKTDATA profiles */
PERMIT  IRRPTAUTH.DOZ1ADMT.* CL(PTKTDATA) +
        ID(DOZ1ADMT) ACCESS(UPDATE)
```

```
PERMIT  IRRPTAUTH.DOZ2ADMT.* CL(PTKTDATA) +
        ID(DOZ2ADMT) ACCESS(UPDATE)
PERMIT  DOZ1ADMT             CL(PTKTDATA) +
        ID(DOZ1ADMT) ACCESS(UPDATE)
PERMIT  DOZ2ADMT             CL(PTKTDATA) +
        ID(DOZ2ADMT) ACCESS(UPDATE)
/* refresh RACF changes */
SETROPTS RACLIST (PTKTDATA) REFRESH
SETROPTS RACLIST (FACILITY) REFRESH
SETROPTS REFRESH GENERIC(*) RACLIST(PTKTDATA)
```

A.1.4 Installing the DSNTIJRT job

DSNTIJRT creates DB2 tables, packages, and stored procedures that are required for DB2 routines that are provided for DB2 administration. This process includes creating and granting the DB2 objects that are required for running the administrative task scheduler. If you do not run DSNTIJRT and the administrative task scheduler starts, the administrative task scheduler issues error message DSNA679I. DSNTIJRT creates the following objects:

- Tables
  - SYSIBM.ADMIN_TASKS
  - SYSIBM.ADMIN_TASKS_HIST
- Temporary tables that are used by stored procedures
  - SYSIBM.BIN_REC_INPUT
  - SYSIBM.BIN_REC_OUTPUT
  - SYSIBM.BUFFERPOOL_STATUS
  - SYSIBM.DATA_SHARING_GROUP
  - SYSIBM.DB_STATUS
  - SYSIBM.DB2_CMD_OUTPUT
  - SYSIBM.DB2_SYSPARM
  - SYSIBM.DB2_THREAD_STATUS
  - SYSIBM.DDF_CONFIG
  - SYSIBM.DSLIST
  - SYSIBM.DSN_SUBCMD_OUTPUT
  - SYSIBM.JES_SYSOUT
  - SYSIBM.JOB_JCL
  - SYSIBM.SERVICE_SQL_OUTPUT
  - SYSIBM.SMS_INFO
  - SYSIBM.SMS_OBJECTS
  - SYSIBM.SYSLOG
  - SYSIBM.SYSTEM_HOSTNAME
  - SYSIBM.TEXT_REC_INPUT
  - SYSIBM.TEXT_REC_OUTPUT
  - SYSIBM.USS_CMD_OUTPUT
  - SYSIBM.UTILITY_JOB_STATUS
  - SYSIBM.UTILITY_OBJECTS
  - SYSIBM.UTILITY_RETCODE
  - SYSIBM.UTILITY_SORT_OBJ
  - SYSIBM.UTILITY_SORT_OUT
  - SYSIBM.UTILITY_STMT
  - SYSIBM.UTILITY_SYSPRINT

- Stored procedures and user-defined table functions

  ADMT uses the following stored procedures for task scheduling and administrative routine enablement:

  – Administrative task scheduler routines
    - DSNADM.ADMIN_TASK_LIST
    - DSNADM.ADMIN_TASK_OUTPUT
    - DSNADM.ADMIN_TASK_STATUS
    - SYSPROC.ADMIN_TASK_ADD
    - SYSPROC.ADMIN_TASK_CANCEL
    - SYSPROC.ADMIN_TASK_REMOVE
    - SYSPROC.ADMIN_TASK_UPDATE
  – Administrative enablement routines
    - SYSPROC.ADMIN_COMMAND_DB2
    - SYSPROC.ADMIN_COMMAND_DSN
    - SYSPROC.ADMIN_COMMAND_UNIX
    - SYSPROC.ADMIN_DS_BROWSE
    - SYSPROC.ADMIN_DS_DELETE
    - SYSPROC.ADMIN_DS_LIST
    - SYSPROC.ADMIN_DS_RENAME
    - SYSPROC.ADMIN_DS_SEARCH
    - SYSPROC.ADMIN_DS_WRITE
    - SYSPROC.ADMIN_INFO_HOST
    - SYSPROC.ADMIN_INFO_SMS
    - SYSPROC.ADMIN_INFO_SQL
    - SYSPROC.ADMIN_INFO_SSID
    - SYSPROC.ADMIN_INFO_SYSLOG
    - SYSPROC.ADMIN_INFO_SYSPARM
    - SYSPROC.ADMIN_JOB_CANCEL
    - SYSPROC.ADMIN_JOB_FETCH
    - SYSPROC.ADMIN_JOB_QUERY
    - SYSPROC.ADMIN_JOB_SUBMIT
    - SYSPROC.ADMIN_UTL_EXECUTE
    - SYSPROC.ADMIN_UTL_MODIFY
    - SYSPROC.ADMIN_UTL_MONITOR
    - SYSPROC.ADMIN_UTL_SCHEDULE
    - SYSPROC.ADMIN_UTL_SORT
    - SYSPROC.DSN_WLM_APPLENV
    - SYSPROC.GET_CONFIG
    - SYSPROC.GET_MESSAGE
    - SYSPROC.GET_SYSTEM_INFO

- Packages

  – DSNADM.DSNADMDW
  – DSNADM.DSNADMGC
  – DSNADM.DSNADMGU
  – DSNADM.DSNADMGV
  – DSNADM.DSNADMGW
  – DSNADM.DSNADMIH
  – DSNADM.DSNADMIV
  – DSNADM.DSNADMIZ
  – DSNADM.DSNADMJF
  – DSNADM.DSNADMJP
  – DSNADM.DSNADMJQ
  – DSNADM.DSNADMJS
  – DSNADM.DSNADMSB

– DSNADM.DSNADMSS
– DSNADM.DSNADMTA
– DSNADM.DSNADMTC
– DSNADM.DSNADMTD
– DSNADM.DSNADMTH
– DSNADM.DSNADMTL
– DSNADM.DSNADMTO
– DSNADM.DSNADMTR
– DSNADM.DSNADMTS
– DSNADM.DSNADMTU
– DSNADM.DSNADMUM
– DSNADM.DSNADMUS
– DSNADMSI.DSNADMSI

A.1.5 ADMTPROC DSNZPARM

The **ADMTPROC DSNZPARM** contains the JCL procedure that is used to start the DB2 administrative task scheduler that is associated with the DB2 member. To disable the scheduler, provide a blank value for this parameter. In our environment, we configured the following JCL procedure names in **ADMTPROC DSNZPARM**:

- Member D0Z1: D0Z1ADMT
- Member D0Z2: D0Z2ADMT

A.2 Administrative scheduler operation

Figure A-1 on page 484 provides an architecture overview of operating ADMT in data sharing.

In data sharing, ADMT provides one administrative scheduler STC per DB2 member, with each ADMT instance running in the same LPAR as its corresponding DB2 member. The ADMT STC names are unique across the data sharing group. In our environment, each ADMT STC uses its own STC user, which must be different from the user that is specified in the **DFLTUID** parameter of the STC JCL. The ADMT STCs share one VSAM tasklist data set and the ADMT DB2 tables.

A.2.1 Starting ADMT

ADMT is started by DB2 during startup and stopped manually unless you provide the ADMT start **STOPONDB2STOP** parameter during ADMT start, as shown in Example A-7. If you use that parameter, ADMT is stopped as part of the DB2 shutdown.

Example A-7 ADMT parameter STOPONDB2STOP

```
//STARTADM EXEC PGM=DSNADMT0,DYNAMNBR=100,REGION=0K,
// PARM=('DB2SSID=&DB2SSID',
// ' DFLTUID=&DFLTUID',
// ' TRACE=&TRACE'
// ' MAXTHD=&MAXTHD'
// ' ERRFREQ=1440'
// ' STOPONDB2STOP')
```

In our environment, ADMT is not stopped with the DB2 shutdown because we do not use the **STOPONDB2TOP** parameter. Upon a successful ADMT start, we observed the runtime messages that are shown in Figure A-3.

```
DSNA671I  DSNA6MAI THE ADMIN SCHEDULER DOZ1ADMT IS STARTING
DSNA672I  DSNA6MAI START COMMAND FOR ADMIN SCHEDULER DOZ1ADMT NORMAL COMPLETION
```

Figure A-3 ADMT start messages

When you stop DB2, ADMT loses its connection to DB2 and writes out the message that is shown in Figure A-4.

```
DSNA679I DSNA6BUF THE ADMIN SCHEDULER DOZ2ADMT CANNOT ACCESS TASK LIST
SYSIBM.ADMIN_TASK
DB2 CODE X'00F30002' IN IFI IDENTIFY
```

Figure A-4 ADMT DB2 unavailable message

A.2.2 Manually operating ADMT

You can stop and start ADMT any time and you can use ADMT modify commands to change its runtime behavior. In our environment, we used the commands that are shown in Example A-8 to operate the ADMT tasks manually for both DB2 members.

Example A-8 Commands operating ADMT

```
RO SC63,S DOZ1ADMT /* start DOZ1ADMT in SC63 */
RO SC64,S DOZ2ADMT /* start DOZ2ADMT in SC64 */
RO SC63,S DOZ1ADMT /* stop DOZ1ADMT in SC63 */
RO SC64,S DOZ2ADMT /* stop DOZ2ADMT in SC64 */
RO SC63,F DOZ1ADMT,appl=shutdown /* stop DOZ1ADMT in SC63 */
RO SC64,F DOZ2ADMT,appl=shutdown /* stop DOZ1ADMT in SC64 */
RO SC63,F DOZ1ADMT,appl=trace=on /* start trace in DOZ1ADMT in SC63 */
RO SC64,F DOZ2ADMT,appl=trace=on /* start trace in DOZ2ADMT in SC64 */
RO SC63,F DOZ1ADMT,appl=trace=off /* stop trace in DOZ1ADMT in SC63 */
RO SC64,F DOZ2ADMT,appl=trace=off /* stop trace in DOZ2ADMT in SC64 */
```

A.3 Using ADMT for DB2STOP, DB2START, and statistics monitoring

We used the administrative scheduler to trigger batch jobs in case a DB2 member is stopped or started. We used the administrative scheduler to run the RUNSTATS utility on objects that have no or outdated statistics. To implement this functionality, we completed the following tasks:

1. Create a REXX exec library for storing the @OSCMDS REXX program that runs DB2 commands during DB2STOP event processing.
2. Create a Sysplex-wide JES2 include library for job skeletons to be used across both ADMT instances.
3. Create LPAR-specific JCL libraries to include JCL members to cater to the system affinity of an ADMT-submitted batch JCL.

4. Call the ADMIN_TASK_ADD stored procedure to add DB2START and DB2STOP job submission tasks for both members of the data sharing group.
5. Call the ADMIN_TASK_ADD stored procedure to add calls to the ADMIN_UTL_MONITOR stored procedure to monitor and resolve outdated statistics on user objects and on the DSNDB06.SYSTSKEYS table space.

A.3.1 DB2START processing

With ADMT, you can define tasks for DB2START event processing. In the example that is shown in Example A-9, we start the SYSPROC.ADMIN_TASK_ADD stored procedure to direct ADMT to submit JCL member D0Z1STRT of JCL library DB0ZM.D0ZGADMT.JCL whenever member D0Z1 is started. We ran a similar SQL call statement to enable DB2START processing for DB2 member D0Z2.

Example A-9 ADMT DB2START ADMIN_TASK_ADD invocation

```
CALL SYSPROC.ADMIN_TASK_ADD
  (NULL,NULL,NULL,NULL,
  NULL,NULL,NULL,'DB2START',NULL,NULL,'D0Z1',
   NULL,NULL,NULL,'DB0ZM.D0ZGADMT.JCL','D0Z1STRT','YES',
   'D0Z1STRT','D0Z1 START',?,?)
```

Upon successful completion, we used the ADMIN_TASK_LIST user-defined function (UDF) to list the DB2START events that are registered in the administrative scheduler. The query result is shown in Figure A-5.

```
SELECT
      substr(TRIGGER_TASK_NAME,1,8) as TASKNAME
    , DB2_SSID
    , SUBSTR(JCL_LIBRARY,1,18) AS JCL_LIBRARY
    , JCL_MEMBER
    , JOB_WAIT
    , TASK_NAME
    , DESCRIPTION
    , CREATOR
    , LAST_MODIFIED
FROM table(DSNADM.ADMIN_TASK_LIST()) as tasklist
WHERE TRIGGER_TASK_NAME = 'DB2START';
---------+---------+---------+---------+---------+---------+---------+---
TASKNAME  DB2_SSID  JCL_LIBRARY        JCL_MEMBER  JOB_WAIT  TASK_NAME
---------+---------+---------+---------+---------+---------+---------+---
DB2START  D0Z1      DB0ZM.D0ZGADMT.JCL  D0Z1STRT    YES       D0Z1STRT
DB2START  D0Z2      DB0ZM.D0ZGADMT.JCL  D0Z2STRT    YES       D0Z2STRT
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

Figure A-5 Query DB2START events

JCL library and system affinity

The SQL call statement that is shown in Example A-9 on page 493 references JCL library DB0ZM.D0ZGADMT.JCL as a data set that stores the JCL member. Job submission by ADMT requires affinity with the system in which the ADMT's DB2 member runs; otherwise, job D0Z1STRT fails during DB2 command processing. Job D0Z1STRT ensures system affinity through hardcoded JES2 JCL control statements, which requires extra care in case a DB2 member is moved to a different LPAR. To solve this problem, we created two JCL libraries, one for system SC63 and one for system SC64:

- DB0ZM.SC63.JCL
- DB0ZM.SC64.JCL

We then reference these data sets by defining a common data set alias name that uses the **&SYSNAME** symbolic variable in the data set alias definition. The alias name is identical to the JCL library name that we used in the ADMT task definition in Example A-9 on page 493. With this technique, the alias name references the appropriate system-related JCL data set, depending on the system (SC63 or SC64) from which the reference is made. The define alias control statement that we used is shown in Example A-10.

Example A-10 Define JCL data set alias using symbolicrelate

```
def alias (name('DB0ZM.D0ZGADMT.JCL') symbolicrelate('db0zm.&sysname..jcl')
```

JCL member D0Z1STRT

JCL member D0Z1STR is used to issue a series of DB2 commands that you usually want to run soon after DB2 becomes available. In our scenario, the commands we run have the following purpose:

- Start trace IFCID 318 to enable dynamic statement cache statistics.
- Start audit trace class 10 to capture detail information about authorization failures.
- Issue a **START PROFILE** command to activate the profile that we defined in 4.3.17, "Using DB2 profiles" on page 180.
- Display an activated trace.
- Display the utility status.
- Display the databases in restricted status.
- Display the spaces in restricted status.

The D0Z1STRT JCL that we created for D0Z1 ADMT DB2START processing is shown in Example A-11.

Example A-11 DB2START D0Z1STRT JCL

```
//D0Z1STRT JOB (ZACCTNUM),REGION=0M,
//         CLASS=A,
//         MSGLEVEL=(1,1)
/*JOBPARM S=SC64,L=9999 1
// JCLLIB ORDER=(DB0ZM.D0ZGADMT.INCLUDE) 2
// SET SSID=D0Z1 3
// INCLUDE MEMBER=&SSID.STRT 4
```

1. We coded a JOBPARM JES control statement to define system affinity. The job that is shown in Example A-11 runs on system SC63.
2. The JCLLIB statement refers to a library that is used to include JCL templates that are used in DB2 data sharing across administrative scheduler instances for job submission.

3. We set the SSID variable to the name of the DB2 subsystem ID. The variable is then used for resolving include member names and to pass the DB2 subsystem IDs parameter for JCL and program parameters processing.
4. The JCL shown in Example A-11 on page 494 uses the SSID variable to include the JCL template D0Z1STRT from JCLLIB data set DB0ZM.D0ZGADMT.INCLUDE.

JCL include member D0Z1STRT

Example A-11 on page 494 references include template D0Z1STRT. D0Z1STRT contains a JCL template that contains a JCL job step to run a series of DB2 commands against the DB2 system that are referred to by the SSID variable. The D0Z1STRT JCL template that we use is shown in Example A-12.

Example A-12 JCL template D0Z1STRT

```
//STRT01   EXEC PGM=IKJEFT01,DYNAMNBR=20,TIME=1440,
// PARM='DSN S(&SSID.)' 1
//STEPLIB DD DISP=SHR,DSN=DB0ZT.SDSNEXIT
//        DD DISP=SHR,DSN=DB0ZT.SDSNLOAD
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
-START TRACE (P) CLASS(30) DEST(SMF)  IFCID(318) 2
-START TRACE (AUDIT) CLASS(10) DEST(SMF)
-START PROFILE
-DIS   PROFILE
-DIS TRACE
-DIS UTIL(*)
-DIS DB(*) RESTRICT LIMIT(*)
-DIS DB(*) SPACE(*) RESTRICT LIMIT(*)
END
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
```

1. The SSID variable is passed in by the D0Z1STRT job described in “JCL member D0Z1STRT” on page 494.
2. This part of the JCL shows the DB2 commands that are required to complete the tasks that are described in “JCL member D0Z1STRT” on page 494.

Administrative scheduler runtime messages

When we started DB2 member D0Z1, we observed the ADMT runtime messages that are shown in Figure A-6, which resulted from D0Z1ADMT DB2START processing.

```
 SDSF OUTPUT DISPLAY D0Z1ADMT STC24540  DSID     2 LINE 34      COLS 21- 100
 COMMAND INPUT ===>                                             SCROLL ===> CSR
$HASP100 D0Z1STRT ON INTRDR                             FROM STC24540 D0Z1ADMT
IRR010I  USERID D0ZGADMT IS ASSIGNED TO THIS JOB.
```

Figure A-6 Administrative scheduler DB2START messages

The ADMT trace data provided the information about the execution of D0Z1STRT, as shown in Figure A-7.

```
(IITHD) Receiving DB2 START event
(IIEVENT)  - DB2 Subsystem = "D0Z1"
(IIEVENT)  - Event = -1
(triggerSchedules) entering
(db2_OPEN) threadid=251464000000000D connected]
(TTHD000) Signal received with command = 1
(TTHD000) Execution begins for task = 2
(TTHD000) Execution begins at time 2012-08-12-14.22.47.000000
(TTHD000) num invocations = 6
(TTHD000) PassTicket generated for user = "D0ZGADMT"
(TTHD000) logged in
(TTHD000ÝJ¨) allocating JCL internal reader data set
(TTHD000ÝJ¨) opening JCL data set = "//'DB0ZM.D0ZGADMT.JCL(D0Z1STRT)'"
(TTHD000ÝJ¨) opening JCL internal reader data set
(TTHD000ÝJ¨) writing records to JCL internal reader data set
(TTHD000ÝJ¨) written records = 7
(TTHD000ÝJ¨) closing JCL data set
(TTHD000ÝJ¨) closing JCL internal reader data set
(TTHD000ÝJ¨) deallocating JCL internal reader data set
(TTHD000ÝJ¨) jobid = "JOB24643"
(TTHD000ÝJ¨) JCL job submitted, jobid = "JOB24643"
(TTHD000ÝJ¨) waiting for job status... (TTHD000ÝJ¨) execution duration (in nb polls) = 1
(TTHD000ÝJ¨) status found for JCL job = "JOB24643"
(TTHD000ÝJ¨) max_rc = 0
(TTHD000ÝJ¨) comp_type = 1
(TTHD000) logged out
(TTHD000) Execution status COMPLETED
(TTHD000) Execution ends at time 2012-08-12-14.22.48.000000
```

Figure A-7 Administrative scheduler DB2START trace

Verifying the status of DB2START processing

You can verify the status of DB2START processing by using the `ADMIN_TASK_STATUS` table UDF to query the ADMT status. The result of the query that we ran is provided in Figure A-8.

```
SELECT
      SUBSTR(TASK_NAME,1,8)    AS TASKNAME
    , SUBSTR(STATUS,1,10)      AS STATE
    , NUM_INVOCATIONS AS #INV
    , SUBSTR(CHAR(START_TIMESTAMP),1,19) AS BETS
    , SUBSTR(CHAR(END_TIMESTAMP),1,19)   AS ENTS
    , JOB_ID
    , DB2_SSID AS SSID
FROM table(DSNADM.ADMIN_TASK_STATUS()) as taskstatus
where task_name = 'D0Z1STRT'
---------+---------+---------+---------+---------+---------+---------+---------
TASKNAME  STATE     #INV  BETS                ENTS                JOB_ID   SSID
---------+---------+---------+---------+---------+---------+---------+---------
D0Z1STRT  COMPLETED    6  2012-08-12-14.22.47 2012-08-12-14.22.48 JOB24643 D0Z1
```

Figure A-8 Query DB2START processing status

The query that is shown in Figure A-8 on page 496 provides information about the most recent DB2START event run. You can use the UDF to obtain a history of recent runs. You can limit the number of rows to be returned by passing a numeric input parameter in the UDF interface. An example of such a query and its processing result is illustrated in Figure A-9.

```
SELECT
      SUBSTR(TASK_NAME,1,8)    AS TASKNAME
    , SUBSTR(STATUS,1,10)      AS STATE
    , NUM_INVOCATIONS AS #INV
    , SUBSTR(CHAR(START_TIMESTAMP),1,19) AS BETS
    , SUBSTR(CHAR(END_TIMESTAMP),1,19)   AS ENTS
    , JOB_ID
    , DB2_SSID AS SSID
FROM table(DSNADM.ADMIN_TASK_STATUS(10)) as taskstatus
where task_name = 'D0Z1STRT'
---------+---------+---------+---------+---------+---------+---------+--------
TASKNAME  STATE     #INV BETS                ENTS                JOB_ID   SSID
---------+---------+---------+---------+---------+---------+---------+--------
D0Z1STRT  COMPLETED    1 2012-08-12-02.03.46 2012-08-12-02.03.48 JOB24557 D0Z1
D0Z1STRT  COMPLETED    2 2012-08-12-02.14.27 2012-08-12-02.14.30 JOB24565 D0Z1
D0Z1STRT  COMPLETED    3 2012-08-12-03.36.19 2012-08-12-03.36.20 JOB24584 D0Z1
D0Z1STRT  COMPLETED    4 2012-08-12-03.41.18 2012-08-12-03.41.19 JOB24594 D0Z1
D0Z1STRT  COMPLETED    5 2012-08-12-14.11.31 2012-08-12-14.11.32 JOB24632 D0Z1
D0Z1STRT  COMPLETED    6 2012-08-12-14.22.47 2012-08-12-14.22.48 JOB24643 D0Z1
DSNE610I NUMBER OF ROWS DISPLAYED IS 6
```

Figure A-9 Query DB2START history

A.3.2 DB2STOP processing

With ADMT, you can define tasks for DB2STOP event processing. In Example A-13, we start the SYSPROC.ADMIN_TASK_ADD stored procedure to inform ADMT to submit JCL member D0Z1STOP of JCL library DB0ZM.D0ZGADMT.JCL whenever member D0Z1 is stopped. We ran a similar SQL call statement to enable DB2STOP processing for DB2 member D0Z2.

Example A-13 ADMT DB2STOP ADMIN_TASK_ADD invocation

```
CALL SYSPROC.ADMIN_TASK_ADD
  (NULL,NULL, NULL,NULL,
  NULL,NULL,NULL,'DB2STOP',NULL,NULL,'D0Z1',
   NULL,NULL,NULL,'DB0ZM.D0ZGADMT.JCL','D0Z1STOP','YES',
   'D0Z1STOP','DB0Z1 STOP',?,?)
```

Upon successful completion, we used the ADMIN_TASK_LIST user-defined function (UDF) to list the DB2STOP events that are registered in the administrative scheduler. The query result is shown in Figure A-10.

```
SELECT
      substr(TRIGGER_TASK_NAME,1,8) as TASKNAME
    , DB2_SSID
    , SUBSTR(JCL_LIBRARY,1,18) AS JCL_LIBRARY
    , JCL_MEMBER
    , JOB_WAIT
    , TASK_NAME
    , DESCRIPTION
    , CREATOR
    , LAST_MODIFIED
FROM table(DSNADM.ADMIN_TASK_LIST()) as tasklist
WHERE TRIGGER_TASK_NAME = 'DB2STOP';
---------+---------+---------+---------+---------+---------+---------+-
TASKNAME  DB2_SSID  JCL_LIBRARY        JCL_MEMBER  JOB_WAIT  TASK_NAME
---------+---------+---------+---------+---------+---------+---------+-
DB2STOP   D0Z1      DB0ZM.D0ZGADMT.JCL  D0Z1STOP    YES       D0Z1STOP
DB2STOP   D0Z2      DB0ZM.D0ZGADMT.JCL  D0Z2STOP    YES       D0Z2STOP
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
```

Figure A-10 Query DB2STOP events

JCL member D0Z1STOP

JCL member D0Z1STOP is used to run a series of DB2 commands that you usually want to run when DB2 shuts down. In our scenario, the commands that we run have the following purposes:

- Display the DB2 threads.
- Display an activated trace.
- Display the utility status.
- Display the databases in restricted status.
- Display the spaces in restricted status.

The D0Z1STOP JCL that we created for D0Z1 ADMT DB2STOP processing is shown in Example A-14.

Example A-14 DB2STOP D0Z1STOP JCL

```
//D0Z1STOP JOB (ZACCTNUM),REGION=0M,
//         CLASS=A,
//         MSGLEVEL=(1,1)
/*JOBPARM S=SC63,L=9999
// JCLLIB ORDER=(DB0ZM.D0ZGADMT.INCLUDE)
// SET SSID=D0Z1
// INCLUDE MEMBER=&SSID.STOP
```

The SSID and JOBPARM setting and the JCLLIB statement that are used in Example A-14 are similar to the ones in Example A-11 on page 494. For information about these settings, see Example A-11 on page 494.

JCL include member D0Z1STOP

Example A-14 on page 498 references include template D0Z1STOP. D0Z1STOP contains a JCL template that consists of a JCL job step that runs a series of DB2 commands against the DB2 system that is referred to by the SSID variable. The D0Z1STOP JCL template that we use is shown in Example A-15.

Example A-15 JCL template D0ZASTOP

```
//STOP01   EXEC PGM=IKJEFT01,DYNAMNBR=20,TIME=1440,
// PARM='%@OSCMD' 1
//STEPLIB DD DISP=SHR,DSN=DB0ZT.SDSNEXIT
//        DD DISP=SHR,DSN=DB0ZT.SDSNLOAD
//        DD DISP=SHR,DSN=DB0ZM.RUNLIB.LOAD
//SYSEXEC DD DISP=SHR,DSN=DB0ZM.D0ZGADMT.EXEC 2
//CMDIN   DD DISP=SHR,DSN=DB0ZM.D0ZGADMT.INCLUDE(&SSID.STOC) 3
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
```

1. In DB2STOP processing, you cannot use the TSO batch DSN processor to process DB2 commands because DB2 has been stopped and does not allow for any further work to be submitted through the traditional DB2 interfaces. Thus, we use SDSF REXX to perform DB2 command processing through an operating system console. The logic of the SDSF REXX program is illustrated in Example A-17.
2. The @OSCMD REXX program is stored in the PO data set that is referenced by the **SYSEXEC DD** statement.
3. JCL DD statement CMDIN refers to the data set that contains the DB2 commands to be run through SDSF REXX in case of DB2STOP processing.

CMDIN data set

In our environment, the CMDIN data set contains the DB2 commands that shown in Example A-16.

Example A-16 CMDIN DB2 console commands

```
-D0Z1 DIS TRACE
-D0Z1 DIS THD(*) LIMIT(*)
-D0Z1 DIS UTIL(*)
-D0Z1 DIS DB(*) RESTRICT LIMIT(*)
-D0Z1 DIS DB(*) SPACE(*) RESTRICT LIMIT(*)
```

@OSCMD SDSF REXX program

During DB2STOP processing, ADMT submits a batch job that runs REXX program @OSCMD to process a series of DB2 commands through the z/OS console interface The DB2 commands are provided through the JCL CMDIN DD data set. The REXX program that is run is shown in Example A-17.

Example A-17 @OSCMD REXX program

```
/* REXX */
/*
  Author.........: Josef.Klitsch@ch.ibm.com
  Function.......: Use SDSF REXX to execute z/OS command and
                   send output to standard output
```

```
                   For further details on using SDSF REXX see
                   Implementing REXX support in SDSF, SG24-7419
                   http://www.redbooks.ibm.com/abstracts/sg247419.html

 Input..........: Commands to be executed to be provided in CMDIN DD
*/
trace off
/* set console name to jobname                       */
isfcons = MVSVAR('SYMDEF',JOBNAME )

/* Load the SDSF environment and abort on failure */
IsfRC = isfcalls( "ON" )
if IsfRC <> 0 then do
  say "RC" IsfRC "returned from isfcalls( ON )"
  exit IsfRC
end

/* read commands from CMDIN                        */
call readcmds

/* issue commands and display output               */
do xi=1 to CMDIN.0
   call runcmds
   call displayresponses
   call displaycmdoutput
end


/* Unload the SDSF environment */
call isfcalls "OFF"
exit 0

/* Read commands to be executed from CMDIN DD      */
readcmds:
 ADDRESS TSO,
 "EXECIO * diskr CMDIN (STEM CMDIN. FINIS"
 if RC > 4 then
    do
      say "Error during EXECIO CMDIN"
      exit 12
    end
 do i=1 to CMDIN.0
  CMDIN.i =  "'"||strip(CMDIN.i)||"'"
 end
return

/* issue commands                             */
runcmds:
 address SDSF "isfexec /"||CMDIN.xi
 if RC <> 0 then do
    Say "RC" RC "returned from ..."
    call DisplayMessages
    exit 12
 end
```

```
return

/* Display the user log associated with the action */
displaycmdoutput:
 say isfulog.0 "user log lines"
 do i = 1 to isfulog.0
    say " '"isfulog.i"'"
 end
return
/* Display the responses associated with the action */
displayresponses:
if ifsresp.0 > 0 then
    say isfresp.0 "response lines"
 do i = 1 to isfresp.0
    say " '"isfresp.i"'"
 end
return

/* Display the messages associated with the action */
DisplayMessages:
   if ifsrmsg.0 > 0 then
    do
      say "isfmsg: '"isfmsg"'"
      say isfmsg2.0 "long messages in the isfmsg2 stem:"
    end
   do i = 1 to isfmsg2.0
      say " '"isfmsg2.i"'"
   end
return
```

Administrative scheduler runtime messages

When we stopped DB2 member D0Z1, we observed the ADMT runtime messages that are shown in Figure A-11, which resulted from D0Z1ADMT DB2STOP event processing.

```
$HASP100 D0Z1STOP ON INTRDR                            FROM STC24540 D0Z1ADMT
IRR010I  USERID D0ZGADMT IS ASSIGNED TO THIS JOB.
DSNA679I  DSNA6BUF THE ADMIN SCHEDULER D0Z1ADMT CANNOT ACCESS TASK LIST
SYSIBM.ADMIN_TASK
DB2 CODE X'00F30002' IN IFI IDENTIFY
```

Figure A-11 Administrative scheduler DB2STOP messages

The ADMT trace data provided the information that is shown in Figure A-12 on the execution of D0Z1STOP event processing.

```
(IITHD) Receiving DB2 STOP event
(IIEVENT)  - DB2 Subsystem = "D0Z1"
(IIEVENT)  - Event = 0
(IIEVENT) ending with RC = x00000000
(TTHD000) Signal received with command = 1
(TTHD000) Execution begins for task = 1
(TTHD000) Execution begins at time 2012-08-12-14.20.34.000000
(TTHD000) num invocations = 6
(TTHD000) PassTicket generated for user = "D0ZGADMT"
(TTHD000) logged in
(TTHD000ÝJ¨) allocating JCL internal reader data set
(TTHD000ÝJ¨) opening JCL data set = "//'DB0ZM.D0ZGADMT.JCL(D0Z1STOP)'"
(TTHD000ÝJ¨) opening JCL internal reader data set
(TTHD000ÝJ¨) writing records to JCL internal reader data set
(TTHD000ÝJ¨) written records = 7
(TTHD000ÝJ¨) closing JCL data set
(TTHD000ÝJ¨) closing JCL internal reader data set
(TTHD000ÝJ¨) deallocating JCL internal reader data set
(TTHD000ÝJ¨) jobid = "JOB24638"
(TTHD000ÝJ¨) JCL job submitted, jobid = "JOB24638"
(TTHD000ÝJ¨) waiting for job status...
(modifyStatus) task 1, status=RUNNING on D0Z1 at 16:2012-08-12-14.20.34.000000
(modifyStatus) admin record(current) at 35: incons=1, <3 tasks
(modifyStatus) status successfully updated
(TTHD000ÝJ¨) execution duration (in nb polls) = 6
(TTHD000ÝJ¨) status found for JCL job = "JOB24638"
(TTHD000ÝJ¨) max_rc = 0
(TTHD000ÝJ¨) comp_type = 1
(TTHD000) logged out
(TTHD000) Execution status COMPLETED
(TTHD000) Execution ends at time 2012-08-12-14.20.40.000000
(modifyStatus) task 1, status=COMPLETED on D0Z1 at 17:2012-08-12-14.20.40.000000
(modifyStatus) admin record(current) at 36: incons=1, <3 tasks
(modifyStatus) status successfully updated
```

Figure A-12 Administrative scheduler DB2STOP trace

Verifying the status of DB2STOP processing

You can verify the status of DB2STOP processing by using the ADMIN_TASK_STATUS table UDF for querying the ADMT status. The result of the query that we ran is provided in Figure A-13.

```
SELECT
      SUBSTR(TASK_NAME,1,8)    AS TASKNAME
    , SUBSTR(STATUS,1,10)      AS STATE
    , NUM_INVOCATIONS AS #INV
    , SUBSTR(CHAR(START_TIMESTAMP),1,19) AS BETS
    , SUBSTR(CHAR(END_TIMESTAMP),1,19)   AS ENTS
    , JOB_ID
    , DB2_SSID AS SSID
FROM table(DSNADM.ADMIN_TASK_STATUS()) as taskstatus
where task_name = 'DOZ1STOP'
---------+---------+---------+---------+---------+---------+---------+------
TASKNAME  STATE    #INV BETS                ENTS              JOB_ID   SSID
---------+---------+---------+---------+---------+---------+---------+------
DOZ1STOP  COMPLETE    6 2012-08-12-14.20.3  2012-08-12-14.20.4 JOB24638 DOZ1
```

Figure A-13 Query the DB2STOP processing status

The query that is shown in Figure A-13 provides information about the most recent DB2STOP event run. You can use the UDF also to obtain a history of recent runs. You can limit the number of rows to be returned by passing a numeric input parameter in the UDF interface. An example of such a query and its processing result is illustrated in Figure A-14.

```
SELECT
      SUBSTR(TASK_NAME,1,8)    AS TASKNAME
    , SUBSTR(STATUS,1,10)      AS STATE
    , NUM_INVOCATIONS AS #INV
    , SUBSTR(CHAR(START_TIMESTAMP),1,19) AS BETS
    , SUBSTR(CHAR(END_TIMESTAMP),1,19)   AS ENTS
    , JOB_ID
    , DB2_SSID AS SSID
FROM table(DSNADM.ADMIN_TASK_STATUS(10)) as taskstatus
where task_name = 'DOZ1STOP'
---------+---------+---------+---------+---------+---------+---------+--------
TASKNAME  STATE     #INV BETS                ENTS               JOB_ID   SSID
---------+---------+---------+---------+---------+---------+---------+--------
DOZ1STOP  COMPLETED    2 2012-08-12-02.13.25 2012-08-12-02.13.2 JOB24560 DOZ1
DOZ1STOP  COMPLETED    3 2012-08-12-03.34.05 2012-08-12-03.34.0 JOB24579 DOZ1
DOZ1STOP  COMPLETED    4 2012-08-12-03.38.50 2012-08-12-03.38.5 JOB24588 DOZ1
DOZ1STOP  COMPLETED    5 2012-08-12-03.42.36 2012-08-12-03.42.3 JOB24596 DOZ1
DOZ1STOP  COMPLETED    6 2012-08-12-14.20.34 2012-08-12-14.20.4 JOB24638 DOZ1
DSNE610I NUMBER OF ROWS DISPLAYED IS 5
```

Figure A-14 Query the DB2STOP history

A.3.3 Autonomic statistics monitoring

In our example, we use autonomic statistics monitoring to automatically identify, collect, and maintain accurate statistics in DB2.

For autonomic monitoring, DB2 relies on scheduled calls to the ADMIN_UTL_MONITOR stored procedure to monitor your statistics. When stale, missing, or conflicting statistics are identified, the ADMIN_UTL_EXECUTE stored procedure starts RUNSTATS within defined maintenance windows and resolves the problems. The ADMIN_UTL_EXECUTE stored procedure uses the options that are defined in RUNSTATS profiles to start the RUNSTATS stored procedure. The ADMIN_UTL_MODIFY stored procedure is called at regular intervals to clean up the log file and alert history.

Autonomic statistics and ADMT overview

DB2 uses interactions between the administrative scheduler, certain DB2 -supplied stored procedures, and certain catalog tables for autonomic statistics maintenance.

Figure A-15 illustrates the relationships between the various objects that DB2 uses for autonomic statistics maintenance.

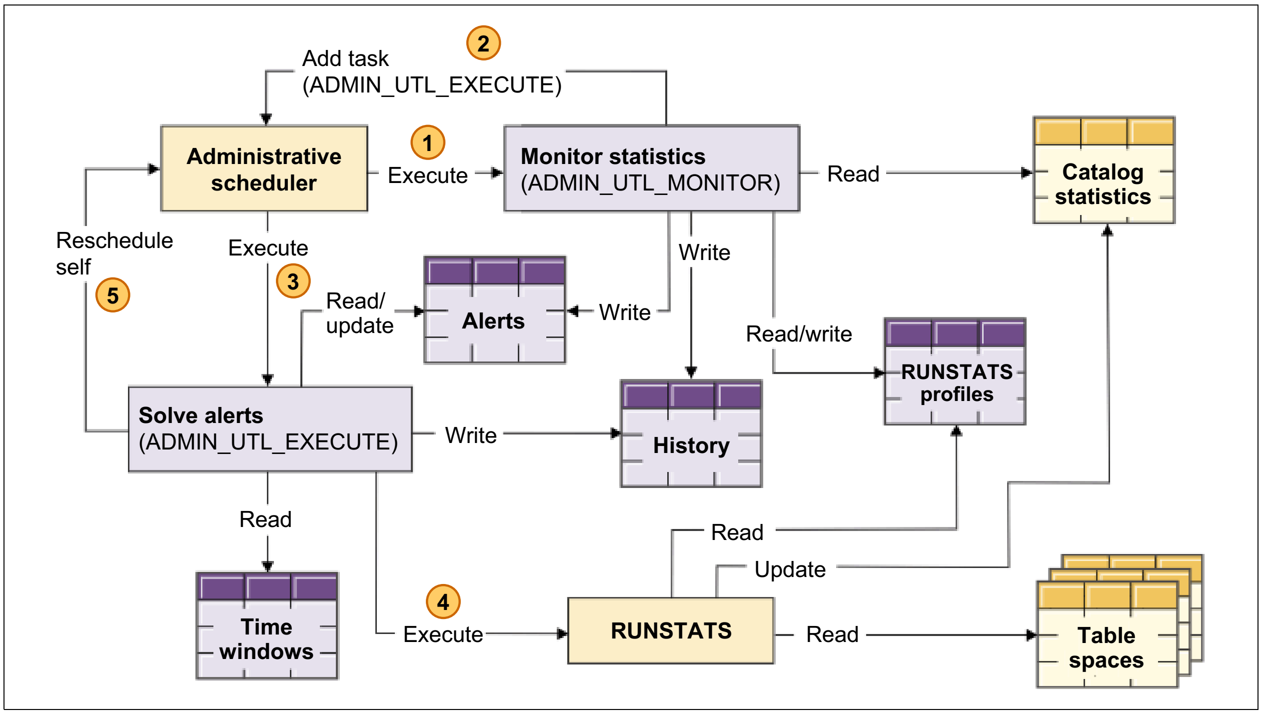


Figure A-15 Object interactions for autonomic statistics maintenance in DB2

DB2 uses the following actions to implement autonomic statistics maintenance:

1. The administrative task scheduler issues calls to the ADMIN_UTL_MONITOR stored procedure according to the schedule that you specify.
2. When the ADMIN_UTL_MONITOR detects missing, out-of-date, or conflicting statistics, it issues a call to the ADMIN_TASK_ADD stored procedure to schedule an immediate run of the ADMIN_UTL_EXECUTE stored procedure.
3. The administrative scheduler calls the ADMIN_UTL_EXECUTE stored procedure.
4. When the call to the ADMIN_UTL_EXECUTE stored procedure occurs within a time window that you specify, it starts the RUNSTATS utility to solve alerts.
5. When the call to the ADMIN_UTL_EXECUTE stored procedure occurs outside of a specified time window, the ADMIN_UTL_EXECUTE stored procedure issues a call to the ADMIN_TASK_ADD stored procedure to reschedule its own execution to the next time window.

Scheduling autonomic statistics monitoring

We scheduled autonomic statistics monitoring for the following group of DB2 objects:

- User table and index spaces
- DB2 catalog table space DSNDB06.SYSTSKEYS on the first day of each month at 1:00 a.m.

User table and index spaces

We ran the SQL **CALL** statement that is shown in Example A-18 to schedule autonomic statistics monitoring for user table and index spaces every day at 1:00 a.m.

Example A-18 Statistics monitoring user objects

```
CALL SYSPROC.ADMIN_TASK_ADD
  (NULL,
   NULL,
   null,
   null,
   NULL,
   NULL,
   '0 1 * * *',
   NULL,
   NULL,
   NULL,
   NULL, r
   'SYSPROC',
   'ADMIN_UTL_MONITOR',
   'SELECT ''statistics-scope=profile,restrict-ts="DBNAME <> ''''DSNDB06''''"'',
0, 0 ,'''' from SYSIBM.SYSDUMMY1',
   NULL,
   NULL,
   NULL,
   'STATSMON1', e
   'statistics monitoring on user tablespaces every day at 1 am',
   ?,
   ?
  )
 ;
```

DB2 catalog table space DSNDB06.SYSTSKEY

We ran the SQL **CALL** statement that is shown in Example A-19 to schedule autonomic statistics monitoring for DB2 catalog table space on the first day of each month at 1:00 a.m.

Example A-19 Statistics monitoring DSNDB06.SYSTSKEY

```
CALL SYSPROC.ADMIN_TASK_ADD
  (NULL,
   NULL,
   null,
   null,
   NULL,
   NULL,
   '0 1 1 * *',
   NULL,
   NULL,
   NULL,
```

```
   NULL,
   'SYSPROC',
   'ADMIN_UTL_MONITOR',
   'SELECT ''statistics-scope=profile,restrict-ts="DBNAME = ''''DSNDB06'''' AND
NAME =''''SYSTSKEY'''' "'', 0, 0 ,'''' from SYSIBM.SYSDUMMY1',
   NULL,
   NULL,
   NULL,
   'STATSMON2',
   'statistics monitoring systskey tablespace on first day on each month day at 1
am',
   ?,
   ?
  )
 ;
```

Upon successful completion, we used the ADMIN_TASK_LIST user-defined function (UDF) to list the autonomic statistics monitoring tasks that are registered in the administrative scheduler. The query result is shown in Figure A-16.

```
SELECT
      substr(TASK_NAME,1,10)        as TASKNAME
    , substr(POINT_IN_TIME,1,10   ) as PIT
    , substr(PROCEDURE_SCHEMA,1,10) as STPSCHEMA
    , substr(PROCEDURE_NAME  ,1,20) as STPNAME
    , substr(DESCRIPTION     ,1,40) as DESCRIPTION
    , CREATOR
FROM table(DSNADM.ADMIN_TASK_LIST()) as tasklist
WHERE PROCEDURE_NAME = 'ADMIN_UTL_MONITOR';
---------+---------+---------+---------+---------+---------+---------+---------
TASKNAME    PIT       STPSCHEMA STPNAME           DESCRIPTION
---------+---------+---------+---------+---------+---------+---------+---------
STATSMON1   0 1 * * *  SYSPROC   ADMIN_UTL_MONITOR  statistics monitoring
STATSMON2   0 1 1 * *  SYSPROC   ADMIN_UTL_MONITOR  statistics monitoring
```

Figure A-16 Query statistics monitoring tasks

Administrative scheduler runtime messages

When the administrative scheduler triggered the ADMIN_UTIL_MONITOR stored procedure, the ADMT trace data provided the information that is shown in Figure A-17 on page 507.

```
(TTHD000) Signal received with command = 1
(TTHD000) Execution begins for task = 6
(TTHD000) Execution begins at time 2012-09-21-09.38.00.000000
(TTHD000) PassTicket generated for user = "DOZGADMT"
(TTHD000ÝP¨) starting
(TTHD000ÝP¨) connected]
(TTHD000ÝP¨) stored procedure schema = "SYSPROC"
(TTHD000ÝP¨) stored procedure name = "ADMIN_UTL_MONITOR"
(TTHD000ÝP¨) P parmÝ0¨ type=449-0 length=30000 name = "MONITOR_OPTIONS"
(TTHD000ÝP¨) O parmÝ1¨ type=493-0 length=8 name = "HISTORY_ENTRY_ID"
(TTHD000ÝP¨) O parmÝ2¨ type=497-0 length=4 name = "RETURN_CODE"
(TTHD000ÝP¨) O parmÝ3¨ type=449-0 length=1331 name = "MESSAGE"
(TTHD000ÝP¨) num(columns) = 4
(TTHD000ÝP¨) num(variables) = 128
(TTHD000ÝP¨) columnÝ0¨ type=448 length=79 addr = x24E2263E
(TTHD000ÝP¨) columnÝ1¨ type=497 length=8 addr = x24E22693
(TTHD000ÝP¨) columnÝ2¨ type=497 length=4 addr = x24E226A1
(TTHD000ÝP¨) columnÝ3¨ type=449 length=1331 addr = x24E226AB
(TTHD000ÝP¨) "SYSPROC"."ADMIN_UTL_MONITOR"
(TTHD000ÝP¨) call stored procedure, SQLCODE = 0
(setSQLStatus)  DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
(TTHD000ÝP¨) out parmÝ1¨ = "0x0000011D"
(TTHD000ÝP¨) out parmÝ2¨ = "0x00000000"
(TTHD000ÝP¨) out parmÝ3¨ = ""
(TTHD000ÝP¨) disconnected]
(TTHD000) logged out
(TTHD000) Execution status COMPLETED
(TTHD000) Execution ends at time 2012-09-21-09.38.00.000000
```

Figure A-17   ADMIN_UTL_MONITOR ADMT trace information

Verifying the status of statistics monitoring processing

The trace information that is shown in Figure A-17 provides processing information for ADMT task number 6 and processing begin at 2012-09-21-09.38.00.000000. We use the query that is shown in Example A-20 to verify the processing status of that task number.

Example A-20   Query for verifying the status of a task

```
SELECT
      SUBSTR(TASK_NAME,1,10)   AS TASKNAME
    , SUBSTR(STATUS,1,10)      AS STATE
    , NUM_INVOCATIONS AS #INV
    , SUBSTR(CHAR(START_TIMESTAMP),1,19) AS BETS
    , SUBSTR(CHAR(END_TIMESTAMP),1,19)   AS ENTS
    , SQLCODE
    , DB2_SSID AS SSID
    , SUBSTR(MSG,1,40) AS MSG
FROM table(DSNADM.ADMIN_TASK_STATUS(10)) as taskstatus
where task_name LIKE 'STATSMON%'
and start_timestamp = '2012-09-21-09.38.00.000000'
---------+---------+---------+---------+---------+---------+---------+-------
TASKNAME    STATE             #INV  BETS                 ENTS
---------+---------+---------+---------+---------+---------+---------+-------
STATSMON1   COMPLETED           26  2012-09-21-09.38.00  2012-09-21-09.38.01
```

```
STATSMON2   COMPLETED           26  2012-09-21-09.38.00  2012-09-21-09.38.00
```

The query output that is shown in Example A-20 on page 507 confirms a status of `COMPLETED` for both of our statistics monitoring tasks.

Verifying the RUNSTATS utility output

Autonomic statistics monitoring in our example environment triggers RUNSTATS whenever missing or inconsistent statistics are detected. The administrative scheduler calls the SYSPROC.ADMIN_UTL_EXECUTE stored procedure for triggering and controlling RUNSTATS. Upon RUNSTATS completion, you can query table SYSIBM.SYALERTS to verify that RUNSTATS completed successfully and to review the RUNSTATS utility output.

We created the SQL table UDF shown in Example A-21 to retrieve the RUNSTATS utility output of a table space. We provide the table space name and qualifier as input parameters in the SQL table UDF interface.

Example A-21   SQL table UDF to obtain the RUNSTATS output

```
CREATE FUNCTION UTILOUTPUT
(CREATOR VARCHAR(12), OBJECT VARCHAR(32))
 RETURNS TABLE
 ( STARTTS TIMESTAMP,
   STATUS  VARCHAR(32),
   OUTPUT  CLOB(2 M))
 LANGUAGE SQL READS SQL DATA NO EXTERNAL ACTION DETERMINISTIC
 RETURN
   WITH
   Q1 (ID,CREATOR, OBJECT) AS
   (SELECT
     ALERT_ID
    ,SUBSTR(TARGET_QUALIFIER,1,32)
    ,SUBSTR(TARGET_OBJECT,1,08)
    FROM SYSIBM.SYSAUTOALERTS
     WHERE TARGET_QUALIFIER =  UTILOUTPUT.CREATOR AND
           TARGET_OBJECT    =  UTILOUTPUT.OBJECT
     ORDER BY ALERT_ID DESC
     FETCH FIRST ROW ONLY)
   ,Q2 (STARTTS, STATUS,OUTPUT) AS
   (SELECT STARTTS, STATUS, OUTPUT FROM SYSIBM.SYSAUTOALERTS A,Q1
    WHERE  A.ALERT_ID = Q1.ID  ORDER BY STARTTS)
   SELECT * FROM Q2
```

In the example that is shown in Example A-22, we use the SQL table UDF shown in Example A-21 to obtain the RUNSTATS utility output of the most recent utility that is run for table space DSNADMDB.DSNADMTS.

Example A-22   Query for recent RUNSTATS for table space DSNADMDB.DSNADMTS

```
SELECT output
 FROM TABLE(UTILOUTPUT('DSNADMDB','DSNADMTS')) AS A
OUTPUT
2012-09-21 09:38:02.487888> 1DSNU000I    265 09:38:01.88 DSNUGUTC - OUTPUT START
2012-09-21 09:38:02.487899>  DSNU1045I   265 09:38:01.96 DSNUGTIS - PROCESSING S
2012-09-21 09:38:02.487910> 0DSNU050I    265 09:38:02.11 DSNUGUTC -  RUNSTATS TA
2012-09-21 09:38:02.487920>   PROFILE
2012-09-21 09:38:02.487930>  DSNU1361I -D0Z2 265 09:38:02.11 DSNUGPRF - THE STAT
```

```
2012-09-21 09:38:02.487940>  ADMIN_TASKS HAS BEEN USED
2012-09-21 09:38:02.487950>  DSNU1368I   265 09:38:02.11 DSNUGPRB - PARSING STAT
2012-09-21 09:38:02.487961>  DSNU1369I   265 09:38:02.11 DSNUGPRB - PARSING STAT
```

A.4 Additional information

For more information about how to implement and configure the administrative task scheduler, see following DB2 for z/OS manuals:

- *DB2 10 for z/OS Installation and Migration Guide*, GC19-2974
- *DB2 10 for z/OS Administration Guide*, SC19-2968
- *DB2 10 for z/OS Managing Performance*, SC19-2978

B

Configuration and workload

This appendix shows the configurations of the different platform environments that are used during this project and the DayTrader application workload.

This appendix covers the following topics:

- Configurations
- The DayTrader application workload
- Using the DayTrader application

B.1 Configurations

This section describes the environment in which we performed our tests. Figure B-1 shows our starting z/OS configurations. We used a DB2 10 for z/OS data sharing configuration that contains two members, D0Z1 and D0Z2, on LPARs SC63 and SC64. The processors on WTSCPLX2 were shared across all LPARs. We also built a WebSphere Application Server V 8.5 network deployment cell across these two LPARS and a cluster named MZSR014.

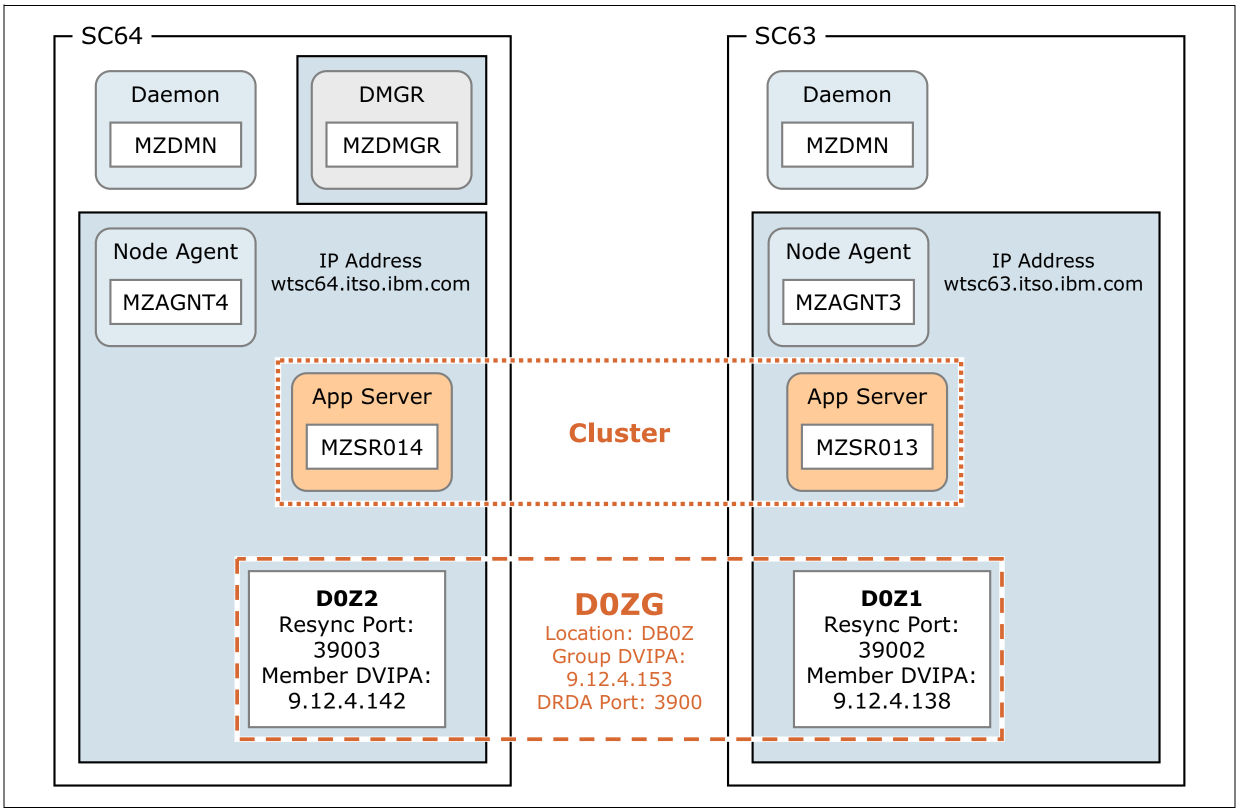


Figure B-1 Our DB2 for z/OS configuration

B.2 The DayTrader application workload

This section provides a description of the DayTrader application workload, which was used for our tests.

B.2.1 The IBM DayTrader performance benchmark sample for WebSphere Application Server

The IBM DayTrader performance benchmark sample provides a suite of IBM-developed workloads for characterizing the performance of the WebSphere Application Server. The workloads consist of an end-to-end web application and a full set of primitives. The applications are a collection of Java classes, Java servlets, JavaServer Pages, web services, and Enterprise beans that are built to open Java Platform, Enterprise Edition APIs. Together, these provide versatile and portable test cases that measure aspects of scalability and performance.

Figure B-2 provides an overview of the DayTrader application workload.

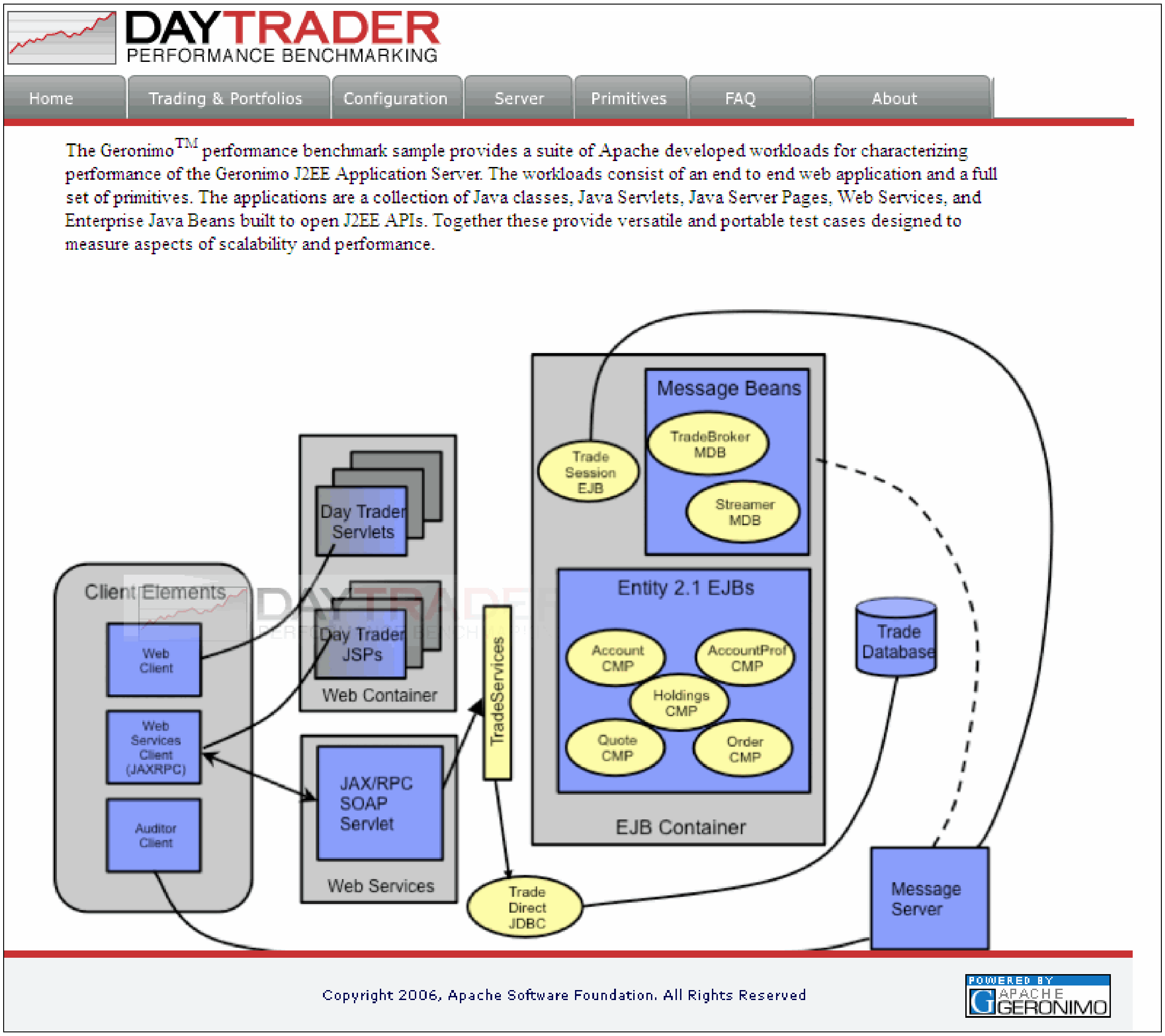


Figure B-2 DayTrader overview

Our environment

The environment consists of the following elements:

- z/OS R13
- WebSphere Application Server V8.5
- IBM Data Server Driver for JDBC and SQLJ Fix Pack 6

> **Note:** The steps show the installation of trade application with the assumption that the software was installed.

DayTrader installation

Download your copy of the DayTrader application from the following web page:

https://cwiki.apache.org/GMOxDOC20/daytrader.html

Extract the files from the downloaded package.

DB2 for z/OS

Create the DayTrader database by completing the following steps. The skeleton JCL is prepared in the DayTrader package.

1. Upload the `trade6db_pakage.jcl` file in the `tradeinstall/zOS` directory to a data set in your TSO environment on your z/OS system.

   > **Note:** The `trade6db_pakage.jcl` is stored in EBCDIC format, so no conversions are necessary.

2. Customize the JCL by following the instructions in the JCL.
3. Submit the job to create the table spaces, tables, and indexes for the DayTrader database.

Installing the DayTrader application on WebSphere Application Server for z/OS V8.5

The DayTrader application installation script defines and installs the necessary resources in your installation of WebSphere Application Server. The script provides the following installation options:

- DB2 for Linux, UNIX, and Windows (type 4 driver)
- DB2 for z/OS (type 2 driver)
- Oracle

> **Tip:** You cannot choose an option to use a type 4 driver for DB2 for z/OS. But you must choose DB2 for z/OS or your application will not run. After you run the installation script, you can modify your data source setting from the WebSphere Application Server administrative console. We show how to change them later.

- Username

  This is the user ID that is used to install the DayTrader application; in our example, we used “mzadmin”.
- Password

  This is the password for the user name.
- WebSphere Application Server installation

  If you are using global security or cluster installation, you are prompted by this option.
- WebSphere Application Server node

Installing the nodes for the DayTrader application

Now, you have lists of nodes to choose from. In our example, we used a cluster that is named “mzsr014””. You have the following options:

- Backend DB type

  Choose from db2, oracle, or db2zos. You must choose “db2zos” when running against DB2 for z/OS server.
- DB driver path

  Enter the path for your type 4 driver path. In our installation, we used `/usr/lpp/db2/d0zg/jdbc/classes/db2jcc.jar` and `/usr/lpp/db2/d0zg/jdbc/classes/db2jcc_lisence_cisuz.jar`.

- DB name

  The location name of DB2 for z/OS. In our installation, “D0ZG” is the location name for our DB2 data sharing group.

- DB username

  The user ID that is used to connect to DB2 for z/OS. In our installation, we used “Rajesh”.

- DB password

  The password that is associated with the DB username.

> **Tip:** If your scripts do not complete, check your WebSphere Application Server administrative console to see whether any of configurations or installations are done. If you rerun your script, you manually must delete what is configured or installed from your last installation.

After you are done with running scripts, go to your WebSphere Application Server administrative console. The DayTrader application should be installed, but it might not be started, as shown in Figure B-3, after the installation. You do not need to start the DayTrader application now.

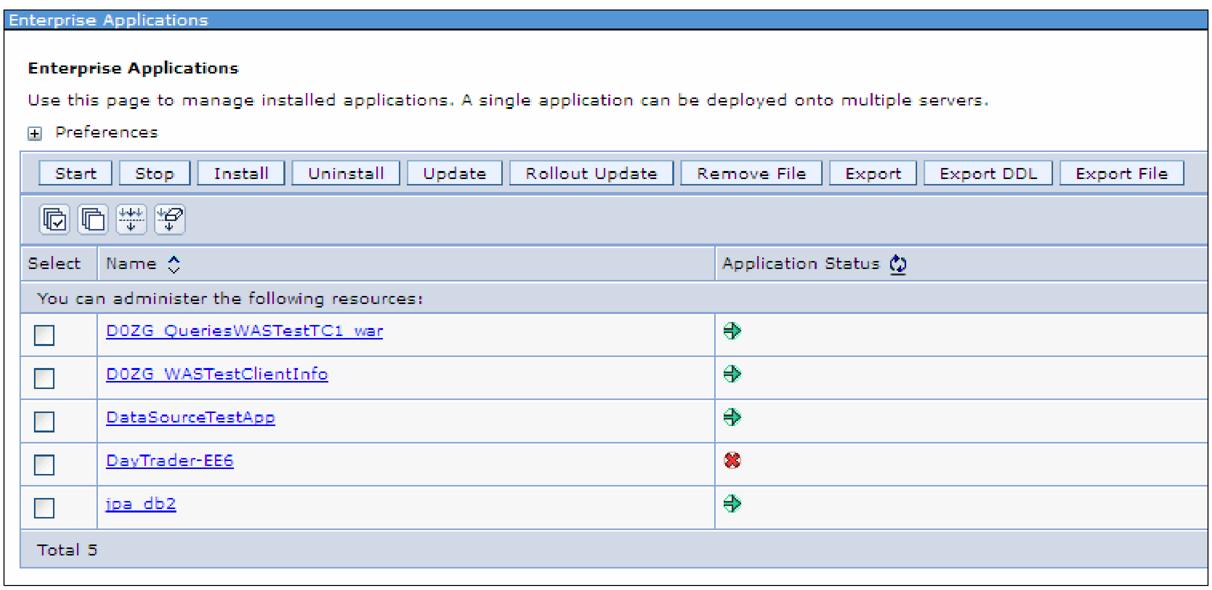


Figure B-3 WebSphere Application Server admin console after installation

After running the scripts, the administration console of WebSphere Application Server opens. Click **Resources** → **JDBC** → **JDBC Provider**. You should see a new JDBC provider that is defined, as shown in Figure B-4.

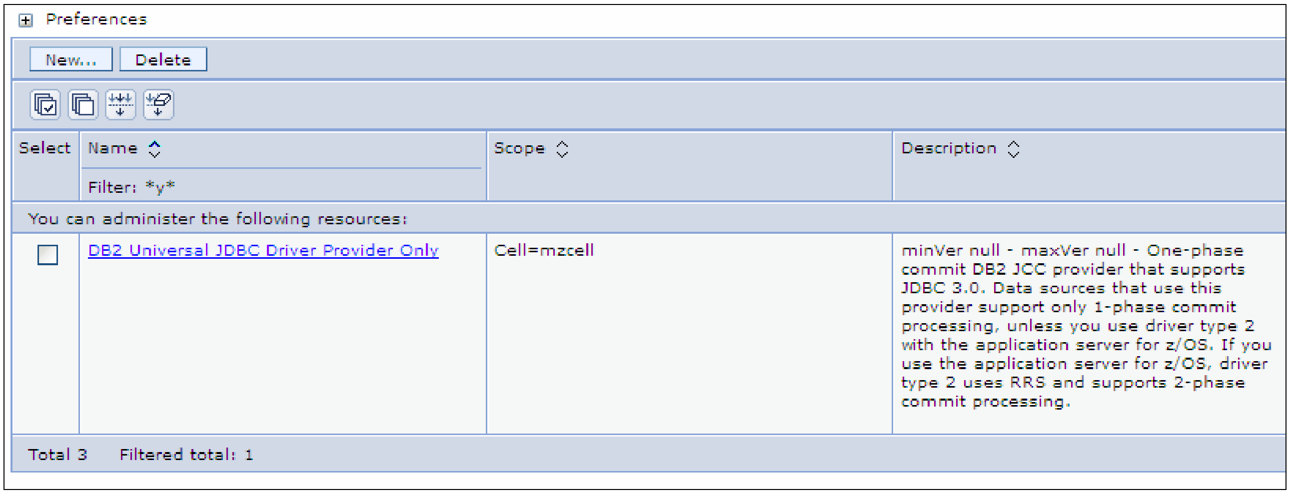


Figure B-4 JDBC Provider that is defined by the configuration script

Customize your data source settings so you can connect to DB2 for z/OS through the network, where the default installation for “db2zos” is the type 2 driver. In our example, we also performed the setup for sysplex workload balancing and a type 4 connection.

In the administration console of WebSphere Application Server, click **Resources** → **JDBC** → **JDBC Provider**. You see lists of data sources that are defined in your installation. You should see “TradeDataSource”; if not, check your scope match with a node that you specify at script or “All scopes”.

Figure B-5 shows the data source for our example.

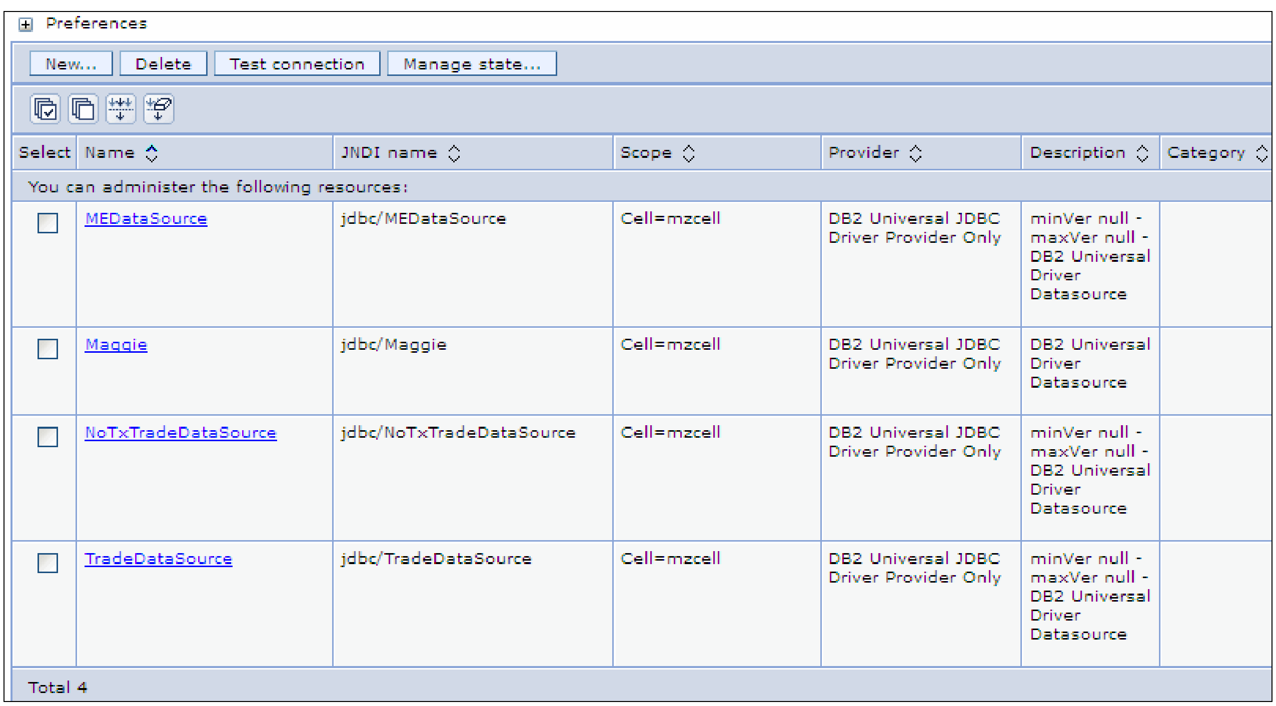


Figure B-5 TradeDataSource from WebSphere Application Server administration console

Click **TradeDataSource** to access the settings for your DayTrader application data source. At the bottom, you find the Driver type, Server name, and Port number. Change Driver type to “4”, Server name to an IP address or the domain name for your DB2 for z/OS, and change the Port number to the DRDA service port. In our example, the settings are the ones that are shown in Figure B-6. In addition, we added two properties that are related to sysplex workload balancing in the data source custom properties.

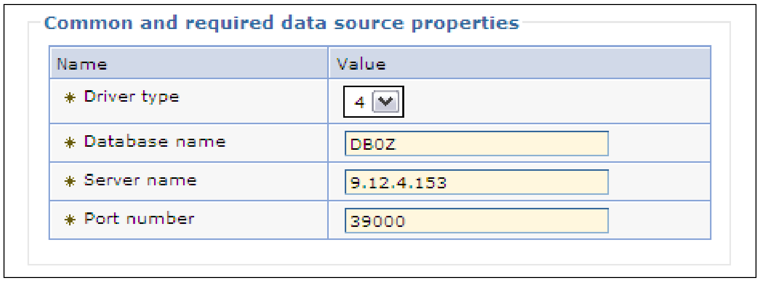


Figure B-6 Modify the data source for the type 4 connection

Go to **Servers** → **Application servers**, and restart the application server in which the DayTrader application is installed. After restarting, navigate to **Applications** → **Enterprise Application** to verify that the DayTrader application started.

Your DayTrader application should be accessible from your browser. You must access your installation of the DayTrader application to finish your installation.

Navigate to **Configuration (Re)-populate Trade Database** to finish your installation. This populates your DayTrader database with fictitious users and stocks, as shown in Figure B-7. This step takes some time. You can close your browser, but you cannot see the status if you do so.

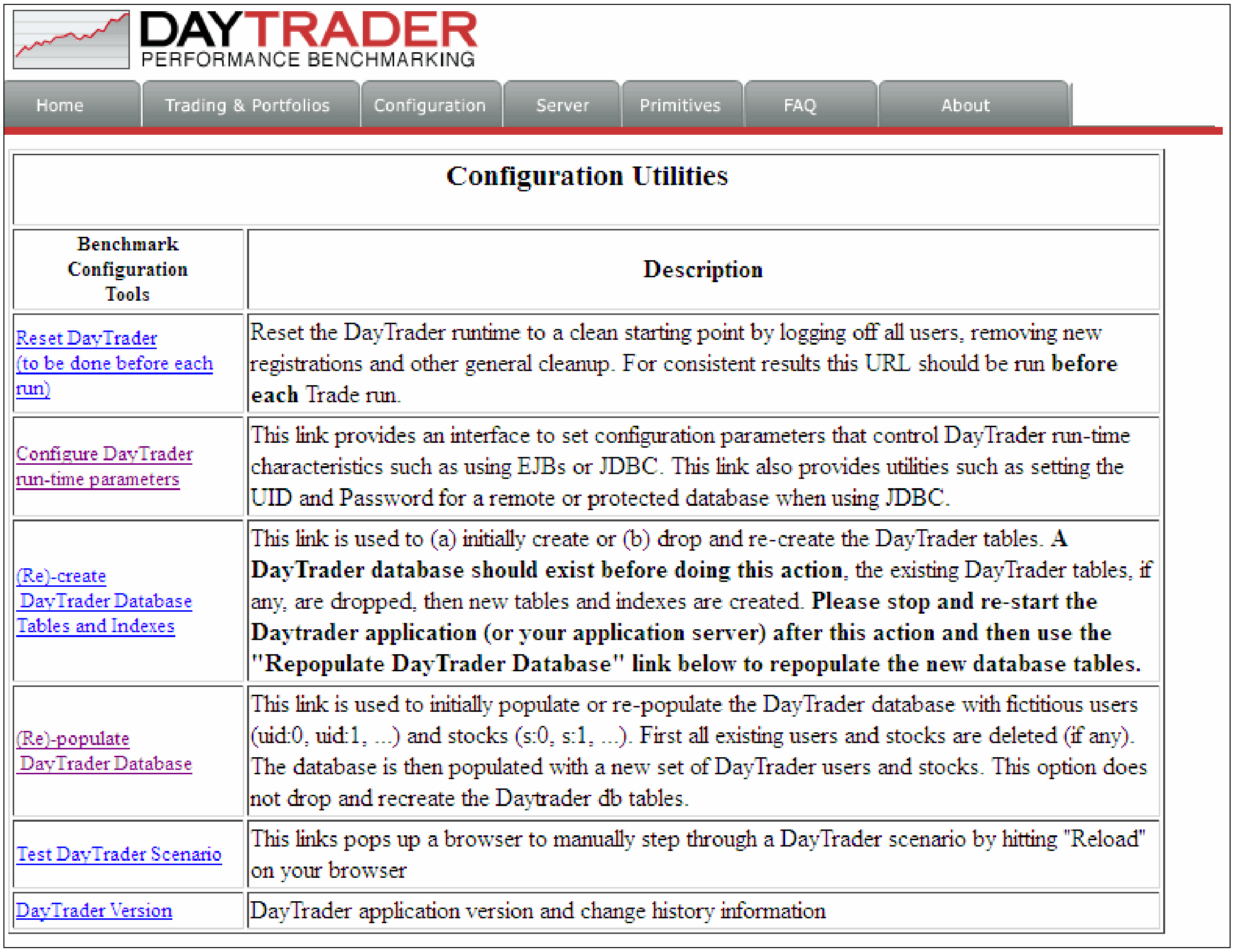


Figure B-7 Finish installation by populating the DayTrader database

> **Tip:** If your application does not work, you might need to ask your WebSphere Application Server administrator for help.
>
> To work through issues on your own, see *Approach to Problem Determination in WebSphere Application Server V6*, REDP-4073, found at:
>
> `http://www.redbooks.ibm.com/abstracts/redp4073.html?Open`
>
> In addition, *WebSphere Application Server V6: Default Messaging Provider Problem Determination*, REDP-4076, found at the following web page, might be useful:
>
> `http://www.redbooks.ibm.com/abstracts/redp4076.html?Open`

B.3 Using the DayTrader application

In this section, we briefly explain how the DayTrader application works.

Click **Go Trade!** from the left menu pane of the window shown in Figure B-8.

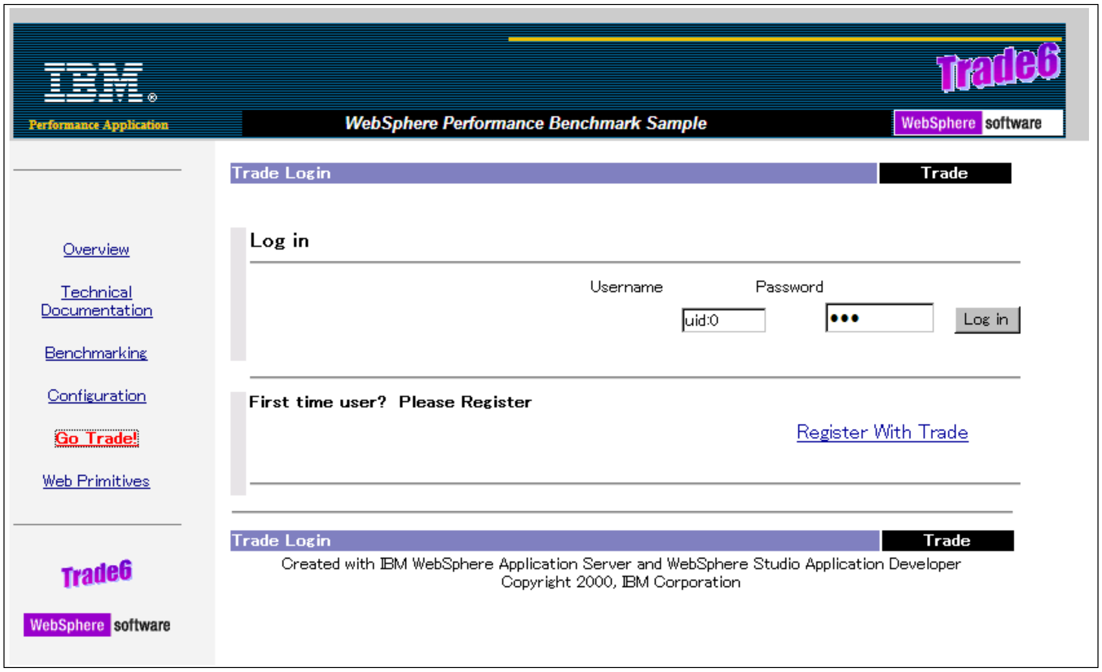


Figure B-8 Go Trade! window

The window that is shown in Figure B-9 opens. Click **Log in** (the Username and Password are already entered) to get started, or create an account by clicking **Register With Trade**.

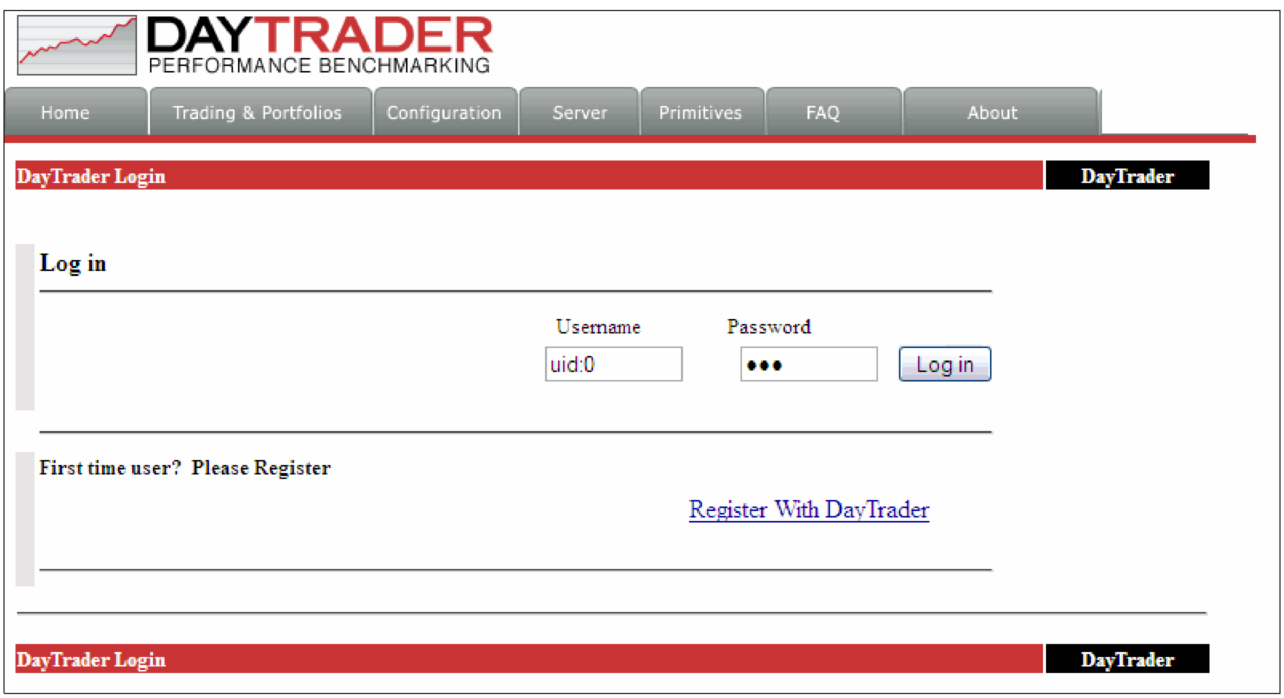


Figure B-9 Verify your installation by logging in to the DayTrader application

After you log in to the DayTrader application, you see the DayTrader Home window (Figure B-10). Click **Portfolio** to start trading.

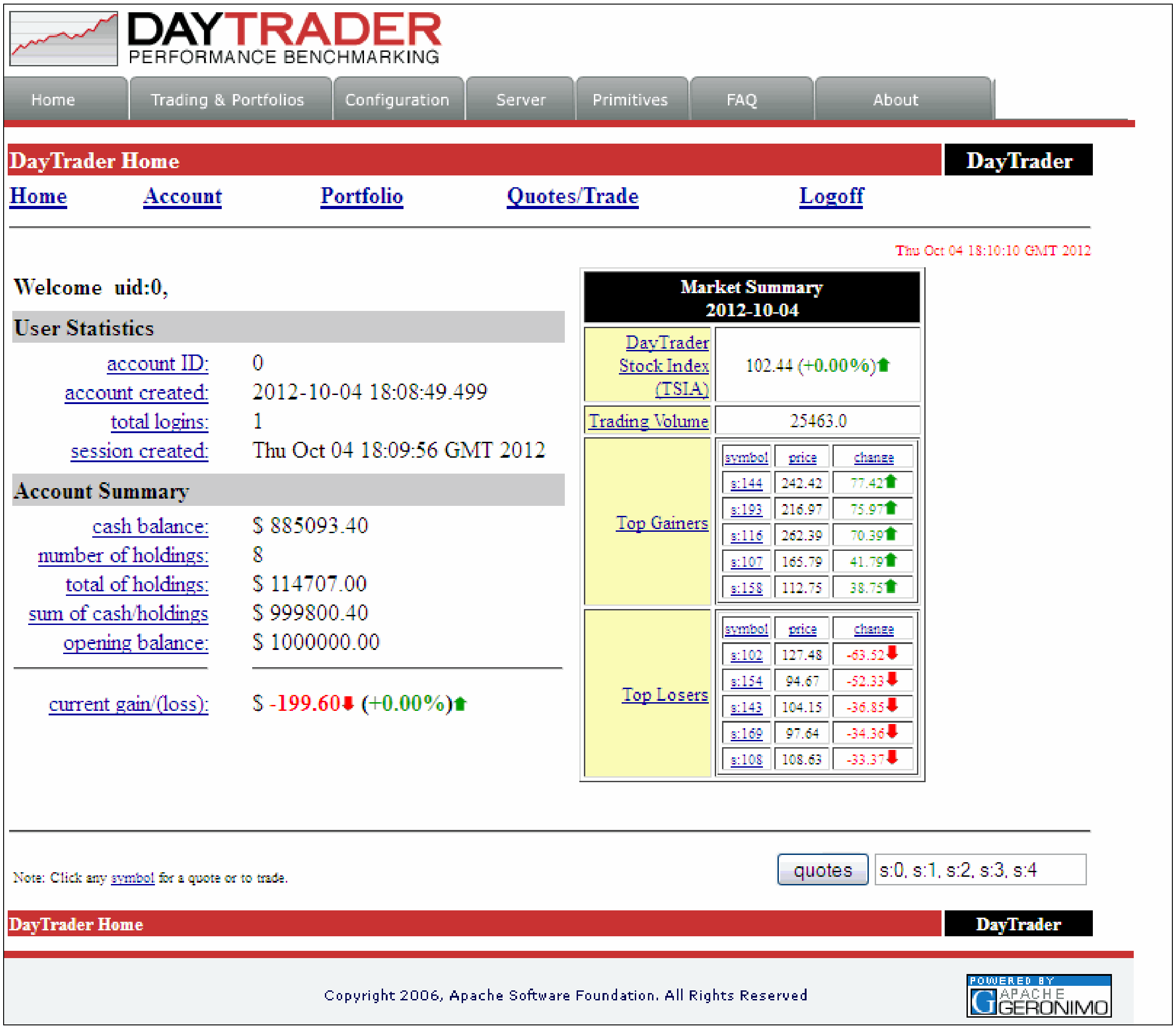
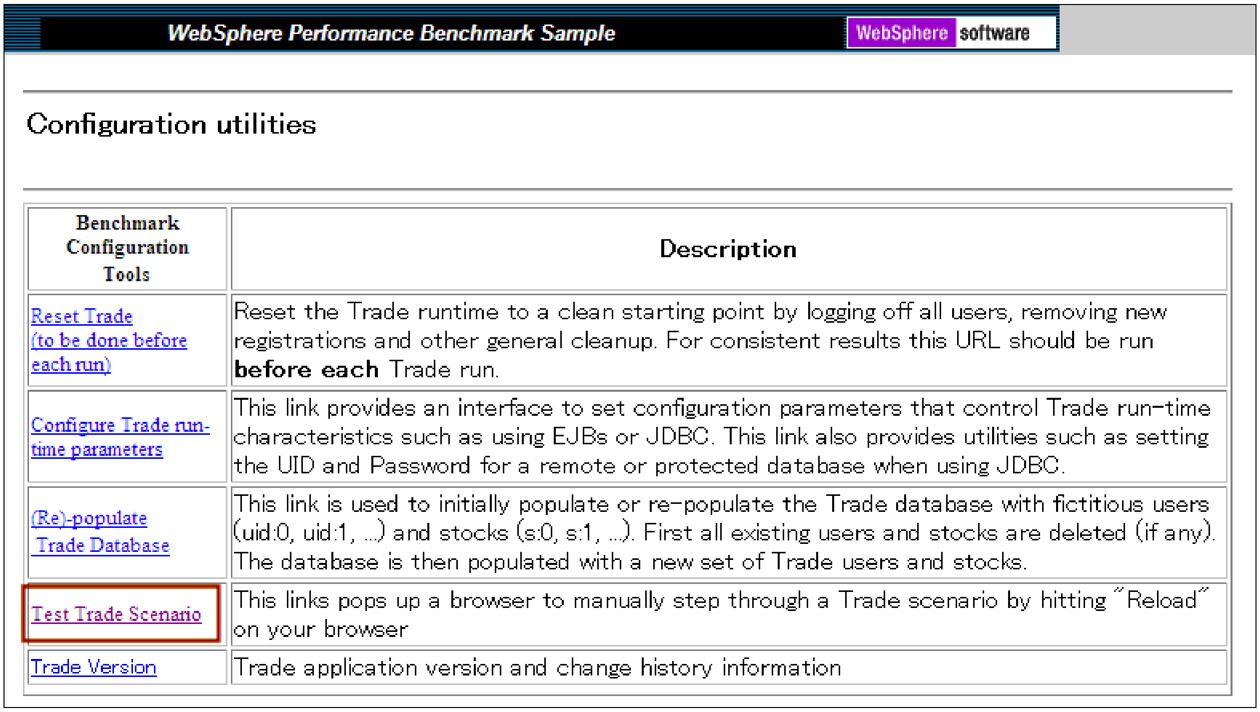


Figure B-10 DayTrader Home window

After you verify your installation (by clicking all the menus), you can start your workload by using the Test Trade Scenario.

Click **Configuration** in the left menu pane (shown in Figure B-8 on page 519) and then click **Test Trade Scenario**, as shown in Figure B-11. A new window opens. In that window, click **Reload**.


WebSphere Performance Benchmark Sample

WebSphere software

Configuration utilities

Benchmark Configuration Tools	Description
Reset Trade (to be done before each run)	Reset the Trade runtime to a clean starting point by logging off all users, removing new registrations and other general cleanup. For consistent results this URL should be run **before each** Trade run.
Configure Trade run-time parameters	This link provides an interface to set configuration parameters that control Trade run-time characteristics such as using EJBs or JDBC. This link also provides utilities such as setting the UID and Password for a remote or protected database when using JDBC.
(Re)-populate Trade Database	This link is used to initially populate or re-populate the Trade database with fictitious users (uid:0, uid:1, ...) and stocks (s:0, s:1, ...). First all existing users and stocks are deleted (if any). The database is then populated with a new set of Trade users and stocks.
Test Trade Scenario	This links pops up a browser to manually step through a Trade scenario by hitting "Reload" on your browser
Trade Version	Trade application version and change history information

Figure B-11 Test Trade scenario

You can use any load testing tool to create a workload to run by clicking **Test Trade Scenario**. For our test, we used Apache JMeter. This is a Java -based application that tests functional behavior and measures performance. It is available from the following web page:

http://jakarta.apache.org/

Apache JMeter can load and performance test various server types, but because the DayTrader application provides the Test Trade Scenario, you can use it to test a scenario by using an HTTP request.

C

Setting up a WebSphere Application Server test environment on IBM Data Studio

This appendix provides an overview of how the IBM Data Studio can be used as a development and test environment for Java Platform, Enterprise Edition applications.

This appendix covers the following topics:

- Download of freely available WebSphere Application Server products for development
- How to install these products
- Definitions of the JDBC drivers and data sources

C.1 Installing WebSphere Application Server Developer Tools into IBM Data Studio

The IBM Data Studio Version 3.1.1.0 full client comes with a rich set of tools for database development. It also has tools that help handle Java -related tasks. A full Java Platform, Enterprise Edition application can be developed with IBM Data Studio. But you cannot test the application in an application server because it is not prepared for it. Because IBM Data Studio is an Eclipse -based tool, it can be extended by the WebSphere Application Server Developer Tools, which are a subset of IBM Rational Application Developer plug-ins that can be downloaded at no additional charge at the following website:

http://public.dhe.ibm.com/ibmdl/export/pub/software/websphere/wasdev/V8.5/wdtupdate/

After you download the compressed file containing the server tools repository (`wdt-update-site_8.5.0.WDT85-I20120530_0920.zip`, in our case), it can be installed in to IBM Data Studio by completing the following steps:

1. Click **Help** → **Install new software**.
2. In the **Add Repository** window, click **Archive**.
3. Browse to the location of the compressed file of IBM WebSphere Application Server Developer Tools for Eclipse. Select the file and then click **Open**.

4. You should see a selection menu, as shown in Figure C-1, where you can select the versions of the server adapters that you need. No server installation is done now; you get the only adapter software, which lets you connect to a separate server installation. Thus, you must use an existing application server installation or install either a full version of WebSphere Application Server or the new Liberty profile, which is optimized for developer productivity and web / mobile application deployment.

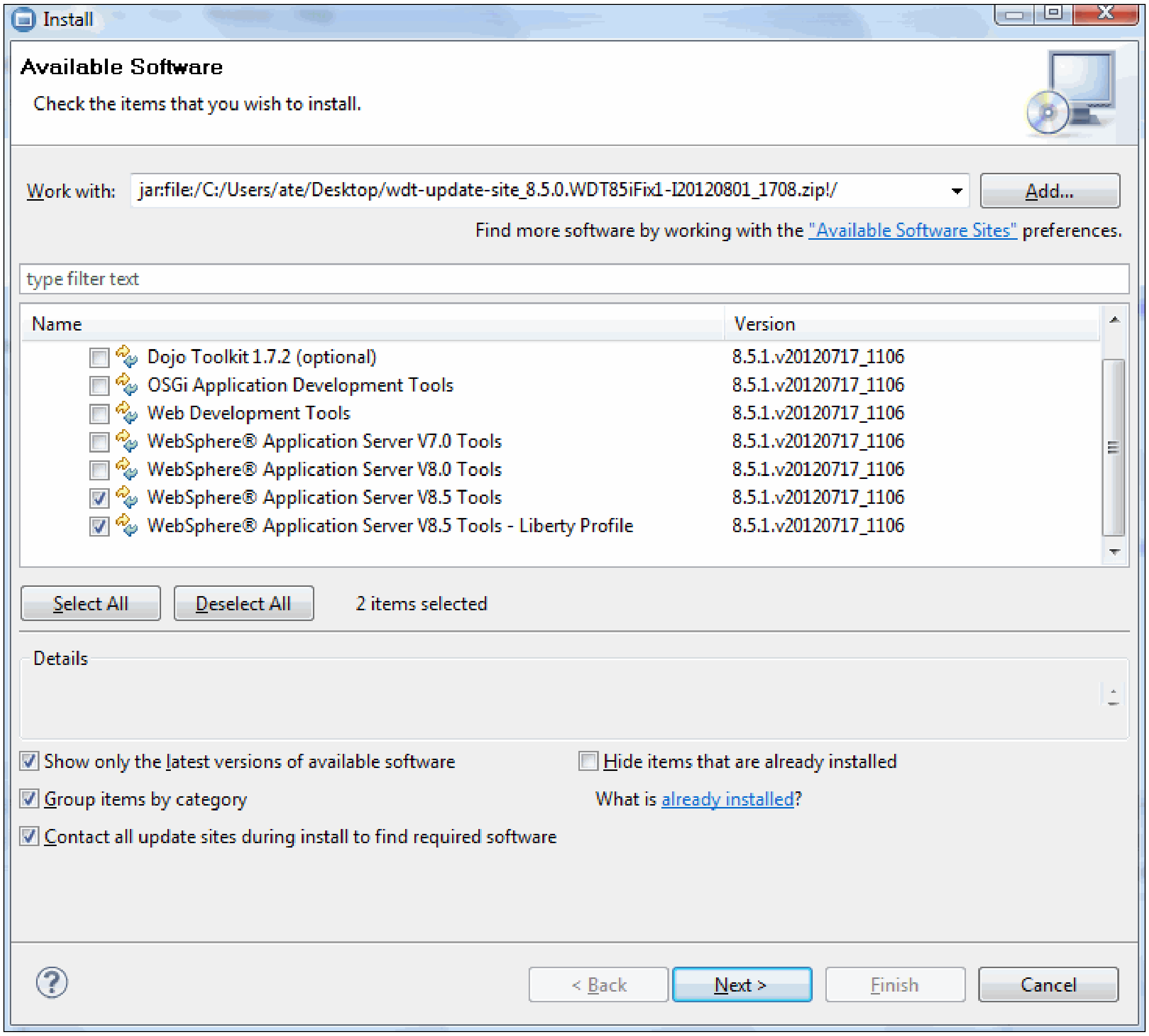


Figure C-1 Install the WebSphere Application Server Developer Tools

C.1.1 WebSphere Application Server for Developers V8.5

The WebSphere Application Server for Developers V8.5 package is functionally equivalent to the WebSphere Application Server V8.5 package, but it is licensed for development use only. WebSphere Application Server for Developers is an easy-to-use development environment to build and test Java and Java Platform, Enterprise Edition applications. It provides simplified and no additional charge access to enable developers to build and test in the same environment that ultimately supports their applications.

The installation of the application server is well documented and does not need to be repeated here. For more information, go to the following website:

`http://www.ibm.com/software/webservers/appserv/developer/index.html`

C.1.2 WebSphere Application Server Liberty Profile

WebSphere Application Server V8.5 includes a Liberty profile, which is a highly composable and dynamic application server profile. It is a stand-alone product that must be installed independently from the WebSphere Application Server product, which has knowledge about profiles, but the Liberty profile is different.

Download the Liberty profile from the following website:

https://www.ibm.com/developerworks/mydeveloperworks/blogs/wasdev/entry/download_wlp

In the directory that you chose to download the file into, run **`java -jar wlp-developers-8.5.0.0.jar`** and follow the installation instructions, which are straightforward.

D

IBM OMEGAMON XE for DB2 performance database

OMEGAMON XE for DB2 provides a performance database (PDB), which you can use to store historical information in DB2 tables. Using these tables can be useful for problem determination, application profiling, KPI monitoring, and capacity planning.

In this appendix, we introduce the OMEGAMON PDB, and outline how to create the PDB database, and extract, transform, and load (ETL) DB2 trace information into the PDB tables. We used this functionality to implement the activity that is described at 4.4, "Tivoli OMEGAMON XE for DB2 Performance Expert for z/OS" on page 201.

The appendix covers the following topics:

- Introduction
- Creating the performance database
- Extracting, transforming, and loading accounting and statistics data
- Additional information

D.1 Introduction

The PDB consists of a set of tables that you can populate with information from DB2 statistics, accounting, performance, locking, and audit traces. The population process is also referred to as *extract, transform, and load* (ETL). We provide an overview of the PDB ETL process in Figure D-1.

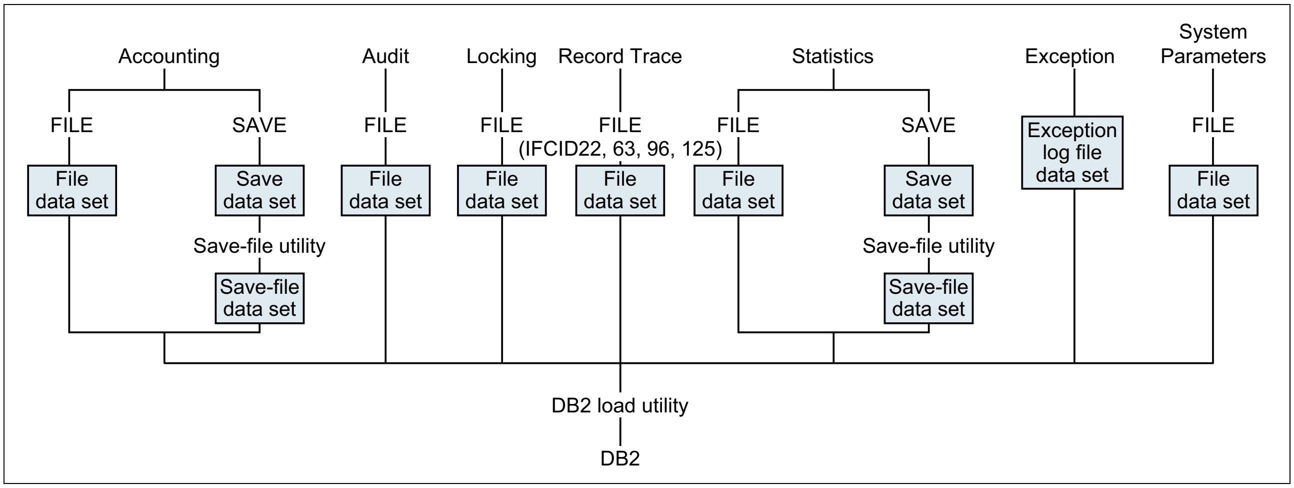


Figure D-1 OMEGAMON PDB ETL overview

As indicated in Figure D-1, ETL processes non-aggregated (FILE format) and aggregated (SAVE format) information.

- Aggregated information

  Several records are summarized by specific identifiers. In a report, each entry represents aggregated data. You run the **SAVE** subcommand to generate a VSAM data set that contains the aggregated data. When the data is saved, you use the Save-File utility to generate a DB2 -loadable data set. As you might have noticed in Figure D-1, this format is supported only for statistics and accounting trace information. This option is useful if you must process huge volumes of accounting information.

- Non-aggregated information

  For non-aggregated data, each record is listed in the order of occurrence. In a trace, each entry represents non-aggregated data. You run the **FILE** subcommand to generate a data set that contains non-aggregated data. This format is supported for all DB2 trace information. Analyzing non-aggregated accounting information can be useful if you want to use the report capabilities of SQL to drill down on thread level accounting information. In our scenario, the volume of DB2 trace information is not expected to be large. We therefore decided to load the PDB tables with non-aggregated information.

With PDB ETL, you can process DB2 trace data of the following input formats:

- System Measurement Facility (SMF) record types 100 (statistics), 101 (accounting), and 102 (performance and audit).
- Generalized Trace Facility (GTF).
- OMPE ISPF interface (collect report data).
- Batch program FPEZCRD. For an example of how to run program FPEZCRD in batch, refer the JCL sample that is provided in the RKO2SAMP library, member FPEZCRDJ.
- Near term history sequential data sets.

In our DB2 environment, we processed DB2 traces that we collected through SMF and GTF.

D.1.1 Performance database structure

The PDB database design is provided by OMEGAMON and comes with a set of tables to store DB2 trace data of the following information categories:

- Accounting
- Audit
- Exceptions
- Locking
- Record trace
- Statistics
- System parameters

For this book, we focused on using non-aggregated accounting and statistics information. If you need details about using the PDB for the other information categories, see Chapter 5, “Advanced reporting concepts. The Performance Database and the Performance Warehouse”, in *IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS Reporting User's Guide*, SH12-6927.

Accounting tables

Figure D-2 shows the accounting table categories that are provided by the performance database. PDB stores each data type in its own DB2 table.

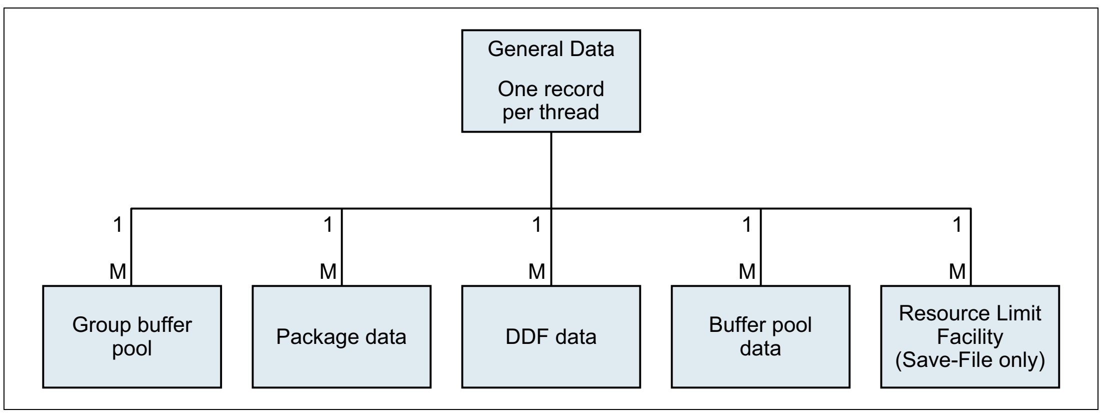


Figure D-2 PDB structure accounting tables

OMPE provides two sets of accounting tables:

- FILE accounting tables, which detailed information so that you can use SQL to query accounting information on a thread level.
- SAVE accounting tables, which store aggregated data so that you can use SQL to query summarized accounting information of the time interval boundary.

FILE accounting tables

Each table type that is shown in Figure D-2 stores the following information:

- General data: General accounting information (one row per thread)
- Group buffer pool: For each thread, one row per group buffer pool that is being used
- Package data: For each thread, one row per package that is being used
- Buffer pool: For each thread, one row per buffer pool that is being used
- Resource limit facility: One row per resource limit type that is encountered

SAVE accounting tables

Each table type that is shown in Figure D-2 on page 529 stores the following aggregated information:

- General data: General accounting information, one row per aggregation interval
- Group buffer pool: For each aggregation interval, one row per group buffer pool that is being used
- Package data: For each aggregation interval, one row per package that is being used
- Buffer pool: For each aggregation interval, one row per buffer pool that is being used

Accounting table DDL and load statements

OMPE provides sample create table DDL, load utility control statement templates, and table metadata descriptions in the RKO2SAMP library members that are shown Table D-1 and Table D-2. We used these templates to create and load these accounting tables.

Table D-1 FILE accounting table DDL and load statements

Table name	Type	RKO2SAMP create table DDL	RKO2SAMP load utility statements	RKO2SAMP table metadata documentation
DB2PMFACCT_BUFFER	Buffer pool data	DGOACFBU	DOGALFBU	DGOABFBU
DB2PMFACCT_GENERAL	General data	DGOACFGE	DGOALFGE	DGOABFGE
DB2PMFACCT_GBUFFER	Group buffer pool	DGOACFGP	DGOALFGP	DGOABFGP
DB2PMFACCT_PROGRAM	Package data	DGOACFPK	DGOALFPK	DGOABFPK
DB2PMFACCT_DDF	DDF data	DGOACFDF	DGOALFDF	DGOABFDF

Table D-2 SAVE accounting table DDL and load statements

Table name	Type	RKO2SAMP create table DDL	RKO2SAMP load utility statements	RKO2SAMP table metadata documentation
DB2PMSACCT_BUFFER	Buffer pool data	DGOACSBU	DOGALSBU	DOGABSBU
DB2PMSACCT_GENERAL	General data	DGOACSGE	DOGALSGE	DOGABSGE
DB2PMFACCT_GBUFFER	Group buffer pool	DGOACSGP	DOGALSGP	DOGABSGP
DB2PMFACCT_PROGRAM	Package data	DGOACSPK	DOGALSPK	DOGABSPK
DB2PMFACCT_DDF	DDF data	DGOACSDF	DOGALSDF	DOGABSDF

Statistics tables DDL and load statements

Figure D-3 shows the structure of each of the statistics tables in the performance database. PDB stored each data type in its own DB2 table.

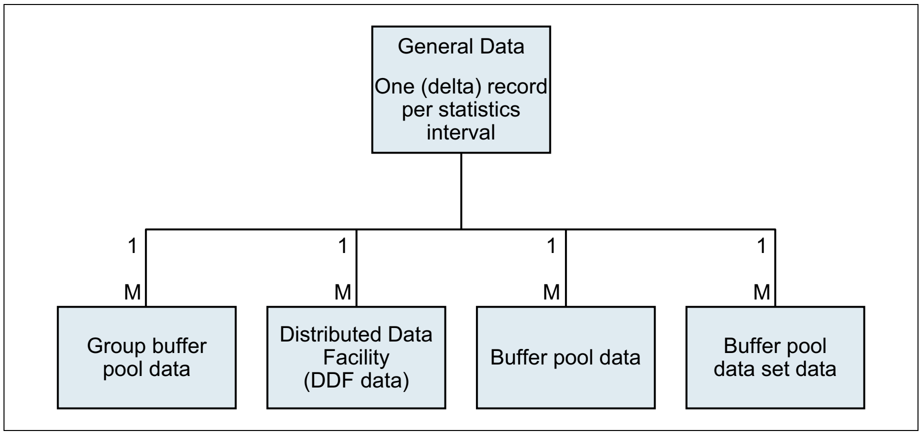


Figure D-3 PDB structure statistics tables

In our environment, we generate loadable input records in the FILE data format. In that format, each table type that is shown in Figure D-3 stores the following information:

- General data: One row for each Statistics delta record, containing data from IFCID 1 and 2. A delta record describes the activity between two consecutive statistics record pairs.
- Group buffer pool data: One row per group buffer pool that is active at the start of the corresponding delta record.
- DDF data: One row per remote location that is participating in distributed activity by using the DB2 private protocol and one row for all remote locations that used DRDA.
- Buffer pool data: One row per buffer pool that is active at the start of the corresponding delta record.
- Buffer pool data set data: One row for each open data set that has an I/O event rate at least one event per second during the reporting interval. To obtain that statistics trace information, you must activate statistics trace class 9.

OMEGAMON provides sample create table DDL, load utility control statement templates, and table metadata descriptions in the RKO2SAMP library members that are shown in Table D-3. We used these templates to create and load these statistics tables.

Table D-3 Statistics table DDL and load statements

Table name	Type	RKO2SAMP create Table DDL	RKO2SAMP load utility statements	RKO2SAMP table metadata documentation
DB2PM_STAT_GENERAL	General data	DGOSCGEN	DGOSLGEN	DGOSBGEN
DB2PM_STAT_GBUFFER	Group buffer pool data	DGOSCGBP	DGOSLGBP	DGOSBGBP

Table name	Type	RKO2SAMP create Table DDL	RKO2SAMP load utility statements	RKO2SAMP table metadata documentation
DDB2PM_STAT_DDF	DDF data	DGOSCDDF	DGOSLDDF	DGOSBDDF
DB2PM_STAT_BUFFER	Buffer pool data	DGOSCBUF	DGOSLBUF	DGOSBBUF

D.2 Creating the performance database

We used the create table DDL RK02SAMP library members that are described in D.1.1, “Performance database structure” on page 529 to create the PDB accounting and statistics tables. To create the PDB, we performed the following activities:

- Create a DB2 for z/OS database to store the PDB tables.
- Customize PDB create table DDL.
- Create PDB tables.

D.2.1 Creating a DB2 z/OS database

We ran the SQL shown in Example D-1 to create the DB2 for z/OS database that we used to create the PDB tables. In our PDB environment, table spaces use buffer pool BP1, and index spaces use BP2.

Example D-1 PDB create DB2 z/OS database

```
CREATE DATABASE PMPDB
  BUFFERPOOL BP1
  INDEXBP    BP2
  CCSID      EBCDIC
  STOGROUP   SYSDEFLT;
```

D.2.2 Customizing the PDB create table DDL

The OMEGAMON -provided PDB create table DDL statements require customization, as OMEGAMON does not provide an interface for providing PDB table qualifier, database, and table space names. In addition, the PDB provided database design does not provide create table space DDL and does not provide for indexes that are required to ensure uniqueness of data and to support query performance. To perform this customization, we performed the following tasks:

- Generate a create table DDL data set that contains all DDL statements.
- Modify a create table DDL to reflect the PDB database name and table qualifier.

Generating a create table DDL data set

We ran the JCL that is shown in Example D-2 on page 533 to merge the accounting and statistics create DDL statements that are shown in Table D-1 on page 530, Table D-2 on page 530, and Table D-3 on page 531 in to a data set. For application profiling, we run queries on aggregated accounting information we created in the OMPE accounting tables that are described in “SAVE accounting tables” on page 530.

Example D-2 PDB generate create table DDL data set

```
//S1GEN   EXEC PGM=IEBGENER
//SYSUT1   DD *
  SET CURRENT SCHEMA = 'PDB';
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOACFBU)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOACFDF)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOACFGE)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOACFGP)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOACFPK)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOACSBU)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOACSDF)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOACSGE)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOACSGP)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOACSPK)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOACSRF)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOSCBUF)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOSCDDF)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOSCGBP)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOSCGEN)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOSCSET)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOWCSFP)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOWC106)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOWC201)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOWC202)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOWC230)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOWC256)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOXCBND)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOXCBRD)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOXCCHG)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOXCCNT)
```

```
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOXCDDL)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOXCDML)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOXCFAI)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOXCSQL)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//   DD DISP=SHR,DSN=<omhlq>.RKO2SAMP(DGOXCUTI)
//   DD DISP=SHR,DSN=DB2R3.SG.PM.DDL(SEMIKOLO)
//SYSUT2   DD DISP=SHR,DSN=DB2R3.PM.CNTL($04DDLTB)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
```

Customizing a create table DDL

Next, we customize the create table DDL data set that we generated in Example D-2 on page 533. You might notice that we set the current schema to control the table qualifier and that we inserted a semicolon to separate the create table statements for SQL batch processing. We ran the ISPF edit command that is illustrated in Figure D-4 to modify the DDL to use the database that we created in D.2.1, “Creating a DB2 z/OS database” on page 532 for table creation.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
 -------------------------------------------------------------------------------
 EDIT       DB2R3.PM.CNTL($03DDL) - 01.00                    Columns 0010 0072
Command ===> c 'IN DB2PM.' 'IN PMPDB.' ALL                    Scroll ===> CSR
 ****** ***************************** Top of Data *****************************
 000001   SET CURRENT SCHEMA = 'PDB';
 000002 --**Start of Specifications*******************************************
 000003 --*                                                                  *
 000004 --* MODULE-NAME      = DGOSCBUF                                      *
 000005 --* DESCRIPTIVE-NAME = SQL for creating Statistics Buffer Pool Table  *
```

Figure D-4   Customize a create table DDL

Creating table spaces

The create table DDL statements reference the following table spaces in the table space clause:

- PMPDB.TSPAFBU
- PMPDB.TSPAFDF
- PMPDB.TSPAFGE
- PMPDB.TSPAFGP
- PMPDB.TSPAFPK
- PMDB.TSPASBU
- PMDB.TSPASDF
- PMDB.TSPASGE
- PMDB.TSPASGP
- PMDB.TSPASPK
- PMPDB.TSPSBUF
- PMPDB.TSPSDDF
- PMPDB.TSPSGBP

- PMPDB.TSPSGEN
- PMPDB.TSPSSET

As these table spaces do not yet exist, we used the create table space DDL template that is shown in Example D-3 to create these table spaces. The template supports table space compression and uses the primary and secondary space quantity sliding scale feature to take advantage of autonomic space management.

Example D-3 Create table space template

```
CREATE TABLESPACE <tsname>
  IN PMPDB
  USING STOGROUP SYSDEFLT
  PRIQTY -1 SECQTY -1
  ERASE  NO
  FREEPAGE 0 PCTFREE 5
  GBPCACHE CHANGED
  TRACKMOD YES
  LOGGED
  SEGSIZE 64
  BUFFERPOOL BP1
  LOCKSIZE ANY
  LOCKMAX SYSTEM
  CLOSE YES
  COMPRESS YES
  CCSID     EBCDIC
  DEFINE YES
  MAXROWS 255;
```

D.2.3 Creating the PDB accounting and statistics tables

Now, the DB2 for z/OS database PMPDB and the table spaces that are required for the tables are created and a generated a data set with customized create table DDL statements exists. Next, we run the batch JCL that is shown in Example D-4 to run the create table DDL statements that we customized in “Customizing a create table DDL” on page 534.

Example D-4 Batch JCL PDB accounting and statistics table creation

```
//S10TEP2  EXEC PGM=IKJEFT1B,DYNAMNBR=20,TIME=1440
//STEPLIB DD DISP=SHR,DSN=DB0ZT.SDSNEXIT
//        DD DISP=SHR,DSN=DB0ZT.SDSNLOAD
//        DD DISP=SHR,DSN=DB0ZT.RUNLIB.LOAD
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(D0ZG)
 RUN  PROGRAM(DSNTEP2) PLAN(DSNTEP10)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD DISP=SHR,DSN=DB2R3.PM.CNTL($04DDLTB)
```

D.3 Extracting, transforming, and loading accounting and statistics data

Next, we extract, transform, and load (ETL) DB2 accounting and statistics trace information in to the PDB tables that we created in D.2.3, "Creating the PDB accounting and statistics tables" on page 535. The ETL process consists of the following processing steps:

1. Extract and transform DB2 trace data into an OMEGAMON Performance Expert (OMPE) FILE formatted data set
2. Load the OMPE FILE formatted data set into DB2 tables.
3. Extract transform and DB2 trace date into an OMEGAMON Performance Expert SAVE formatted data set.
4. Load the OMPE SAVE data into DB2 tables.

D.3.1 Extracting and transforming DB2 trace data into the FILE format

We ran the batch JCL that is shown in Example D-5 to extract and to transform SMF DB2 accounting and statistics data into the OMEGAMON XE for DB2 PE FILE format. We obtained the accounting and statistics data in a sequential data set that we later use for the DB2 LOAD utility to load the data into DB2 Performance Database accounting and statistics tables.

Example D-5 OMPE extract and transform DB2 trace data into FILE format

```
//* ---------------------------------------------------------------
//*DOC Extract and transform accounting and statistics trace data
//*DOC into Omegamon PE FILE format
//* ---------------------------------------------------------------
//DB2PM1  EXEC PGM=DB2PM,REGION=0M
//STEPLIB  DD  DISP=SHR,DSN=<omhlq>.RKANMOD
//INPUTDD  DD  DISP=SHR,DSN=SMF.DUMP.G0033V00
//STFILDD1 DD  DISP=(NEW,CATLG,DELETE),DSN=DB2R3.PM.STAT.FILE,
//             SPACE=(CYL,(050,100),RLSE),
//             UNIT=SYSDA,
//             DATACLAS=COMP /*trigger DFSMS compression */
//ACFILDD1 DD  DISP=(NEW,CATLG,DELETE),DSN=DB2R3.PM.ACCT.FILE,
//             SPACE=(CYL,(050,100),RLSE),
//             UNIT=SYSDA,
//             DATACLAS=COMP /*trigger DFSMS compression */
//JOBSUMDD DD  SYSOUT=A
//DPMLOG   DD  SYSOUT=A
//SYSOUT   DD  SYSOUT=A
//SYSIN    DD  *
GLOBAL
   INCLUDE(SSID(DB1S)) TIMEZONE(-1)
STATISTICS
      FILE DDNAME(STFILDD1)
ACCOUNTING
      FILE DDNAME(ACFILDD1)
EXEC
```

D.3.2 Extracting and transforming DB2 trace data into the SAVE format

We ran the batch JCL that is shown in Example D-6 to extract and to transform SMF DB2 accounting data into OMEGAMON XE for DB2 PE accounting SAVE format. We obtained the accounting data in a sequential data set, which we later use as input for the DB2 LOAD utility to load the data into DB2 Performance Database save accounting tables

Example D-6 Extract and transform accounting SAVE format

```
//IDC01   EXEC PGM=IDCAMS
//* ==============================================================
//* Def Cluster source: RKO2SAMP(DGOPJAMI)
//* ==============================================================
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
  DELETE (DB2SMF.WASRB.ACCTLOAD) NONVSAM
  SET MAXCC = 0
  DELETE (DB2SMF.WASRB.ACCTSAVE    )          CLUSTER
  SET MAXCC = 0
  DEFINE CLUSTER -
     (NAME(DB2SMF.WASRB.ACCTSAVE    ) -
      CYL(100,40) -
      BUFFERSPACE(40960) -
      KEYS(255 0) -
      REUSE -
      RECORDSIZE(2800 4600) -
     ) -
  DATA  (CISZ(8192)) -
  INDEX (CISZ(4096))
//SAVE02  EXEC PGM=DB2PM,REGION=0M
//STEPLIB  DD  DISP=SHR,DSN=OMEGA5RT.SC63.RKANMOD
//INPUTDD  DD  DISP=SHR,DSN=DB2SMF.WASRB.SC63.T4.SMFDB2
//ACSAVDD  DD  DISP=SHR,DSN=DB2SMF.WASRB.ACCTSAVE
//DPMLOG   DD  SYSOUT=A
//JOBSUMDD DD  SYSOUT=A
//SYSOUT   DD  SYSOUT=A
//SYSIN    DD  *
GLOBAL
    INCLUDE(SUBSYSTEMID(D0Z*))
    TIMEZONE(+4)
ACCOUNTING
      /* 1 minute interval */
      REDUCE INTERVAL(1) BOUNDARY(60)
      SAVE
EXEC
//CONV03  EXEC PGM=DGOPMICO,PARM=CONVERT,COND=(0,NE)
//STEPLIB  DD  DISP=SHR,DSN=OMEGA5RT.SC63.RKANMOD
//SYSPRINT DD  SYSOUT=*
//INPUT    DD  DSN=DB2SMF.WASRB.ACCTSAVE,DISP=SHR
//OUTPUT   DD  DSN=DB2SMF.WASRB.ACCTLOAD,
//             DISP=(NEW,CATLG,DELETE),
//             SPACE=(CYL,(200,10),RLSE),
//             UNIT=SYSDA,
//             DCB=(RECFM=VB,LRECL=9072,BLKSIZE=9076)
```

D.3.3 Preparing a load job

Loading data into DB2 tables requires that a DB2 load utility batch JCL be available for batch job submission. To prepare the required batch JCL, we performed the following tasks:

- Consolidate and customize load utility control statements for loading PDB accounting and statistics data.
- Provide batch JCL for DB2 load utility job submission.

Load utility control statements

We ran the batch JCL that is shown in Example D-7 and Example D-8 to merge the load utility control statements that we referenced in “Accounting table DDL and load statements” on page 530 and in “Statistics tables DDL and load statements” on page 531 into a consolidated data set.

Example D-7 Merge statistics and accounting file load utility control statements

```
//S1GEN   EXEC PGM=IEBGENER
//SYSUT1   DD *
--OPTIONS PREVIEW
//         DD DISP=SHR,DSN=<omhlq>.TKO2SAMP(DGOSLBUF)
//         DD DISP=SHR,DSN=<omhlq>.TKO2SAMP(DGOSLDDF)
//         DD DISP=SHR,DSN=<omhlq>.TKO2SAMP(DGOSLGBP)
//         DD DISP=SHR,DSN=<omhlq>.TKO2SAMP(DGOSLGEN)
//         DD DISP=SHR,DSN=<omhlq>.TKO2SAMP(DGOSLSET)
//         DD DISP=SHR,DSN=<omhlq>.TKO2SAMP(DGOALFBU)
//         DD DISP=SHR,DSN=<omhlq>.TKO2SAMP(DGOALFDF)
//         DD DISP=SHR,DSN=<omhlq>.TKO2SAMP(DGOALFGE)
//         DD DISP=SHR,DSN=<omhlq>.TKO2SAMP(DGOALFGP)
//         DD DISP=SHR,DSN=<omhlq>.TKO2SAMP(DGOALFPK)
//SYSUT2   DD DISP=SHR,DSN=DB2R3.PM.CNTL($08LOATB)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
```

Example D-8 Merge accounting save load utility control statements

```
//COPY1    EXEC PGM=IEBGENER,DYNAMNBR=20,TIME=1440
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSUT2 DD DISP=SHR,DSN=DB2R3.SG.PM.LOAD(LOADACCS)
//SYSUT1   DD *
-- OPTIONS PREVIEW
//   DD DISP=SHR,DSN=<LOADHLQ>.RKO2SAMP(DGOALSBU)
//   DD DISP=SHR,DSN=<LOADHLQ>.RKO2SAMP(DGOALSDF)
//   DD DISP=SHR,DSN=<LOADHLQ>.RKO2SAMP(DGOALSGE)
//   DD DISP=SHR,DSN=<LOADHLQ>.RKO2SAMP(DGOALSGP)
//   DD DISP=SHR,DSN=<LOADHLQ>.RKO2SAMP(DGOALSPK)
//   DD DISP=SHR,DSN=<LOADHLQ>.RKO2SAMP(DGOALSRF)
```

We then modified the generated data set to reflect the table qualifier and the appropriate input DD statement and implemented the load utility options that we needed to use. Here are the load options that we use:

- **RESUME YES**
- **LOG NO**
- **KEEPDICTIONARY**
- **NOCOPYPEND**

D.3.4 Loading accounting and statistics tables

We use the DB2 load utility to load the data that is referred to in D.3.1, “Extracting and transforming DB2 trace data into the FILE format” on page 536 and in D.3.2, “Extracting and transforming DB2 trace data into the SAVE format” on page 537 into the PDB accounting and statistics tables.

D.3.5 Maintaining PDB tables

Your DB2 installation regularly performs Image Copy, Runstats, and Reorg on your tables to comply with your recovery requirements and to support good query performance.

Image copy

We ran the batch JCL that is shown in Example D-9 to perform image copy on PDB accounting and statistics tables.

Example D-9 Image copy batch JCL

```
//COPY EXEC DSNUPROC,SYSTEM=DB1S,
//             LIB='SYS1.DSNDB1S.SDSNLOAD',
//             UID='PDBCOPY'
//DSNUPROC.SYSIN  DD  *
--OPTIONS PREVIEW
  TEMPLATE TPDB  DSN DB1SIC.IC.&DB..&TS..D&DATE..T&TIME.
           DATACLAS COMP
  LISTDEF  LPDB  INCLUDE TABLE      PDB.*
  COPY LIST LPDB  COPYDDN(TPDB) CHANGELIMIT(0) PARALLEL
```

Runstats

Because we configured the administrative scheduler to perform autonomic statistics maintenance on non-catalog table spaces, there was no need to plan any further Runstats activity.

Reorg

We ran the batch JCL that is shown in Example D-10 to perform Reorg on PDB accounting and statistics tables.

Example D-10 Reorg batch JCL

```
//REORG1 EXEC DSNUPROC,SYSTEM=DB1S,
//             UID='PDBREO'
//DSNUPROC.SYSIN DD *
--OPTIONS PREVIEW
  LISTDEF  LPDB INCLUDE TABLE PDB.*
  TEMPLATE TCOPY   DSN DB1SIC.IC.&DB..&TS..D&DATE..T&TIME.
```

```
          DATACLAS COMP
 TEMPLATE TSYSUT1 DSN(DB1SIC.&DB..&TS..&UTILID..SYSUT1)
          DISP(NEW,DELETE,KEEP)
          DATACLAS COMP
 TEMPLATE TSORTOUT DSN(DB1SIC.&DB..&TS..&UTILID..SORTOUT)
          DISP(NEW,DELETE,KEEP)
          DATACLAS COMP
 TEMPLATE TPUNCH   DSN(DB1SIC.&DB..&TS..&UTILID..PUNCH  )
          DISP(NEW,DELETE,KEEP)
          DATACLAS COMP
 TEMPLATE TSYSREC DISP(NEW,DELETE,KEEP)
          DSN(DB1SIC.&DB..&TS..&UTILID..SYSREC)
          DATACLAS COMP

 REORG TABLESPACE LIST LPDB
   LOG NO
   SHRLEVEL REFERENCE
   SORTDATA
   SORTDEVT SYSDA
   SORTNUM 4
   UNLDDN TSYSREC
   WORKDDN(TSYSUT1,TSORTOUT)
   STATISTICS
   COPYDDN(TCOPY)
   PUNCHDDN(TPUNCH)
```

D.4 Sample query for application profiling

We created the DB2 SQL table UDF that is shown in Example D-11 to provide an interface for querying the DB2PMSACCT_GENERAL and DB2PMSACCT_BUFFER PDB tables for application profiling. The UDF receives two input parameters and joins DB2 general and buffer pool accounting information. The result is filtered by the DB2 client application information and the connection type (RRS or DRDA) to provide profiling information for a particular clientApplicationInformation for JDBC type 2 (connection type RRS) or for JDBC type 4 (connection type DRDA) applications.

Example D-11 OMPE SQL table UDF

```
CREATE FUNCTION ACCOUNTING
(CLIENTAPPLICATION VARCHAR(128),
 CONNTYPE          CHAR(8)      )
 RETURNS TABLE (
   "DateTime"              VARCHAR(16)
 , "ClientApplication"     VARCHAR(40)
 , "Elapsed"               DECIMAL(9,2)
 , "TotCPU"                DECIMAL(9,2)
 , "TotzIIP"               DECIMAL(9,2)
 , DB2CPU                  DECIMAL(9,2)
 , "DB2zIIP"               DECIMAL(9,2)
 , "Commit"                INTEGER
 , SQL                     INTEGER
 , "Locks"                 INTEGER
 , "RowsFetched"           INTEGER
 , "RowsInserted"          INTEGER
```

```
 , "RowsUpdated"            INTEGER
 , "RowsDeleted"            INTEGER
 , "GetPage"                INTEGER
 ,"AVG-Time"                DECIMAL(15, 6)
 ,"AVG-CPU"                 DECIMAL(15, 6)
 ,"Time/SQL"                DECIMAL(15, 6)
 ,"CPU/SQL"                 DECIMAL(15, 6)
 ,"AVG-SQL"                 DECIMAL(15, 6)
 ,"LOCK/Tran"               DECIMAL(15, 6)
 ,"LOCK/SQL"                DECIMAL(15, 6)
 ,"GETP/Tran"               DECIMAL(15, 6)
 ,"GETP/SQL"                DECIMAL(15, 6)
 )
 LANGUAGE SQL READS SQL DATA NO EXTERNAL ACTION
 DETERMINISTIC
 RETURN
WITH
Q1 AS
(SELECT
   substr(char(INTERVAL_TIME),1,16        ) AS DATETIME
 , CLIENT_TRANSACTION
 , DECIMAL(CLASS1_ELAPSED,9,2             ) AS ELAPSED
 , DECIMAL(CLASS1_CPU_NNESTED+CLASS1_CPU_STPROC+CLASS1_CPU_UDF
   +CLASS1_IIP_CPU,9,2                    ) AS CPU
 , DECIMAL(CLASS1_IIP_CPU,9,2             ) AS ZIIP
 , DECIMAL(CLASS2_CPU_NNESTED+CLASS2_CPU_STPROC+CLASS2_CPU_UDF
   +CLASS2_IIP_CPU,9,2                    ) AS DB2CPU
 , DECIMAL(CLASS2_IIP_CPU,9,2             ) AS DB2ZIIP
 , DECIMAL(COMMIT,9,2                     ) AS COMMIT
 , DECIMAL(SELECT+INSERT+UPDATE+DELETE+FETCH+MERGE,9,2) AS SQL
 , DECIMAL(LOCK_REQ,9,2                   ) AS LOCKS
 , INTEGER(ROWS_FETCHED                   ) AS ROWS_FETCHED
 , INTEGER(ROWS_INSERTED                  ) AS ROWS_INSERTED
 , INTEGER(ROWS_UPDATED                   ) AS ROWS_UPDATED
 , INTEGER(ROWS_DELETED                   ) AS ROWS_DELETED
 FROM DB2PMSACCT_GENERAL
  WHERE CONNECT_TYPE = ACCOUNTING.CONNTYPE
   AND CLIENT_TRANSACTION = ACCOUNTING.CLIENTAPPLICATION
   AND COMMIT > 0 ),
Q2 AS
(SELECT
   substr(char(INTERVAL_TIME),1,16        ) AS DATETIME
 , CLIENT_TRANSACTION
 , decimal(SUM(BP_GETPAGES),9,2           ) AS GETPAGE
 FROM DB2PMSACCT_BUFFER
  WHERE CONNECT_TYPE = ACCOUNTING.CONNTYPE
   AND CLIENT_TRANSACTION = CLIENTAPPLICATION
  GROUP BY substr(char(INTERVAL_TIME),1,16), CLIENT_TRANSACTION ),
Q3 AS
(SELECT Q1.*, Q2.GETPAGE FROM Q1, Q2 WHERE
 (Q1.DATETIME,Q1.CLIENT_TRANSACTION) =
 (Q2.DATETIME,Q2.CLIENT_TRANSACTION) AND Q1.SQL > 0),
Q4 AS
(SELECT Q3.*,
 ELAPSED/COMMIT  as "AVG-Time",
```

```
     CPU/COMMIT  as "AVG-CPU",
 ELAPSED/SQL     as "Time/SQL",
     CPU/SQL     as "CPU/SQL",
     SQL/COMMIT  as "AVG-SQL",
   LOCKS/COMMIT  as "LOCK/Tran",
   LOCKS/SQL     as "LOCK/SQL",
 GETPAGE/COMMIT  as "GETP/Tran",
 GETPAGE/SQL     as "GETP/SQL"
 FROM Q3)
SELECT * FROM Q4
```

For each interval, the UDF returns the following information:

- DateTime: Interval date and time
- ClientApplication: Client application name
- Elapsed: Total elapsed time
- TotCPU: Total CPU time, including the time that was processed on a zIIP processor
- TotzIIP: Total zIIP processor time
- DB2CPU: DB2 part of the total CPU time
- DB2zIIP: DB2 part of the zIIP processor time
- Commit: Total number of commits
- SQL: Total number of SQL **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **FETCH**, and **MERGE** statements
- Locks: Total number of lock requests
- RowsFetched: Number of rows that were fetched
- RowsInserted: Number of rows that were inserted
- RowsUpdated: Number of rows that were updated
- RowsDeleted: Number of rows that were deleted
- GetPage: Number of getpage requests
- AVG-Time: Average elapsed time
- AVG-CPU: Average CPU time, including zIIP time
- Time/SQL: Average elapsed time per SQL
- CPU/SQL: Average CPU time per SQL
- AVG-SQL: Average number of SQL per commit
- LOCK/Tran: Average number of lock requests per commit
- LOCK/SQL: Average number of locks per SQL
- GETP/Tran: Average number of getpage requests per commit
- GETP/SQL: Average number of getpage requests per SQL

D.5 Using the UDF for application profiling

We used the query that is shown in Example D-12 to start the UDF for JDBC type 2 (connection type RRS) application profiling.

Example D-12 Starting UDF for JBC driver Type 4

```
select * from
   table(accounting('TraderClientApplication','RRS')) a
   order by "DateTime"     ;
---------+---------+---------+---------+---------+---------+---------+---------+
DateTime          ClientApplication                             Elapsed       To
---------+---------+---------+---------+---------+---------+---------+---------+
2012-08-14-22.39  TraderClientApplication                         11.96
2012-08-14-22.41  TraderClientApplication                       2417.46        3
```

For more information about how we used the UDF in our application scenario, see Chapter 8, “Monitoring WebSphere Application Server applications” on page 361.

D.6 Additional information

For more information about using the OMEGAMON Performance Expert PDB, see the following resources:

- Chapter 5, “Advanced reporting concepts”, in *IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS, Reporting User's Guide*, SH12-6927
- *A Deep Blue View of DB2 Performance: IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS*, SG24-72244

E

SMF 120 records subtypes 1, 3, 7, and 8

This appendix contains sample output of the SMF Browser program that can be used to format the SMF record type 120 records that are created by WebSphere applications.

We provide sample output for the following subtypes:

- Server activity record: Subtype 1
- Server interval record: Subtype 3
- WebContainer activity record: Subtype 7
- WebContainer activity record: Subtype 8

E.1 Server activity record: Subtype 1

As described in 8.3.1, "Using SMF 120 records" on page 377, we used the following commands to generate a summary and a detailed report of the SMF 120 records that were collected during one of the runs using the DayTrader sample application:

```
java -cp bbomsmfv.jar:batchsmf.jar com.ibm.ws390.sm.smfview.SMF
'INFILE(BART.WAS.TEST1.SMF120)' 'PLUGIN(PERFSUM,/tmp/smf120sum.txt)'
java -cp bbomsmfv.jar:batchsmf.jar com.ibm.ws390.sm.smfview.SMF
'INFILE(BART.WAS.TEST1.SMF120)' 'PLUGIN(DEFAULT,/tmp/smf120detail.txt)'
```

The second parm of the **PLUGIN** option indicates the file that the output is directed to.

We provide this information to give you a better feel for the different types of information that is available through the different SMF 120 subtype records. We provide both the summary and the detailed output for each of the subtype records that are mentioned above.

Example E-1 shows the summary output of an SMF 120 subtype 1 (120.1) server activity record.

Example E-1 Subtype 1 summary

```
=========================================== = Date:Fri Aug 10 19:58:06 EDT 2012   SysID: SC64, Page: 136

SMF -Record Time     Server   Bean/WebAppName  Bytes   Bytes  # of  El.Time  CPU_Time(uSec)     Other SMF 120.9
Numbr -Type hh:mm:ss Instance  Method/Servlet  toSvr   frSvr Calls   (msec)   Tot-CPU      zAAP Sections Present
1---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+ ----------------
  679 120.1 19:58:06 MZSR014                                                     882
                                                  942    8038
```

Example E-2 shows the detailed output of an SMF 120 subtype 1 (120.1) server activity record. It is the detailed output of the same record that is shown in Example E-1.

Example E-2 Subtype 1 detail

```
--------------------------------------------------------------------------------
Record#: 679;
  Type: 120; Size: 480; Date: Fri Aug 10 19:58:06 EDT 2012;
  SystemID: SC64; SubsystemID: WAS; Flag: 94;
  Subtype: 1 (SERVER ACTIVITY);

  #Triplets: 4;
  Triplet #: 1; offsetDec: 76; offsetHex: 4c; lengthDec: 32; lengthHex: 20; count: 1;
  Triplet #: 2; offsetDec: 108; offsetHex: 6c; lengthDec: 216; lengthHex: d8; count: 1;
  Triplet #: 3; offsetDec: 324; offsetHex: 144; lengthDec: 100; lengthHex: 64; count: 1;
  Triplet #: 4; offsetDec: 424; offsetHex: 1a8; lengthDec: 28; lengthHex: 1c; count: 2;

  Triplet #: 1; Type: ProductSection;
    Version: 4; Codeset: IBM-1047; Endian: 1; TimeStampFormat: 1 (S390STCK64);
    IndexOfThisRecord: 1; Total # records: 1; Total # triplets: 4;

  Triplet #: 2; Type: ServerActivitySection;
    HostName          : WTSC64;
    ServerName        : MZSR01;
    ServerInstanceName: MZSR014;
    ServerType        : J2EE Server;
    CellName          : MZCELL;
    NodeName          : MZNODE4;
```

```
  #ServerRegions: 1;
   ASID1: 252; ASID2: 0; ASID3: 0; ASID4: 0; ASID5: 0;
   UserCredentials: ;
   ActivityType: 1 (method request);
   ActivityID                * ca00265f 3cf7d4eb 000002a4 00000062 *
                             * 090c0609 -------- -------- -------- *
   WlmEnclaveToken           * 0000011c 012075bf -------- -------- *
   ActivityStartTime         * ca00265f 3cf7d4eb 00000000 00000000 *
   ActivityStopTime          * ca00265f 418d6820 00000000 00000000 *
   #InputMethods      : 1;
   #GlobalTransactions: 0;
   #LocalTransactions : 2;
   WLM enclave CPU time: 882;

 Triplet #: 3; Type: CommSessionSection;
   CommSessionHandle         * 08a1e8d0 00000000 -------- -------- *
   CommSessionAddress: ip addr=9.12.6.9 port=24012;
   CommSessionOptimization: 5 (HTTP session optimization);
   DataReceived: 942; DataTransferred: 8038;
   DataReceived 8 byte field: 942;
   DataTransferred 8 byte field: 8038;

 Triplet #: 4; Type: JvmHeapSection;
   Server Region ASID: 252;
   Heap Type: 3;
   Garbage Collection Count: 313;
   free Storage : 30751064;
   total Storage: 63307776;

 Triplet #: 5; Type: JvmHeapSection;
   Server Region ASID: 252;
   Heap Type: 4;
   Garbage Collection Count: 0;
   free Storage : 78369816;
   total Storage: 201326592;

--------------------------------------------------------------------------------
```

E.2 Server interval record: Subtype 3

Example E-3 shows the summary output of an SMF 120 subtype 3 (120.3) server interval record. It contains similar information to subtype 1, but it is created only at the SMF interval (server_SMF_interval_length) that you specified (3 minutes in our case). Interval records are a good starting point to get a quick idea about how the work is doing while having a minimum impact, but when you must drill deeper, or when the interval is too wide, it is not as useful.

Example E-3 Subtype 3 summary

```
SMF -Record Time     Server   Bean/WebAppName  Bytes   Bytes  # of  El.Time  CPU_Time(uSec)     Other SMF 120.9
Numbr -Type hh:mm:ss Instance  Method/Servlet  toSvr   frSvr Calls   (msec)   Tot-CPU      zAAP Sections Present
1---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+ ----------------
 5622 120.3 19:58:06 MZSR014                 2851637  26988724              2917645
```

Example E-4 shows the detailed output of an SMF 120 subtype 3 (120.3) server interval record. It is the detailed output of the same record that is shown in Example E-3 on page 547.

Example E-4 Subtype 3 detail

```
--------------------------------------------------------------------------------
Record#: 5622;
  Type: 120; Size: 548; Date: Fri Aug 10 19:58:06 EDT 2012;
  SystemID: SC64; SubsystemID: WAS; Flag: 94;
  Subtype: 3 (SERVER INTERVAL);

  #Triplets: 3;
  Triplet #: 1; offsetDec: 64; offsetHex: 40; lengthDec: 32; lengthHex: 20; count: 1;
  Triplet #: 2; offsetDec: 96; offsetHex: 60; lengthDec: 308; lengthHex: 134; count: 1;
  Triplet #: 3; offsetDec: 404; offsetHex: 194; lengthDec: 144; lengthHex: 90; count: 1;

  Triplet #: 1; Type: ProductSection;
    Version: 4; Codeset: IBM-1047; Endian: 1; TimeStampFormat: 1 (S390STCK64);
    IndexOfThisRecord: 1; Total # records: 1; Total # triplets: 3;

  Triplet #: 2; Type: ServerIntervalSection;
    HostName          : WTSC64;
    ServerName        : MZSR01;
    ServerInstanceName: MZSR014;
    ServerType        : J2EE Server;
    CellName          : MZCELL;
    NodeName          : MZNODE4;
    SampleStartTime            * ca00265e 5809c227 00000000 00000000 *
    SampleStopTime             * ca002661 347fb56b 00000000 00000000 *
    #GlobalTransactions: 0; #LocalTransactions: 8773;
    #Active CS: 41;
    #ActiveLocal  CS: 0;
    #ActiveRemote CS: 0;
    # Bytes transferred 4 byte fields
      ToServer: 2851637; FromServer: 26988724;
      LocalToServer: 0; LocalFromServer: 0;
      RemoteToServer: 0; RemoteFromServer: 0;
      Transferred to Server from http clients: 2851637;
      Transferred from Server to http clients: 26988724;
      Transferred to Server from SIP clients: 0;
      Transferred from Server to SIP clients: 0;
    #Http Communication Sessions attached and active during interval: 41;
    #SIP Communication Sessions attached and active during interval: 0;
    Total WLM enclave CPU time: 2917645;
    # Bytes transferred 8 byte fields
      ToServer: 2851637;
      FromServer: 26988724;
      LocalToServer: 0;
      LocalFromServer: 0;
      RemoteToServer: 0;
      RemoteFromServer: 0;
      Transferred to Server from http clients: 2851637;
      Transferred from Server to http clients: 26988724;
      Transferred to Server from SIP clients: 0;
      Transferred from Server to SIP clients: 0;

    Triplet #: 3; Type: ServerRegionSection;
      ASID                     * 000000fc -------- -------- -------- *

      #HeapIdSections: 2;
      Triplet #: 3.1; offsetDec: 32; offsetHex: 20; length: 56; count: 1;
```

```
   Triplet #: 3.2; offsetDec: 88; offsetHex: 58; length: 56; count: 1;

   Triplet #: 1; Type: HeapIdSection;
     HeapID                * 00000003 -------- -------- -------- *
     Total Garbage Collection Count: 9;
     Minimum Total Storage: 63307776;
     Maximum Total Storage: 63307776;
     Average Total Storage: 63307776;
     Minimum Free Storage : 14107448;
     Maximum Free Storage : 59786912;
     Average Free Storage : 29656271;

   Triplet #: 2; Type: HeapIdSection;
     HeapID                * 00000004 -------- -------- -------- *
     Total Garbage Collection Count: 0;
     Minimum Total Storage: 201326592;
     Maximum Total Storage: 201326592;
     Average Total Storage: 201326592;
     Minimum Free Storage : 77059096;
     Maximum Free Storage : 78894104;
     Average Free Storage : 77984765;

--------------------------------------------------------------------------------
```

E.3 WebContainer activity record: Subtype 7

Example E-5 shows the summary output of an SMF 120 subtype 7 (120.7) WebContainer activity record.

Example E-5 Subtype 7 summary

```
SMF -Record Time     Server   Bean/WebAppName  Bytes   Bytes  # of  El.Time  CPU_Time(uSec)     Other SMF 120.9
Numbr -Type hh:mm:ss Instance  Method/Servlet  toSvr   frSvr Calls   (msec)   Tot-CPU      zAAP Sections Present
1---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+ ----------------

  680 120.7 19:58:06 MZSR014
                               /account.jsp                              0        32
                               TradeAppServlet                          13       591
                              DayTrader-EE6#web.war
```

Example E-6 shows the detailed output of an SMF 120 subtype 7 (120.7) WebContainer activity record. It is the detailed output of the same record that is shown in Example E-5.

Example E-6 Subtype 7 detail

```
--------------------------------------------------------------------------------
Record#: 680;
  Type: 120; Size: 1148; Date: Fri Aug 10 19:58:06 EDT 2012;
  SystemID: SC64; SubsystemID: WAS; Flag: 94;
  Subtype: 7 (WEB CONTAINER ACTIVITY);

  # Triplets: 4;
  Triplet #: 1; offsetDec: 76; offsetHex: 4c; lengthDec: 32; lengthHex: 20; count: 1;
  Triplet #: 2; offsetDec: 108; offsetHex: 6c; lengthDec: 156; lengthHex: 9c; count: 1;
  Triplet #: 3; offsetDec: 264; offsetHex: 108; lengthDec: 16; lengthHex: 10; count: 1;
  Triplet #: 4; offsetDec: 280; offsetHex: 118; lengthDec: 868; lengthHex: 364; count: 1;

  Triplet #: 1; Type: ProductSection;
    Version: 2; Codeset: Unicode; Endian: 1; TimeStampFormat: 1 (S390STCK64);
    IndexOfThisRecord: 1; Total # records: 1; Total # triplets: 4;
```

```
Triplet #: 2; Type: WebContainerActivitySection;
  HostName         : WTSC64;
  ServerName       : MZSR01;
  ServerInstanceName: MZSR014;
  CellName         : MZCELL;
  NodeName         : MZNODE4;
  WlmEnclaveToken          * 0000011c 012075bf -------- -------- * ......Í×........ * Cp1047
  ActivityID              * ca00265f 3cf7d4eb 000002a4 00000062 * -..¬.7MÔ...u...Â * Cp1047
                         * 090c0609 -------- -------- -------- * ................ * Cp1047
  ActivityStartTime         * ca00265f 3cf7d4eb 00000000 00000000 *
  ActivityStopTime          * ca00265f 418d6820 00000000 00000000 *

Triplet #: 3; Type: HttpSessionManagerActivitySection;
  # http sessions created: 0; # http sessions invalidated: 0; # http sessions active: 0; Average session life time: 0
                                                                                                   Ýsec*10**-3¨

Triplet #: 4; Type: WebApplicationActivitySection;
  Name: DayTrader-EE6#web.war;

  # Servlets: 2;
  Triplet #: 4.1; offsetDec: 284; offsetHex: 11c; length: 292; count: 1;
  Triplet #: 4.2; offsetDec: 576; offsetHex: 240; length: 292; count: 1;

  Triplet #: 4.1; Type: ServletActivitySection;
    Name: /account.jsp;
    ResponseTime: 0 Ýsec*10**-3¨;
    # errors: 0;
    Loaded by this request: 0;
    Loaded since (raw): 13912f79b14;
    Loaded since: Fri Aug 10 19:56:20 EDT 2012;
    CPU Time: 32;

  Triplet #: 4.2; Type: ServletActivitySection;
    Name: TradeAppServlet;
    ResponseTime: 13 Ýsec*10**-3¨;
    # errors: 0;
    Loaded by this request: 0;
    Loaded since (raw): 13912f79a56;
    Loaded since: Fri Aug 10 19:56:20 EDT 2012;
    CPU Time: 591;

--------------------------------------------------------------------------------
```

E.4 WebContainer activity record: Subtype 8

Example E-7 shows the summary output of an SMF 120 subtype 8 (120.8) WebContainer interval record. It contains information that is similar to subtype 7, but it is written only at the specified server_SMF_interval_length (every 3 minutes, in our case).

Example E-7 Subtype 8 summary

```
SMF -Record Time     Server   Bean/WebAppName  Bytes   Bytes  # of  El.Time  CPU_Time(uSec)     Other SMF 120.9
Numbr -Type hh:mm:ss Instance  Method/Servlet  toSvr   frSvr Calls   (msec)   Tot-CPU      zAAP Sections Present
1---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+ ----------------
 5623 120.8 19:58:06 MZSR014
                               /tradehome.jsp                  84        1        90 -Av
                               /marketSummary.jsp              39        1        38 -Av
                               /order.jsp                       5        1        32 -Av
                               /account.jsp                    19        1        33 -Av
                               /register.jsp                    2        1        20 -Av
                               /displayQuote.jsp             1155        3       205 -Av
                               TradeAppServlet               3386       11       555 -Av
```

```
 /welcome.jsp                      2         1          9 -Av
 /portfolio.jsp                   38         1         70 -Av
 /quote.jsp                     1155         3        262 -Av
DayTrader-EE6#web.war
```

Example E-8 shows the detailed output of an SMF 120 subtype 8 (120.8) WebContainer interval record. It is the detailed output of the same record that is shown in Example E-7 on page 550.

Example E-8 Subtype 8 detail

```
--------------------------------------------------------------------------------
Record#: 5623;
  Type: 120; Size: 3824; Date: Fri Aug 10 19:58:06 EDT 2012;
  SystemID: SC64; SubsystemID: WAS; Flag: 94;
  Subtype: 8 (WEB CONTAINER INTERVAL);

  # Triplets: 4;
  Triplet #: 1; offsetDec: 76; offsetHex: 4c; lengthDec: 32; lengthHex: 20; count: 1;
  Triplet #: 2; offsetDec: 108; offsetHex: 6c; lengthDec: 128; lengthHex: 80; count: 1;
  Triplet #: 3; offsetDec: 236; offsetHex: ec; lengthDec: 44; lengthHex: 2c; count: 1;
  Triplet #: 4; offsetDec: 280; offsetHex: 118; lengthDec: 3544; lengthHex: dd8; count: 1;

  Triplet #: 1; Type: ProductSection;
    Version: 2; Codeset: Unicode; Endian: 1; TimeStampFormat: 1 (S390STCK64);
    IndexOfThisRecord: 1; Total # records: 1; Total # triplets: 4;

  Triplet #: 2; Type: WebContainerIntervalSection;
    HostName          : WTSC64;
    ServerName        : MZSR01;
    ServerInstanceName: MZSR014;
    CellName          : MZCELL;
    NodeName          : MZNODE4;
    SampleStartTime            * ca00265e 5809c227 00000000 00000000 *
    SampleStopTime             * ca002661 347fb56b 00000000 00000000 *

  Triplet #: 3; Type: HttpSessionManagerIntervalSection;
    http sessions #created: 233; #invalidated: 0;
    http sessions #active: 0; Min #active: 0; Max #active: 0;
    Average session life time: 0;
    Average session invalidate time: 0;
    http sessions #finalized: 0; #tracked: 0;
    http sessions #min live: 0; #max live: 0;

  Triplet #: 4; Type: WebApplicationIntervalSection;
    Name: DayTrader-EE6#web.war;

    # Servlets loaded: 0;
    # Servlets: 10;
    Triplet #: 4.1; offsetDec: 384; offsetHex: 180; length: 316; count: 1;
    Triplet #: 4.2; offsetDec: 700; offsetHex: 2bc; length: 316; count: 1;
    Triplet #: 4.3; offsetDec: 1016; offsetHex: 3f8; length: 316; count: 1;
    Triplet #: 4.4; offsetDec: 1332; offsetHex: 534; length: 316; count: 1;
    Triplet #: 4.5; offsetDec: 1648; offsetHex: 670; length: 316; count: 1;
    Triplet #: 4.6; offsetDec: 1964; offsetHex: 7ac; length: 316; count: 1;
    Triplet #: 4.7; offsetDec: 2280; offsetHex: 8e8; length: 316; count: 1;
    Triplet #: 4.8; offsetDec: 2596; offsetHex: a24; length: 316; count: 1;
   Triplet #: 4.9; offsetDec: 2912; offsetHex: b60; length: 316; count: 1;
   Triplet #: 4.10; offsetDec: 3228; offsetHex: c9c; length: 316; count: 1;
```

```
Triplet #: 4.1; Type: ServletIntervalSection;
  Name: /tradehome.jsp;
  # requests: 84;
  AverageResponseTime: 1 Ýsec*10**-3¨;
  MinimumResponseTime: 1 Ýsec*10**-3¨;
  MaximumResponseTime: 8 Ýsec*10**-3¨;
  # errors: 0;
  Loaded since (raw): 13912f79ad3;
  Loaded since: Fri Aug 10 19:56:20 EDT 2012;
  Average CPU Time: 90;
  Minimum CPU Time: 66;
  Maximum CPU Time: 1053;

Triplet #: 4.2; Type: ServletIntervalSection;
  Name: /marketSummary.jsp;
  # requests: 39;
  AverageResponseTime: 1 Ýsec*10**-3¨;
  MinimumResponseTime: 1 Ýsec*10**-3¨;
  MaximumResponseTime: 8 Ýsec*10**-3¨;
  # errors: 0;
  Loaded since (raw): 13912f79ade;
  Loaded since: Fri Aug 10 19:56:20 EDT 2012;
  Average CPU Time: 38;
  Minimum CPU Time: 28;
  Maximum CPU Time: 995;

Triplet #: 4.3; Type: ServletIntervalSection;
  Name: /order.jsp;
  # requests: 5;
  AverageResponseTime: 1 Ýsec*10**-3¨;
  MinimumResponseTime: 1 Ýsec*10**-3¨;
  MaximumResponseTime: 1 Ýsec*10**-3¨;
  # errors: 0;
  Loaded since (raw): 13912f79bed;
  Loaded since: Fri Aug 10 19:56:20 EDT 2012;
  Average CPU Time: 32;
  Minimum CPU Time: 25;
  Maximum CPU Time: 59;

Triplet #: 4.4; Type: ServletIntervalSection;
  Name: /account.jsp;
  # requests: 19;
  AverageResponseTime: 1 Ýsec*10**-3¨;
  MinimumResponseTime: 1 Ýsec*10**-3¨;
  MaximumResponseTime: 1 Ýsec*10**-3¨;
  # errors: 0;
  Loaded since (raw): 13912f79b14;
  Loaded since: Fri Aug 10 19:56:20 EDT 2012;
  Average CPU Time: 33;
  Minimum CPU Time: 24;
  Maximum CPU Time: 80;

Triplet #: 4.5; Type: ServletIntervalSection;
  Name: /register.jsp;
  # requests: 2;
  AverageResponseTime: 1 Ýsec*10**-3¨;
  MinimumResponseTime: 1 Ýsec*10**-3¨;
  MaximumResponseTime: 2 Ýsec*10**-3¨;
  # errors: 0;
  Loaded since (raw): 13912f79d7d;
```

```
  Loaded since: Fri Aug 10 19:56:21 EDT 2012;
  Average CPU Time: 20;
  Minimum CPU Time: 13;
  Maximum CPU Time: 38;

Triplet #: 4.6; Type: ServletIntervalSection;
  Name: /displayQuote.jsp;
  # requests: 1155;
  AverageResponseTime: 3 Ýsec*10**-3¨;
  MinimumResponseTime: 1 Ýsec*10**-3¨;
  MaximumResponseTime: 48 Ýsec*10**-3¨;
  # errors: 0;
  Loaded since (raw): 13912f79b2c;
  Loaded since: Fri Aug 10 19:56:20 EDT 2012;
  Average CPU Time: 205;
  Minimum CPU Time: 154;
  Maximum CPU Time: 333;

Triplet #: 4.7; Type: ServletIntervalSection;
  Name: TradeAppServlet;
  # requests: 3386;
  AverageResponseTime: 11 Ýsec*10**-3¨;
  MinimumResponseTime: 1 Ýsec*10**-3¨;
  MaximumResponseTime: 323 Ýsec*10**-3¨;
  # errors: 0;
  Loaded since (raw): 13912f79a56;
  Loaded since: Fri Aug 10 19:56:20 EDT 2012;
  Average CPU Time: 555;
  Minimum CPU Time: 199;
   Maximum CPU Time: 5739;

 Triplet #: 4.8; Type: ServletIntervalSection;
   Name: /welcome.jsp;
   # requests: 2;
   AverageResponseTime: 1 Ýsec*10**-3¨;
   MinimumResponseTime: 1 Ýsec*10**-3¨;
   MaximumResponseTime: 1 Ýsec*10**-3¨;
   # errors: 0;
   Loaded since (raw): 13912f79b92;
   Loaded since: Fri Aug 10 19:56:20 EDT 2012;
   Average CPU Time: 9;
   Minimum CPU Time: 7;
   Maximum CPU Time: 26;

 Triplet #: 4.9; Type: ServletIntervalSection;
   Name: /portfolio.jsp;
   # requests: 38;
   AverageResponseTime: 1 Ýsec*10**-3¨;
   MinimumResponseTime: 1 Ýsec*10**-3¨;
   MaximumResponseTime: 2 Ýsec*10**-3¨;
   # errors: 0;
   Loaded since (raw): 13912f79b38;
   Loaded since: Fri Aug 10 19:56:20 EDT 2012;
   Average CPU Time: 70;
   Minimum CPU Time: 32;
   Maximum CPU Time: 145;

 Triplet #: 4.10; Type: ServletIntervalSection;
   Name: /quote.jsp;
   # requests: 1155;
```

```
    AverageResponseTime: 3 Ýsec*10**-3¨;
    MinimumResponseTime: 1 Ýsec*10**-3¨;
    MaximumResponseTime: 48 Ýsec*10**-3¨;
    # errors: 0;
    Loaded since (raw): 13912f79b23;
    Loaded since: Fri Aug 10 19:56:20 EDT 2012;
    Average CPU Time: 262;
    Minimum CPU Time: 203;
    Maximum CPU Time: 414;

--------------------------------------------------------------------------------
```

F

Sample IBM Data Server Driver for JDBC and SQLJ trace

This appendix provides a sample output of an IBM Data Server Driver for JDBC and SQLJ trace of a short transaction from the DayTrader workload.

Example F-1 contains this output. It was captured with the **TRACE_ALL** setting, so it contains the most detailed information.

There is much information in a JCC trace, but the trace data sets can get larger quickly. Therefore, complete the following tasks:

- Activate the traces for the shortest possible time.
- Try to make the trace as selective as possible, both with regard to which applications are traced and the level of detail that is specified for the trace.
- Set up circular tracing, as described in “Specifying the JCC trace at the driver configuration properties level” on page 463.

Example F-1 Sample JCC trace of a single (short) transaction

```
[jcc][Time:2012-11-16-21:49:01.949][Thread:WebSphere WLM Dispatch Thread t=007bd580][Connection@13361385]isInDB2UnitOfWork () returned
false
[jcc][Time:2012-11-16-21:49:08.222][Thread:WebSphere WLM Dispatch Thread t=007bd580][Connection@13361385]clearWarnings () called
[jcc][Time:2012-11-16-21:49:08.222][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@41d88590]getFetchSize () returned 0
[jcc][Time:2012-11-16-21:49:08.222][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@41d88590]setString (1, uid:0)
called
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-21:49:08.222][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@41d88590]executeQuery () called
[jcc][t4] [time:2012-11-16-21:49:08.222][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:431]Before Executing,
AutoCommit=false RLSCONV=242
[jcc][t4] [time:2012-11-16-21:49:08.222][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:815]====== connected to primary
server = true
[jcc][t4][time:2012-11-16-21:49:08.222][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:600][findTransportObject called.
wantConnect = true haveTransport_ = false]
[jcc]findBestSysplexMember [time:2012-11-16-21:49:08.222][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:200] selected(first
time) currMember:0 currBestMember: 0 currMemberPriority: 33 currMemberRatio: 0.67346936 memberTransportsInUse_: 0 total connection: 0
[jcc]findBestSysplexMember [time:2012-11-16-21:49:08.222][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:200]{SWLBG@51321542:
/9.12.4.153 39000 DBOZ 4 2 0 49
[jcc]findBestSysplexMember [time:2012-11-16-21:49:08.222][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:200] member:0
{SWLBN@b81cb1cf: /9.12.4.138 39000 33 0.67346936 false 10 1 2 50 0
[jcc]findBestSysplexMember [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:200] member:1
{SWLBN@f2a28587: /9.12.4.142 39000 16 0.3265306 false 0 1 0 0 0
[jcc]findBestSysplexMember [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:200] bestMember: 0
{SWLBN@b81cb1cf: /9.12.4.138 39000 33 0.67346936 false 10 1 2 50 0
```



```
[jcc]findBestSysplexMember [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:222] incrTranCount -
WebSphere WLM Dispatch Thread t=007bd580 bestmemberIndex: 0     {SWLBN@b81cb1cf: /9.12.4.138 39000 33 0.67346936 false 10 1 2 51 1
[jcc][Time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][Connection@13361385]isClosed () returned false
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:701]getting transport object from
pool with timeToDeadLine: 0
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:620]poolKey.connectionNeedsReset_ =
false
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:625] getNewServerList_ = false
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:635]got Transport: {T4GTPK@e1897bf9:
9.12.4.138 39000 rajesh 3 DB0Z 1 153873318 38 false true DB2XADataSource@3cc7dc83:1,2147483647 a@e6c9ee52 true
com.ibm.db2.jcc.t4.vb@9a255c79[SET CURRENT SCHEMA = 'SG248074']
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:640]newTransport.dbConnected_ = true
resetConnectionAtFirstSql_ = false
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:210] comparing special register size.
On connection 1 on transport 1 genericSQLSetPiggyBackCommand=com.ibm.db2.jcc.am.eg@a32c1b6b {T4GTPK@e1897bf9: 9.12.4.138 39000 rajesh 3
DB0Z 1 153873318 38 false true DB2XADataSource@3cc7dc83:1,2147483647 a@e6c9ee52 true com.ibm.db2.jcc.t4.vb@9a255c79[SET CURRENT SCHEMA =
'SG248074']
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:220]On Connection: : SET CURRENT
SCHEMA = 'SG248074'
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:220]On Transport : : SET CURRENT
SCHEMA = 'SG248074'
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:220]Old SpecialRegs:
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread
t=007bd580][tracepoint:230]piggybackCommandQueue=com.ibm.db2.jcc.am.fg@d673b200 count_=0 next_=null previous_=null head_=null tail_=null
piggybackCommand_=null
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:220]New SpecialRegs:
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:220]PiggyBackCmds  :
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread
t=007bd580][tracepoint:230]piggybackCommandQueue=com.ibm.db2.jcc.am.fg@d673b200 count_=0 next_=null previous_=null head_=null tail_=null
piggybackCommand_=null
[jcc][t4][time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:500][matchMGRLVL. T4Agent Manager
Level  10 8 10 9 5 7 0 0 TranportObject Manager Level  10 8 10 9 5 7 0 0]
[jcc][t4][time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:660][findTransportObject exited.
Comparing special register size. On connection 1 on transport 1]
[jcc][Time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][T4XAResource@7e1f760f]makeEntryCurrent(new,old) (0,
0) called
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:210] comparing special register size.
On connection 1 on transport 1 genericSQLSetPiggyBackCommand=com.ibm.db2.jcc.am.eg@a32c1b6b {T4GTPK@e1897bf9: 9.12.4.138 39000 rajesh 3
DB0Z 1 153873318 38 false true DB2XADataSource@3cc7dc83:1,2147483647 a@e6c9ee52 true com.ibm.db2.jcc.t4.vb@9a255c79[SET CURRENT SCHEMA =
'SG248074']
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:220]On Connection: : SET CURRENT
SCHEMA = 'SG248074'
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:220]On Transport : : SET CURRENT
SCHEMA = 'SG248074'
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:220]Old SpecialRegs:
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread
t=007bd580][tracepoint:230]piggybackCommandQueue=com.ibm.db2.jcc.am.fg@d673b200 count_=0 next_=null previous_=null head_=null tail_=null
piggybackCommand_=null
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:220]New SpecialRegs:
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:220]PiggyBackCmds  :
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread
t=007bd580][tracepoint:230]piggybackCommandQueue=com.ibm.db2.jcc.am.fg@d673b200 count_=0 next_=null previous_=null head_=null tail_=null
piggybackCommand_=null
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:322]T4XAResource saving Transport[0]:
{T4GTPK@e1897bf9: 9.12.4.138 39000 rajesh 3 DB0Z 1 153873318 38 false true DB2XADataSource@3cc7dc83:1,2147483647 a@e6c9ee52 true
com.ibm.db2.jcc.t4.vb@9a255c79[SET CURRENT SCHEMA = 'SG248074'] Xid: {DB2Xid: formatID(-1), gtrid_length(0), bqual_length(0), data()}
freeEntry: false
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:324]Before:
conn_.resetConnectionAtFirstSql_=false
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:203]T4XAResource sw to Transport[0]:
{T4GTPK@e1897bf9: 9.12.4.138 39000 rajesh 3 DB0Z 1 153873318 38 false true DB2XADataSource@3cc7dc83:1,2147483647 a@e6c9ee52 true
com.ibm.db2.jcc.t4.vb@9a255c79[SET CURRENT SCHEMA = 'SG248074'] Xid: {DB2Xid: formatID(-1), gtrid_length(0), bqual_length(0), data()}
freeEntry: false
[jcc][t4] [time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:325]After:
conn_.resetConnectionAtFirstSql_=false
[jcc][Time:2012-11-16-21:49:08.223][Thread:WebSphere WLM Dispatch Thread t=007bd580][Connection@13361385]getDB2Correlator () returned
::9.12.6.9.65123.CA7B405C24DB
[jcc][Connection@13361385] BEGIN TRACE_CONNECTS
[jcc][Connection@13361385] Successfully connected to server jdbc:db2://9.12.4.153:39000/DB0Z
[jcc][Connection@13361385] User: rajesh
[jcc][Connection@13361385] Database product name: DB2
[jcc][Connection@13361385] Database product version: DSN10015
[jcc][Connection@13361385] Driver name: IBM DB2 JDBC Universal Driver Architecture
[jcc][Connection@13361385] Driver version: 3.64.82
[jcc][Connection@13361385] DB2 Application Correlator: ::9.12.6.9.65123.CA7B405C24DB.0000
[jcc][Connection@13361385] END TRACE_CONNECTS
[jcc][t4]   DRDA manager levels: { SQLAM=10, AGENT=10, CMNTCPIP=5, RDB=8, SECMGR=9, XAMGR=7, SYNCPTMGR=0, RSYNCMGR=0 }
[jcc][Time:2012-11-16-21:49:08.224][Thread:WebSphere WLM Dispatch Thread t=007bd580][Connection@13361385]setDB2ClientUser
(TraderClientUser) called
[jcc][Time:2012-11-16-21:49:08.224][Thread:WebSphere WLM Dispatch Thread t=007bd580][Connection@13361385]setDB2ClientWorkstation
(TraderClientWorkstation) called
[jcc][Time:2012-11-16-21:49:08.224][Thread:WebSphere WLM Dispatch Thread
t=007bd580][Connection@13361385]setDB2ClientApplicationInformation (TraderClientApplicationInformation) called
```

```
[jcc][Time:2012-11-16-21:49:08.224][Thread:WebSphere WLM Dispatch Thread t=007bd580][Connection@13361385]setDB2ClientAccountingInformation
(TraderClientAccountingInformation) called
[jcc][t4] [time:2012-11-16-21:49:08.224][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:208]T4XAResource setting
XACallInfo[0] Xid: {DB2Xid: formatID(-1), gtrid_length(0), bqual_length(0), data()} freeEntry: false
[jcc][t4][time:2012-11-16-21:49:08.224][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:1][Request.flush]
[jcc][t4]       SEND BUFFER: PRPSQLSTT              (ASCII)           (EBCDIC)
[jcc][t4]       0 1 2 3 4 5 6 7   8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
[jcc][t4] 0000  0056D05100010050  200D004421134442  .V.Q...P ..D!.DB  ..}....&........
[jcc][t4] 0010  305A202020202020  2020202020202020  0Z                .!..............
[jcc][t4] 0020  4E554C4C49442020  2020202020202020  NULLID            +.<<............
[jcc][t4] 0030  20205359534C4E33  3030202020202020    SYSLN300        .....<+.........
[jcc][t4] 0040  202020205359534C  564C303100060008      SYSLVL01....   .......<.<......
[jcc][t4] 0050  1900A0000000                        ......            ......
[jcc][t4]
[jcc][t4]       SEND BUFFER: SQLATTR                (ASCII)           (EBCDIC)
[jcc][t4] 0000  0037D05300010031  2450000000002746  .7.S...1$P....'F  ..}......&......
[jcc][t4] 0010  4F52205245414420  4F4E4C5920574954  OR READ ONLY WIT  |.......|+<.....
[jcc][t4] 0020  4820455854454E44  454420494E444943  H EXTENDED INDIC  ......+.....+...
[jcc][t4] 0030  41544F525320FF                      ATORS .           ..|....
[jcc][t4]
[jcc][t4]       SEND BUFFER: SQLSTT                 (ASCII)           (EBCDIC)
[jcc][t4] 0000  00A3D04300010009D  2414000000009373  ...C....$......s  .t}...........l.
[jcc][t4] 0010  656C656374202A20  66726F6D206F7264  elect * from ord  .%........?_.?..
[jcc][t4] 0020  6572656A62206F20  7768657265206F2E  erejb o where o.  ...|..?.......?.
[jcc][t4] 0030  6F72646572737461  747573203D202763  orderstatus = 'c  ?....../........
[jcc][t4] 0040  6C6F736564272041  4E44206F2E616363  losed' AND o.acc  %?......+..?./..
[jcc][t4] 0050  6F756E745F616363  6F756E746964203D  ount_accountid =  ?.>../..?.>.....
[jcc][t4] 0060  202873656C656374  20612E6163636F75   (select a.accou  ....%..../../..?.
[jcc][t4] 0070  6E7469642066726F  6D206163636F756E  ntid from accoun  >......?_./..?.>
[jcc][t4] 0080  74656A6220612077  6865726520612E70  tejb a where a.p  ..|../......./..
[jcc][t4] 0090  726F66696C655F75  7365726964203D20  rofile_userid =   .?..%...........
[jcc][t4] 00A0  3F29FF                              ?).               ...
[jcc][t4]
[jcc][t4]       SEND BUFFER: OPNQRY                 (ASCII)           (EBCDIC)
[jcc][t4] 0000  0091D0510002008B  200C004421134442  ...Q.... ..D!.DB  .j}.............
[jcc][t4] 0010  305A202020202020  2020202020202020  0Z                .!..............
[jcc][t4] 0020  4E554C4C49442020  2020202020202020  NULLID            +.<<............
[jcc][t4] 0030  20205359534C4E33  3030202020202020    SYSLN300        .....<+.........
[jcc][t4] 0040  202020205359534C  564C303100060008      SYSLVL01....   .......<.<......
[jcc][t4] 0050  211400007FFF0005  215D0100081900E0  !.......!]......  ...."....).....\
[jcc][t4] 0060  0000000005214703  0005214BF1000C21  .....!G...!K...!  ............1...
[jcc][t4] 0070  3700000000001000  00000C2136000000  7..........!6...  ................
[jcc][t4] 0080  0000003000000C21  340000000000A000  ...0...!4.......  ................
[jcc][t4] 0090  00                                  .                 .
[jcc][t4]
[jcc][t4]       SEND BUFFER: SQLDTA                 (ASCII)           (EBCDIC)
[jcc][t4] 0000  0027D00300020021  2412001000100676  .'.....!$......v  ..}.............
[jcc][t4] 0010  D03F7FFF0671E4D0  0001000D147A0000  .?...q.......z..  }."...U}.....:..
[jcc][t4] 0020  00057569643A30                      ..uid:0           .......
[jcc][t4]
[jcc][t4] [time:2012-11-16-21:49:08.224][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:101]Request flushed.
[jcc][t4] [time:2012-11-16-21:49:08.224][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:102]Reply to be filled.
[jcc][t4][time:2012-11-16-21:49:08.225][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:2][Reply.fill]
[jcc][t4]       RECEIVE BUFFER: SQLCARD             (ASCII)           (EBCDIC)
[jcc][t4]       0 1 2 3 4 5 6 7   8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
[jcc][t4] 0000  0057D05300010051  2408000000000030  .W.S...Q$......0  ..}.............
[jcc][t4] 0010  3030303044534E20  2020202000000000  0000DSN     ....  ......+.........
[jcc][t4] 0020  0000000000000000  054056CDEC000000  .........@V.....  ......... ......
[jcc][t4] 0030  0000000000202020  2020202020202020  .....             ................
[jcc][t4] 0040  00104442305A2020  2020202020202020  ..DB0Z            .....!..........
[jcc][t4] 0050  202000000000FF                        .....           .......
[jcc][t4]
[jcc][t4]       RECEIVE BUFFER: MONITORRD           (ASCII)           (EBCDIC)
[jcc][t4] 0000  0046D04300010040  1C00000C19010000  .F.C...@........  ..}.... ........
[jcc][t4] 0010  00000000006C000C  1915000000000000  .....l..........  .....%..........
[jcc][t4] 0020  00070018116D4430  5A312E4442305A2E  .....mD0Z1.DB0Z.  ....._..!.....!.
[jcc][t4] 0030  44305A312E444230  5A47000C112E4453  D0Z1.DB0ZG....DS  ..!.....!.......
[jcc][t4] 0040  4E3130303135                        N10015            +.....
[jcc][t4]
[jcc][t4]       RECEIVE BUFFER: OPNQRYRM            (ASCII)           (EBCDIC)
[jcc][t4] 0000  002CD05200020026  2205000611490000  .,.R...&"....I..  ..}.............
[jcc][t4] 0010  0006210224170005  215001000C215B00  ..!.$...!P...![.  .........&....$.
[jcc][t4] 0020  00024177CE4A3000  05214BF1          ..Aw.J0..!K.      ...........1
[jcc][t4]
[jcc][t4]       RECEIVE BUFFER: QRYDSC              (ASCII)           (EBCDIC)
[jcc][t4] 0000  0078D05300020072  241A077800050101  .x.S...r$..x....  ..}.............
[jcc][t4] 0010  250C705090000000  2501000020077800  %.pP....%... .x.  ...&............
[jcc][t4] 0020  050101330C705191  0000002501017FFF  ...3.pQ....%....  .......j......".
[jcc][t4] 0030  077800050201D024  76D00F0E0250001A  .x.....$v....P..  ......}..}...&..
[jcc][t4] 0040  5100FF5100FF0F0E  020A000850001A02  Q..Q........P...  ............&...
[jcc][t4] 0050  00040300045100FF  0300040778000503  .....Q......x...  ................
[jcc][t4] 0060  01E00971E0540001  D000010778000504  ...q.T......x...  .\..\...}.......
[jcc][t4] 0070  01F00671F0E00000                    ...q....          .0..0\..
```

```
[jcc][t4]
[jcc][t4]        RECEIVE BUFFER: QRYDTA              (ASCII)           (EBCDIC)
[jcc][t4] 0000   0085D05300028008  241B00000077FF00  ...S....$....w..  .e}.............
[jcc][t4] 0010   0000000000000249  5C00F2F0F1F260F1  .......I\.....`.  ........*.2012-1
[jcc][t4] 0020   F160F1F660F2F14B  F4F94BF0F14BF8F9  .`..`..K..K..K..  1-16-21.49.01.89
[jcc][t4] 0030   F2F0F0F000000382  A4A8000006839396  ................  2000...buy...clo
[jcc][t4] 0040   A285840000000000  0019421C42640000  ..........B.Bd..  sed.............
[jcc][t4] 0050   0000000000F2F0F1  F260F1F160F1F660  .........`..`..`  .....2012-11-16-
[jcc][t4] 0060   F2F14BF4F94BF0F1  4BF2F9F9F0F0F000  ..K..K..K.......  21.49.01.299000.
[jcc][t4] 0070   001F420000000000  000005A27AF1F6F8  ..B.........z...  ...........s:168
[jcc][t4] 0080   0000001771                          ....q             .....
[jcc][t4]
[jcc][t4]        RECEIVE BUFFER: ENDQRYRM            (ASCII)           (EBCDIC)
[jcc][t4] 0000   0026D05200020020  220B000611490004  .&.R... "....I..  ..}.............
[jcc][t4] 0010   001621104442305A  2020202020202020  ..!.DB0Z          .......!........
[jcc][t4] 0020   202020202020                                          ......
[jcc][t4]
[jcc][t4]        RECEIVE BUFFER: SQLCARD             (ASCII)           (EBCDIC)
[jcc][t4] 0000   0057D05300020051  2408000000006430  .W.S...Q$.....d0  ..}.............
[jcc][t4] 0010   3230303044534E58  52464E2000FFFFFF  2000DSNXRFN ....  ......+...+.....
[jcc][t4] 0020   9200000000000000  0000000000000000  ................  k...............
[jcc][t4] 0030   0000000000202020  2020202020202020  .....             ................
[jcc][t4] 0040   00104442305A2020  2020202020202020  ..DB0Z            .....!..........
[jcc][t4] 0050   202000000000FF                        .....           .......
[jcc][t4]
[jcc][t4]        RECEIVE BUFFER: MONITORRD           (ASCII)           (EBCDIC)
[jcc][t4] 0000   006AD00300020064  1C00000C19010000  .j.....d........  .|}.............
[jcc][t4] 0010   0000000001060024  1914800000002012  .......$...... .  ................
[jcc][t4] 0020   1114160835117280  0000000000000000  ....5.r.........  ................
[jcc][t4] 0030   00A0000000000000  0000000C19150000  ................  ................
[jcc][t4] 0040   0000000000070018  116D44305A312E44  .........mD0Z1.D  ........._..!...
[jcc][t4] 0050   42305A2E44305A31  2E4442305A47000C  B0Z.D0Z1.DB0ZG..  ..!...!.....!...
[jcc][t4] 0060   112E44534E313030  3135              ..DSN10015        ....+.....
[jcc][t4]
[jcc][t4] [time:2012-11-16-21:49:08.225][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:510]initXid -
com.ibm.db2.jcc.t4.yb@7e1f760f xid_ = {DB2Xid: formatID(-1), gtrid_length(0), bqual_length(0), data()} t4Connection_.currXACallInfoOffset_
= 0
[jcc][t4] [time:2012-11-16-21:49:08.225][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:431]After Executing, AutoCommit=false
RLSCONV=240
[jcc][Time:2012-11-16-21:49:08.225][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@41d88590]executeQuery () returned
com.ibm.db2.jcc.t4.i@9a783140
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 3.0797499999999998ms | network: 0.921125ms | server:
0.37ms [STMT@1104709008]
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-21:49:08.225][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]next () called
[jcc][Time:2012-11-16-21:49:08.225][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]next () returned true
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 0.15337499999999998ms | network: 0.0ms | server: 0.0ms
[STMT@1104709008]
[jcc][Time:2012-11-16-21:49:08.225][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getInt (orderID) called
[jcc][Time:2012-11-16-21:49:08.225][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getInt (8) called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getInt () returned 8002
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getString (orderType) called
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getString (3) called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getString () returned buy
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 0.03125ms | network: 0.0ms | server: 0.0ms
[STMT@1104709008]
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getString (orderStatus) called
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getString (4) called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getString () returned closed
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 0.025875ms | network: 0.0ms | server: 0.0ms
[STMT@1104709008]
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getTimestamp (openDate) called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getTimestamp (7) called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getTimestamp () returned
2012-11-16 21:49:01.299
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getTimestamp (completionDate)
called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getTimestamp (2) called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getTimestamp () returned
2012-11-16 21:49:01.892
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getDouble (quantity) called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getDouble (6) called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getDouble () returned 100.0
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getBigDecimal (price) called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getBigDecimal (5) called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getBigDecimal () returned 194.21
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getBigDecimal (orderFee) called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getBigDecimal (1) called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getBigDecimal () returned 24.95
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getString (quote_symbol) called
[jcc][SystemMonitor:start]
```

```
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getString (10) called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]getString () returned s:168
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 0.028499999999999998ms | network: 0.0ms | server: 0.0ms
[STMT@1104709008]
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@7b1f5c83]getFetchSize () returned 0
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@7b1f5c83]setString (1, completed)
called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@7b1f5c83]setTimestamp (2,
2012-11-16 21:49:08.226) called
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@7b1f5c83]setInt (3, 8002) called
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@7b1f5c83]executeUpdate () called
[jcc][t4] [time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:432]Before Executing,
AutoCommit=false RLSCONV=240
[jcc][t4] [time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:816]====== connected to primary
server = true
[jcc][t4][time:2012-11-16-21:49:08.226][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:1][Request.flush]
[jcc][t4]        SEND BUFFER: PRPSQLSTT              (ASCII)           (EBCDIC)
[jcc][t4]        0 1 2 3 4 5 6 7   8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
[jcc][t4] 0000   0056D05100010050  200D004421134442  .V.Q...P ..D!.DB  ..}....&........
[jcc][t4] 0010   305A202020202020  2020202020202020  0Z                .!..............
[jcc][t4] 0020   4E554C4C49442020  2020202020202020  NULLID            +.<<............
[jcc][t4] 0030   20205359534C4E33  3030202020202020    SYSLN300        .....<+.........
[jcc][t4] 0040   202020205359534C  564C303100100008      SYSLVL01....  .......<.<......
[jcc][t4] 0050   1900A0000000                        ......            ......
[jcc][t4]
[jcc][t4]        SEND BUFFER: SQLATTR                (ASCII)           (EBCDIC)
[jcc][t4] 0000   0029D05300010023  2450000000001957  .).S...#$P.....W  ..}......&......
[jcc][t4] 0010   4954482045585445  4E44454420494E44  ITH EXTENDED IND  ........+.....+.
[jcc][t4] 0020   494341544F525320  FF                ICATORS .         ....|....
[jcc][t4]
[jcc][t4]        SEND BUFFER: SQLSTT                 (ASCII)           (EBCDIC)
[jcc][t4] 0000   0059D04300010053  2414000000004975  .Y.C...S$.....Iu  ..}.............
[jcc][t4] 0010   7064617465206F72  646572656A622073  pdate orderejb s  ../...?.....|...
[jcc][t4] 0020   6574206F72646572  737461747573203D  et orderstatus =  ...?....../.....
[jcc][t4] 0030   203F2C20636F6D70  6C6574696F6E6461   ?, completionda  .....?_.%...?>./
[jcc][t4] 0040   7465203D203F2077  68657265206F7264  te = ? where ord  .............?..
[jcc][t4] 0050   65726964203D203F  FF                erid = ?.         .........
[jcc][t4]
[jcc][t4]        SEND BUFFER: EXCSQLSTT              (ASCII)           (EBCDIC)
[jcc][t4] 0000   005BD05100020055  200B004421134442  .[.Q...U ..D!.DB  .$}.............
[jcc][t4] 0010   305A202020202020  2020202020202020  0Z                .!..............
[jcc][t4] 0020   4E554C4C49442020  2020202020202020  NULLID            +.<<............
[jcc][t4] 0030   20205359534C4E33  3030202020202020    SYSLN300        .....<+.........
[jcc][t4] 0040   202020205359534C  564C303100100005      SYSLVL01....  .......<.<......
[jcc][t4] 0050   2105F100081900E0  000000            !..........       ..1....\...
[jcc][t4]
[jcc][t4]        SEND BUFFER: SQLDTA                 (ASCII)           (EBCDIC)
[jcc][t4] 0000   0057D00300020051  2412001600100C76  .W.....Q$......v  ..}.............
[jcc][t4] 0010   D03F7FFF25002003  00040671E4D00001  .?..%. ....q....  }."........U}..
[jcc][t4] 0020   0037147A00000009  636F6D706C657465  .7.z....complete  ...:.....?_.%...
[jcc][t4] 0030   6400323031322D31  312D31362D32312E  d.2012-11-16-21.  ................
[jcc][t4] 0040   34392E30382E3232  3630303030303030  49.08.2260000000  ................
[jcc][t4] 0050   30300000001F42                      00....B           .......
[jcc][t4]
[jcc][t4] [time:2012-11-16-21:49:08.227][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:101]Request flushed.
[jcc][t4] [time:2012-11-16-21:49:08.227][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:102]Reply to be filled.
[jcc][t4][time:2012-11-16-21:49:08.230][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:2][Reply.fill]
[jcc][t4]        RECEIVE BUFFER: SQLCARD             (ASCII)           (EBCDIC)
[jcc][t4]        0 1 2 3 4 5 6 7   8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
[jcc][t4] 0000   0057D05300010051  2408000000000030  .W.S...Q$......0  ..}.............
[jcc][t4] 0010   3030303044534E20  2020202000000000  0000DSN     ....  ......+.........
[jcc][t4] 0020   0000000000000000  003FF83568000000  .........?.5h...  ..........8.....
[jcc][t4] 0030   0000000000202020  2020202020202020  .....             ................
[jcc][t4] 0040   00104442305A2020  2020202020202020  ..DB0Z            .....!..........
[jcc][t4] 0050   202000000000FF                        .....           .......
[jcc][t4]
[jcc][t4]        RECEIVE BUFFER: MONITORRD           (ASCII)           (EBCDIC)
[jcc][t4] 0000   0046D04300010040  1C00000C19010000  .F.C...@........  ..}.... ........
[jcc][t4] 0010   000000000018000C  1915000000000000  ................  ................
[jcc][t4] 0020   00070018116D4430  5A312E4442305A2E  .....mD0Z1.DB0Z.  ....._..!.....!.
[jcc][t4] 0030   44305A312E444230  5A47000C112E4453  D0Z1.DB0ZG....DS  ..!.....!.......
[jcc][t4] 0040   4E3130303135                        N10015            +.....
[jcc][t4]
[jcc][t4]        RECEIVE BUFFER: RDBUPDRM            (ASCII)           (EBCDIC)
[jcc][t4] 0000   0026D05200020020  2218000611490000  .&.R... "....I..  ..}.............
[jcc][t4] 0010   001621104442305A  2020202020202020  ..!.DB0Z          .......!........
[jcc][t4] 0020   202020202020                                          ......
[jcc][t4]
[jcc][t4]        RECEIVE BUFFER: SQLCARD             (ASCII)           (EBCDIC)
[jcc][t4] 0000   0057D05300020051  2408000000000030  .W.S...Q$......0  ..}.............
[jcc][t4] 0010   3030303044534E20  2020202000000000  0000DSN     ....  ......+.........
[jcc][t4] 0020   0000000000000000  01FFFFFFFF000000  ................  ................
```

```
[jcc][t4] 0030   0000000000202020  2020202020202020  .....             ................
[jcc][t4] 0040   00104442305A2020  2020202020202020  ..DBOZ            .....!..........
[jcc][t4] 0050   202000000000FF                        .....           .......
[jcc][t4]
[jcc][t4]        RECEIVE BUFFER: MONITORRD           (ASCII)           (EBCDIC)
[jcc][t4] 0000   006AD00300020064  1C00000C19010000  .j.....d........  .|}.............
[jcc][t4] 0010   000000000CCC0024  1914800000002012  .......$...... .  ................
[jcc][t4] 0020   1114160835120025  0000000000000000  ....5..%........  ................
[jcc][t4] 0030   00A1000000000000  0000000C19150000  ................  .~..............
[jcc][t4] 0040   0000000000070018  116D44305A312E44  .........mDOZ1.D  ........._..!...
[jcc][t4] 0050   42305A2E44305A31  2E4442305A47000C  BOZ.DOZ1.DBOZG..  ..!...!.....!...
[jcc][t4] 0060   112E44534E313030  3135              ..DSN10015        ....+.....
[jcc][t4]
[jcc][t4] [time:2012-11-16-21:49:08.230][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:432]After Executing, AutoCommit=false
RLSCONV=240
[jcc][Time:2012-11-16-21:49:08.230][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@7b1f5c83]executeUpdate () returned
1
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 4.1976249999999995ms | network: 3.7115ms | server:
3.3000000000000003ms [STMT@2065652867]
[jcc][Time:2012-11-16-21:49:08.230][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@7b1f5c83]getMoreResults () called
[jcc][Time:2012-11-16-21:49:08.230][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@7b1f5c83]getMoreResults () returned
false
[jcc][Time:2012-11-16-21:49:08.230][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@7b1f5c83]getUpdateCount () called
[jcc][Time:2012-11-16-21:49:08.230][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@7b1f5c83]getUpdateCount () returned
-1
[jcc][Time:2012-11-16-21:49:08.230][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@7b1f5c83]clearParameters () called
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]next () called
[jcc][Time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]closeX ({DB2Xid: formatID(-1),
gtrid_length(0), bqual_length(0), data()}, com.ibm.db2.jcc.t4.T4XAConnection@13361385) called
[jcc][Time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]markClosed () called
[jcc][Time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]next () returned false
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 0.10062499999999999ms | network: 0.0ms | server: 0.0ms
[STMT@1104709008]
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]close () called
[jcc][Time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][ResultSet@9a783140]closeX ({DB2Xid: formatID(-1),
gtrid_length(0), bqual_length(0), data()}, com.ibm.db2.jcc.t4.T4XAConnection@13361385) called
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 0.08199999999999999ms | network: 0.0ms | server: 0.0ms
[STMT@1104709008]
[jcc][Time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@41d88590]getMoreResults () called
[jcc][Time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@41d88590]getMoreResults () returned
false
[jcc][Time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@41d88590]getUpdateCount () called
[jcc][Time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@41d88590]getUpdateCount () returned
-1
[jcc][Time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@41d88590]clearParameters () called
[jcc][SystemMonitor:start]
[jcc][Time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][Connection@13361385]commit () called
[jcc][t4][time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:1][Request.flush]
[jcc][t4]        SEND BUFFER: RDBCMM                 (ASCII)           (EBCDIC)
[jcc][t4]        0 1 2 3 4 5 6 7   8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
[jcc][t4] 0000   000FD00100010009  200E0005119FF2    ........ ......   ..}...........2
[jcc][t4]
[jcc][t4] [time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:101]Request flushed.
[jcc][t4] [time:2012-11-16-21:49:08.231][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:102]Reply to be filled.
[jcc][t4][time:2012-11-16-21:49:08.233][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:2][Reply.fill]
[jcc][t4]        RECEIVE BUFFER: ENDUOWRM            (ASCII)           (EBCDIC)
[jcc][t4]        0 1 2 3 4 5 6 7   8 9 A B C D E F   0123456789ABCDEF  0123456789ABCDEF
[jcc][t4] 0000   0030D0520001002A  220C000611490004  .0.R...*"....I..  ..}.............
[jcc][t4] 0010   001621104442305A  2020202020202020  ..!.DBOZ          .......!........
[jcc][t4] 0020   2020202020200005  2115010005119FF2        ..!.......  ...............2
[jcc][t4]
[jcc][t4]        RECEIVE BUFFER: SQLCARD             (ASCII)           (EBCDIC)
[jcc][t4] 0000   000BD05300010005  2408FF            ...S....$..       ..}........
[jcc][t4]
[jcc][t4]        RECEIVE BUFFER: SQLSTT              (ASCII)           (EBCDIC)
[jcc][t4] 0000   002FD00300010029  2414000000001F53  ./.....)$......S  ..}.............
[jcc][t4] 0010   4554204355525245  4E54205343484540  ET CURRENT SCHEM  ........+......(
[jcc][t4] 0020   41203D2027534732  343830373427FF    A = 'SG248074'.   ...............
[jcc][t4]
[jcc][t4] [time:2012-11-16-21:49:08.233][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:720]parseRLSCONV: 119f=f2
[jcc][Connection@13361385] DB2 LUWID: ::9.12.6.9.65123.CA7B405C24DB.0007
[jcc][t4] [time:2012-11-16-21:49:08.233][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:300]currXACallInfoOffset : 0commit
[jcc][t4] [time:2012-11-16-21:49:08.233][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:204]parseSQLSTTList :
{T4GTPK@e1897bf9: 9.12.4.138 39000 rajesh 3 DBOZ 1 153873318 38 false true DB2XADataSource@3cc7dc83:1,2147483647 a@e6c9ee52 true
com.ibm.db2.jcc.t4.vb@9a255c79[SET CURRENT SCHEMA = 'SG248074']
[jcc][t4] [time:2012-11-16-21:49:08.233][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:220]On Connection: : SET CURRENT
SCHEMA = 'SG248074'
[jcc][t4] [time:2012-11-16-21:49:08.233][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:220]On Transport : : SET CURRENT
SCHEMA = 'SG248074'
[jcc]determineMemberNumber [time:2012-11-16-21:49:08.233][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:250]WebSphere WLM
Dispatch Thread t=007bd580 i= 0
```

```
[jcc][t4] [time:2012-11-16-21:49:08.233][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:222] freeing Transport - WebSphere
WLM Dispatch Thread t=007bd580 bestMemberIndex: 0        {SWLBN@b81cb1cf: /9.12.4.138 39000 33 0.67346936 false 10 1 2 51 0
{T4GTPK@e1897bf9: 9.12.4.138 39000 rajesh 3 DB0Z 1 153873318 38 false true DB2XADataSource@3cc7dc83:1,2147483647 a@e6c9ee52 true
com.ibm.db2.jcc.t4.vb@9a255c79[SET CURRENT SCHEMA = 'SG248074']
[jcc][t4] [time:2012-11-16-21:49:08.233][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:220]On Connection: : SET CURRENT
SCHEMA = 'SG248074'
[jcc][t4] [time:2012-11-16-21:49:08.233][Thread:WebSphere WLM Dispatch Thread t=007bd580][tracepoint:220]On Transport : : SET CURRENT
SCHEMA = 'SG248074'
[jcc][Time:2012-11-16-21:49:08.233][Thread:WebSphere WLM Dispatch Thread t=007bd580][Connection@13361385]commit () returned null
[jcc][Thread:WebSphere WLM Dispatch Thread t=007bd580][SystemMonitor:stop] core: 2.367375ms | network: 1.9597499999999999ms | server:
0.0ms
[jcc][Time:2012-11-16-21:49:08.269][Thread:WebSphere WLM Dispatch Thread t=007bd580][Connection@13361385]isInDB2UnitOfWork () returned
false
[jcc][Time:2012-11-16-21:49:11.040][Thread:WebSphere WLM Dispatch Thread t=007bd580][Connection@13361385]clearWarnings () called
[jcc][Time:2012-11-16-21:49:11.041][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@41d88590]getFetchSize () returned 0
[jcc][Time:2012-11-16-21:49:11.041][Thread:WebSphere WLM Dispatch Thread t=007bd580][PreparedStatement@41d88590]setString (1, uid:0)
called
```

G

External user-defined functions

In this appendix, we provide the details about two external user-defined functions (UDF) that are used in our scenarios, which are described in Chapter 4, “DB2 infrastructure setup” on page 99.

The UDF routines are:

- UDF GRACFGRP
- UDF BIGINT

G.1 UDF GRACFGRP

The external UDF GRACFGRP is an assembler program that extracts the RACF groups that the current UDF caller is connected to from the RACF ACEE control block, which was created by DB2 because of the SECURITY USER UDF attribute. GRACFGRP then returns an XML document that contains the RACF group names as a VARCHAR scalar value. We used this UDF in the scenario that is described in 4.1.9, “WebSphere Application Server and DB2 security” on page 126. Listings of the DDL and the ASM are provided in Example G-1 and Example G-2.

Example G-1 DDL for UDF GRACFGRP

```
CREATE FUNCTION JOSEF.GRACFGRP
  RETURNS VARCHAR(32000)
  EXTERNAL NAME 'GRACFGRP'
  LANGUAGE ASSEMBLE
  NOT DETERMINISTIC
  PARAMETER CCSID EBCDIC
  PARAMETER STYLE DB2SQL
  FENCED
  CALLED ON NULL INPUT
  NO SQL
  EXTERNAL ACTION
  NO PACKAGE PATH
  NO SCRATCHPAD
  NO FINAL CALL
  ALLOW PARALLEL
  DBINFO
  NO COLLID
  WLM ENVIRONMENT WLMENV1
  ASUTIME NO LIMIT
  STAY RESIDENT YES
  PROGRAM TYPE SUB
  SECURITY USER
  STOP AFTER SYSTEM DEFAULT FAILURES
  INHERIT SPECIAL REGISTERS
  RUN OPTIONS 'NOTEST(ALL,INSPIN,,*)'
```

Example G-2 Assembler listing of GRAFGRP

```
TITLE 'GRACFGRP (Get RACF groups a user is connected to )'     00010000
**********************************************************************
* Author.....: josef.klitsch@ch.ibm.com
* Date.......: 5th May 2012
* Function...:
*  Implements an SQL UDF external scalar function to return the RACF
*  groups the current RACF user is connected to. GRACFGRP obtains the
*  list of RACF groups through the UDF's ACEE control block. For the
*  ACEE to be available to the GRACFGRP program the UDF <MUST> be
*  defined with "SECURITY USER".
*
*  The scalar value is returned as VARCHAR to contain an XML document
*  like the sample shown below:
*      <GROUPS>
*       <USER>JOSEF   </USER>
```

```
*       <GROUP>RACFGRP1</GROUP>
*       <GROUP>RACFGRP2</GROUP>
*       <GROUP>RACFGRP3</GROUP>
*      </GROUPS>
*
* SQL DDL....:
*      CREATE FUNCTION GRACFGRP ()
*       RETURNS VARCHAR(32000)
*       CCSID EBCDIC FOR SBCS DATA
*       SPECIFIC GRACFGRP
*       EXTERNAL NAME GRACFGRP
*       LANGUAGE ASSEMBLE
*       PARAMETER STYLE DB2SQL
*       SECURITY USER
*       FENCED
*       CALLED ON NULL INPUT
*       NO SQL
*       ALLOW PARALLEL
*       DBINFO
*       WLM ENVIRONMENT WLMENV1
*       ASUTIME NO LIMIT;
*
* Interface..:
*      select gracfgrp() from sysibm.sysdummy1;
*      --> returns the XML document shown above
*
*     pureXML query:
*     ==============
*       SELECT T.* FROM XMLTABLE
*       ('$d/GROUPS/GROUP'
*         PASSING XMLPARSE (DOCUMENT gracfgrp()) AS "d"
*       COLUMNS
*        "RACF User"    VARCHAR(08) PATH '../USER/text()',
*        "RACF Group"   VARCHAR(08) PATH './text()'
*       ) AS T
*       ;
*
*      pureXML query result:
*      =====================
*       RACF User RACF Group
*       --------- ----------
*       JOSEF     RACFGRP1
*       JOSEF     RACFGRP2
*       JOSEF     RACFGRP3
*
***********************************************************************
         YREGS                                                          01740000
GRACFGRP CEEENTRY AUTO=WORKSIZE,BASE=R11,MAIN=NO,PLIST=OS               01750000
         USING WORKAREA,R13                                             01760000
         L R9,0(R1)                get pointer TO return parm
         USING RACFGRP,R9                                               01760000
         L R7,4(R1)                get pointer to indicator variable
         MVC 0(2,R7),=AL2(0)       indicate return value
         MVC  XML01#,XML01
         MVC  XML11#(XML11L),XML11
```

```
         MVC  RACFLEN,XML01L
A0010    DS   0H                                                     02940000
         L     R5,CVTPTR           ADDRESS MVS CVT
         L     R7,CVTRAC-CVT(,R5)  RACF CVT ADDRESS
         LTR   R7,R7               IF RACF CVT ADDRESS ZERO,
         BZ    A0080               RACF IS NOT EVEN INSTALLED
         USING RCVT,R7             SET BASE FOR RACF CVT
A0011    DS   0H                                                     02940000
         L     R8,PSAAOLD-PSA      GET CURRENT ASCB ADDRESS AND
         USING ASCB,R8             SET MAPPING ADDRESSABILITY
         L     R6,PSATOLD-PSA      CURRENT TCB ADDRESS               04990000
         L     R6,TCBSENV-TCB(,R6) GET TASK LEVEL ACEE               05000000
         LTR   R6,R6               TASK LEVEL ACEE AVAILABLE         05010000
         USING ACEE,R6             ESTABLISH BASE FOR ACEE
         BZ    A0015               NO,GO TRY ADDRESS SPACE           05020000
         CLC   ACEEACEE,=C'ACEE'   DOSE IT LOOK LIKE AN ACEE?        05030000
         BE    A0017               YES,THEN USE IT                   05040000
A0015    DS    0H                                                    05050000
         L     R6,ASCBASXB         GET ADDRESS SPACE EXTENSION BLOCK 05060000
         L     R6,ASXBSENV-ASXB(,R6)  GET ACEE ADDRESS               05070000
         LTR   R6,R6               DOES AN ACEE EXIST?  IF NOT,      05080000
         BZ    A0080               SKIP AROUND CONNECTED GROUP NAME  05090000
         CLC   ACEEACEE,=C'ACEE'   DOES IT LOOK LIKE AN ACEE?        05100000
         BNE   A0080               NO, THEN CAN'T DO GROUPS          05110000
         DROP  R8                  DROP ASCB BASE REG                05120000
A0017    DS    0H                  CHECK LIST OF GROUPS OPTION       00010001
         MVC   XML11U#,ACEEUSRI
         TM    RCVTOPTX,RCVTLGRP   IS LIST OF GROUPS CHECKING ACTIVE 05150000
         BZ    A0080               NO, THEN CAN'T DO GROUPS          05160000
         DROP  R7                  DROP RCVT BASE REG                05170000
A0020    DS    0H                  CHECK LIST OF GROUPS OPTION       00010001
*   INITIALIZE                                                       05270000
         L     R5,ACEEFCGP         CONNECT GROUP BLOCK               05280000
         LTR   R5,R5               ENSURE THE BLOCK IS THERE         05290000
         BZ    A0080               THE COUNT IS ZERO, SKIP IT        05300000
         USING CGRP,R5             SET BASE FOR CONNECT GROUP        05310000
         CLC   CGRPID,=C'CGRP'     DOES IT LOOK LIKE A CGRP BLOCK?   05320000
         BNE   A0080               NO, GROUP NAMES NOT AVAILABLE     05330000
         SLR   R3,R3               CLEAR THE COUNTER REGISTER        05340000
         ICM   R3,B'0011',CGRPNUM  GET THE NUMBER OF CONNECT GROUPS  05350000
         ST    R3,SECCOUNT         SAVE COUNT OF GROUPS              05360000
         BZ    A0080               BR IF NO GROUP NAMES AVAILABLE    05370000
         LA    R2,CGRPENT          POINT TO CONNECT GROUP ENTRIES    05380000
         USING CGRPENTD,R2         SET BASE FOR CONNECT GROUP ENTRIES 05390000
         LA    R4,GRPS             ADDRESS OF SECONDARY IDS          05410000
         USING SGRP,R4             SET BASE FOR SECONDARY GROUPS     05420000
         DROP  R5                  DROP CGRP BASE REG        @TU25003 05460000
         LH    R7,RACFLEN
*   COPY GROUP NAMES                                                 05490000
A0026    DS    0H                  COUNTER SET FOR MOVING            05500000
         TM    CGRPNAME,X'BF'      SEE IF THE GROUP IS VALID         05510000
         BNM   A0027               BR IF NULL, BLANK, OR FF          05520000
         MVC   SGRPXML1,XML12                                        05530000
         MVC   SGRPNAME,CGRPNAME   MOVE THE GROUP NAME               05530000
         MVC   SGRPXML2,XML122                                       05530000
```

```
         LA    R4,SGRPNEXT         POINT TO NEXT SECONDARY AUTHID      05540000
A0027    DS    0H                  BYPASS UPDATING SECONDARY LIST      05570000
         LA    R2,L'CGRPENT(,R2)   POINT TO NEXT CONNECT GROUP         05590000
         AH    R7,=AL2(SGRPLEN)
         BCT   R3,A0026            BR UNTIL ALL GROUP NAMES EXAMINED   05600000
         B     A0060               MOVING IS COMPLETED                 05610000
         DROP  R2                  DROP CGRPENTD BASE REG              05620000
A0060    DS   0H                   Moving groups is complete           02940000
         AH    R7,=AL2(L'XML02)
         STH   R7,RACFLEN
         MVC  0(L'XML02,R4),XML02
         B    A0099
A0080    DS   0H                   Can't do groups                     02940000
*-------------------------------------------------------------------* 02910000
*        TERMINATION                                                * 02920000
*-------------------------------------------------------------------* 02930000
A0099    DS   0H                   Terminate                           02940000
         MVC   0(2,R1),=H'0'       RC=0                                03020000
         CEETERM RC=0                                                  03030000
SECLEN   DC    Y(SGRPNEXT-SGRP)    LENGTH OF A SECONDARY AUTHID ENTRY
PPA      CEEPPA
         LTORG                                                         19981000
XML01    DC C'<GROUPS>'
XML02    DC C'</GROUPS>'
XML11    DC C'<USER>'
XMLUSER  DC CL8' '
         DC C'</USER>'
XML11L   EQU *-XML11
XML01L   DC AL2(*-XML01-L'XML02)
XML12    DC C'<GROUP>'
         DC CL8' '
XML122   DC C'</GROUP>'
XML12L   EQU *-XML12
*-------------------------------------------------------------------* 18200000
*        VARIABLES                                                  * 18210000
*-------------------------------------------------------------------* 18220000
WORKAREA DSECT                                                         18290000
         ORG   *+CEEDSASZ          Space for dynamic save area         18300000
SAVEREGS DS  16F                       Copy of caller's registers
*                                                                      18310000
SECCOUNT DS    F                   COUNT OF SECONDARY IDS
         DS   0D                   On doubleword boundary              19800000
WORKSIZE EQU   *-WORKAREA                                              19810000
*                                                                      19820000
*-------------------------------------------------------------------* 19830000
*        DSECTs                                                     * 19840000
*-------------------------------------------------------------------* 19850000
RACFGRP  DSECT
RACFLEN  DS    H
RACFGRPS DS    CL1024              LIST OF RACF GROUPS
         ORG   RACFGRPS
XML01#   DC    C'<GROUPS>'
XML11#   DC    C'<USER>'
XML11U#  DC    CL8' '
         DC    C'</USER>'
```

```
GRPS     DS    CL(*-RACFGRPS+L'RACFGRPS)
*
SGRP     DSECT
SGRPXML1 DS    CL(L'XML12)
SGRPNAME DS    CL8                 MOVE SECONDARY GROUPS HERE
SGRPXML2 DS    CL(L'XML122)
SGRPLEN  EQU   *-SGRP
SGRPNEXT EQU   *                   NEXT SECONDARY NAME STARTS HERE
         CEEDSA                                                         19950000
         CEECAA                                                         19970000
         CVT   DSECT=YES                                                19971000
         ICHPRCVT                                                       19972000
         IHAACEE                                                        19973000
         ICHPCGRP                                                       19974000
         IHAPSA                                                         19975000
         IKJTCB                                                         19976000
         IKJPSCB                                                        19977000
         IKJUPT                                                         19978000
IEFAJCTB DSECT                                                          19979000
         IEFAJCTB                                                       19979100
         IEZJSCB                                                        19979200
         IHAASCB                                                        19979300
         IHAASXB                                                        19979400
         END   GRACFGRP                                                 19990000
```

G.2 UDF BIGINT

The external UDF BIGINT is a COBOL program that can be used to convert a `CHAR` string to BIGINT. We used it in the scenario that is described in 4.3.23, “SYSPROC.ADMIN_DS_LIST stored procedure” on page 197. Listings of the DDL and the COBOL program are provided in Example G-3 and Example G-4 on page 569.

Example G-3 DDL for UDF BIGINT

```
-- DROP   FUNCTION BIGINT ;
 CREATE FUNCTION BIGINT
    (NAME VARCHAR(00008))
   RETURNS BIGINT
   EXTERNAL NAME 'UDFDOUBL'
   LANGUAGE COBOL
   DETERMINISTIC
   PARAMETER STYLE DB2SQL
   FENCED
   RETURNS NULL ON NULL INPUT
   NO EXTERNAL ACTION
   NO SCRATCHPAD
   NO FINAL CALL
   ALLOW PARALLEL
   DBINFO
   NO COLLID
   WLM ENVIRONMENT WLMENV3
   ASUTIME NO LIMIT
   STAY RESIDENT YES
   PROGRAM TYPE SUB
```

```
SECURITY DB2
STOP AFTER SYSTEM DEFAULT FAILURES
INHERIT SPECIAL REGISTERS
;
```

Example G-4 COBOL listing for UDF BIGINT

```
000100  CBL APOST,MAP,XREF,RENT,TRUNC(BIN),TEST                          00010000
000200 Identification Division.                                          00020000
000300 Program-ID. 'UDFDOUBL'.                                           00030001
000400*****************************************************              00040000
000500* UDF interface to convert a CHAR string to BIGINT  *              00050001
000600*  Interface:                                      .*              00060000
000700*   select BIGINT('<VARCHAR>'   ) from             .*              00070001
000800*   sysibm.sysdummy1;                              .*              00080000
000805*                                                  .*              00610000
0008§0* josef.klitsch@ch.ibm.com                         .*              00610000
001100*                                                  .*              00110000
001200* Create Function DDL:                             .*              00120000
001300* -------------------                              .*              00130000
001400*  CREATE FUNCTION BIGINT                          .*              00140001
001500*     (NAME VARCHAR(00008))                        .*              00150001
001600*    RETURNS DOUBLE                                .*              00160001
001700*    EXTERNAL NAME 'UDFDOUBL'                      .*              00170001
001800*    LANGUAGE COBOL                                .*              00180000
001900*    DETERMINISTIC                                 .*              00190000
002000*    PARAMETER STYLE DB2SQL                        .*              00200000
002100*    FENCED                                        .*              00210000
002200*    RETURNS NULL ON NULL INPUT                    .*              00220000
002300*    NO EXTERNAL ACTION                            .*              00230000
002400*    NO SCRATCHPAD                                 .*              00240000
002500*    NO FINAL CALL                                 .*              00250000
002600*    ALLOW PARALLEL                                .*              00260000
002700*    DBINFO                                        .*              00270000
002800*    NO COLLID                                     .*              00280000
002900*    WLM ENVIRONMENT WLMENV3                       .*              00290000
003000*    ASUTIME NO LIMIT                              .*              00300000
003100*    STAY RESIDENT YES                             .*              00310000
003200*    PROGRAM TYPE SUB                              .*              00320000
003300*    SECURITY DB2                                  .*              00330000
003400*    STOP AFTER SYSTEM DEFAULT FAILURES            .*              00340000
003500*    INHERIT SPECIAL REGISTERS                     .*              00350000
003600*    ;                                             .*              00360000

006200*****************************************************              00620000
006300 Data Division.                                                    00630000
006400 Working-Storage Section.                                          00640000
006500*    EXEC SQL INCLUDE SQLCA  END-EXEC.                             00650000
013600*==========================================================*      01360000
013700 LINKAGE SECTION.                                                  01370000
013800    01 UDFPARM1.                                                   01380004
015500       49 UDFPARM1-LEN           PIC 9(4) USAGE BINARY.            01390004
015600       49 UDFPARM1-TEXT          PIC X(8).                         01400004
014400    01 UDFPARM2                  PIC S9(18) USAGE COMP.            01440008
          01 UDFPARM2-X REDEFINES UDFPARM2 PIC X(8).                     01441003
```

```
    01 UDF-RIND1                  PIC S9(4) USAGE COMP.
       88 UDF-RIND1-OK            VALUE ZERO.
       88 UDF-RIND1-NODATA        VALUE -1.
    01 UDF-RIND2                  PIC S9(4) USAGE COMP.
       88 UDF-RIND2-OK            VALUE ZERO.
       88 UDF-RIND2-NODATA        VALUE -1.
    01 UDF-SQLSTATE               PIC X(5).
       88 UDF-SQLSTATE-OK         VALUE '00000'.
       88 UDF-SQLSTATE-FAIL       VALUE '38999'.
    01 UDF-FUNC.
        49 UDF-FUNC-LEN           PIC 9(4) USAGE BINARY.
        49 UDF-FUNC-TEXT          PIC X(137).
    01 UDF-SPEC.
        49 UDF-SPEC-LEN           PIC 9(4) USAGE BINARY.
        49 UDF-SPEC-TEXT          PIC X(128).
           88 UDF-SPEC-TEXT-CHAR  VALUE 'MD5CHAR'.
           88 UDF-SPEC-TEXT-CLOB  VALUE 'MD5CLOB'.
    01 UDF-DIAG.
        49 UDF-DIAG-LEN           PIC 9(4) USAGE BINARY.
           88 UDF-DIAG-LEN-INIT   VALUE 70.
        49 UDF-DIAG-TEXT          PIC X(70).
           88 UDF-DIAG-TEXT-INIT  VALUE SPACE.
    01 UDF-DBINFO.
*      Location length and name
       02 UDF-DBINFO-LOCATION.
          49 UDF-DBINFO-LLEN      PIC 9(4) USAGE BINARY.
          49 UDF-DBINFO-LOC PIC   X(128).
*      Authorization ID length and name
       02 UDF-DBINFO-AUTHORIZATION.
          49 UDF-DBINFO-ALEN      PIC 9(4) USAGE BINARY.
          49 UDF-DBINFO-AUTH      PIC X(128).
*      CCSIDs for DB2 for OS/390
       02 UDF-DBINFO-CCSID        PIC X(48).
       02 UDF-DBINFO-CDPG REDEFINES UDF-DBINFO-CCSID.
          03 DB2-CCSIDS OCCURS 3 TIMES.
             04 DB2-SBCS          PIC 9(9) USAGE BINARY.
             04 DB2-DBCS          PIC 9(9) USAGE BINARY.
             04 DB2-MIXED         PIC 9(9) USAGE BINARY.
          03 DB2-ENCODING-SCHEME  PIC 9(9) USAGE BINARY.
          03 DB2-CCSID-RESERVED   PIC X(8).
*      Schema length and name
       02 UDF-DBINFO-SCHEMA0.
          49 UDF-DBINFO-SLEN      PIC 9(4) USAGE BINARY.
          49 UDF-DBINFO-SCHEMA    PIC X(128).
*      Table length and name
       02 UDF-DBINFO-TABLE0.
          49 UDF-DBINFO-TLEN      PIC 9(4) USAGE BINARY.
          49 UDF-DBINFO-TABLE     PIC X(128).
*      Column length and name
       02 UDF-DBINFO-COLUMN0.
          49 UDF-DBINFO-CLEN      PIC 9(4) USAGE BINARY.
          49 UDF-DBINFO-COLUMN    PIC X(128).
*      DB2 release level
       02 UDF-DBINFO-VERREL       PIC X(8).
*      unused
```

```
020000        02 FILLER PIC              X(2).                            02000000
020100*       Database Platform                                           02010000
020200        02 UDF-DBINFO-PLATFORM     PIC 9(9) USAGE BINARY.           02020000
020300*       # of entries in Table Function column list                  02030000
020400        02 UDF-DBINFO-NUMTFCOL     PIC 9(4) USAGE BINARY.           02040000
020500*       reserved                                                    02050000
020600        02 UDF-DBINFO-RESERV1      PIC X(24).                       02060000
020700*       Unused                                                      02070000
020800        02 FILLER                  PIC X(2).                        02080000
020900*       Pointer to Table Function column list                       02090000
021000        02 UDF-DBINFO-TFCOLUMN     POINTER.                         02100000
021100*       Pointer to Application ID                                   02110000
021200        02 UDF-DBINFO-APPLID       POINTER.                         02120000
021300*       reserved                                                    02130000
021400        02 UDF-DBINFO-RESERV2      PIC X(20).                       02140000
021500*                                                                   02150000
021600 01  APPLICATION-ID          PIC X(32).                             02160000
021700 Procedure Division using  UDFPARM1,                                02170000
021800                           UDFPARM2,                                02180000
021900                           UDF-RIND1,                               02190000
022000                           UDF-RIND2,                               02200000
022100                           UDF-SQLSTATE,                            02210000
022200                           UDF-FUNC,                                02220000
022300                           UDF-SPEC,                                02230000
022400                           UDF-DIAG,                                02240000
022500                           UDF-DBINFO.                              02250000
023500  A00-CONTROL SECTION.                                              02350000
023600  A0010.                                                            02360000
025100     SET UDF-SQLSTATE-FAIL TO TRUE                                  02361001
023700     IF UDF-RIND1 >= 0                                              02370000
025100        SET UDF-SQLSTATE-OK TO TRUE                                 02510001
              INITIALIZE UDFPARM2                                         02520001
025500        MOVE UDFPARM1-TEXT(1:UDFPARM1-LEN)                          02550004
                   TO UDFPARM2-X(9 - UDFPARM1-LEN:UDFPARM1-LEN)           02560005
030500     END-IF.                                                        03050000
030600  A0099.                                                            03060000
030700      goback.                                                       03070000
```

H

ClientInfo dynamic web project

In 5.5, “Configuring client information in WebSphere Application Server” on page 246, we use the ClientInfo dynamic web application for setting and verifying DB2 client information from a servlet application.

We used IBM Rational Application Developer for WebSphere Software for servlet development and tested the servlet functionality in WebSphere Application Server V8R5 on Windows and on z/OS. Upon successful servlet testing, we exported the ClientInfo dynamic web application and created the `ClientInfo.war` web archive file (WAR file). The WAR file includes the Java source and class files. You can download the `ClientInfo.war` file as described in Appendix I, “Additional material” on page 587.

In this appendix, we describe the ClientInfo dynamic web project, how to access the ClientInfo Java source files by using standard tools, and how to install the ClientInfo application in WebSphere Application Server.

This appendix covers the following topics:

- The ClientInfo dynamic web project
- Accessing the ClientInfo.war file from your workstation
- Installing the ClientInfo web application
- Starting the ClientInfo web application
- Testing the ClientInfo web application
- Testing the ClientInfoJDBC30API servlet
- Testing the ClientInfoJDBC40 servlet
- Testing the ClientInfoWSAPI servlet
- Testing the ClientInfoWLM servlet

H.1 The ClientInfo dynamic web project

The ClientInfo application consists of the servlets that are shown in Table H-1, with each servlet using a different interface for setting DB2 client information:

Table H-1 ClientInfo servlet functionality

Servlet name	Interface that is used for setting the DB2 client information
ClientInfoJDBC30API	com.ibm.db2.jcc.DB2Connection class interfaces: ► setDB2ClientUser ► setDB2ClientWorkstation ► setDB2ClientApplicationInformation ► setDB2ClientAccountingInformation The DB2Connection interfaces are started directly through the com.ibm.websphere.rsadapter.WSCallHelper.jdbcCall interface.
ClientInfoJDBC40API	java.sql.Connection class setClientInfo interface
ClientInfoWSAPI	com.ibm.websphere.rsadapter.WSConnection class setClientInformation interface
ClientInfoWLM	SQL CALL SYSPROC.WLM_SET_CLIENT_INFO stored procedure

If you use the Java EE perspective of Rational Application Developer for WebSphere Software, the structure of the ClientInfo dynamic web project with its Java source files looks like the structure that is shown in Figure H-1.

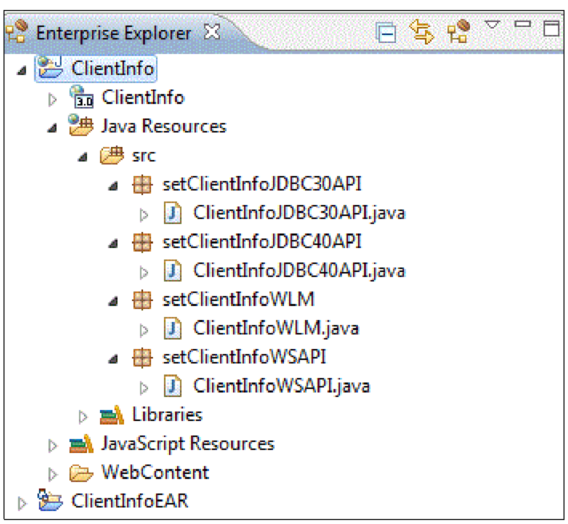


Figure H-1 ClientInfo project that is shown in the Java EE perspective

H.2 Accessing the ClientInfo.war file from your workstation

After the WAR file is downloaded to your workstation, you can access its content by using standard tools that can process archive files (see Figure H-2).

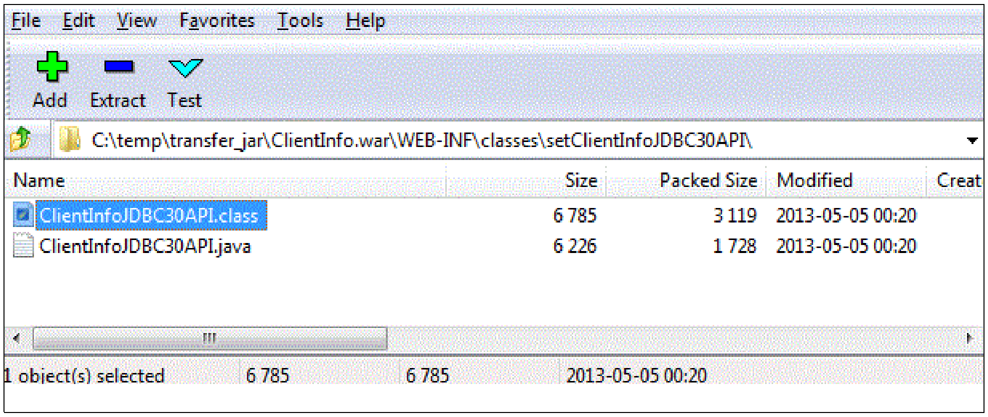


Figure H-2 Opening the ClientInfo.war file

H.3 Installing the ClientInfo web application

To install the ClientInfo dynamic web application, you must log in to the WebSphere Integrated Solution Console (ISC) to complete the following tasks:

1. To define JDBC providers, follow the instructions that are provided in 5.2.1, “Defining a DB2 JDBC XA provider” on page 210 and 5.3.1, “Defining a DB2 JDBC provider” on page 223. If a JDBC provider exists, you do not need to perform this task.
2. To define a data source to access your DB2 server using JDBC type 2, follow the instructions that are provided in 5.2.3, “Defining a JDBC type 4 XA data source” on page 218; for the definition of JDBC type 4, follow the instructions in 5.3.3, “Defining a JDBC type 2 data source” on page 233. For the ClientInfo application to function, the JNDI name of the data source must be jdbc/Josef.

3. To select the `ClientInfo.war` file for installation, click **Applications** → **New Applications** → **Enterprise Applications**, select **Local file system**, and click **Next**, as shown in Figure H-3.

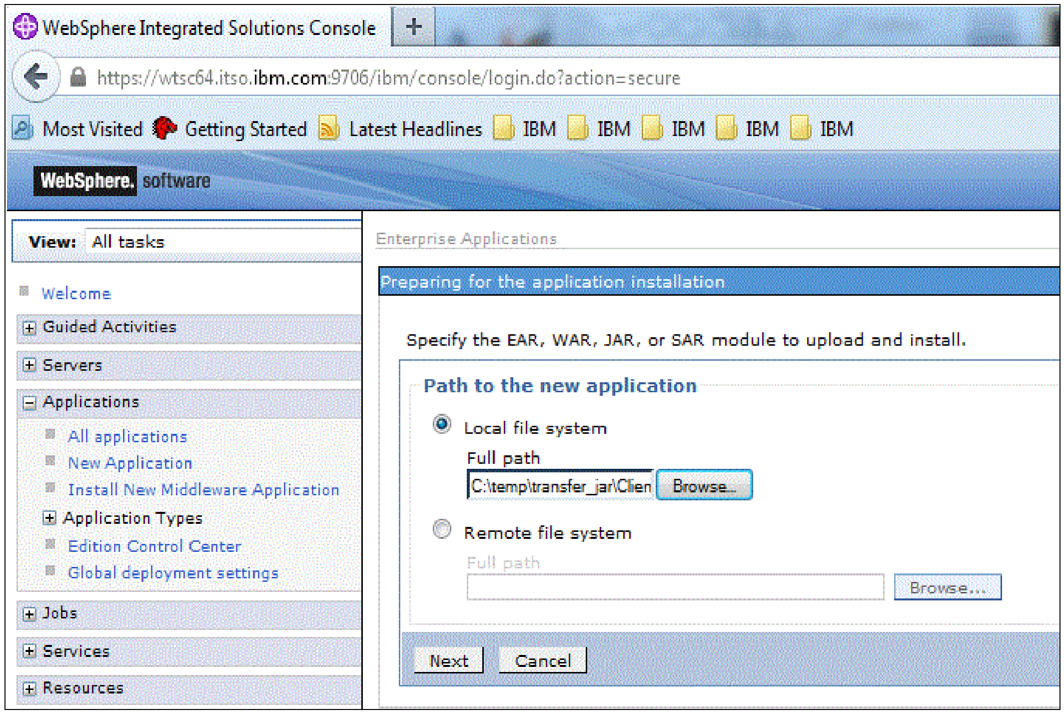


Figure H-3 Install ClientInfo application from local file system

4. Select **Fast Path** and click **Next**, as shown in Figure H-4.

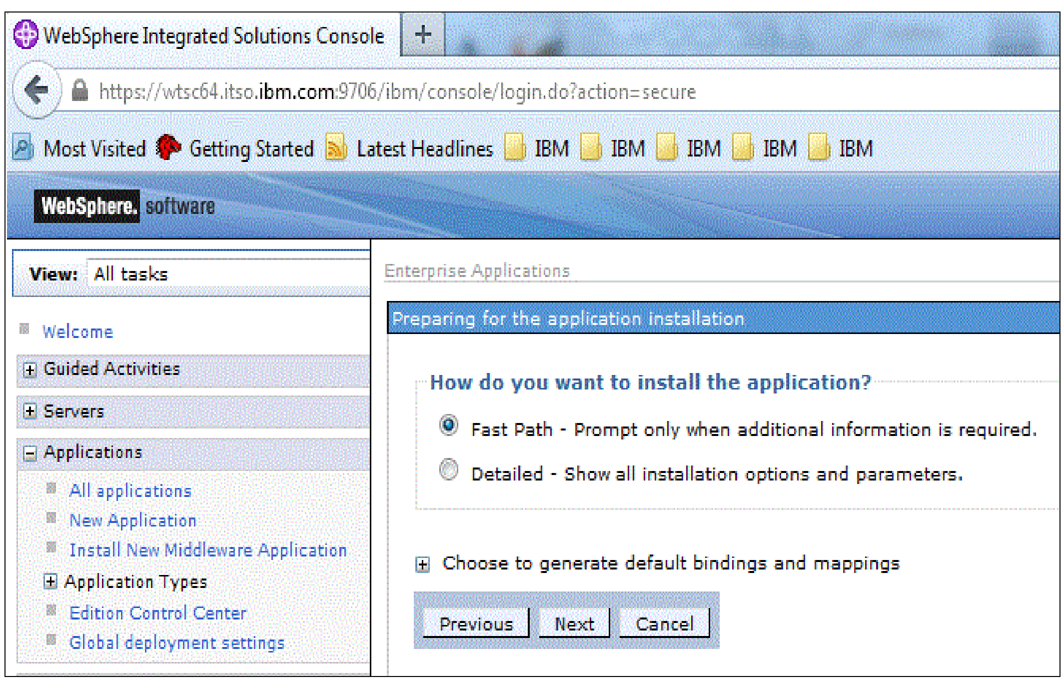


Figure H-4 How to install the application

5. The Step 1: Select installation options window opens (Figure H-5). Leave the options at their defaults and select **Next**.

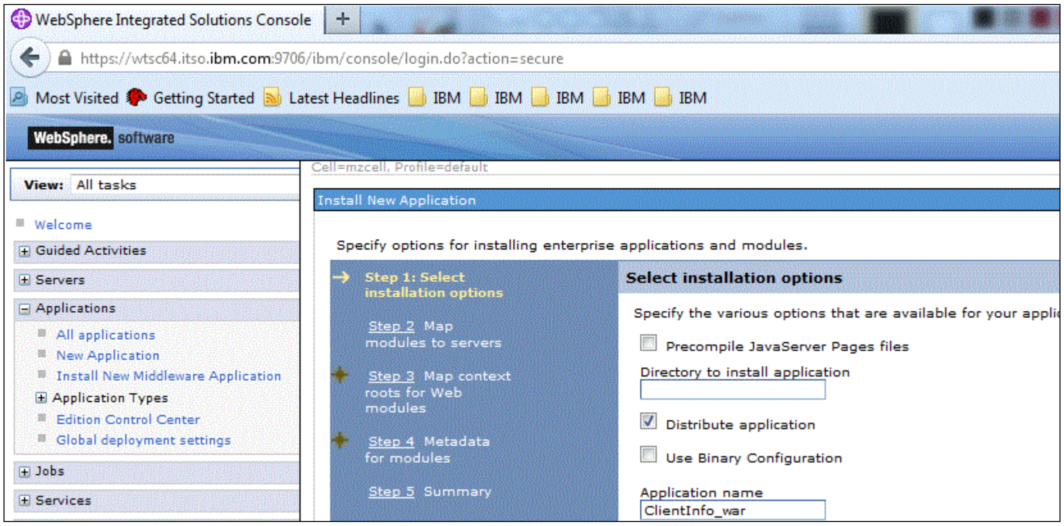


Figure H-5 Step 1: Select installation options window

6. The Step 2: Map modules to servers window opens (Figure H-6). Choose the server that you want to install the application in and click **Next**.

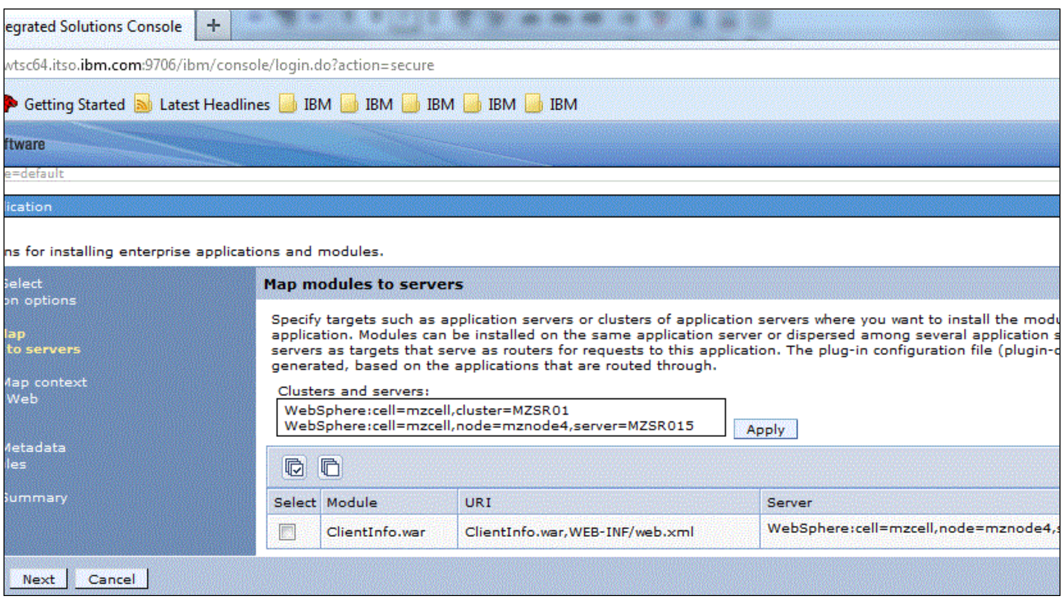


Figure H-6 Step 2: Map modules to servers window

7. The Step 3: Map context roots for Web modules window opens (Figure H-7). Enter the **/ClientInfo** context root name and click **Next**.

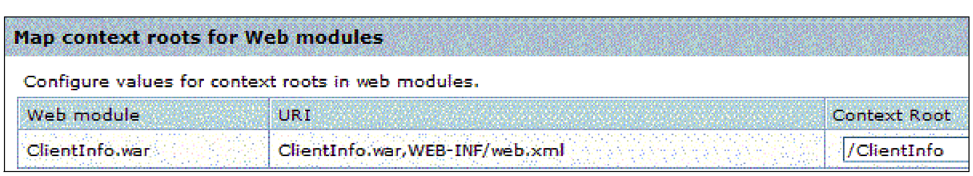


Figure H-7 Step 3: Map context roots for Web modules window

8. The Step 4: Metadata for modules window opens (Figure H-8). Leave the options at their defaults and click **Next**.

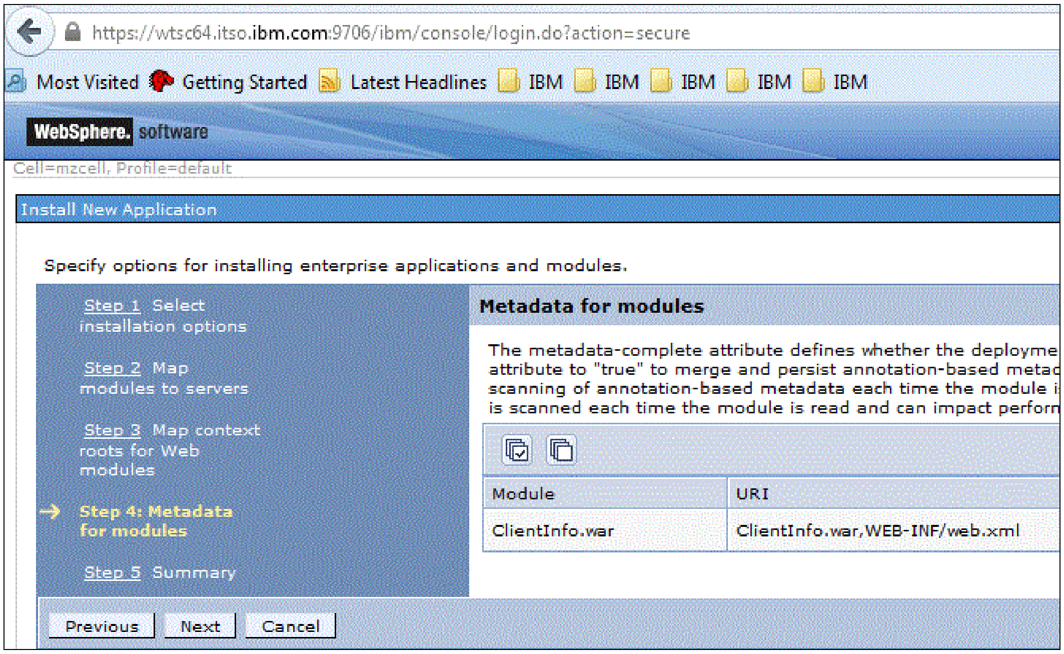


Figure H-8 Step 4: Metadata for modules window

9. The Step 5: Summary window opens (Figure H-9). Click **Finish**.

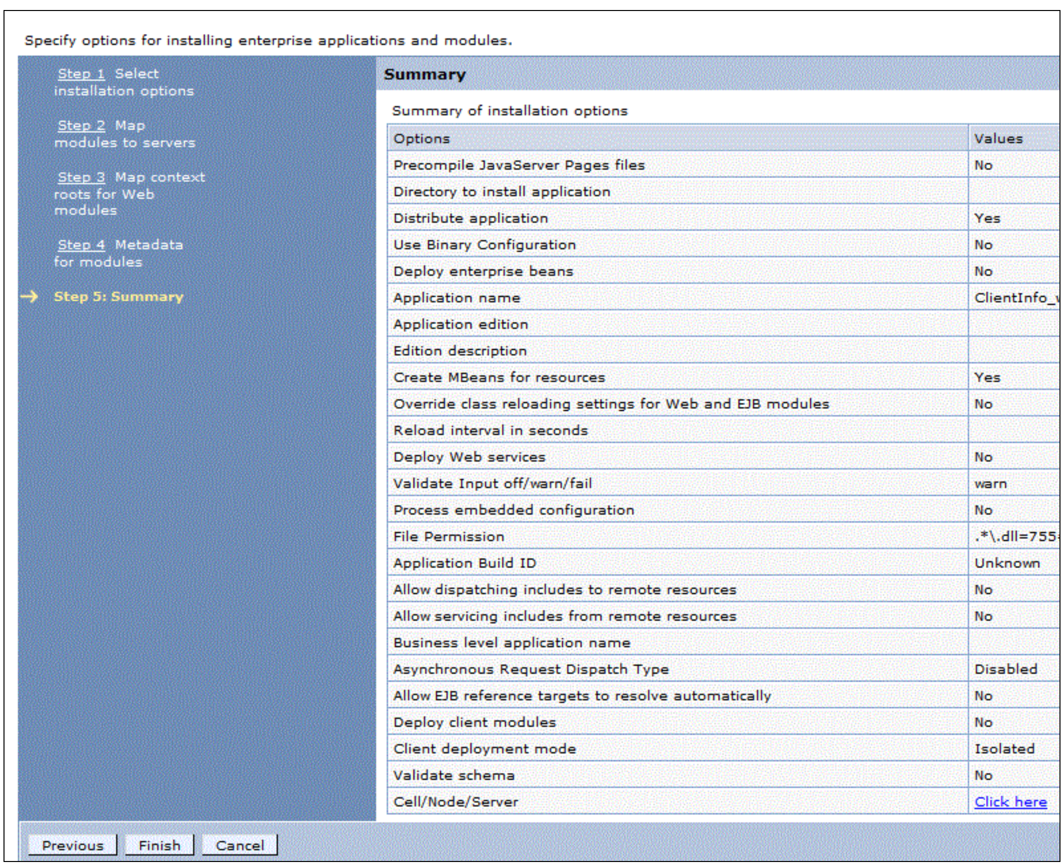


Figure H-9 Step 5: Summary window

10.A window opens and shows a successful application installation (Figure H-10). Click **Review**.

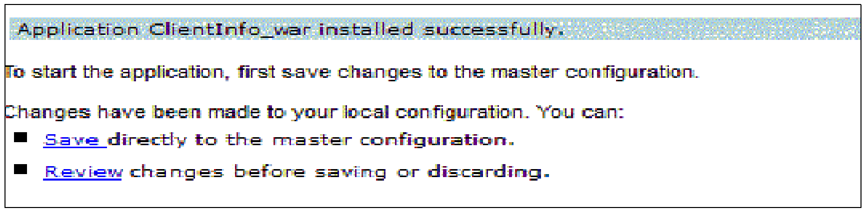


Figure H-10 Application Clientinfo_war installed successfully

11.In the New Application window that opens (Figure H-11), click **Synchronize changes with Nodes** and click **Save**.

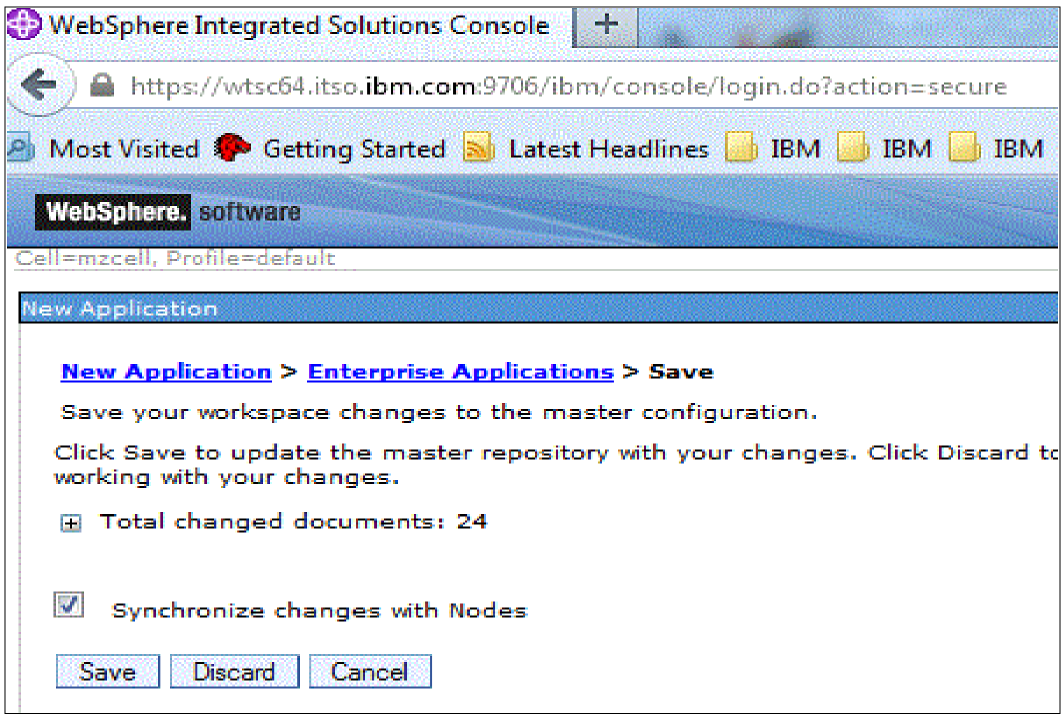


Figure H-11 Synchronize changes with nodes

12. You have successfully installed the ClientInfo application, as shown in Figure H-12. Click **OK** to continue.

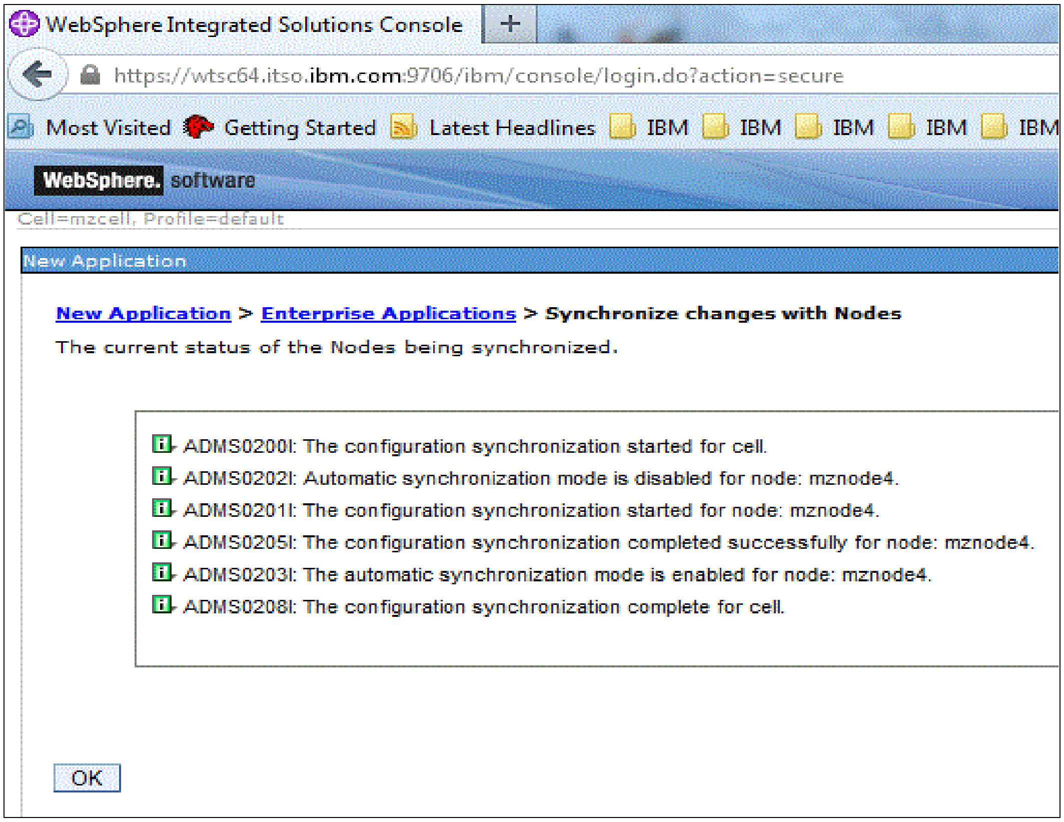


Figure H-12 ClientInfo application installed successfully

H.4 Starting the ClientInfo web application

To start the ClientInfo web application, complete the following steps:

1. Click **All applications**, check the **ClientInfo_war** check box, and click **Submit Action**, as shown in Figure H-13.

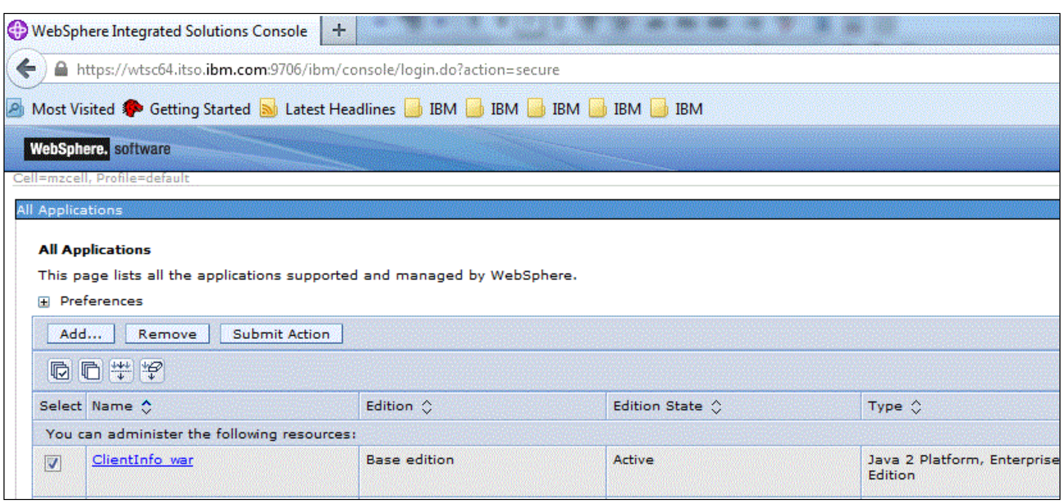


Figure H-13 Panel 1 starting the ClientInfo Application

2. Upon successful completion, ISC confirms a successful start of the application by displaying a green arrow in the application status column. The job log of the servant region shows the runtime messages that are listed in Figure H-14 to confirm successful application start.

```
BBOO0222I: ADMN1008I: An attempt is made to start the ClientInfo_war  330
application. (User ID =
wtsc64.itso.ibm.com/server:mzcell_mznode4_MZSR015)
BBOO0222I: WSVR0190I: Starting composition unit  331
WebSphere:cuname=ClientInfo_war in BLA
WebSphere:blaname=ClientInfo_war.
BBOO0222I: WSVR0200I: Starting application: ClientInfo_war
BBOO0222I: WSVR0204I: Application: ClientInfo_war  Application build  333
level: Unknown
BBOO0222I: WSVR0221I: Application started: ClientInfo_war
BBOO0222I: WSVR0191I: Composition unit WebSphere:cuname=ClientInfo_war  335
in BLA WebSphere:blaname=ClientInfo_war started.
```

Figure H-14 Servant region application start messages

H.5 Testing the ClientInfo web application

After you start the ClientInfo application successfully, you can enter the URLs shown in Table H-2 to start the servlet applications from your browser.

Table H-2 URLs for testing the ClientInfo servlet applications

Servlet name	URL
ClientInfoJDBC30API	http://<server>:<portno>/ClientInfo/JDBC30API
ClientInfoJDBC40API	http://<server>:<portno>/ClientInfo/JDBC40API
ClientInfoWSAPI	http://<server>:<portno>/ClientInfo/WSAPI
ClientInfoWLM	http://<server>:<portno>/ClientInfo/WLM

H.6 Testing the ClientInfoJDBC30API servlet

Upon successful ClientInfoJDBC30API servlet start, you receive the output that is shown in Figure H-15.

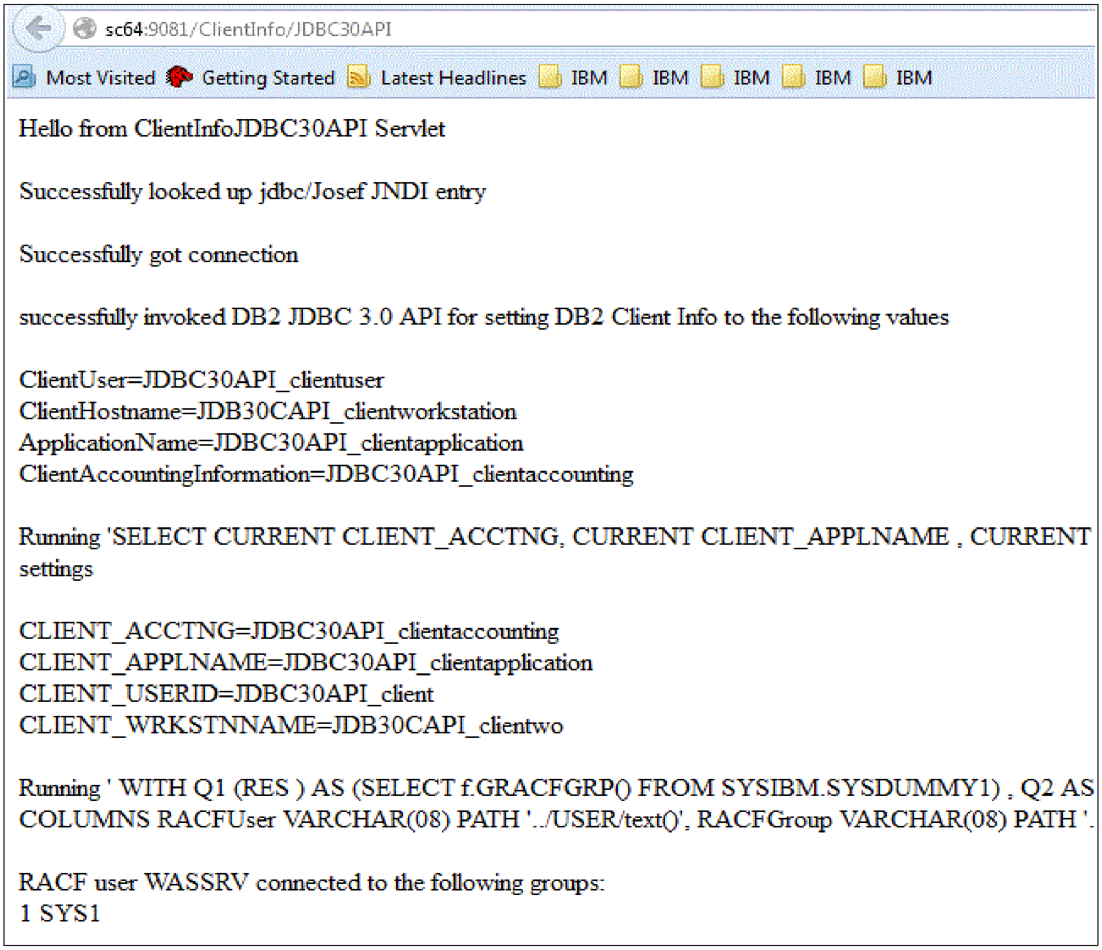


Figure H-15 Testing the ClientInfoJDBC30API servlet

For a description about the result that is returned by the ClientInfoJDBC30 servlet, see 5.5.3, “Setting DB2 client information in a WebSphere Java application” on page 255.

H.7 Testing the ClientInfoJDBC40 servlet

Upon successful ClientInfoJDBC40API servlet start, you receive the output that is shown in Figure H-16 on page 583.

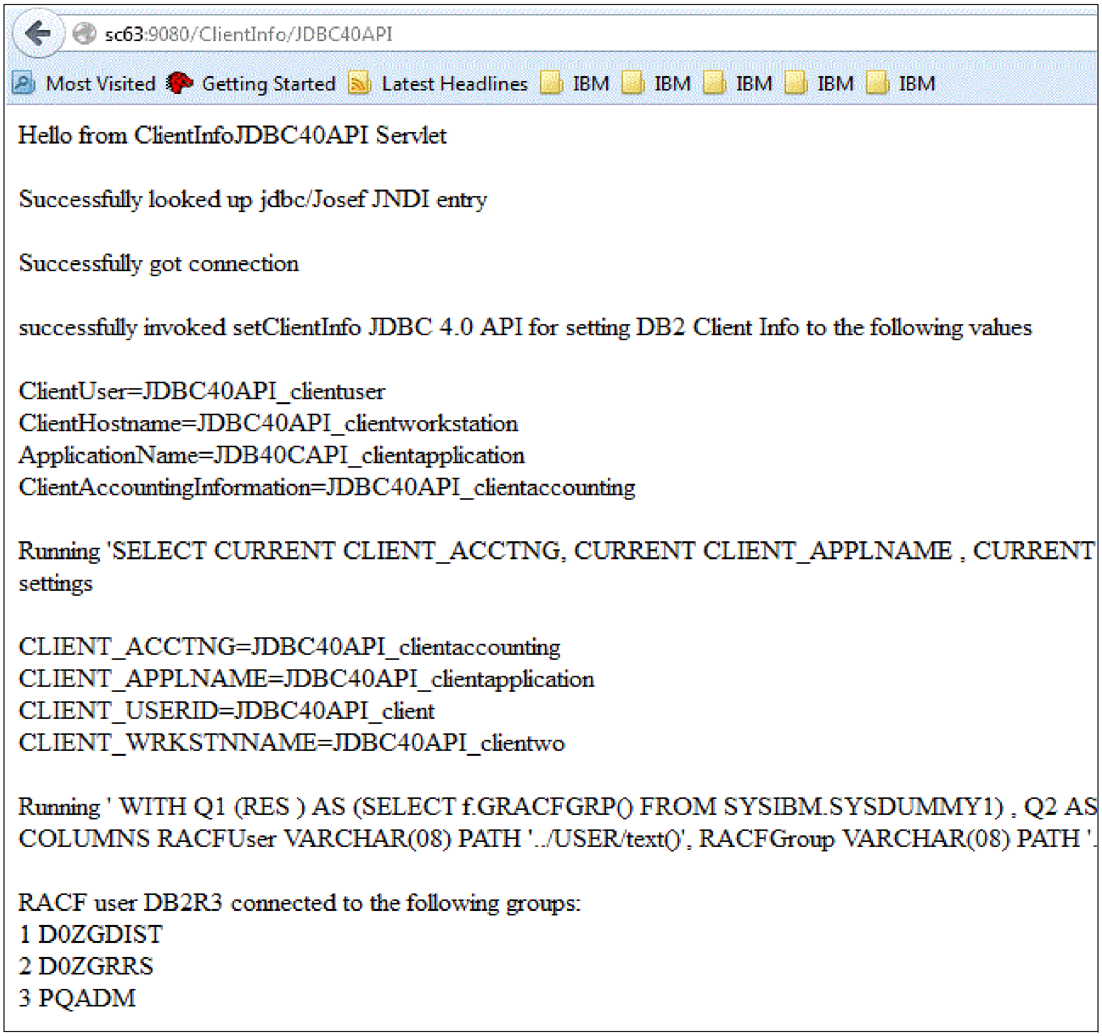


Figure H-16 Testing the ClientInfoJDBC40API servlet

For a description about the result that is returned by the ClientInfoJDBC40 servlet, see 5.5.3, "Setting DB2 client information in a WebSphere Java application" on page 255.

H.7.1 Common pitfalls when using the JDBC 4.0 setClientInfo API

During our testing of ClientInfoJDBC40API, we received the error message that is shown in Figure H-17, which indicates that the JDBC 4.0 java.sql.Connection.setClientInfo API was not supported by the application server runtime environment, even though the JDBC provider we were using explicitly had the `db2jcc4.jar` file in its class path.

```
java.sql.SQLFeatureNotSupportedException: DSRA1300E: Feature is not
implemented: Connection.setClientInfo
   at com.ibm.ws.rsadapter.AdapterUtil.notSupportedX(AdapterUtil.java:1460)
   at
com.ibm.ws.rsadapter.jdbc.WSJdbcConnection.setClientInfo(WSJdbcConnection.java:
5101)
```

Figure H-17 setClientInfo SQLFeatureNotSupportedException

Further analysis of the application server environment showed that we had several JDBC providers defined, some with `db2jcc4.jar` and some with `db2jcc.jar` in their class path. After we changed the JDBC procedures to use only the `db2jcc4.jar` file, the problem was resolved and the ClientInfoJDBC40API servlet ran successfully.

H.8 Testing the ClientInfoWSAPI servlet

Upon successful ClientInfoWSAPI servlet start, you receive the browser output that is shown in Figure H-18.

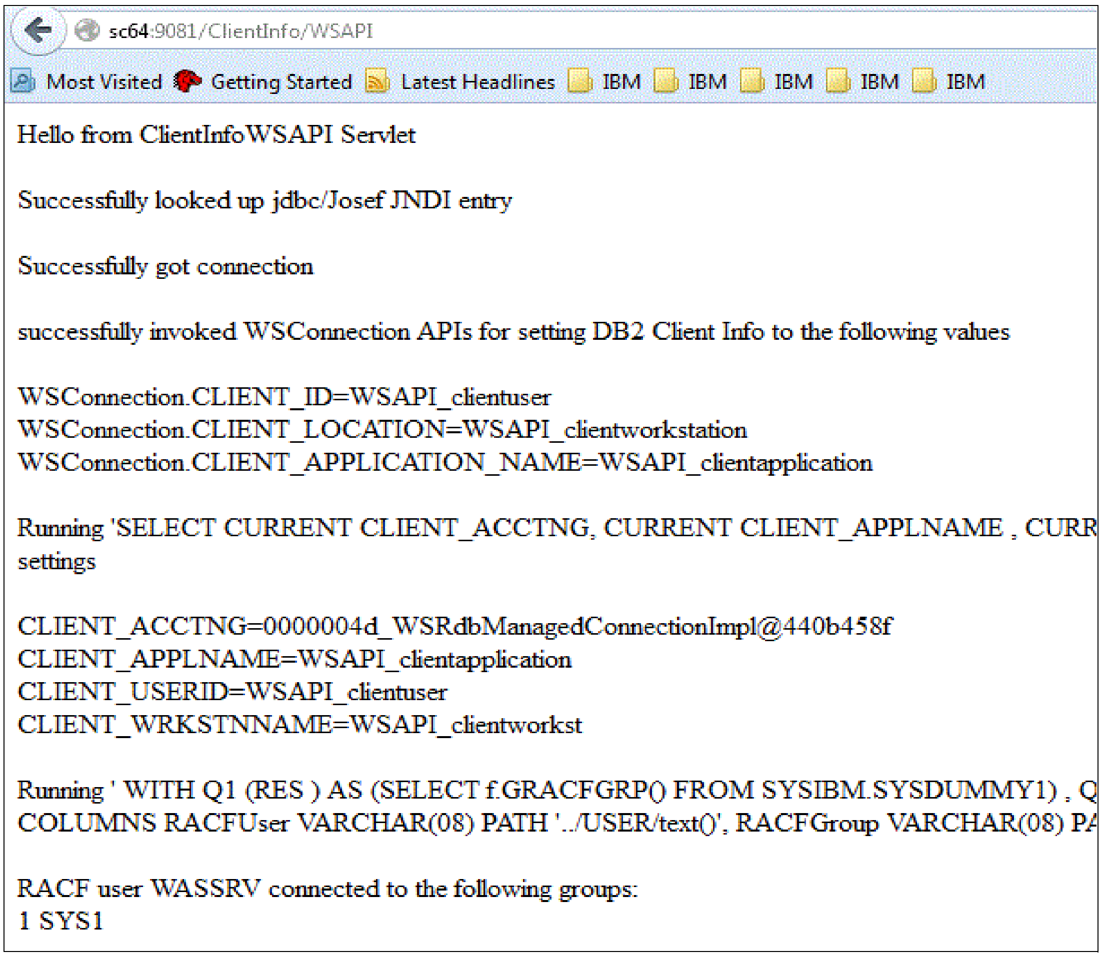


Figure H-18 Testing the ClientInfoWSAPI servlet

For a description about the result that is returned by the ClientInfoWSAPI servlet, see 5.5.3, “Setting DB2 client information in a WebSphere Java application” on page 255.

H.9 Testing the ClientInfoWLM servlet

Upon successful ClientInfoWLM servlet start, you receive the browser output that is shown in Figure H-19.

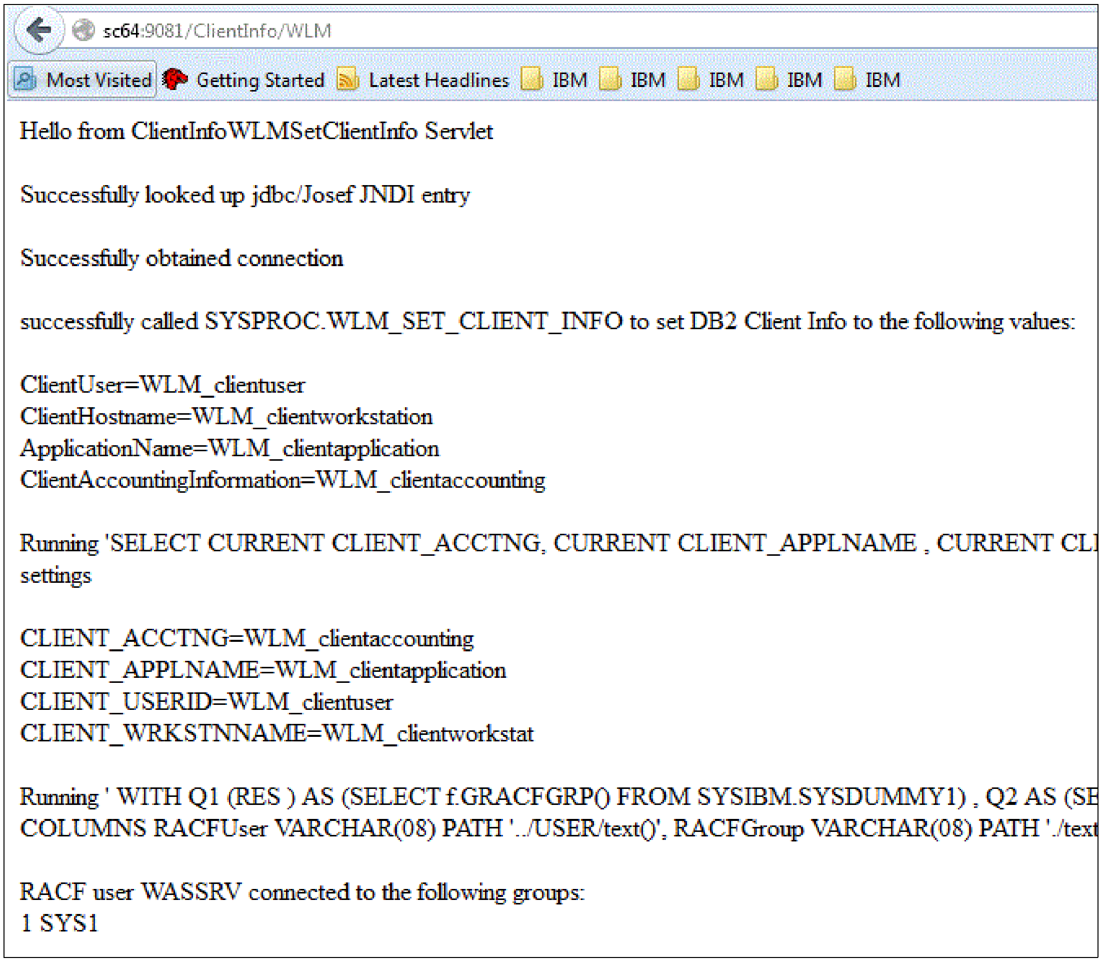


Figure H-19 Testing the ClientInfoWLM servlet

For a description about the result that is returned by the ClientInfoWLM servlet, see 5.5.3, "Setting DB2 client information in a WebSphere Java application" on page 255.

I

Additional material

This book refers to additional material that can be downloaded from the Internet, as described in the following sections.

Locating the web material

The web material that is associated with this book is available in softcopy on the Internet from the IBM Redbooks web server. Point your web browser at:

`ftp://www.redbooks.ibm.com/redbooks/SG248074`

Alternatively, you can go to the IBM Redbooks website at:

ibm.com/redbooks

Select **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG248074.

Using the web material

The additional web material that accompanies this book includes the following files:

File name	*Description*
Clientifo.war	The Java source and class files that are described in Appendix H, "ClientInfo dynamic web project" on page 573.

System requirements for downloading the web material

The web material requires the following system configuration:

Hard disk space:	2 MB minimum
Operating System:	Windows
Processor:	Intel 386 or higher
Memory:	16 MB

Downloading and extracting the web material

Create a subdirectory (folder) on your workstation, and extract the contents of the compressed file into this folder.

Related publications

The publications that are listed in this section are considered suitable for a more detailed discussion of the topics that are covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide more information about the topic in this document. Some publications referenced in this list might be available in softcopy only.

- *Achieving the Highest Levels of Parallel Sysplex Availability in a DB2 Environment*, REDP-3960
- *DB2 9 for z/OS Data Sharing: Distributed Load Balancing and Fault Tolerant Configuration*, REDP-4449
- *DB2 9 for z/OS: Buffer Pool Monitoring and Tuning*, REDP-4604
- *DB2 9 for z/OS: Resource Serialization and Concurrency Control*, SG24-4725
- *DB2 10 for z/OS Performance Topics*, SG24-7942
- *DB2 for z/OS: Data Sharing in a Nutshell*, SG24-7322
- *DB2 for z/OS and WebSphere: The Perfect Couple*, SG24-6319
- *A Deep Blue View of DB2 Performance: IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS*, SG24-7224
- *Extremely pureXML in DB2 10 for z/OS*, SG24-7915
- *IBM Data Studio V2.1: Getting Started with Web Services on DB2 for z/OS*, REDP-4510
- *IBM WebSphere Application Server V8 Concepts, Planning, and Design Guide*, SG24-7957
- *Implementing REXX Support in SDSF*, SG24-7419
- *Security Functions of IBM DB2 10 for z/OS*, SG24-7959
- *System z Parallel Sysplex Best Practices*, SG24-7817
- *WebSphere Application Server V8.5 Concepts, Planning, and Design Guide*, SG24-8022

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Other publications

These publications are also relevant as further information sources:

- *DB2 10 for z/OS Administration Guide*, SC19-2968
- *DB2 10 for z/OS Application Programming Guide and Reference for Java*, SC19-2970
- *DB2 10 for z/OS Command Reference*, SC19-2972
- *DB2 10 for z/OS Data Sharing: Planning and Administration*. SC19-2973

- *DB2 10 for z/OS Installation and Migration Guide*, GC19-2974
- *DB2 10 for z/OS Managing Performance*, SC19-2978
- *DB2 10 for z/OS Messages*, GC19-2979
- *IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS, Reporting User's Guide*, SH12-6927
- *Tivoli OMEGAMON XE for DB2 on z/OS Report Reference*, SH12-6963
- *z/OS MVS Planning: Workload Management*, SA22-7602-20
- *z/OS Programming: Resource Recovery*, SA22-7616-11
- *z/OS V1R13 Resource Measurement Facility (RMF) User's Guide*, SC33-7990

Online resources

These websites are also relevant as further information sources:

- DB2 10 for z/OS information

  http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=%2Fcom.ibm.db2z10.doc.comref%2Fsrc%2Fcomref%2Fdb2z_comref.htm
- Download initial Version 10.1 clients and drivers

  http://www.ibm.com/support/docview.wss?rs=4020&uid=swg21385217
- IBM developerWorks DB2 for z/OS preferred practices presentations

  http://www.ibm.com/developerworks/data/bestpractices/db2zos/
- pureQuery

  http://www.ibm.com/developerworks/data/library/techarticle/dm-0708ahadian/
- System z Solution Edition for Application Development

  http://www.ibm.com/systems/z/solutions/editions/appdev/index.html
- WebSphere Application Server z/OS V8 Resource Adapter Failover Lab

  http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102033
- WebSphere glossary

  http://pic.dhe.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.help.glossary.doc/topics/glossary.html
- WebSphere Portal zone

  http://www.ibm.com/developerworks/websphere/zones/portal/
- New to WebSphere

  http://www.ibm.com/developerworks/websphere/newto/

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

D

J

K

L

M

N

T

U

V

W

X

Z

Redbooks

DB2 for z/OS and WebSphere Integration for Enterprise Java Applications

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

DB2 for z/OS and WebSphere Integration for Enterprise Java Applications

Understand Java drivers usage for workload balancing and failover

Tune DB2 and WebSphere on z/OS for best performance

Extend security and accounting to your clients

IBM DB2 for z/OS is a high-performance database management system (DBMS) with a strong reputation in traditional high-volume transaction workloads that are based on relational technology. IBM WebSphere Application Server is web application server software that runs on most platforms with a web server and is used to deploy, integrate, execute, and manage Java Platform, Enterprise Edition applications. In this IBM Redbooks publication, we describe the application architecture evolution focusing on the value of having DB2 for z/OS as the data server and IBM z/OS as the platform for traditional and for modern applications.

This book provides background technical information about DB2 and WebSphere features and demonstrates their applicability presenting a scenario about configuring WebSphere Version 8.5 on z/OS and type 2 and type 4 connectivity (including the XA transaction support) for accessing a DB2 for z/OS database server taking into account high-availability requirements.

We also provide considerations about developing applications, monitoring performance, and documenting issues.

DB2 database administrators, WebSphere specialists, and Java application developers will appreciate the holistic approach of this document.

SG24-8074-00 ISBN 0738438391