DB2 UDB V8 and WebSphere V5 Performance Tuning and Operation Guide

Best performance tuning practices for DB2 UDB and WAS integrated environment

Gets you smoothly up and running DB2 UDB and WAS together

Problem determination scenarios

Whei-Jen Chen
Peter Z. He
Kang Yong Ying
Sunminder S Saini
Andreas Paulsen

ibm.com/redbooks
Note: Before using this information and the product it supports, read the information in “Notices” on page ix.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notices</td>
<td>ix</td>
</tr>
<tr>
<td>Trademarks</td>
<td>x</td>
</tr>
<tr>
<td>Preface</td>
<td>xi</td>
</tr>
<tr>
<td>The team that wrote this redbook.</td>
<td>xii</td>
</tr>
<tr>
<td>Become a published author</td>
<td>xiv</td>
</tr>
<tr>
<td>Comments welcome.</td>
<td>xv</td>
</tr>
<tr>
<td><strong>Chapter 1. Introduction.</strong></td>
<td>1</td>
</tr>
<tr>
<td>1. IBM On Demand era</td>
<td>2</td>
</tr>
<tr>
<td>1.1 The IBM On Demand operating environment</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Why DB2 UDB and WebSphere</td>
<td>9</td>
</tr>
<tr>
<td>1.3 Key areas of performance</td>
<td>10</td>
</tr>
<tr>
<td>1.3.1 Hardware.</td>
<td>11</td>
</tr>
<tr>
<td>1.3.2 Operating system</td>
<td>11</td>
</tr>
<tr>
<td>1.3.3 Application Server and WebServer</td>
<td>12</td>
</tr>
<tr>
<td>1.3.4 Database manager</td>
<td>13</td>
</tr>
<tr>
<td>1.3.5 Application programs</td>
<td>13</td>
</tr>
<tr>
<td>1.4 Performance tuning guidelines</td>
<td>14</td>
</tr>
<tr>
<td>1.4.1 Initial efforts always pay</td>
<td>14</td>
</tr>
<tr>
<td>1.4.2 Tune the identified constraints</td>
<td>14</td>
</tr>
<tr>
<td>1.4.3 Change one parameter at a time</td>
<td>14</td>
</tr>
<tr>
<td>1.4.4 Consider the entire system</td>
<td>15</td>
</tr>
<tr>
<td>1.4.5 Hardware upgradation</td>
<td>15</td>
</tr>
<tr>
<td>1.4.6 Follow performance tuning process and proper documentation</td>
<td>15</td>
</tr>
<tr>
<td>1.5 Process of performance tuning</td>
<td>16</td>
</tr>
<tr>
<td>1.5.1 Developing a performance-improvement process</td>
<td>16</td>
</tr>
<tr>
<td>1.5.2 Performance-tuning limits</td>
<td>18</td>
</tr>
<tr>
<td><strong>Chapter 2. Overview of WebSphere Application Server V5.</strong></td>
<td>19</td>
</tr>
<tr>
<td>2.1 Java 2 Enterprise Edition (J2EE) overview</td>
<td>20</td>
</tr>
<tr>
<td>2.1.1 J2EE platform roles</td>
<td>21</td>
</tr>
<tr>
<td>2.1.2 J2EE benefits</td>
<td>22</td>
</tr>
<tr>
<td>2.1.3 Application components and their containers</td>
<td>23</td>
</tr>
<tr>
<td>2.1.4 Standard services</td>
<td>23</td>
</tr>
<tr>
<td>2.1.5 J2EE packaging</td>
<td>24</td>
</tr>
<tr>
<td>2.2 WebSphere Application Server V5 product family</td>
<td>24</td>
</tr>
<tr>
<td>2.3 WebSphere Application Server runtime architecture</td>
<td>27</td>
</tr>
<tr>
<td>2.3.1 WAS Version 5 base runtime architecture</td>
<td>29</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 2004. All rights reserved.
6.3 WebSphere Application Server tuning ........................................ 193
  6.3.1 WebSphere Queuing Network ....................................... 193
  6.3.2 Configuring WebSphere Application Server queues ............ 194
  6.3.3 Using Performance Advisors ..................................... 205
  6.3.4 JVM memory tuning .............................................. 210
  6.3.5 Application assembly considerations .......................... 214
  6.3.6 Other considerations ........................................... 219
  6.3.7 Application best practice for performance ..................... 223

Chapter 7. Monitoring and tuning of DB2 UDB V8 ......................... 237
  7.1 Tools for monitoring and tuning ................................... 238
    7.1.1 Snapshot monitor ........................................... 238
    7.1.2 Event monitor ............................................. 246
    7.1.3 Explain utilities .......................................... 250
    7.1.4 DB2 Diagnostic Log (DB2DIAG.LOG) ....................... 251
    7.1.5 Health Center/Memory Visualizer .......................... 252
    7.1.6 Design Advisor ............................................ 253
    7.1.7 Configuration Advisor ................................... 253
  7.2 Application tuning .................................................. 254
    7.2.1 Database design ............................................ 254
    7.2.2 SQL tuning ................................................ 265
    7.2.3 Stored procedures ......................................... 268
    7.2.4 Declared global temporary tables ......................... 269
    7.2.5 Concurrency ............................................... 270
  7.3 System tuning ...................................................... 272
    7.3.1 Tuning the buffer pools ................................... 273
    7.3.2 Table management ......................................... 275
    7.3.3 Index management ......................................... 276
    7.3.4 Prefetcher ................................................ 279
    7.3.5 Cleaner .................................................... 280
    7.3.6 Sort heap ................................................ 280
    7.3.7 Locking .................................................... 281
    7.3.8 Logging .................................................... 284
    7.3.9 Tablespace ................................................. 285

Chapter 8. DB2 UDB V8 and WAS V5 integrated performance .............. 287
  8.1 WebSphere data sources ........................................... 288
    8.1.1 Connection pooling ....................................... 288
    8.1.2 Prepared statement cache .................................. 292
    8.1.3 Monitoring WebSphere application on DB2 UDB server ...... 301
    8.1.4 Tuning WebSphere DataSources ............................ 303
    8.1.5 Best practices ............................................. 307
  8.2 Persistent session .................................................. 311
8.2.1 Enable database persistence .................................................. 313
8.2.2 Session management tuning .................................................. 315
8.2.3 Using larger DB2 page size for database persistence ................. 321
8.2.4 Single vs. multi-row schemas in database persistence ............... 322
8.2.5 What is written to the persistent session database ................... 324
8.2.6 Invalidating sessions .......................................................... 327
8.2.7 Session performance best practices ....................................... 328
8.3 Enterprise JavaBeans ............................................................ 332
   8.3.1 EJB performance considerations ....................................... 332
8.4 Application considerations for performance in database access ....... 332

Chapter 9. Integrated troubleshooting ........................................... 337
9.1 Problem determination methodology ....................................... 338
9.2 Diagnostic information collection and analysis ........................... 340
   9.2.1 DB2 UDB V8 diagnostic information collection and analysis ...... 341
   9.2.2 WAS V5 diagnostic information collection and analysis .......... 353
9.3 Problem determination scenarios .......................................... 359
   9.3.1 Connectivity scenario .................................................... 359
   9.3.2 Concurrency scenario .................................................... 372
   9.3.3 High I/O consumption scenario ...................................... 387
   9.3.4 High CPU utilization scenario ....................................... 391

Appendix A. Trade3 application ...................................................... 401
Introduction .................................................................................... 402
Trade3 application deployment ....................................................... 402
Web Performance Tool ................................................................. 407

Related publications .................................................................... 411
IBM Redbooks ............................................................................. 411
Other publications ........................................................................ 411
Online resources .......................................................................... 412
How to get IBM Redbooks ........................................................... 413
Help from IBM ............................................................................. 413

Index .......................................................................................... 415
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 give 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites 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.

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 illustrates 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. 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 IBM’s application programming interfaces.
Trademarks

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

<table>
<thead>
<tr>
<th>AIX 5L™</th>
<th>e-business on demand™</th>
<th>Purify®</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX®</td>
<td>Encina®</td>
<td>Rational®</td>
</tr>
<tr>
<td>Balance®</td>
<td>Everyplace®</td>
<td>Redbooks™</td>
</tr>
<tr>
<td>ClearCase®</td>
<td>IBM®</td>
<td>Redbooks™</td>
</tr>
<tr>
<td>CICS®</td>
<td>ibm.com®</td>
<td>S/390®</td>
</tr>
<tr>
<td>Distributed Relational Database Architecture™</td>
<td>IMS™</td>
<td>SecureWay®</td>
</tr>
<tr>
<td>Domino®</td>
<td>Informix®</td>
<td>Tivoli®</td>
</tr>
<tr>
<td>DB2 Connect™</td>
<td>iSeries™</td>
<td>VisualAge®</td>
</tr>
<tr>
<td>DB2 Universal Database™</td>
<td>OS/390®</td>
<td>WebSphere®</td>
</tr>
<tr>
<td>DB2®</td>
<td>PowerPC®</td>
<td>xSeries®</td>
</tr>
<tr>
<td>DRDA®</td>
<td>pSeries®</td>
<td>z/OS®</td>
</tr>
<tr>
<td></td>
<td></td>
<td>zSeries®</td>
</tr>
</tbody>
</table>

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, 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 Sun Microsystems, Inc. in the United States, other countries, or both.

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

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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

This IBM® Redbook helps you design, develop, operate, monitor, and tune DB2® UDB V8 and WebSphere® Application Server (WAS) V5 based applications in UNIX® and Windows® environments. It also covers using DB2 Connect™ in a WAS environment.

This book is organized as follows:

► Chapter 1, “Introduction” on page 1, talks about the IBM On Demand operating environment and its roadmap, key areas of performance, and guidelines to tune them. We also discuss the process of performance tuning and recognize the limits of performance tuning in the system.

► Chapter 2, “Overview of WebSphere Application Server V5” on page 19, provides a brief overview of Java™ 2 platform Enterprise Edition (J2EE), including J2EE architecture, standard services provided by J2EE, application components and their corresponding containers, platform roles, as well as the benefits derived from adoption of J2EE. We also provide an overview of the WebSphere Application Server (WAS) V5.0.2, which is the premier J2EE and Web services technology-based application platform, and offers one of the first production-ready application servers for the deployment of enterprise Web services solutions for dynamic e-business on demand™.

► Chapter 3, “Overview of DB2 UDB V8” on page 47, covers the design of DB2 UDB V8. This chapter gives you a brief overview of DB2 UDB and discusses topics like the product family, the architecture, and the tools. With this knowledge you are prepared to understand the following parts of the publication where we discuss performance tuning and problem determination on DB2 UDB.

► Chapter 4, “DB2 UDB V8 and WAS V5 integrated environment” on page 75, discusses the topology selection in the WebSphere Application Server and DB2 UDB integrated environment that implements the multi-tier application model, and how WebSphere Application server interacts with DB2 UDB.

► Chapter 5, “Operational setup” on page 117, discusses the general steps to get DB2 UDB V8 and WebSphere Application Server V5 working together to serve specific business and application requirements. The new DB2 EJB sample application named “AcessEmployee” is introduced in detail to let you have a starting point to show you how DB2 UDB V8 and WebSphere Application Server V5 fit together. In addition, we also discuss the available JDBC drivers shipped with DB2 UDB V8 and the differences between those drivers.
Chapter 6, “WebSphere Application Server V5 performance tuning” on page 169, discusses tuning an existing WebSphere environment for performance and J2EE application best practices for performance. Performance tuning methodology and guidelines are also provided. We also cover the Performance Monitoring infrastructure (PMI) and performance tuning and monitoring tools found in WebSphere Application Server V5.

Chapter 7, “Monitoring and tuning of DB2 UDB V8” on page 237, covers performance tuning on DB2 UDB. It starts with the setup of a new project and the gathering of information about the project and the system. We show what to think of during the development phase, as well as the things to be done when the system is up and running. Learn how to monitor the system and how to use the tools provided by DB2 UDB for tuning and maintaining your system.

Chapter 8, “DB2 UDB V8 and WAS V5 integrated performance” on page 287, provides a description of the components that have an important performance impact in a WebSphere Application Server and DB2 UDB integrated environment, and offers some best-practices guidelines for their usage.

Chapter 9, “Integrated troubleshooting” on page 337, describes some common methods of diagnosing the problems related to DB2 UDB and/or WebSphere Application Server using diagnostic log files and traces. Also to further elaborate the understanding we have covered problem scenarios where you will learn the problem determination methodology and mechanisms to resolve those problems.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.
Whei-Jen Chen is a Project Leader at the International Technical Support Organization, San Jose Center. She has extensive experience in application development, database design and modeling, and DB2 system administration. Whei-Jen is an IBM Certified Solutions Expert in Database Administration and Application Development as well as an IBM Certified IT Specialist.

Peter Z He is a Software Developer with the DB2/WebSphere Solution Integration Team at the IBM Toronto Lab. Peter has been working extensively with DB2 and WebSphere technologies. He designed and led the development of the IBM Video Central software and the DB2 V81 J2EE sample application, Access Employee. He also participated in the development of DB2 Content Manager V8, Rational® ClearCase®, and WebSphere Commerce, and provided System architecture, J2EE application design, and DBA consulting to customers. He holds a Ph.D. from York University and is a DB2 Certified Solutions Expert.

Kang Yong Ying is a Senior IT Specialist from the Technical Sales Support, IBM China. He has extensive experience in database software for distributed platforms and also familiar with WebSphere Application Server products. He is also an IBM Certified Advanced Technical Expert-DB2 for Clusters. His areas of expertise include DB2 UDB administration and troubleshooting, as well as Business Intelligence solution implementation.

Andreas Paulsen is a DB2-certified IT Architect working for the DB2 Software Services Organization in Berlin, Germany. He worked as a Consultant for middleware solutions for the Informix® Company (before acquired by IBM), and
previously he developed database solutions in two other companies on Informix databases. Currently he is working on migration projects to bring databases from competitors to DB2 UDB. His knowledge also includes WebSphere and ETL processes. Andreas holds a Masters degree in Computer Science from the Technical University of Berlin.

**Sunminder S Saini** is an Advisory IT Specialist from the e-Business Software Center, IBM Software Group, IBM India. He has seven years of experience in the enterprise applications and distributed environment. He is certified for DB2 UDB Administration, Managing and Optimizing-Informix Dynamic Server, WebSphere Application Server, and WebSphere Studio Application Developer. His areas of expertise include DB2 UDB/Informix and WebSphere administration. He has written extensively on DB2 UDB Administration for Linux, Windows, and UNIX.

We would like to thank the following people for their contributions to this project:

Grant Hutchison  
Rahul Kitchlu  
Yongli An  
**IBM Toronto Laboratory**

Curt Cotner  
**IBM Silicon Valley Laboratory**

Jerry Yang  
**IBM Business Consultanting Services**

Angelo Sironi  
**IBM Software Group - Italy**

Emma Jacobs  
**International Technical Support Organization, San Jose Center**

Julie Czubik  
**International Technical Support Organization, Poughkeepsie Center**

### Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You’ll team with IBM technical professionals, Business Partners and/or customers.
Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

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 Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

► Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks

► Send your comments in an Internet note to:
  redbook@us.ibm.com

► Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. QXXE  Building 80-E2
  650 Harry Road
  San Jose, California 95120-6099
Introduction

This is an On Demand world where companies must be more responsive, adaptive, focused, and resilient to compete in the business world. To attain these attributes, an information technology infrastructure must be in place to support the business and provide true values to businesses.

IBM provides a platform for e-business On Demand, which enables a robust environment to build your information and applications. A flexible set of offerings allows our customers to get started today in building an On Demand environment in support of their businesses.

This chapter talks about the IBM On Demand operating environment and its roadmap, key areas of performance, and guidelines. We also go through the process of performance tuning and recognize the performance limits of your system.

The topics discussed are:

- IBM On Demand era
- Why DB2 UDB and WebSphere
- Key areas of performance
- Performance tuning guidelines
- Process of performance tuning
1.1 IBM On Demand era

Welcome to a whole new era—e-business On Demand.

We are entering a new era in business, the On Demand era. An era in which everyone demands that you are more responsive, more flexible, and more resilient than ever before.

It is a new era in business where you demand partners with the vision to foresee where the world is going and the resources to help you make sure that your ideas deliver the results you expect.

1.1.1 The IBM On Demand operating environment

The e-business On Demand is not just a vision. It is the mechanism by which businesses can be more responsive, variable, focused, and resilient to cope with ever-increasing pressures and competition from the global economy.

Overview
On Demand businesses need On Demand operating environments. The On Demand operating environment extends integration of people, processes and information to include the entire value chain. It virtualizes IT resources to improve utilization and align IT expenses with business needs. It is based on open standards and it leverages automated technologies to manage IT resources, allowing you to focus on running your business.

To achieve your business goals, you have to be efficient, integrated, flexible and responsive. For these objectives, new technology is available to help you to:

- Reduce cost and improve speed.
- Integrate people, processes and information across and beyond the enterprise.
- Create and deploy new business processes as needed.
- Adapt existing applications and business logic to meet changing requirements.

There are four major attributes of an on demand business:

- Responsive
  Able to sense and respond to dynamic, unpredictable changes in demand, supply, pricing, labor, competition, capital markets, and the needs of its customers, partners, suppliers, and employees.
- **Variable**
  Able to adapt processes and cost structures to reduce risk while maintaining high productivity and financial predictability.

- **Focused**
  Able to concentrate on its core competencies and differentiating capabilities.

- **Resilient**
  Able to manage changes and external threats while consistently meeting the needs of all of its constituents.

IBM provides an IT infrastructure for a business to be successful. We call this infrastructure the on demand operating environment. The IBM on demand operating environment is the end-to-end enabling of an IT infrastructure. It is an integrated platform, based on open standards, that can enable rapid deployment and integration of business applications and processes, combined with an environment that allows true virtualization and automation of the infrastructure.

*Figure 1-1  Overview of an on demand operating environment*
In Figure 1-1 on page 3 we show the three main capabilities of On Demand solution offerings.

- Integration
  The efficient and flexible combination of resources to optimize operations across and beyond the enterprise. It is about people, processes, and information.

- Automation
  The capability to reduce the complexity of management to enable better use of assets, improve availability and resiliency, and reduce costs based on business policy and objectives.

- Virtualization
  Provides a single, consolidated view of and easy access to all available resources in a network—no matter where the data resides.

**What integration is**
Integration is a more efficient and flexible way to combine resources to optimize operations across and beyond the enterprise. It provides a single, consolidated view of all available resources in a network—no matter where they reside.

![Diagram of integration](image)

*Figure 1-2  On Demand integration*

Figure 1-2 shows the key areas of integration that can be used to consolidate the on demand operating environment.
**People integration**

People integration deals with the process of simplifying the end-user requirements by providing more productive tools and standard interfaces to access applications and data anytime, anyplace.

- **Simplicity for end-user**
  
  In order to simplify the end-user process, IBM provides the tools and facilities that make employees more productive and make it easier for customers, suppliers, and partners to do business with the company.

- **Secure role-based interactions**
  
  This ensures privacy and the protection of data while meeting the dynamic demands of the users. IBM provides an On Demand infrastructure so that people can be served with what they need, when they need it. Such facilities require a solution that includes integrated, role-based policies.

- **Standardize access to applications**
  
  Providing standardized interfaces to existing and new applications not only helps users to quickly adapt the new applications, but also help the company to quickly re-deploy resources to meet the changing demands of the industry.

- **Access to data anytime, anywhere**
  
  This allows users to access resources anytime, anyplace using desktop systems, laptops, ATMs, PDAs, cell phones, and other devices.

**Process integration**

Business requirements change everyday and so do business applications. In order to maintain the consistency among various business applications we need to have common standards and open technologies to integrate the processes.

- **Consistent business process**
  
  A consistent business modeling process is not dependent upon underlying product implementations. By providing a consistent model of business processes, you can easily adapt applications as the business needs change.

- **Integrate applications**
  
  An On Demand operating environment provides the infrastructure required for applications to integrate themselves by using common standards and open technologies.

- **External connectivity**
  
  Businesses are changing rapidly. The cost and time required to rewrite applications or redesign processes compatible with existing tools is unacceptable. On Demand infrastructure provide standards through which
applications and processes can be connected to other external applications and processes.

**Information integration**

Information lying on different operating environments and architectures needs consolidation for greater performance and availability.

- **Integrated information**

  An On Demand environment must provide a mechanism with which to manage the most appropriate placement of data based on cost or business needs. Either it consolidates information onto a single platform or provides access to data kept in different places, depending on business needs.

- **Distributed information**

  Data and content can be accessed independently of its location or platform in an On Demand environment. The ability to access data without making any impact to an existing IT infrastructure helps corporations gain greater return on existing information assets.

- **Data consolidation**

  The On Demand environment provides data placement management which helps information to be consolidated for greater performance and availability.

IBM is changing the way we develop and deliver our products to meet on demand capabilities. Integration across people, processes and information provides the real strength to work in the On Demand era.

**What automation is**

Automation is the capability to reduce the complexity of deployment and monitoring, create a better use of assets, improve availability, and protect an IT infrastructure to meet the business needs with little or no human intervention.

Figure 1-3 on page 7 provides the blueprint given by IBM to assist customers to implement automation capabilities to enhance their businesses.
The bottom layer of Figure 1-3 shows that IBM has a full portfolio of software and system resources with built-in autonomic capabilities, which provide the most advanced level of infrastructure to implement automation.

The second layer from the bottom shows the key automation capabilities:

- **Availability** helps to ensure that systems are 24x7.
- **Security** keeps systems protected from threats and unwanted intruders.
- **Optimization** provides tools to ensure that all the resources are running at optimal performance and efficiency levels.
- **Provisioning** focuses on the self-configuring and self-healing.

The next layer, Infrastructure Based Orchestration, implements coordination across the core automation disciplines: Provisioning, availability, optimization, and open standards.

The top-most layer shows End-to-End Business Service Management tools required to visualize an IT environment in business service terms. It aligns and measures Service Level Agreements by the needs of the lines of business. These tools are required to manage service levels; and connect, monitor, and manage business processes end-to-end for complete linkage of IT and business processes.
What virtualization is

Virtualization is the process of providing an On Demand operating environment where resources can be used efficiently based on the business requirements. Virtualization enables the sharing of resources and it helps to lower the Total Cost of Ownership (TCO) of an IT infrastructure.

Figure 1-4  On Demand virtualization

Figure 1-4 shows the four major areas of virtualization.

- Servers
  By building an environment where the servers can be shared across geographies, business units, processes, and applications, allowing for consolidation.

- Storage
  Providing access to data, regardless of its physical location, provides cost effective storage media and more efficient data sharing.

- Distributed systems
  Enables advance distributed systems like grid computing, which allows sharing resources across the enterprise, systems, servers, clusters and storage devices dynamically to meet business needs. It helps to attain optimal utilization of resources and provide better return on investment.
Networking

The network resources can be shared among different enterprises using technologies such as VPNs, VLANs, IP virtualization, and more. It is very critical to manage the network resource optimally.

Overall, a virtualized environment provides simplified access to data and IT resources on demand. It reduces the need to purchase additional hardware and software. The savings realized from reduced capital expenditures can be reinvested in other areas to help grow the business.

1.2 Why DB2 UDB and WebSphere

The IBM WebSphere® software platform is an integral component of an on demand operating environment. WebSphere software provides a robust platform to develop, run, integrate and access business applications.

The IBM WebSphere Application Server is a high-performance and extremely scalable transaction engine for dynamic e-business applications. WebSphere software enables business process and application integration within and outside the enterprise. It provides a platform to develop consistent user interfaces to business applications, and provides a scalable and reliable foundation to develop and deploy applications.

WebSphere continues the evolution to a single Web services-enabled, Java™ 2 Enterprise Edition (J2EE) application server and development environment that addresses the essential elements needed for an on demand operating environment. With WebSphere Application Server V5.0.2, WebSphere demonstrates its continued commitment to the realization of the IBM e-business on demand vision with new platforms and important functional enhancements.

WebSphere Application Server delivers an on demand environment with:

- A build-to-integrate platform by integration-ready applications that leverage existing software assets.
- Maximize ROI and lower costs with superior developer productivity with “portal-like” integration.
- Create a competitive advantage and optimize price/performance while meeting the changing demands of dynamic e-business with a virtualized environment featuring industry-leading reliability, scalability, performance, and security.

The DB2 UDB database is the most advanced self-managing, self-configuring, self-optimizing database in the world.
Innovative manageability DB2 UDB Version 8.1 provides significant automation capabilities. Robust e-business foundation DB2 UDB Version 8.1 provides performance, scalability and availability.

It provides integrated business intelligence for customers to gain a faster Return on Investment (ROI) from their data. DB2 UDB Version 8.1 provides sophisticated Business Intelligence capabilities to easily organize information to perform faster and more insightful queries.

Application development upon DB2’s leadership in open standards, IBM delivers a new Development Center that makes it easier for developers to build and deploy applications for either JAVA or Microsoft® environments.

IBM DB2 software plays a critical role in the on demand operating environment infrastructure. It supports all the key e-business on demand attributes: integrated, open, virtualized, and autonomic. Here are a few of the DB2 capabilities supporting these attributes:

- **Integrated**
  DB2 UDB Version 8.1 helps solve critical business problems by integrating information across the entire enterprise by leveraging the federated Web Services and XML and support for both structured and unstructured information.

- **Open**
  Strong commitment to support Linux, Java, XML, Web services, grid computing, multi-vendor, and multi-platform standards.

- **Virtualized**
  Provides a clustered scalability to support expansion of a virtualized information environment.

- **Autonomic**
  Self-tuning capabilities of DB2 Universal Database™ including self-configuring, self-optimizing, and self-managing capabilities, and rapid DB2 deployment.

### 1.3 Key areas of performance

Performance is the way a computer system behaves given a particular workload. Performance is measured in terms of system response time, throughput, and availability. Performance is also affected by the resources available in your system and how well those resources are used and shared.
In general, you tune your system to improve its cost-benefit ratio. Specific goals could include:

- Processing a larger, or more demanding, work load without increasing processing costs; for example, to increase the work load without buying new hardware or using more processor time.
- Obtaining faster system response times, or higher throughput, without increasing processing costs.
- Reducing processing costs without degrading service to your users.

Translating performance from technical terms to economic terms is difficult. Performance tuning certainly costs money in terms of user time as well as processor time, so before you undertake a tuning project, weigh its costs against its possible benefits. In the following sections we discuss a few key areas of performance tuning.

### 1.3.1 Hardware

A balanced system is a system that has a sensible set of CPU, memory, disks, and network connections. There should be no bottleneck, which would make the other components ineffective. All parts of a balanced system are used appropriately during normal business operation workloads. When you acquire the hardware, please consider the following:

- Type of workload and the CPU power required for the various transactions
- Expected size of the database data
- The maximum number of concurrently connected users
- The maximum number of concurrent jobs
- Number of transactions within a certain period of time

### 1.3.2 Operating system

Diagnosing some problems related to memory, swap files, CPU, disk storage, and other resources requires a thorough understanding of how a given operating system manages these resources. At a minimum, defining resource-related problems requires knowing how much of that resource exists, and what resource limits may exist per user.

Here is some important configuration information that you need to obtain:

- Operating system patch level, installed software, and upgrade history
- Swap and file cache settings
- User data and file resource limits and per-user process limit
Inter-Processor communication (IPC) resource limits (message queues, shared memory segments, semaphores)

1.3.3 Application Server and WebServer

By tuning application server settings, you can control how an application server provides services for running applications and their components. WebSphere Application Server contains interrelated components that must be harmoniously tuned to support the custom needs of your end-to-end e-business application. Below are few important areas of consideration.

JVM
The application server, being a Java process, requires a Java virtual machine (JVM) to run, and to support the Java applications running on it. As part of configuring an application server, you can fine-tune settings that enhance system use of the JVM.

Web container
One of the parts of each WebSphere Application Server is a Web container. To route servlet requests from the Web server to the Web containers, the product establishes a transport queue between the Web server plug-in and each Web container. The Web container is initially created with default property values suitable for simple Web applications. However, these values might not be appropriate for more complex Web applications.

EJB container
One of the parts of each application server in WebSphere Application Server is an EJB container. An EJB container is automatically created when you create an application server. After the EJB container is deployed, you can change the parameters to make adjustments that improve performance.

Data sources
A data source is used to access data from the database. Certain parameters reveal how the number of physical connections within a connection pool can change performance.

Object Request Broker
An Object Request Broker (ORB) manages the interaction between clients and servers, using the Internet InterORB Protocol (IIOP). It supports client requests and responses received from servers in a network-distributed environment.
Session management
IBM WebSphere Application Server session support has features for tuning session performance and operating characteristics, particularly when sessions are configured in a distributed environment. These options support the administrator flexibility in determining the performance and failover characteristics for their environment.

1.3.4 Database manager
When a DB2 UDB instance or a database is created, a corresponding configuration file is created with default parameter values. You can modify these parameter values to improve performance.

- Use sufficient agents for the workload.
- Do not allow DB2 UDB to needlessly close and open files.
- Do not allow extended lock waits.
- Manage DB2 sort memory conservatively and do not mask sort problems with large SORTHEAPs.
- Analyze table access activity and identify tables with unusually high rows read per transaction or overflow counts.
- Analyze the performance characteristics of each tablespace, and seek to improve the performance of the tablespaces with the slowest read times, longest write times, highest physical I/O read rates, worst hit ratios, and access attributes that are inconsistent with expectations.
- Create multiple buffer pools, and make purposeful assignments of tablespaces to buffer pools such that access attributes are shared.
- Examine DB2 UDB SQL statement Event Monitor information to discover which SQL statements are consuming the largest proportions of computing resources, and take corrective actions.
- Reevaluate configuration and physical design settings once high-cost SQL is eliminated.

1.3.5 Application programs
Whether the program is new or purchased, small or large, the developers, the installers, and the prospective users have assumptions about the use of the program, such as:

- Who is going to use the program
- Situations in which the program will be run
How often those situations will arise and at what times of the hour, day, month, or year

Whether those situations will also require additional uses of existing programs

Which systems the program will run on

How much data will be handled, and from where

Whether data created by or for the program will be used in other ways

Unless these ideas are elicited as part of the design process, they will probably be vague, and the programmers will almost certainly have different assumptions than the prospective users. Even in the apparently trivial case in which the programmer is also the user, leaving the assumptions unarticulated makes it impossible to compare design to assumptions in any rigorous way. Worse, it is impossible to identify performance requirements without a complete understanding of the work being performed.

### 1.4 Performance tuning guidelines

The following guidelines should help you develop an overall approach to performance tuning.

#### 1.4.1 Initial efforts always pay

This is the basic law that most of the performance benefits usually come from initial efforts. While designing the system try to get the requirements/facts, like disk sizes, disk I/O rates, number of users, transaction and batch processing load on CPU, transaction rates, memory requirements, availability, network bandwidth, backup requirements. If not taken care of, these factors later on generally produce smaller benefits and initially require more effort.

#### 1.4.2 Tune the identified constraints

Identify the primary cause of performance bottleneck. If you try to tune resources that have little or no effect on response time, it can actually make subsequent tuning processes more difficult. If there is any significant improvement potential, it lies in improving the performance of the resources that are major factors in the response time. The best approach is to tune identified constraints.

#### 1.4.3 Change one parameter at a time

Even though you are sure that all the changes are going to be beneficial, do not change more than one performance tuning parameter at a time, otherwise you
have no way of evaluating how much each change contributed. Changing one parameter at a time helps you to evaluate whether the change does what you want. Remember, every time you change a parameter to improve one area, you almost always affect at least one other area that you may not have considered.

**Tune one level of your system at a time**

Tune one level of your system at a time. This helps you to understand how much each area contributed. Consider the below-mentioned areas while tuning processes:

- Hardware
- Network
- Operating system
- Application server
- Web server
- Database manager
- Application programs
- SQL statements

**1.4.4 Consider the entire system**

Beyond some limits, software alone cannot help much for further performance improvements. You might need to add more storage, additional CPUs, more memory, or a combination of these.

**1.4.5 Hardware upgradation**

Upgrading hardware is not an easy job inspite of good budget approvals from the management. You can spend money on additional disk storage only to find that you do not have the processing power or the channels to exploit it. So it is very critical to identify which portion of hardware should be upgraded. Also, upgrading hardware does not always work. Sometimes even additional storage or processing power cannot help much to improve performance.

**1.4.6 Follow performance tuning process and proper documentation**

Try to follow a proper performance tuning methodology and keep every step documented. Before applying any tuning step one must be ready with the scripts to revert that step. This saves a lot of time, as tuning is an iterative process and the possibility of applying same steps again and again usually remains very high.
1.5 Process of performance tuning

Develop a plan for performance monitoring and tuning by taking the inputs from the user, and recognizing the limits of tuning in your system.

Performance tuning is an iterative process. Depending upon the result of monitoring, adjust the configuration of the hardware, operating system, database server, and application server, and make changes to the applications that use this infrastructure. Base your performance monitoring and tuning decisions on your knowledge of the kinds of applications that use the data and the patterns of data access. Different kinds of applications have different performance requirements.

1.5.1 Developing a performance-improvement process

Choose performance criteria and set performance objectives based on those criteria. The main overall performance criteria of computer systems are response time and throughput.

Response time is the elapsed time between when a request is submitted and when the response from that request is returned. Examples include how long a database query takes, or how long it takes to echo characters to the terminal, or how long it takes to access a Web page.

Throughput is a measure of the amount of work over a period of time. In other words, it is the number of workload operations that can be accomplished per unit of time. Examples include database transactions per minute, kilobytes of a file transferred per second, kilobytes of a file read or written per second, or Web server hits per minute.
Chapter 1. Introduction

Performance tuning is primarily a matter of optimal resource management and correct system-parameter setting. Tuning the workload and the system for efficient resource use consists of the following steps:

1. Define performance objectives.
   a. Determine how the results will be measured.
   b. Quantify and prioritize the objectives.
2. Develop and execute a performance monitoring plan.
3. List the constraints, limitations, and boundaries of the system.
4. Identify the workloads on the system.
5. Identify the critical resources that limit the system’s performance.
6. Minimize the workload’s critical-resource requirements:
   a. Use the most appropriate resource, if there is a choice.
   b. Reduce the critical-resource requirements of individual programs or system functions.
   c. Structure for parallel resource use.
7. Make one change at a time.
8. Repeat steps 4 through 7 until objectives are met or resources are saturated.
9. Apply additional resources, if necessary.

Figure 1-5  Performance tuning process
1.5.2 Performance-tuning limits

Every system has its own limitations and boundaries. Beyond certain limits you start releasing that even after putting a lot of efforts in tuning process only a small amount of efficiency benefits reaped from the system.

Try to consider how much time and money your management is ready to spend on improving system performance, and how much the end user will benefit from this process. Take care of budgets because sometimes significant performance improvement requires additional infrastructure, that is, you might need to add more disk storage, faster CPU, additional CPUs, more main memory, faster communication links, or a combination of these.

However, there is a point beyond which tuning cannot help. At this point, consider revising your goals and expectations within the limits of your environment.
Overview of WebSphere Application Server V5

This chapter provides a brief overview of Java 2 Platform Enterprise Edition (J2EE), including J2EE architecture, standard services provided by J2EE, application components and their corresponding containers, and platform roles, as well as the benefits derived from adoption of J2EE.

In this chapter we also provide an overview of WebSphere Application Server (WAS) V5.0.2, which is the premier J2EE and Web services technology-based application platform and offers one of the first production-ready application servers for the deployment of enterprise Web services solutions for dynamic e-business on demand. The following topics are discussed for WebSphere Application Server V5.0.2:

- WebSphere Application Server product family
- WebSphere Application Server runtime architecture
- WebSphere Application Server V5 tools
- WAS V5.0.2 and Supported J2EE APIs
2.1 Java 2 Enterprise Edition (J2EE) overview

The Java 2 platform Enterprise Edition (J2EE) specification is the standard and the approach for developing, assembling, deploying, running and managing multi-tier server-centric enterprise applications. It defines a set of coordinated specifications that applies to all aspects of architecture and developing large-scale applications. In general, J2EE encourages the adoption of a multiple-tier architecture and a strict separation between business logic and the presentation tier.

The standard architecture defined by the J2EE specification is composed of the following elements:

- Standard application model for developing multi-tier applications
- Standard platform for hosting applications
- Compatibility Test Suite (CTS) for verifying that J2EE platform products comply with the J2EE platform standard
- Reference implementation providing an operational definition of the J2EE platform

The J2EE platform specification describes the run-time environment for a J2EE application. This environment includes application components, containers, and resource manager drivers. The elements of this environment communicate with a specified set of standard services. Figure 2-1 on page 21 shows the basic J2EE architecture and the logical relationships between architectural elements.
IBM WebSphere Application Server Version 5 is a fully J2EE compatible product. It has completed the full J2EE certification test suite. The product supports all of the J2EE 1.3 APIs, and exceeds many with its extensions. You can check the list of J2EE-compatible products posted by Sun Microsystems at:

http://java.sun.com/j2ee/compatibility.html

2.1.1 J2EE platform roles

The J2EE platform also defines a number of distinct roles performed during the application development and deployment life cycle:

- The J2EE Product Provider is the implementor and supplier of a J2EE product that includes the component containers, J2EE platform APIs, and other features that define the J2EE specification.

- The Application Component Provider creates Web components, enterprise beans, applets, or application clients to be used in J2EE applications.
The Application Assembler takes a set of components developed by component providers and assembles them in the form of an enterprise archive (EAR) file.

The Tool Provider provides tools used for the development and packaging of application components.

The Deployer is responsible for deploying an enterprise application into a specific operational environment.

The System Administrator is responsible for the operational environment in which the application runs.

Product providers and tool providers have a product focus. Application component providers and application assemblers focus on the application. Deployers and system administrators focus on the run time.

These roles help identify the tasks and people involved. Understanding this separation of roles is important, because it helps to determine the approach to take when developing and deploying J2EE applications.

### 2.1.2 J2EE benefits

The J2EE specification provides customers a standard by which to compare J2EE offerings from vendors and develop applications that run on any J2EE-compliant platform. Comprehensive, independent Compatibility Test Suites ensure vendor compliance with J2EE standards.

Some benefits of deploying to a J2EE-compliant architecture include:

- A simplified architecture based on standard components, services, and clients, that takes advantage of the write-once, run-anywhere Java technology.

- J2EE containers provide for the separation of business logic from resource and life cycle management, which means that developers can focus on writing business logic and simplify their development.

- Services providing integration with existing systems, including Java Database Connectivity (JDBC), Java Message Service (JMS), Java Interface Definition Language (Java IDL), the JavaMail API, and Java Transaction API (JTA and JTS) for reliable business transactions.

- Scalability to meet demand, by distributing containers across multiple systems and using database connection pooling, for example.

- A better choice of application development tools and components from vendors providing standard solutions.
A flexible security model that provides single sign-on support, integration with legacy security schemes, and a unified approach to securing application components.

The J2EE specifications are the result of an industry-wide effort that involves a large number of contributors. IBM has contributed in defining more than 80 percent of the J2EE APIs.

2.1.3 Application components and their containers

The J2EE programming model has four types of application components, which reside in four types of containers in the Application Server:

- Enterprise JavaBeans - Executed by the EJB container
- Servlets and JavaServer Pages files - Executed by the Web container
- Application clients - Executed by the application client container
- Applets - Executed by the applet container

For more descriptions of components and containers that apply specifically to the IBM WebSphere Application Server, see “Architectural features” on page 33.

J2EE containers provide the run-time support of the application components. There must be one container for each application component type in a J2EE application. By having a container between the application components and the set of services, the J2EE specification can provide a federated view of the APIs for the application components.

A container provides the APIs to the application components used for accessing the services. It can also handle security, resource pooling, state management, as well as naming and transaction issues.

2.1.4 Standard services

The J2EE platform provides components with a set of standard services. See the Sun Web page http://java.sun.com/products/ for descriptions of each standard service:

- HTTP and HTTPS
- Java Transaction API (JTA)
- Remote Method Invocation/Internet Inter-ORB Protocol (RMI/IIOP)
- Java Interface Definition Language (Java IDL)
- Java DataBase Connectivity (JDBC) API
- Java Message Service (JMS)
- Java Naming and Directory Interface (JNDI)
- JavaMail API and the JavaBeans Activation Framework (JAF)
- Java API for XML Parsing (JAXP)
2.1.5 J2EE packaging

Perhaps the most significant change introduced by the J2EE specification is how application components are packaged for deployment.

During a process called assembly, J2EE components are packaged into modules. Modules are then packaged into applications. Applications can be deployed on the Application Server. Each module and application contains a J2EE deployment descriptor. The deployment descriptor is an XML file providing instructions for deploying the application.

The ultimate source of J2EE information is the specification available from the Sun Microsystems Web site (http://java.sun.com). For more information about J2EE, including the list of the specification levels that comprise the J2EE 1.3 specification, see http://java.sun.com/j2ee/1.3/docs/#specs.

2.2 WebSphere Application Server V5 product family

Note: WebSphere Application Server V5.0.2 is the latest available release of WebSphere Application Server V5 at the time of writing. We will use the terms WebSphere Application Server V5.0.2 and WebSphere Application Server V5 interchangeably throughout this book.

IBM WebSphere Application Server Version 5 is IBM's fully compatible implementation of a J2EE platform conforming to J2EE 1.3 Specification. It delivers flexible configuration and deployment options. As your requirements change, you can migrate smoothly to the WAS configurations with more functionality and service. The IBM WebSphere Application Server product family is available in several different configurations that are designed to meet a wide range of customer requirements. Each edition addresses a distinct set of scenarios and needs. WebSphere Application Server includes:

- WebSphere Application Server Express
  
  This edition is a light-weight server for static content, servlets, and JSP pages, but does not support enterprise beans. It addresses the needs of mid-market companies and departments within larger organizations, and is a cost-effective starting point for businesses that want to have a presence on the Web.
The Express Edition provides a combination of development tools and application servers in a single integrated package geared toward developing Web-centric applications. It also provides a simplified programming model that allows you to create new Web applications and to convert existing static applications to dynamic applications. For more information about IBM WebSphere Application Server Express, visit the Express Edition’s home page at:

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

WebSphere Application Server (base configuration)

This edition addresses the basic programming and execution needs of desktop developers and single-server production scenarios. The execution environment for this edition addresses standards-based programming for Web and component-based programming, as well as Web services.

The administration model for this edition presumes a single-server environment—no clustering for failover or workload balancing, nor centralized administration of multiple server instances. However, you can add a stand-alone node to a centrally administered network (the cell) at any time after installing the next product, which controls the cell.

For more information about IBM WebSphere Application Server, see the IBM WebSphere Application Server home page at:

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

WebSphere Application Server Network Deployment

The Network Deployment edition delivers world-class caching, high availability, and industry-leading Web services support on top of the base J2EE 1.3 WebSphere Application Server foundation with support for the JDK 1.4 client container. It provides centralized administration of multiple server instances, as well as basic clustering and caching support. This configuration is designed to add non-programming enhancements to the features provided in the base configuration. These enhancements add scalability features, allowing you to run applications on multiple servers and on multiple physical nodes.

In addition to the features included with the base configuration of WebSphere Application Server you get:

- The Deployment Manager, allowing you to centrally manage a number of different application server instances and clustering for workload management and failover.

- The Edge Components, including Load Balancer, Caching Proxy, and IBM HTTP Server. These features provide the edge-of-network functions required to set up a classic DMZ in front of the application server.
A private UDDI Registry for easier deployment of internal Web services applications and a Web Services Gateway for external applications that request Web services to use rich security features to access an internal Web services provider application.

Web Services Gateway enables Web services invocation by users from outside the firewall with the benefit of robust security protection.

For more information, you could refer to the IBM WebSphere Application Server Network Deployment home page at:

http://www.ibm.com/software/webservers/appserv/was/network/

WebSphere Application Server Enterprise

This edition employs a service orientation to dynamically integrate IT assets within and across enterprises, and addresses large information technology production scenarios for applications that are designed according to the J2EE programming model and Web services. It enables advanced transactional connectivity to coordinate interactions with multiple back-end systems across the network. It delivers J2EE and Web services innovations to tackle some of today's toughest coding challenges. By using Dynamic workload management it increases application performance and flexibility. This capability allows the system to monitor the workload on each server in a cluster and automatically route the work to a server with the lightest workload.

For more information about IBM WebSphere Application Server Enterprise, visit the Enterprise edition's home page at:


WebSphere Application Server for z/OS®

This edition integrates the WebSphere Application Server product and the Network Deployment product into a single package that runs on a z/OS platform. It is a J2EE 1.3-compatible Web services application server specifically designed to utilize the unique qualities of service provided by IBM zSeries® hardware and z/OS operating system. It is functionally equivalent to the programming model of the Network Deployment, providing for greater flexibility in development. It provides improved deployment and management functions for a friendlier and more flexible environment, in a manner consistent with the Network Deployment.

For more information about IBM WebSphere Application Server for z/OS, visit its home page at:


Figure 2-2 on page 27 shows WebSphere Application Server's product packaging for version 5. It shows you the components available in different packaging configurations and the difference between different packages. Please
be aware that the Express and z/OS editions are not pictured in the figure. If you want to obtain more information for those editions, please visit their home pages mentioned in the previous pages.

Figure 2-2   WebSphere Application Server V5 packaging

2.3 WebSphere Application Server runtime architecture

Access to enterprise business logic and data from heterogeneous devices and technologies poses a significant technical challenge: The implementation of logic and data needs to be client neutral. Using a multi-tier architectural approach is a natural way to deal with this issue.

WebSphere Application Server provides the application logic layer in a multi-tier architecture, enabling client components to interact with data resources and legacy applications. Collectively, the multi-tier architectures are programming models that enable the distribution of application functionality across multiple independent systems, typically:

► Client components running on local workstations
► Processes running on remote servers
A discrete collection of databases, resource managers, and mainframe applications

Be aware that these tiers are just logical tiers. Enterprise applications can be partitioned in a number of tiers that are logically independent—although they may physically reside on the same machine or run in the same process.

The very back end tier (Tier-3) of an enterprise application is the Data Server, as illustrated on the right side of the Figure 2-3. The data server may need to handle a combination of legacy databases and newly created information repositories. The services provided by this tier are protected from direct access by the client components residing within a secure network. Interaction must occur through the Tier-2 processes, as shown in Figure 2-3.

The second-tier processes are commonly referred to as the application logic layer. These processes manage the business logic of the application, and are permitted to access the third-tier services. The business logic tier can be implemented by using Enterprise Java Beans. EJBs naturally map to business entities and business processes and, from a data persistence standpoint, they
can be mapped to existing databases or to new tables. The EJB tier incorporates all aspects of the business logic, including the process flow.

The core of application functions and data represented by Tier-2 and Tier-3 in the figure can then be accessed in a variety of ways. They can be accessed by traditional “thick” clients, which are represented by the Tier-1 clients in the figure. Typically, these clients require highly functional and rich GUIs, and access the enterprise applications within the company's firewalls, that is, within the Intranet. Applications can be accessed from the outside world, through the Internet, either via a browser-based interface (represented by the Tier-0 Web browsers) via hand-held devices, or via business-to-business interfaces.

Servlets and JSPs, encapsulated in Tier 1, will be responsible for offering a suitable rendition of the business data and methods to Tier-0 customers. This is what we call presentation logic. It enables the user to interact with the Tier-2 processes in a secure and intuitive manner. Tier 1 is in essence the only segment of the whole architecture that needs to be adapted to fit different access methods.

All these tiers must communicate with each other. Open standard protocols and exposed APIs simplify this communication. You can write client components in any programming language, such as Java or C++. These clients run on any operating system, by speaking with the application logic layer. Databases in the very back end tier can be of any design, if the application layer can query and manipulate them.

### 2.3.1 WAS Version 5 base runtime architecture

The WebSphere Application Server V5 base configuration includes only a single stand-alone application server process. Its configuration files are in XML and could be administered by the Administrative client programs such as Administrative Console or the wsadmin scripting tool. There is no node agent or Deployment Manager involved in this configuration. No coordination between application server processes is supported in the base configuration. Each application server instance has to be separately administered.

Figure 2-4 on page 30 illustrates the main components running in the node for a standalone WebSphere Application Server base configuration.
For architecture features introduction, please refer to “Architectural features” on page 33.

2.3.2 WAS Version 5 Network Deployment runtime architecture

IBM WebSphere Application Server Network Deployment configuration includes support for multiple nodes, each with a node agent process and several application servers, all coordinated within an administrative cell by the Deployment Manager process. Clusters of load-balanced application servers can be configured within a Network Deployment cell. The configuration and application binaries of all components in the cell are managed by the Deployment Manager and synchronized out to local copies on each of the nodes.

Figure 2-5 on page 31 shows the main components running in the node and Deployment Manager for a Network Deployment environment.
For your better understanding of the architecture for base configuration and Network Deployment configuration, here we briefly explain some terminologies that are commonly used in a WebSphere Application Server runtime environment.

**Node**

A node is a logical grouping of managed servers. A node usually corresponds to a physical computer system with a distinct IP host address. Node names usually are identical to the host name for the computer.

A node agent manages all WebSphere Application Server servers on a node. The node agent represents the node in the management cell.

**Cell**

Cells are arbitrary, logical groupings of one or more nodes in a WebSphere Application Server distributed network. A cell is a configuration concept, a way for administrators to logically associate nodes with one another. Administrators
define the nodes that make up a cell according to whatever criteria make sense in their organizational environments.

Administrative configuration data is stored in XML files. A cell retains master configuration files for each server in each node in the cell. Each node and server also have their own local configuration files. Changes to a local node or server configuration file are temporary, if the server belongs to the cell. While in effect, local changes override cell configurations. Changes at the cell level to server and node configuration files are permanent. Synchronization occurs at designated events, such as when a server starts.

The following components belong to IBM WebSphere Application Server Network Deployment only.

**Deployment managers**
Deployment managers are administrative agents that provide a centralized management view for all nodes in a cell, as well as management of clusters and workload balancing of application servers across one or several nodes in some editions. WebSphere Application Server for z/OS uses WLM as the primary vehicle for workload balancing.

A deployment manager hosts the administrative console. A deployment manager provides a single, central point of administrative control for all elements of the entire WebSphere Application Server distributed cell.

**Node agents**
Node agents are administrative agents that route administrative requests to servers.

A node agent is a server that runs on every host computer system that participates in the WebSphere Application Server Network Deployment product. It is purely an administrative agent and is not involved in application-serving functions. A node agent also hosts other important administrative functions such as file transfer services, configuration synchronization, and performance monitoring.

**Dispatcher**
The Dispatcher component is part of the Load Balancer component set of the IBM WebSphere Application Server Edge Components product. The Edge components are included in WebSphere Application Server Network Deployment. The Dispatcher performs intelligent load balancing by using server availability, capability, workload, and other criteria you can define, to determine where to send a TCP/IP request. You can use the Dispatcher to distribute HTTP
requests among Application Server instances that are running on multiple physical machines.

**Clusters**
Clusters are sets of servers that are managed together and participate in workload management. The servers that are members of a cluster can be on different host machines, as opposed to the servers that are part of the same node and must be located on the same host machine.

A cell can have no clusters, one cluster, or multiple clusters. Servers that belong to a cluster are members of that cluster set and must all have identical application components deployed on them. Other than the applications configured to run on them, cluster members do not have to share any other configuration data. One cluster member might be running on a huge multi-processor enterprise server system while another member of that same cluster might be running on a small laptop. The server configuration settings for each of these two cluster members are very different, except in the area of application components assigned to them. In that area of configuration, they are identical.

A vertical cluster has cluster members on the same node. A horizontal cluster has cluster members on multiple nodes.

A network dispatcher routes application access among cluster members by server-weighting, to provide better distribution control.

WebSphere Application Server can respond to increased use of an enterprise application by automatically replicating the application to additional cluster members as needed. This lets you deploy an application on a cluster instead of on a single node, without considering workload.

### 2.3.3 Architectural features

This section describes the major components within IBM WebSphere Application Server.

**HTTP server**
IBM WebSphere Application Server works with an HTTP server to handle requests for servlets and other dynamic content from Web applications.

The HTTP server and Application Server communicate using the WebSphere HTTP plug-in for the HTTP server. The HTTP plug-in uses an easy-to-read XML configuration file to determine whether a request is handled by the Web server or the Application Server. The HTTP plug-in uses the standard HTTP protocol to
communicate with the Application Server though you can configure it to use secure HTTPS, if required. The HTTP plug-in is available for popular Web servers.

**Application Server**
The WebSphere Application Server collaborates with the Web server by exchanging client requests and application responses. You can define multiple Application Servers, each running in its own Java virtual machine (JVM).

**EJB container**
The EJB container provides the run-time services needed to deploy and manage EJB components, known as enterprise beans. It is a server process that handles requests for both session and entity beans.

The enterprise beans (inside EJB modules) installed in an Application Server do not communicate directly with the server; instead, an EJB container provides an interface between the enterprise beans and the server. Together, the container and the server provide the bean run-time environment.

The container provides many low-level services, including threading and transaction support. From an administrative viewpoint, the container manages data storage and retrieval for the contained beans. A single container can manage more than one EJB JAR file.

**Web container**
Servlets and JavaServer Pages (JSP) files are server-side components used to process requests from HTTP clients, such as Web browsers. They handle presentation and control of the user interaction with the underlying application data and business logic. They can also generate formatted data, such as XML, for use by other application components.

The Web container processes servlets, JSP files, and other types of server-side includes Pre-J2EE servlets run in a servlet engine. Each Web container automatically contains a single session manager.

When handling servlets, the Web container creates a request object and a response object, and then invokes the servlet service method. The Web container invokes the servlet destroy() method when appropriate and unloads the servlet, after which the JVM performs garbage collection.

**Application client container**
Application clients are Java programs that typically run on a desktop computer with a graphical user interface (GUI). They have access to the full range of J2EE server-side components and services.
Chapter 2. Overview of WebSphere Application Server V5

The application client container handles Java application programs that access enterprise beans, Java Database Connectivity (JDBC), and message queues that belong to the Java Message Service (JMS). The J2EE application client program runs on client machines. This program follows the same Java programming model as other Java programs; however, the J2EE application client depends on the application client run time to configure its execution environment, and uses the Java Naming and Directory Interface (JNDI) name space to access resources.

**Applet container**
An applet is a client Java class that typically executes in a Web browser, but can also run in a variety of other client applications or devices.

Applets are often used in combination with HTML pages to enhance the user experience provided by a Web browser. They can also shift some of the processing workload from the server to the client.

The applet container handles Java applets embedded in HyperText Markup Language (HTML) documents that reside on a client machine that is remote from the application server. With this type of client, the user accesses an enterprise bean in the Application Server through the Java applet in the HTML document.

**Embedded HTTP server**
The HTTP handling capability embedded within the application server supports a connection from an HTTP client to the Application Server. An HTTP client can connect to a Web server and the HTTP plug-in can forward the request to the Application Server.

**Virtual host**
A virtual host is a configuration enabling a single host machine to resemble multiple host machines. Resources associated with one virtual host cannot share data with resources associated with another virtual host, even if the virtual hosts share the same physical machine.

Administrators can use virtual hosts to associate Web applications with a particular host configured to run the application.

**Session Management**
WebSphere Application Server provides facilities, grouped under the heading Session Management, that support the javax.servlet.http.HttpSession interface described in the Servlet API specification.

When an HTTP client interacts with a servlet, the state information associated with a series of client requests is represented as an HTTP session and identified
by a session ID. Session Management is responsible for managing HTTP sessions, providing storage for session data, allocating session IDs, and tracking the session ID associated with each client request through the use of cookies or URL rewriting techniques. Session Management can store session-related information in several ways:

- In application server memory (the default). This information cannot be shared with other application servers.
- In a database. This storage option is known as database-persistent sessions.
- In another WebSphere Application Server instance. This storage option is known as memory-to-memory sessions.

The last two options are referred to as distributed sessions. Distributed sessions are essential for using HTTP sessions for failover facility. When an application server receives a request associated with a session ID that it currently does not have in memory, it can obtain the required session state by accessing the external store (database or memory-to-memory). If distributed session support is not enabled, an application server cannot access session information for HTTP requests that are sent to servers other than the one where the session was originally created. Session Management implements caching optimizations to minimize the overhead of accessing the external store, especially when consecutive requests are routed to the same application server.

**JNDI support**

Naming is used by clients of WebSphere Application Server applications to obtain references to objects related to those applications, such as Enterprise JavaBeans (EJBs) homes.

IBM WebSphere Application Server includes a name server to provide shared access to Java components, and an implementation of the javax.naming JNDI package, which supports user access to the WebSphere Application Server name server through the JNDI naming interface.

WebSphere Application Server's JNDI implementation is based on version 1.2 of the JNDI interface, and was tested with version 1.2.1 of Sun's JNDI Service Provider Interface (SPI).

**Messaging support using JMS**

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface. JMS provides a common way for Java programs (clients and J2EE applications) to create, send, receive, and read asynchronous requests as JMS messages.
WebSphere Application Server also supports automatic asynchronous messaging using message-driven beans (a type of enterprise bean defined in the EJB 2.0 specification) and JMS listeners (part of the JMS application server facilities). Messages are automatically retrieved from JMS destinations, optionally within a transaction, then sent to the message-driven bean in an J2EE application, without the application having to explicitly poll JMS destinations.

**Security Service**
IBM WebSphere Application Server Version 5 provides security infrastructure and mechanisms to protect sensitive J2EE resources and administrative resources and to address enterprise end-to-end security requirements on authentication, resource access control, data integrity, confidentiality, privacy, and secure inter-operability. IBM WebSphere Application Server security is based on industry standards such as SSL Security 2.0. Version 5 has an open architecture that processes secure connectivity and inter-operability with Enterprise Information Systems including DB2, CICS®, MQ Series, Domino®, IBM Directory and others with security providers including Tivoli® Access Manager (Policy Director) and WebSEAL secure proxy server.

### 2.4 WebSphere Application Server V5 tools

A number of tools in graphical and non-graphical user interfaces are provided by IBM WebSphere Application Server. In the following sections, we introduce you to some very useful product GUI tools as well as some key scripting and command-line tools. For a complete list of available commands and their detailed information please refer to the WebSphere Application Server V5 InfoCenter at:


#### 2.4.1 Tools for installing, upgrading, and migrating

This subsection provides information about the tools available in WebSphere Application Server V5 for installation, upgrade and migration purpose.

**First Steps**
First Steps is a post-installation ease-of-use tool for directing WebSphere Application Server elements from one place. Options dynamically appear on the First Steps panel, depending on the features you install. With all options present, you can use First Steps to start or stop the application server, verify the installation, access the InfoCenter, run the Application Assembly Tool, access the administrative console, access the Samples Gallery, or launch the product registration.
First Steps starts automatically at the end of the installation. If it is not running, start First Steps from the IBM WebSphere menu, select **Application Server v5.0 -> First Steps**, or you could invoke it from the command line:

- On Windows platforms: `install_root\bin\firststeps.bat`
- On UNIX-based server platforms: `install_root/bin/firststeps.sh`

**Launch Pad**
This is a graphical interface for launching the product installation. It also provides links to information that you might need for installation.

**Installation wizard**
This is a graphical interface that leads you through the process of installing the product.

**Migration tools**
Command-line tools, such as WASPreUpgrade and WASPostUpgrade, are available to help you migrate from a previous product version.

### 2.4.2 Tools for assembling applications

Assembling is a necessary packaging and configuration step prior to deploying an application onto the server. IBM WebSphere Application Server also provides a couple of tools to assist you in these types of works.

**Application Assembly Tool**
The Application Assembly Tool (AAT) assembles enterprise applications for deployment. For more information, see **Applications -> Assembly tools -> Assembling applications** in IBM WebSphere Application Server V5 InfoCenter.

**Note:** In WebSphere Application Server V5.0.2, for some platforms such as Windows, AIX®, and Linux Intel® operating systems, the Assembly Toolkit replaces the Application Assembly Tool (AAT). To download the Application Server Toolkit product or obtain the latest information about this toolkit, you could visit the Web site:


**Deployment tool**
The AAT calls this command-line tool to generate code for deployment. You can generate code for deployment by either using the Application Assembly Tool
(AAT) or by using the Deployment Tool for Enterprise Java Beans (ejbdeploy) from a command prompt.

**Application Client Resource Configuration Tool**
Use this tool to configure deployment descriptors that define the resources needed by application clients.

**clientUpgrade**
Use this tool to migrate client Java archive (JAR) files from the J2EE 1.2 specification to the J2EE 1.3 specification.

### 2.4.3 Tools for deploying and administering

Deployment is the activity in which applications are placed onto application servers. In a WebSphere Application Server V5 environment, there are a few tools available to help you to deploy applications without needing to customize the application code for each server environment.

**Systems administration tools**
A couple of available systems administration tools exist, including the graphical WebSphere Administrative Console, the WSAdmin scripting client, and other command-line tools to speed up your deployment of applications and facilitate the administration of the system. For more details, please refer to the System Administration topic in the WebSphere Application Server V5 InfoCenter.

**LaunchClient command**
This command-line tool starts application clients. It could be invoked from a command line via `launchClient`.

**XML-SOAP administrative tool**
This graphical interface helps you manage deployed Web services. See Administering deployed Apache SOAP Web services (XML-SOAP administrative tool) in the WebSphere Application Server V5 InfoCenter for more details.

**NameSpaceDump tool**
You can use the dumpNameSpace tool to dump the contents of a name space accessed through a name server. It is very useful for debugging purposes. When you invoke the dumpNameSpace tool, make sure the naming service is active.
2.4.4 Tools for monitoring and tuning

This subsection provides information about the available tools in WebSphere Application Server V5 for monitoring and tuning purposes.

PMI request metrics in WebSphere Administrative Console

This tool collects data by timing requests as the requests travel through components of the product. The PMI request metrics are configurable through the administrative console. The PMI request metrics log time spent in major components, such as the Web container of the Application Server. These data points are recorded in logs and can be written to Application Response Time (ARM) agents used by Tivoli monitoring tools.

Tivoli Performance Viewer

The Tivoli Performance Viewer (TPV) is a Graphical User Interface (GUI) performance monitor for WebSphere Application Server. It is built on the Performance Monitoring Infrastructure (PMI) client API, which also is exposed to third-party development tools. Furthermore, the Performance Advisor in TPV provides advice to help tune systems for optimal performance and gives recommendations on inefficient settings by using collected PMI data. We have more introductions for TPV for specific scenarios in Chapter 6, “WebSphere Application Server V5 performance tuning” on page 169, and Chapter 9, “Integrated troubleshooting” on page 337.

2.4.5 Tools for troubleshooting

This subsection provides information regarding tools available in WebSphere Application Server V5 for troubleshooting purpose.

Log Analyzer

The Log Analyzer takes one or more service or activity logs, merges all of the data, and displays the entries. Based on its symptom database, it analyzes and interprets the error conditions in the log entries to help you diagnose problems. Log Analyzer has a special feature enabling it to download the latest symptom database from the IBM Web site.

The Log Analyzer tool cannot view remote files. If the operating system on which you are running WebSphere Application Server does not support the use of a graphical interface, transfer the file in binary mode to the system on which you are running the Java administrative console. Use the Log Analyzer tool there.
In cases where transferring the file is impractical or inconvenient, use the alternate viewing tool, showlog, to view the service or activity log file:

1. Change the directory to bin directory of the install_root.

2. Run the showlog tool with no parameters to display usage instructions:
   - On Windows systems, run `showlog.bat`.
   - On UNIX systems, run `showlog.sh`.

3. To direct the service or activity log (activity.log) contents to stdout, use the `showlog activity.log` command.

4. To dump the service or activity log to a text file for viewing with a text editor, use the `showlog activity.log textFileName` command.

Collector tool
The Collector tool gathers information about your WebSphere Application Server installation and packages it in a JAR file that can be sent to IBM Customer Support to assist in problem determination and analysis. For more information about the Collector tool, see the Monitoring and Troubleshooting topic in InfoCenter.

Application Server Toolkit
The Application Server Toolkit is included with IBM WebSphere Application Server, but on a separately installable CD. This kit includes debugging functionality that is built on the Eclipse workbench. See Debugging with the Application Server Toolkit in WebSphere Application Server V5 InfoCenter for more details.

2.5 WAS V5.0.2 and Supported J2EE APIs

In this section we introduce you to the new enhancements made in the WebSphere Application Server V5.0.2 and the list of supported J2EE APIs by WebSphere Application Server V5.0.2.

2.5.1 What is new in WebSphere Application Server V5.0.2

WebSphere Application Server V5.0.2 is more than just a “standard fixpack.” In this “mini-release”, WebSphere demonstrates its continued commitment to the realization of the IBM e-business on demand vision with new platforms and important functional enhancements. Along with support for the Java Development Kit (JDK) 1.4 client container, there are improvements in Web Services Inter-operability and Security, Performance and Application tuning, Platform Support, Systems Management, and Database Integration. The
introduction below is categorized by the operational requirements of the on
demand operating environment, that is, integrated, virtualized, open and
autonomic (refer to Chapter 1, “Introduction” on page 1 for more information
about the On Demand strategy).

**Integrated**
WebSphere Application Server 5.0.2 is an integral component of an on demand
operating environment.

**Unleash Interoperable and Secure Web Services**
IBM provides third-generation support for all of the Web services standards
needed for describing and deploying applications or services on a network in a
consistent way so that they can be discovered and invoked in a more secure and
reliable manner. In addition to high-performing support for all of the newly
released Web services standards, WebSphere Application Server V5.0.2 is the
first production level application server to support the June 2003 Board Approved
Draft of the WS-I Basic Profile 1.0. Developers building Web services
applications with WebSphere Application Server V5.0.2 will get a head start on
inter-operating across heterogeneous environments and enterprise boundaries.
As a result, IBM Web services technology can allow a business to be more
efficient, increase the value it can offer to its customers, and better differentiate
itself from its competitors.

**Industry’s Broadest Platform Support**
With V5.0.2, WebSphere Application Server and WebSphere Application Server
Network Deployment add new platforms to their already broad platform support.
Inclusion of the Native Deployment Manager for WebSphere Application Server
for zSeries on Linux completes support for zSeries beyond what was included in
V5.0. The new WebSphere Application Server for Linux on iSeries™ and
pSeries® further demonstrates the WebSphere commitment to open source.

New and completed platforms include:
- Complete support for zSeries on Linux, with the addition of the Native
  Deployment Manager
- Linux on iSeries and pSeries
- Linux Client (United Linux 1.0)
- Solaris 9
- Windows 2003 (.NET) server 32-bit
- Windows XP Professional (Client Support)
- Windows XP (Development and Test only)
Expanded Database Connectivity
A new, integrated DB2 Correlator with WebSphere Trace provides improved serviceability and better debugging when using WAS and DB2 together. WebSphere Application Server V5.0.2 also extends its support to make use of the New JDBC type 2 driver support for DB2 V8.

Virtualized
With Backup Clusters available in WebSphere Application Server Enterprise, customers can automatically configure their system to set up a back-up cluster of servers in case the primary cluster fails—without having to write any code, saving customers the time spent today setting up back-up clusters. If a cluster goes down, the workload is automatically sent to another cluster elsewhere in your network.

Open
WebSphere Application Server 5.02 continuously provides the open services infrastructure that allows companies to deploy a core operating environment that works as a reliable foundation capable of handling high-volume secure transactions and Web services.

Move closer to J2EE 1.4 with JDK 1.4
WebSphere Application Server V5.0 includes support for the JDK 1.4 client container, the first step toward J2EE 1.4 compliance. With JDK 1.4, enterprises can use Java technology to develop more demanding business applications with less effort and in less time. Improved functionality in JDK 1.4 means developers can now spend less time writing custom code to accomplish the same goal as functions that are now part of the core JDK platform. This allows developers to use a single technology to develop, test, and deploy end-to-end enterprise applications and solutions.

More user friendly than ever before
A more functional replacement for the Application Assembly Tool for WebSphere is being made available as a component of the WebSphere Application Server Toolkit. The Assembly Toolkit for WebSphere Application Server, based on the Eclipse Framework (http://www.eclipse.org), works as a natural part of the WebSphere Studio environment. It provides much-improved function, better usability, and a more consistent user interface with the Eclipse toolset. The new Toolkit is an answer to the complex task of packaging J2EE applications. It is an interface that assists users in updating XML for packaging procedures and other value-add tasks that are not directly related to packaging but will help in their deployment.
**Autonomic**
More robust self-management functions are in the WebSphere Application Server 5.0.2.

**Dynamic workload management**
It addresses the aspect of grid computing that monitors and manages application workload. The primary focus of grid computing has been on sharing hardware processor load, such as the heavy numbers crunching of life sciences research. Grid also deals with intelligently sharing application server loads across multiple machines so software runs in parallel, over multiple clusters, thereby allowing transactions to be processed more quickly and efficiently.

Moreover, dynamic workload management dramatically increases application performance and utilization of server resources by acting as an intelligent “traffic cop” within an application server cluster. Previously, application workload running on a cluster of servers was handled with static or fixed weighting algorithms, workload directed to servers based on pre-determined metrics which are based on individual server capacity. With WebSphere, the system can monitor the workload on each server in the cluster and automatically route the application to the one with the lightest workload.

**Significantly enhanced Web services and XML performance**
The new B2B Parser provides extensive XML parsing performance improvements for the Web Services Engine. This feature, combined with other improvements in the Web Services implementation in WebSphere Application Server V5.0.2, provides significant performance advantages. Additionally, Advanced Performance Advisors simplify the administrator’s job by making suggestions on how to set the most critical WebSphere Application Server parameters to maximize performance. Performance Advisors use live data collected from a running system to give tuning advice.

**Enhanced monitoring capabilities**
The new Request Metrics function helps administrators debug and monitor their system by timing individual requests as they travel through WebSphere Application Server components. In addition to the Web container, the enterprise bean container, and the database, WebSphere Application Server 5.0.2 can provide time spent in the Web server.

For more information about new features and enhancements introduced in WebSphere Application Server V5.0.2, please refer to WebSphere Application Server’s home page at:

http://www.ibm.com/software/webservers/appserv/
2.5.2 Supported J2EE APIs in WAS 5.0.2

Table 2-1 shows you the current mainly supported J2EE APIs/Standards in WebSphere Application Server V5.0.2. As mentioned earlier, WebSphere Application Server Version 5 and Version 5.0.2 is a fully J2EE compatible product. It supports all of the J2EE 1.3 APIs and exceeds many with its extensions.

Table 2-1 Supported J2EE APIs in WAS 5.0.2

<table>
<thead>
<tr>
<th>API/Standard</th>
<th>Level</th>
<th>WebSphere Application Server 5.0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>J2EE</td>
<td>1.3</td>
<td>Fully compliant</td>
</tr>
<tr>
<td>EJB</td>
<td>2.0</td>
<td>Support, also support EJB 1.1 for compatibility purpose</td>
</tr>
<tr>
<td>JDK</td>
<td>1.3</td>
<td>Support, also support JDK 1.4 Client Container</td>
</tr>
<tr>
<td>Servlet</td>
<td>2.3</td>
<td>Support</td>
</tr>
<tr>
<td>JSP</td>
<td>1.2</td>
<td>Support</td>
</tr>
<tr>
<td>JTS/JTA</td>
<td>1.0</td>
<td>Support, with distributed transactions</td>
</tr>
<tr>
<td>JMS</td>
<td>1.0.2</td>
<td>Support, with native provider and MQ plug-in</td>
</tr>
<tr>
<td>JDBC</td>
<td>2.0</td>
<td>Support, with 2PC across heterogeneous databases</td>
</tr>
<tr>
<td>JNDI</td>
<td>1.2</td>
<td>Support, with JNDI 1.2 for EJB lookup and CosNaming</td>
</tr>
<tr>
<td>RMI/IIOP</td>
<td>1.0</td>
<td>Support</td>
</tr>
<tr>
<td>JavaMail/JAF</td>
<td>1.2</td>
<td>Support, plus Domino support</td>
</tr>
<tr>
<td>SSL Security</td>
<td>2.0</td>
<td>Support, JSSE and JCE</td>
</tr>
<tr>
<td>XML JAXP</td>
<td>1.0</td>
<td>Support</td>
</tr>
<tr>
<td>J-IDL/CORBA</td>
<td>N/A</td>
<td>Support IIOP 1.2</td>
</tr>
<tr>
<td>J2C</td>
<td>1.0</td>
<td>Support, bean and container managed</td>
</tr>
<tr>
<td>LDAP</td>
<td></td>
<td>Support for SecureWay®, iPlanet, ActiveDirectory</td>
</tr>
<tr>
<td>HTTP</td>
<td>1.1</td>
<td>Support, plus across multiple Web Servers</td>
</tr>
<tr>
<td>SOAP</td>
<td>2.2.2</td>
<td>Support for Web Services</td>
</tr>
<tr>
<td>SOAP-SEC</td>
<td>1.0</td>
<td>Support for Web Services</td>
</tr>
<tr>
<td>JMX</td>
<td>1.0</td>
<td>Support</td>
</tr>
<tr>
<td>XML4J</td>
<td>4.0</td>
<td>Support</td>
</tr>
<tr>
<td>XSL</td>
<td>2.3</td>
<td>Support, with XSL Parser</td>
</tr>
</tbody>
</table>
For further information about WebSphere Application Server, please see WebSphere Application Server Information Center at:

http://publib.boulder.ibm.com/infocenter/wasinfo/index.jsp

Or visit the home page for WebSphere Application Server:

http://www.ibm.com/software/webservers/appserv/
Overview of DB2 UDB V8

In this chapter we introduce the architecture of DB2 UDB V8.1 to provide you with a basic understanding of how DB2 UDB is designed. It is easier to understand what you should think of when there is a need for tuning once you know the DB2 UDB architecture. After a short introduction to the architecture, we show you the key performance-related areas in DB2 UDB. We also introduce some tools that help you to monitor and optimize the database environment.

In this chapter we discuss:

- DB2 UDB product family for Linux, UNIX, and Windows platforms
- DB2 UDB architecture overview
  - Process model
  - Memory model
  - Storage model
  - Key performance related areas
- Tools
3.1 DB2 UDB product family

The DB2 UDB family offers a database server for any application from very small to very large. The DB2 UDB runs on a variety of hardware platforms. It supports both 32-bit and 64-bit hardware including Intel and AMD based systems, IBM PowerPC® server pSeries, iSeries, and zSeries.

IBM's DB2 UDB database software is full-featured, robust, scalable and easy to use. As the market share leader, DB2 UDB provides the foundation of information on demand on Linux, UNIX, Windows, and z/OS platforms (see Figure 3-1). DB2 UDB is specially designed and priced to meet your business needs whether large or small.

3.1.1 DB2 Everyplace®

This database engine is designed to store and process data on mobile and embedded devices. It is a full relational database with a small footprint and a synchronization server.
3.1.2 DB2 Express

This is the newest member of the DB2 UDB product family. The key features include simplified deployment, autonomic management capabilities, application development support and design for 7x24 operation. This flavor of DB2 UDB is aimed at Independent Software Vendors (ISVs) who want to integrate DB2 UDB as part of their applications with low cost and have the capability to expand easily in the future.

3.1.3 DB2 UDB Personal Edition (PE)

This full-function database offering is for single-users and will not accept remote database requests. This offering is available on Windows and Linux. DB2 Personal Edition only has a 32-bit version available.

3.1.4 DB2 UDB Workgroup Server Edition (WSE)

This edition is designed for deployment in a departmental or small business environment that involves a small number of users. The WSE can be deployed in Linux, UNIX and Windows environments with up to four CPUs.

3.1.5 DB2 UDB Enterprise Server Edition (ESE)

ESE is designed to meet the database server needs of midsize to large businesses. The ESE can be deployed in Linux, UNIX, and Windows environments on any size server. The ESE feature set, scalability, reliability, and availability provide the ideal foundation for building data warehouses, transaction processing, or Web-based solutions, as well as a back-end for packaged solutions like ERP, CRM, or SCM.

**Note:** The DB2 UDB Database Partitioning Feature (DPF) is required in order to partition your DB2 UDB ESE database, either within a single server or across multiple servers. The DPF is license-only and does not require any additional products on top of DB2 UDB ESE to be installed on your database server to support database partitioning.

3.2 DB2 architecture overview

Figure 3-2 shows a general overview of the architecture and processes for DB2 UDB V8.
On top of this figure you see the clients connecting to the database using different network protocols. WebSphere also acts as a client. From the view of DB2 UDB, it makes no difference with other DB2 UDB clients when WebSphere requests a connection. With WebSphere, the communication with DB2 UDB is by Java Database Connectivity (JDBC) via a network protocol.

If a client connects to the database he will communicate with a **coordinator agent**. The coordinator agent will be assigned to this application and handles all of the SQL requests. If intra-partition parallelism is enabled, the coordinator agent works with **subagents**. The coordinator agent then has to decide whether to handle the request by itself or to send the request to a number of subagents, depending on the configuration and the number of CPUs.

The **bufferpool** is a piece of memory to where database pages of user table data, index data, and catalog data are temporarily moved from disk storage. DB2 agents read and modify the bufferpool when processing the data requests. If the data is not available in the buffer pool, it needs to be retrieved from the disks into the buffer pool.
Prefetchers retrieve data from disk and move it into the buffer pool before applications need the data. Page cleaners move data from the buffer pool back out to disk. The page cleaners are working in the background to look for no longer needed dirty pages and write them to disk.

All changes to regular data (any data type except BLOBs, Long VARCHAR, and data in Global Temporary tables) and index pages are written to the log buffer. The data in the log buffer is written to disk by the logger. The mechanism of logging the database changes allows you to recover the data to the point of failure when a system crash is encountered.

The deadlock detector checks if a deadlock situation occurred. A deadlock is created when one or more applications are waiting for another application to release a lock on data. Each of the waiting applications is locking data needed by another application. Mutual waiting for the other application to release a lock on held data leads to a deadlock. The deadlock detector will select one of the applications in the deadlock and releases the locks currently held by that “volunteered” application. A negative SQLCODE will be returned to the application containing the “selected” transaction.

### 3.2.1 Process model

DB2 UDB starts various processes to deal with the database tasks. In Figure 3-3 you can find an illustration of the process model DB2 UDB uses.
Figure 3-3   DB2 process model

A single coordinator agent (db2agent) is assigned to a client application when it connects to a database. A coordinator agent works on behalf of an application, and communicates to other agents, using inter-process communication (IPC) or remote communication protocols. Subagents (db2agntp) support the coordinator agents if they are a partitioned database environment or intra-partition parallelism is activated.

DB2 UDB uses a firewall to prevent client applications and implementations of user defined functions (UDF) and stored procedures (SP) from accessing the address space of DB2 UDB. A firewall maintains the integrity of the data in the databases, because it disables application programming errors from overwriting internal buffers or files of the database manager. The firewall also improves reliability, because application errors cannot crash the database manager.

Note: The firewall inside of DB2 acts only for internal purposes and has nothing to do with a firewall that prevents your company from external attacks.
The fenced mode process (\texttt{db2fmp}) is responsible for executing fenced store procedures and UDFs. Fenced code runs in a separate address space behind the firewall.

A \textit{listener process} is between the client and the server processes. This process is responsible for the communication between the client and the server. There are different listener processes to handle different communication protocols.

There can be up to four listeners available in an AIX environment:

- \texttt{db2ipccm}: For local client connections
- \texttt{db2tcpcm}: For TCP/IP connections
- \texttt{db2snacm}: For APPC connections
- \texttt{db2tcpdm}: For TCP/IP discovery tool requests

On the instance level there are several processes and threads like the system controller (\texttt{db2sysc}) that must exist in order for the database server to function. Also, the following threads and processes may be started to carry out various tasks:

- \texttt{db2resyn}: The resync agent that scans the global resync list
- \texttt{db2gds}: The global daemon spawner on UNIX-based systems that starts new processes
- \texttt{db2wdog}: The watchdog on UNIX-based systems that handles abnormal terminations
- \texttt{db2fcmdm}: The fast communications manager daemon for handling inter-partition communication (used only in multi-partitioned databases)
- \texttt{db2pdbc}: The parallel system controller that handles parallel requests from remote nodes (used only in a partitioned database environment)
- \texttt{db2cart}: For archiving log files when accessing a database configured with USEREXIT enabled
- \texttt{db2fmtlg}: For formatting log files, when accessing a database configured with LOGRETAINT enabled, but with USEREXIT disabled
- \texttt{db2panic}: The panic agent that handles urgent requests after agent limits have been reached at a particular node (used only in a partitioned database environment)
- \texttt{dlasync}: A monitor for the Data Links servers, if you have configured DB2 UDB for datalinks

The following list includes some of the important threads/processes used by each database:

- \texttt{db2pfchr}: For buffer pool prefetchers.
- \texttt{db2pclnr}: For buffer pool page cleaners.
- db2loggr: For manipulating log files to handle transaction processing and recovery.
- db2loggw: For writing log records to the log files.
- db2logts: For collecting historical information about which logs are active when a tablespace is modified. This information is ultimately recorded in the DB2TSCHG.HIS file in the database directory. It is used to speed up tablespace rollforward recovery.
- db2dlock: For deadlock detection. In a multi-partitioned database environment, an additional process called db2glock is used to coordinate the information gathered from the db2dlock process on each partition. db2glock runs only on the catalog partition.

3.2.2 Memory model

DB2 UDB has four different types of memory, as shown in Figure 3-4.
The *Database global manager shared memory* is allocated when the database manager is started (*db2start*). The memory remains allocated until the database manager is stopped (*db2stop*). This area contains information that the database manager uses to manage activity across all database connections.

*Database Shared Memory* (also known as *Database Global Memory*) is allocated when a database is activated or connected to for the first time. This memory is used across all applications that might connect to the database. Database Shared Memory can be controlled by the `DATABASE_MEMORY` configuration parameter. By default, this parameter is set to automatic so that DB2 calculates the amount of memory allocated for the database. With the `NUMDB` parameter you can control the maximum count of concurrent active databases. Each database has its own global memory that consumes the biggest part of memory.

Many different memory areas are contained in database shared memory, as shown in Figure 3-5, including:

- Buffer pools
- Lock list
- Database heap—and this includes the log buffer
- Utility heap
- Package cache
- Catalog cache

The memory for an agent becomes allocated when the agent is created. Agent private memory is allocated for the agent and contains memory allocations that are used only by this specific agent, such as the sort heap and the application heap. When a database is already in use by one application, only agent private memory and application global shared memory is allocated for subsequent connecting applications. With the `MAXAPPLS` parameter you specify the maximum number of applications that can simultaneously connect to a single database. This parameter is part of the database configuration and can be set to different values for different databases. The more applications are allowed to connect concurrently to a database the more memory is needed.

The database manager parameters `MAXAGENTS` and `MAX_COORDAGENTS` are not shown in the figure. They limit the number of database manager agents that can exist simultaneously across all active databases in an instance. Together with `MAXAPPLS`, these parameters limit the amount of memory allocated for agent private memory and application global memory.

Figure 3-5 shows what the database manager shared memory and the other portions of memory are used for.
3.2.3 Storage model

Figure 3-6 illustrates how the data is physically stored on disks.
A *tablespace* is a logical layer between the database and data stored in it. By default each database has three table spaces.

- **SYSCATSPACE**: Stores the internal database control tables called catalog tables
- **USERSPACE1**: Stores the user-defined tables
- **TEMPSPACE1**: Used to store temporary tables for operations, such as sorts and reorganizations

There are two different kinds of table spaces called *System Managed Space (SMS)* and *Database Managed Space (DMS)*.

- **SMS**
  
  In a SMS tablespace the operating system is responsible for allocating and managing the space where data is to be stored. SMS is the default table space setting and, in general, easier to administrate.

- **DMS**
  
  The containers for a DMS table space are raw devices or file system files with pre-defined sizes, that is, the disk space is pre-allocated when a DMS table space is defined. Usually, a DMS table space performs better than an SMS.
table space since it does not have to spend time extending the files for the data and the indexes when new rows are inserted.

We recommend using SMS if you cannot estimate the size and the growth of data. Temporary tables with data volume varies from small to large is a good candidate of using SMS table space.

A container is a physical storage device. It can be identified by a directory name (SMS), a file name (DMS), or a raw device (DMS). A table space consists of several containers, as you can see from Figure 3-6.

An extent is a unit of space in a container and contains a number of pages; the default is 32.

A page has a size of 4 KB, 8 KB, 16 KB or 32 KB. You have to define the page size when you create a table space. There is no possibility to change the page size of a table space, once created. The maximum amount of rows in page is 254. Therefore it is necessary to determine the maximum row size of the tables you want to store in the table space to determine what page size fits it best.

### 3.2.4 Key performance-related areas

In this section we provide a deeper view of the performance-related aspects of the DB2 UDB architecture. This will prepare you for Chapter 7, “Monitoring and tuning of DB2 UDB V8” on page 237, where we show you how to tune the database server.

**Buffer pool**

When an application accesses a row of a table for the first time, the database manager places the page containing that row in the buffer pool. The next time any application requests data, the database manager first looks for the buffer pool. If the requested data is in the buffer pool, it can be retrieved without disk access, resulting in faster performance.

The buffer pool is a very important area for data accessing performance. It is a good practice to have as much data as possible that the application frequently needs in the buffer pool. However, allocating excessive amounts of memory for the buffer pool for an application could impact the system performance. In Chapter 7, “Monitoring and tuning of DB2 UDB V8” on page 237, we show you how to monitor and tune the buffer pool to achieve your performance goal.

A buffer pool is assigned to one or many table spaces. You may define more than one buffer pool in a database. If you have a table with data that is constantly used, it may be useful to place this table in a table space with its own buffer pool to make the data memory resident.
With DMS table spaces, the table data, indexes, and long data can be placed in separate table spaces. This makes it possible to have a separate buffer pool for an index to keep the index memory resident and improve the performance.

By default the prefetcher reads contiguous pages from disk and may write them into non-contiguous pages in the buffer pool. With DB2 V8, you can configure a block-based buffer pool. When this feature is activated (during creation of a buffer pool or with the ALTER buffer pool command) the contiguous pages from disk are written into contiguous pages of the buffer pool, when available.

It is recommended to use a separate buffer pool for temporary table spaces. This increases performance for queries that require temporary storage, especially sort-intensive queries.

**Note:** The default buffer pool size is very small and therefore there is a need to tune. In UNIX systems the default size is 1000 4-K pages; in Windows systems it is 250 4-K pages.

**Asynchronous read/write**

I/O is a complex part and also very important for performance. As described earlier, it is good to have prefachers running. These agents are responsible for accessing data before an application needs it. Processes known as I/O servers handle the prefetching. The number of processes can be configured using the configuration parameter `NUM_IOSERVERS`. It is better to configure more I/O servers than needed than not have enough of them. The impact of having excessive I/O servers is almost none because the memory used for the I/O servers is paged out. However, the performance can be dramatically decreased if I/O servers are not enough.

Figure 3-7 shows the steps of how DB2 UDB uses the I/O server and prefetching process. It illustrates the process starting from a client request for data pages until DB2 brings the data from the database server back to the client.
Figure 3-7  Prefetching data using I/O servers

Below we explain the figure:

- (1) The user application passes the SQL request to the database agent that has been assigned to the user application by the database manager.

- (2), (3) The database agent determines that prefetching should be used to obtain the data required to satisfy the SQL request and writes a prefetch request to the I/O server queue.

- (4), (5) The first available I/O server reads the prefetch request from the queue and then reads the data from the table space into the buffer pool. The number of I/O servers that can fetch data from a table space at the same time depends on the number of prefetch requests in the queue and the number of I/O servers configured by the NUM_IOSERVERS database configuration parameter.
(6) The database agent performs the necessary operations on the data pages in the buffer pool and returns the result to the user application.

**Tablespaces and container**

A tablespace can have several containers. The data will be distributed in a round-robin technique across the available containers. Figure 3-8 shows how the extents are allocated. Extent 0 is allocated in the first container, extent 1 in the second, and so forth.

![Distribution of extents in container](image)

**Figure 3-8  Distribution of extents in container**

DB2 V8 introduces some new container management features. It is now allowed to alter the size of container. Another new feature is that you can turn off rebalancing while adding a container. In the previous version the container rebalance process is automatic. This process can impact overall system performance if the table space is big. For detailed information about how to add or extended the containers, please reference Chapter 5 of the *IBM DB2 UDB Administration Guide: Planning*, SC09-4822.

It is important to distribute the container over several disks to allow the database manager to do parallel I/O. If you are using striped devices like RAID devices, it is recommended to use only one container per tablespace. The size of an extent should be $n \times \text{the RAID stripe size}$ with $n >= 1$. The prefetch size should be $n \times \text{extent size}$ and $n \times \text{RAID stripe size} \times \text{count of devices}$, with $n >= 1$.

**Note:** DB2 cannot determine if a RAID device is used. It is important to set the DB2_PARALLEL_IO registry variable to enable parallel I/O. DB2_PARALLEL_IO=“* enables parallel I/O for all tablespaces.
**Database agents**

Agents are processes and they need system resources. For a good performance it is necessary to handle database agents economically with system resources. The number of available agents depends on the database manager configuration parameters MAXAGENTS and NUM_POOLAGENTS.

**Connection concentrator**

For Internet applications with many relatively transient connections, or similar kinds of applications, the connection concentrator improves performance by allowing many more client connections to be processed efficiently. It also reduces memory use for each connection and decreases the number of context switches.

The connection concentrator is new in DB2 UDB V8 and works like a transaction monitor. Therefore there is no more need to buy a commercial product to reduce the number of connections to the database. It is now possible to handle thousands of concurrent connections with hundreds of agents.

**Note:** The connection concentrator is enabled when the value of the database manager parameter MAX_CONNECTIONS is greater than the value of the DBM parameter MAX_COORDAGENTS.

Figure 3-9 shows the concept of the connection concentrator. The left side of the picture shows that without the connection concentrator, the number of coordinator agents is as many as the number of client connections. This consumes a lot of system resources if there are many connections active on a database server. When the connection concentrator is activated, the number of coordinator agents decreases because the coordinator agents are within a pool and accessible for different connections. When the number of coordinator agents shrinks, the number of subagents also reduces, as shown in right side of the figure.

For more details see Chapter 7, “Monitoring and tuning of DB2 UDB V8” on page 237, where the parameter will be discussed in more detail.
Concurrency

Concurrency can be a reason for an application with a poor performance. DB2 UDB puts locks on data used by an application according to the isolation level. How restrictive these locks are depends on the isolation level. The isolation level can be set for an application and depends on the kind and the needs of the application. These levels are from restrictive to non-restrictive:

- **Repeatable Read (RR)**
  RR is the highest isolation level. Locks are held on all referenced rows of a result set within a unit of work. RR guarantees that no changes can be made on the referenced data till the unit of work terminates. For example, if you scan 10,000 rows and apply predicates to them, locks are held on all 10,000 rows, even though only 10 rows qualify.

- **Read Stability (RS)**
  RS locks only the rows that an application retrieves in a unit of work. The Read Stability isolation level ensures that all returned data remains unchanged until the time the application sees the data, even when temporary tables or row blocking is used. Other applications can make changes to other parts, so if the cursor is reponed the result may be different.

- **Cursor Stability (CS)**
  CS locks any row accessed by a transaction of an application while the cursor is positioned on the row. This lock remains in effect until the next row is fetched or the transaction is terminated. However, if any data on a row is changed, the lock must be held until the change is committed to the database.
Uncommitted Read (UR)

UR allows applications to retrieve uncommitted data. So it may occur that the applications see changes or inserts that will become undone.

Table 3-1 is a summary of the isolation levels.

<table>
<thead>
<tr>
<th>Isolation level</th>
<th>Access to uncommitted data</th>
<th>Nonrepeatable read</th>
<th>Phantom read phenomenon</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
</tr>
<tr>
<td>RS</td>
<td>Not possible</td>
<td>Not possible</td>
<td>possible</td>
</tr>
<tr>
<td>CS</td>
<td>Not possible</td>
<td>possible</td>
<td>possible</td>
</tr>
<tr>
<td>UR</td>
<td>possible</td>
<td>possible</td>
<td>possible</td>
</tr>
</tbody>
</table>

The impacts of the isolation level on the performance that may be seen in an application are deadlocks and lock timeouts/lock waits. How to set the Isolation level is discussed in Chapter 7, “Monitoring and tuning of DB2 UDB V8” on page 237 when we talk about application tuning. We will also discuss how to prevent from lock escalation.

In Table 3-2 we introduce the different locks DB2 UDB uses.

<table>
<thead>
<tr>
<th>Lock mode</th>
<th>Applicable object type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN (Intent None)</td>
<td>Table spaces, tables</td>
<td>The lock owner can read any data in the table, including uncommitted data, but cannot update any of it. No row locks are acquired by the lock owner. Other concurrent applications can read or update the table.</td>
</tr>
</tbody>
</table>
### Lock mode

<table>
<thead>
<tr>
<th>Lock mode</th>
<th>Applicable object type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS (Intent Share)</td>
<td>Table spaces, tables</td>
<td>The lock owner can read data in the locked table, but not update this data. When an application holds the IS table lock, the application acquires an S or NS lock on each row read. In either case, other applications can read or update the table.</td>
</tr>
<tr>
<td>NS (Next Key Share)</td>
<td>Rows</td>
<td>The lock owner and all concurrent applications can read, but not update, the locked row. This lock is acquired on rows of a table, instead of an S lock, where the isolation level is either RS or CS on data that is read.</td>
</tr>
<tr>
<td>S (Share)</td>
<td>Rows, tables</td>
<td>The lock owner and all concurrent applications can read, but not update, the locked data. Individual rows of a table can be S locked. If a table is S locked, no row locks are necessary.</td>
</tr>
<tr>
<td>IX (Intent Exclusive)</td>
<td>Table spaces, tables</td>
<td>The lock owner and concurrent applications can read and update data in the table. When the lock owner reads data, an S, NS, X, or U lock is acquired on each row read. An X lock is also acquired on each row that the lock owner updates. Other concurrent applications can both read and update the table.</td>
</tr>
<tr>
<td>Lock mode</td>
<td>Applicable object type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SIX (Share with Intent Exclusive)</td>
<td>Tables</td>
<td>The lock owner can read and update data in the table. The lock owner acquires X locks on the rows it updates, but acquires no locks on rows that it reads. Other concurrent applications can read the table.</td>
</tr>
<tr>
<td>U (Update)</td>
<td>Rows, Tables</td>
<td>The lock owner can update data in the locked row or table. The lock owner acquires X locks on the rows before it updates the rows. Other units of work can read the data in the locked row or table, but cannot attempt to update it.</td>
</tr>
<tr>
<td>NX (Next Key Exclusive)</td>
<td>Rows</td>
<td>The lock owner can read but not update the locked row. This mode is similar to an X lock except that it is compatible with the NS lock.</td>
</tr>
<tr>
<td>NW (Next Key Weak Exclusive)</td>
<td>Rows</td>
<td>This lock is acquired on the next row when a row is inserted into the index of a non-catalog table. The lock owner can read but not update the locked row. This mode is similar to X and NX locks except that it is compatible with the W and NS locks.</td>
</tr>
</tbody>
</table>
In the following we list some concepts with high performance issues and some flycatchers introduced in version 8.

**Data types**

Understanding your data and defining the proper data type to store the data is another element affects the performance. Improper data type design in an application could result in bad performance. For example, it is not a good practice to use VARCHAR whenever a string needs to be stored. Numbers should be stored in number fields and not in character fields. In Chapter 7, “Monitoring and
tuning of DB2 UDB V8” on page 237, we give some advice on how to choose the correct data types.

**String data**
This section describes the three string data types CHAR, VARCHAR and LONG VARCHAR. Each character string is further defined as one of:

- **Bit data**: Data that is not associated with a code page
- **Single-byte character set (SBCS) data**: Data in which every character is represented by a single byte
- **Mixed data**: Data that may contain a mixture of characters from a single-byte character set and a multi-byte character set (MBCS)

When defining the length of a string data type, it is important to keep in mind that the length represents the number of bytes and not the number of characters. If you have a multi-byte code set like Unicode there is a need to reserve more space.

- CHAR(n) specifies a fixed-length column for character string data. The maximum length is 254 bytes.
- VARCHAR(n) specifies a varying-length column for character string data. The maximum length of the string is 4000 bytes. If the length is greater than 254, the column is a long-string column.
- LONG VARCHAR specifies a varying-length column for character string data. The maximum length of a column of this type is 32700 bytes.

**Numeric data**
There are six data types that can be used to store numeric values. The data types are used to store different numeric types and precision. The data is stored using a fixed amount of storage for all numeric data types. The amount of storage required increases as the precision of the number goes up.

- Small integer (SMALLINT) specifies a small integer. Values in a column of this type can range from -32768 through +32767.
- Large integer (INTEGER) specifies a large integer. Values in a column of this type can range from -2147483648 through +2147483647.
- Big integer (BIGINT) is available to store 64-bit integers and can range from -9,223,372,036,854,775,808 to +9,223,372,036,775,807. As platforms include native support for 64-bit integers, the processing is much faster than with decimal, and more precise than double or real.
- Single-precision floating-point (REAL) is a 32-bit approximation of a real number. The number can be zero or can range from -3.402E+38 to -1.175E-37, or from 1.175E-37 to 3.402E+38.
Double-precision floating-point (DOUBLE or FLOAT) specifies a floating-point number that is 64 bits long. Values in a column of this type can range from -1.79769E+308 to -2.225E-307 or +2.225E-307 to +1.79769E+308, or they can be 0.

Decimal (DECIMAL or NUMERIC) specifies a mainframe packed decimal number with an implicit decimal point. The position of the decimal point is determined by the precision and scale of the number. The scale, which is the numbers to the right of the decimal point, cannot be negative or greater than the precision. The maximum precision is 31 digits. Note that numbers that require decimal precision greater than 15 digits may be subject to rounding and conversion errors.

Dates, times, and timestamps
There are three DB2 data types to represent dates and times:

- DATE specifies date values in various formats, as determined by the country code of the database.
- TIME specifies time values in a three-part format. The values range from 0 to 24 for hours (hh) and from 0 to 59 for minutes (mm) and seconds (ss).
- TIMESTAMP combines a date and time and adds an optional microsecond to make a seven-part value of the format yyyy-mm-dd-hh.mm.ss[.nnnnnn].

Index
Indexes are used to speed up the retrieval of data. An index usually takes less space than a table and has a tree structure to reduce the look-up time. But indexes also need space from your storage. What indexes need to be created depends on the SQL the application uses. We introduce a tool later in this chapter that helps you to determine which indexes may be useful.

Type-2 indexes
DB2 Version 8 adds support for type-2 indexes. Here are the main advantages of type-2 indexes:

- They improve concurrency because the next_key locking is reduced to a minimum. Most next-key locking is eliminated by marking the key as having been deleted instead of physically removed the key from the index page.
- An index can be created on columns that have a length greater than 255 bytes.
- In-place table reorg and online table load can be used against a table that has only type-2 indexes defined on it.
- They are required for the new multidimensional clustering (MDC) facility.
All indexes created before DB2 Version 8 are type-1 indexes. Use the REORG INDEXES command to convert type-1 indexes to type-2 indexes. Use the INSPECT command to ascertain the type of index defined on a table. After this conversion, runstats should be performed.

**Stored procedures (SPs)**

Store procedures (SPs) can help reducing the network traffic. A SP can contain several SQL commands and also some business logic. SPs are stored in the database server. Calling a SP takes only one call command and the result is sent back to the application after executing the SQLs in SP at the host. If you do all the SQL commands within your application, the communication between your application and the database is much higher.

**Multidimensional clustering (MDC)**

Multidimensional clustering (MDC) is a new feature of DB2 UDB V8. Multidimensional clustering enables a table to be physically clustered on more than one key or dimension simultaneously. Prior to Version 8, DB2 UDB only supports single-dimensional clustering of data via clustering indexes. Using a clustering index, DB2 UDB attempts to maintain the physical order of data on pages in the key order of the index as records are inserted and updated in the table. Clustering indexes greatly improve the performance of range queries that have predicates containing the key (or keys) of the clustering index, as, with good clustering, only a portion of the table needs to be accessed, and, when the pages are sequential, more efficient prefetching can be performed.

With MDC, these benefits are extended to more than one dimension or clustering key. In the case of query performance, range queries involving any, or any combination of, specified dimensions of the table will benefit from clustering. Not only will these queries access only those pages having records with the correct dimension values, these qualifying pages will be grouped by extents. Furthermore, although a table with a clustering index can become un-clustered over time as space fills up in the table, an MDC table is able to maintain its clustering over all dimensions automatically and continuously, thus eliminating the need to reorganize the table in order to restore the physical order of the data.

**Declared temporary tables**

A declared temporary table is defined by the DECLARED GLOBAL TEMPORARY TABLE statement. Declared temporary tables are used to store
data that does not need to be persistent. A declared temporary table is a temporary table that is only accessible to SQL statements that are issued by the application that created the temporary table. A declared temporary table does not persist beyond the duration of the connection of the application to the database. The creation of a temporary table is not stored in the catalog. In comparison to regular tables, DB2 does not lock declared temporary tables or their rows, and, if you specify the NOT LOGGED parameter when you create it, does not log declared temporary tables or their contents.

In DB2 UDB V8, a new feature is added to the declared temporary tables that allows the creation of index on temporary tables. Prior to version 8, the temporary table needed to be scanned completely, which is very time consuming if the table is big. With indexes, retrieving data from a temporary table is much faster. For a better optimization it is now also possible to update the statistic of temporary tables with runstats.

**Maintenance**
Performance tuning is a constant task. There are maintenance tasks an administrator can do regularly to keep good performance.

**Runstats**
DB2 UDB has an optimizer that looks for the best way to retrieve data. The optimizer uses statistical data to calculate the lowest price for retrieval. With the runstats command the statistical data will be collected and stored in the catalog. So the optimizer knows, for example, how many rows a table has, and can determine if an index scan is useful or just use a table scan.

The statistic data must be updated from time to time. If you have tables with many changes, you should run `runstats` at regular intervals.

**Reorg**
In tables where data are frequently inserted and deleted, the table space becomes more and more fragmented. This has an impact on the performance. To defragment the data, DB2 UDB offers a `reorgchk` command to determine if a table requires re-organization, and `reorg` command to organize the data.

**Rebind**
When using static SQL, the best access plan is determined during the bind time and saved as a package. If your data changes and you run a `runstats`, the static code that was bound to the server will not pick up the new statistic. Therefore it is necessary to do a rebind to force a new calculation of the access plan.
**Application design**

Keep in mind that the tuning of the server is important, but the best-tuned server is slow if the database design or the application is badly designed.

The physical and logical designs of the database and the application are also critical for performance. The database needs to become normalized to eliminate redundant data. But for performance it is sometime useful to not have normalized data. We discus this in Chapter 7, “Monitoring and tuning of DB2 UDB V8” on page 237.

### 3.3 Tools

DB2 UDB provides some tools that help you in several tasks, like performance optimization and maintenances tasks. In the following we introduce four of them.

#### 3.3.1 IBM DB2 UDB Performance Expert for Multiplatforms

IBM DB2 UDB Performance Expert for Multiplatforms offers a comprehensive view that consolidates, reports, analyzes and recommends changes on DB2 UDB performance-related information. The tool includes a Performance Warehouse that stores performance data and analysis tools as well as a Buffer Pool Analyzer that collects data and provides reports on related event activity. With online monitoring capabilities, DB2 UDB Performance Expert monitors DB2 Connect gateways, including application and system-related information. Building on IBM’s autonomic computing and on demand expertise, DB2 UDB Performance Expert also offers recommendations for system tuning to gain optimum throughput. You can use DB2 UDB Performance Expert to:

- Monitor the DB2 UDB database and DB2 applications online and by evaluating report sets.
- View and examine the status of a DB2 instance and its applications while they are currently active, or investigate events and performance problems that happened in the past.
- Monitor individual nodes or an entire system.
- Recognize event exceptions and take appropriate actions by means of a user exit.
- Obtain tuning recommendations.
- Identify trends and anticipate potential bottlenecks.
3.3.2 IBM DB2 UDB Recovery Expert for Multiplatforms

DB2 UDB Recovery Expert for Multiplatforms provides targeted, flexible and automated recovery of database assets. Its easy-to-use environment enables even less experienced DBAs to successfully complete highly sophisticated and efficient recovery techniques in minimal time. DB2 UDB Recovery Expert provides intelligent analysis and diagnostics of altered, incorrect, or missing database assets including tables, indexes, or data. It also automates the process of rebuilding those assets to a correct “point-in-time,” often without taking the database or e-business operations offline.

3.3.3 IBM DB2 High Performance Unload for Multiplatforms

An IBM DB2 UDB tool that helps reduce the maintenance window is DB2 UDB High Performance Unload for Multiplatforms. This product gives customers a fast and efficient tool for unloading and extracting data for movement across the enterprise or for reorganization in place. It delivers high levels of parallelism and distribution of data across multiple logical or physical partitions. It also supports DB2 UDB Enterprise Edition/Enterprise Server Edition (SMP) and data that has been distributed across multiple logical or physical partitions on an MPP system, such as DB2 Enterprise-Extended Edition/Database Partitioning Feature (PDF).

3.3.4 IBM DB2 UDB Table Editor for Multiplatforms

IBM DB2 UDB Table Editor for Multiplatforms quickly and easily accesses, updates, and deletes data across multiple DB2 database platforms, including IBM Informix Dynamic Server 9.x. Through integration with the Control Center, or Web-based interfaces, the tool supports high function editing with full support for DB2 security and user profiles. In addition to editing raw data, you can create editing forms with drag-and-drop graphical user interfaces, giving end users the ability to perform more of the data maintenance activities, therefore freeing up DBA resources.

3.3.5 IBM DB2 UDB Web Query Tool for Multiplatforms

DB2 Web Query Tool for Multiplatforms makes it easy for users at all levels to access enterprise data for query, data comparison, or reporting through multiple interfaces. These interfaces can include traditional Web-based browsers, e-mail clients, and wireless devices such as PDAs, text pagers, and WAP-enabled cellular phones.
This chapter discusses some key WAS components and the topology selection in the WebSphere Application Server and DB2 UDB integrated environment that implements the multi-tier application model, and how WebSphere Application server interacts with DB2 UDB.

The topics covered include:

- Session management
- Enterprise JavaBeans
- Topology section
- J2EE Connector architecture
- DB2 JDBC drivers
- DB2 JDBC providers
- DB2 UDB connectivity with DB2 Connect
4.1 WebSphere session manager

The session manager is part of each Web container, and is responsible for managing HTTP sessions, providing storage for session data, allocating session IDs, and tracking the session ID associated with each client request through the use of cookies, URL rewriting, or SSL session identifier techniques. The session manager allows the WebSphere Application Server administrator to dynamically configure and tune the behavior of all HTTP sessions created by servlets within its application server.

From an application development perspective, servlet and JSP code do not interact directly with the session manager object. Rather, the session manager supports the HttpSession interface, which developers use for session functionality.

All the servlet or JSP developer has to do is create the session and put and get data. This allows the application developer to focus on business logic, and ensures consistent behavior across all of the servlets called by its servlet engine.

4.1.1 HttpSession interface

The Java servlet specification contains the interface javax.servlet.http.HttpSession, which the WebSphere Application Server servlet engine (Web container) supports. HttpSession provides Application Program Interfaces (APIs) that handle many of the details of session access and management.

An HttpSession gets created by calling the javax.servlet.http.HttpServletRequest.getSession() method on the servlet’s request object. If a session does not yet exist, this call creates one.

Some of these APIs have been deprecated with the Java Servlet Specification 2.2. The putValue() and getValue() methods, for example, have been replaced by putAttribute() and getAttribute(), respectively, although the Java Servlet Specification 2.1 methods are still supported.

4.2 Enterprise JavaBeans

In this section we provide a brief overview of EJBs so that we can better understand the methodology of the performance tuning of a WebSphere Application Server and DB2 integrated system and how WebSphere applications interact with DB2.
4.2.1 EJB overview

Enterprise JavaBeans (EJB) is a server-side component architecture for the development and deployment of component-based distributed business applications.

The EJB component model simplifies development of business components that are transactional, scalable, and portable. Enterprise JavaBean servers reduce the complexity of developing business components by providing automatic support for system-level services, such as transactions, security, and database connectivity, thus allowing the developers to concentrate on developing the business logic.

The EJB specification defines a standard, so that different vendors are able to implement these standards. Because this standard defines every essential detail of the architecture, an application written using the Enterprise JavaBeans architecture is scalable, transactional, and multi-user secure. Such an application may be written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification.

Figure 4-1 on page 78 gives an overview for a basic EJB environment:

- The EJB components are running inside the container of an EJB server.
- The container has the connection to the database or to other components.
- An EJB client can access the EJBs from the same Java Virtual Machine (JVM) or from another JVM over remote interfaces. The EJB home component is comparable to a factory for the EJB objects. The EJB objects retrieved from the home components can be also local or remote objects.
EJB 2.0 overview
In this section we have a short look at the new and changed functions in EJB 2.0.

Local interfaces
An EJB client never directly interacts with an EJB object. The client uses the component interface, which defines the methods that are available to the client. The implementation of this interface is provided by the container.

Without having the new local interfaces, the container has to provide only one implementation of this component interface—an implementation for invocations over a network protocol, because the EJB client has to be a remote client. If the EJB client and the EJB object itself are in the same Java Virtual Machine (JVM), an avoidable overhead arises because of the communication layers.

To avoid this overhead, one type of component interface is being added to the EJB specification 2.0: The local component interface.

There are now two types of component interfaces: A local and a remote interface. We can now choose the type that suits our needs.

Container-managed persistence (CMP)
An EJB container and an EJB have to interact with each other. In the case of an CMP entity bean, this interaction is quite complex. This communication is called
a contract.

The EJB specification 2.0 establishes new contracts for CMP. These new contracts are fundamentals for the new functionality and they support more efficient vendor implementations.

The new contracts themselves are transparent to an EJB programmer. From a programmer's point of view, there are only some changes in the implementations. One change that may catch the attention of a programmer is that the bean class is now an abstract class without fields. The getters and setters of the bean class describe the attributes of the bean. It is then the job of the container to generate a concrete class for this bean and to implement the fields and the relationships between entity beans.

The two new primary CMP functions are relationships and EJB QL.

**Container-managed relationships (CMR)**

The main principles of CMR and CMP are comparable. In CMP, we describe our container-managed fields and the container is responsible for persistence. In CMR, we describe the relations between our entity beans and the container is responsible for maintaining the referential integrity.

What a container does in different situations is clearly described in the specification. For example, if we create a one-to-many relationship between bean A and bean B and invoke a set method on instance A2 passing the instance B1, the container has to remove the previous relationship between A1 and B1 because of the 1:n relationship definition. B1 can only belong to one instance of bean A. The container has to remove the prior relationship in the same transaction context to maintain the referential integrity.

The descriptions of the relationships are based on the abstract persistence schema. The abstract schema and the description of the relationships are part of the deployment descriptor.

The members of a container-managed relationship must have a local interface. The container-managed relationships support one-to-many (1:m), one-to-one (1:1), and many-to-many (m:m) relationships.

**EJB query language**

The finder methods in the home interfaces are responsible for locating particular entity objects, but the name and the declaration of the methods are not sufficient information for the container to generate an implementation of the finder method.

Therefore, the bean provider has to provide a description of the finder method. The EJB Architecture Version 1.1 does not specify the format of the finder
method description. So every vendor (that is, every container provider) must offer a way to do this.

There are some obvious drawbacks with this approach. Every container provider offers a different way to describe and implement the finder methods, but all the different implementations have a big drawback in common. Since the EJB 1.1 beans have no object schema, they have to implement and describe the finder methods based on the data schema.

The EJB Query Language (EJB QL) enables us to describe the finder methods based on the object schema in an independent manner. Now our finder methods are supported by every EJB 2.0-compliant container. We do not have to customize our finder methods if we are using another database or another EJB container.

The EJB QL uses an SQL-like syntax to select objects or values. It is a typed expression language.

**EJB types**

There are three types of EJBs: Entity, session, and message-driven beans, as shown in Figure 4-2 on page 83. In this section, we explore their structure, behavior, and container life-cycle relationship in more detail. Additionally, when a particular bean type might be an appropriate choice in the design, an expanded client view is also discussed. Finally, we conclude with an example application scenario that helps us to classify the usage of a particular bean type.

**Session beans**

Session beans are often considered to be extensions or agents of the client, and perform tasks on behalf of the client. Only one client may be processing on a given session bean at a given time, so there is no concurrency of session beans. They are modeled to encapsulate process or workflow-like behavior, such as transferring funds between accounts. Although some session beans may maintain state data, this data is not persistent in the sense that an entity bean represents persistent data.

There are two sub-types of session EJBs—stateless session EJBs, and stateful session EJBs as follows:

- **Stateless session EJB**

  These represent a set of related behaviors (methods) that do not retain client-specific states between invocations. Contrary to popular belief, stateless session beans can, in fact, possess instance variables, but those variables must be shared among multiple potential clients (for example, they should be read-only). This often overlooked fact can be key to understanding some potential uses of stateless session EJBs.
For readers familiar with traditional transaction-processing systems like CICS or Encina®, you can think of each method call to a stateless session EJB as an individual non-conversational transaction.

- **Stateful session EJB**

  Each stateful session EJB is “owned” by a single client, and is uniquely connected to that client. As a result of this, stateful session EJBs may retain client state across method invocations. That is to say, a client may call a method that sets a variable value in one method, and then be assured that another, later, method invocation to retrieve that value will retrieve the same value.

**Entity beans**

Entity beans are typically modeled to represent domain objects, that is, data entities that are stored in a permanent, persistent data store such as a database, and the behavior that can be performed on that data. This is sometimes referred to as objectifying the data, and the attributes of the entity object are mapped to rows in one or more database tables. Some examples of entities might be accounts or customers. Each entity is uniquely identified by its primary key. Since data in a database may be shared by multiple users, so may entity beans. Managing the concurrency of the entity bean is one of the responsibilities of the container.

Depending on the way the persistence is managed, there are two kinds of entity beans: Container-managed persistence (CMP) and bean-managed persistence (BMP) beans. Persistence can be defined as the data access protocol for transferring the state of the object between the entity bean and the underlying data source.

- **CMP beans**

  CMP means that the EJB container handles all database access required by the entity bean. The bean's code contains no database access (SQL) calls. As a result, the bean's code is not tied to a specific persistent storage mechanism (database). This provides independence from the underlying database implementations, and the same entity EJB can be deployed on different J2EE servers that use different databases, without modifying or recompiling the entity EJB code. In other words, your entity beans are more portable.

  The EJB 2.0 specification addresses some of the known limitations in EJB 1.1 by providing a common persistence model that CMPs are developed on. This persistent model is intended to transcend the product lines and make it standard for all EJB vendors. Specifically, the specification provides:

  - An abstract persistence schema—This enables CMP mappings to be done in an abstract way for all vendor tools.
– **EJB Q L**—A standardized query language for finding and locating beans. Although it is not actually SQL, it is an SQL-like language that supports a subset of SQL functions.

– **Container-managed relationships (CMR)**—Standardize how beans are related to other beans, and supports the relationship types of one-to-one, one-to-many, and many-to-many. This is actually included as a part of the abstract persistence schema.

WebSphere Studio Application Developer and VisualAge® for Java have long provided support within the tool set for building schemas and managing relationships of beans. Application Developer Version 5 provides updated tools for the building of beans on this new, abstract model.

- **BMP beans**

  Entity beans that manage their own persistence are called bean-managed persistence (BMP) entity beans.

  With a BMP entity bean, the EJB developer manages the persistent state of the bean by coding database calls, or any type of access to permanent storage, directly into the bean class. This puts the responsibility on the developer to properly manage the persistence of the bean. To do so properly requires understanding how callback methods and other bean life-cycle methods are invoked by the container as part of its persistence service, as is done automatically in CMPs, then emulating that behavior yourself in your own bean.

  It is the developer's responsibility to save and restore the state of the bean when called by the container through the ejbLoad and ejbStore methods—these are the callback methods for the bean type; and to create, find, and/or remove beans through ejbCreate and ejbRemove methods—these are the life-cycle methods of the bean. Most of the time, the developer of BMPs uses JDBC for coding the persistence logic directly into these methods; however, other techniques can also be used, such as SQLJ or CICS transactions.

  Although most of the new features of EJB 2.0 were for CMPs, some of the newer features are available for BMPs as well. Specifically, BMPs also may define new structures of query methods (home and select methods), although developers must code the data source logic themselves. The new EJB query language is not available for BMPs.

**Message-driven beans**

This EJB type is added in the EJB 2.0 specification. A message-driven bean is an asynchronous message consumer, implemented as a Java Message Service (JMS) consumer. Message-driven beans are similar to session beans. They may also represent business process-like behavior, but are invoked asynchronously.
They typically represent integration points for other applications that need to work with the EJB.

**Access intent**

When using CMP EJBs, the SQL and the interface to the database are usually transparent to the programmer. However, there are ways of controlling the database lock isolation level of the CMP auto-generated SQL.

- **EJB 1.1 module**

  The database lock isolation level is indirectly selected based on the isolation level defined for the transaction attributes in the EJB deployment descriptor. This can be set at the method level by the development tool when you build the `.ear` file. This capability has been removed from the EJB 2.0 module. WebSphere Application Server Version 5.0 is compliant with the EJB 2.0 specification; therefore you cannot specify the isolation level on the EJB method level or bean level.

- **EJB 2.0 module**

  The isolation level and read-only method level modifiers that could be defined for EJB1.1 are now part of the access intent mechanism of WebSphere. Isolation levels were specific to JDBC, but because the persistence mechanism is now based on J2C resources, a more abstract mechanism was needed. If the underlying resource is a JDBC resource, then the access intent hints will still be translated to JDBC isolation levels under the covers. If the
resource is not a relational database, then the access intent will be translated to the mechanism appropriate to that resource.

An access intent policy is a named set of properties (access intents) that governs data access for EJB persistence. You can assign a policy to individual methods on an entity bean's home, remote, or local interfaces during assembly. Access intents are available only within EJB 2.x-compliant modules for entity beans with CMP 2.x and for BMPs.

Access intent enables developers to configure applications so that the EJB container and its agents can make performance optimizations for entity bean access. Entity bean methods are configured with access intent policies at the module level. A policy is acted upon by either the combination of the WebSphere EJB container and persistence manager (for CMP entities) or by BMP entities directly. Note that access intent policies apply to entity beans only.

The intent of the access intent mechanism in WebSphere Version 5 is to allow developers to supply the container with optimization hints. The container will use these hints to make decisions about isolation levels, cursor managements, and so forth. The hints are organized into groups called policies. The policies are defined at the module level and applied to individual methods on the bean's interface (local or remote).

**Concurrency control**

Concurrency control is the management of contention for data resources. A concurrency control scheme is considered pessimistic when it locks a given resource early in the data-access transaction and does not release it until the transaction is closed. A concurrency control scheme is considered optimistic when locks are acquired and released over a very short period of time at the end of a transaction.

**Read-ahead hints**

Read-ahead schemes enable applications to minimize the number of database round trips by retrieving a working set of CMP beans for the transaction within one query. Read-ahead involves activating the requested CMP beans and caching the data of related beans (relationships), which ensures that data is present for the beans that are most likely to be needed next by an application. A read-ahead hint is a canonical representation of the related beans that are to be read. It is associated with a finder method for the requested bean type, which must be an EJB 2.x-compliant CMP entity bean.
Specifying access intent in WebSphere
WebSphere V5 has several predefined access intent policies that are useful combinations of the five available access intent attributes:

- Access type
- Collection scope
- Collection increment
- Resource adapter prefetch increment
- Read-ahead hint

The access type hint is relevant for transactions because it has to do with concurrency and update intent. For a more detailed explanation of the access intent policies, please refer to the Application Developer help.

Table 4-1 and Table 4-2 describe the access intent settings, and how they might affect the underlying isolation levels.

Table 4-1  Access intent settings

<table>
<thead>
<tr>
<th>Profile name</th>
<th>Concurrency control</th>
<th>Access type</th>
<th>Transaction isolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsPessimisticRead</td>
<td>pessimistic</td>
<td>read</td>
<td>repeatable read</td>
</tr>
<tr>
<td>wsPessimisticRead</td>
<td>pessimistic</td>
<td>update</td>
<td>repeatable read</td>
</tr>
<tr>
<td>wsPessimisticUpdate-Exclusive</td>
<td>pessimistic</td>
<td>update</td>
<td>serializable</td>
</tr>
<tr>
<td>wsPessimisticUpdate-NoCollision</td>
<td>pessimistic</td>
<td>update</td>
<td>read committed</td>
</tr>
<tr>
<td>wsPessimisticUpdate-WeakestLockAtLoad</td>
<td>pessimistic</td>
<td>update</td>
<td>repeatable read</td>
</tr>
<tr>
<td>wsOptimisticRead</td>
<td>optimistic</td>
<td>read</td>
<td>read committed</td>
</tr>
<tr>
<td>wsOptimisticUpdate</td>
<td>optimistic</td>
<td>update</td>
<td>read committed</td>
</tr>
</tbody>
</table>

Mapping transaction isolation levels to DB2
DB2 accepts a set of isolation levels when you bind an application. The DB2 specification is a little different from the EJB specification, but there is a close match between the two sets.

Table 4-2  DB2 isolation levels

<table>
<thead>
<tr>
<th>Isolation level in JDBC</th>
<th>Isolation level in DB2</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTION_SERIALIZABLE</td>
<td>Repeatable read</td>
<td>RR</td>
</tr>
</tbody>
</table>
**Pessimistic read**
Read locks are held for the duration of the transaction.

**Pessimistic update**
To assure data integrity, locks will be held during the scope of the transaction under which ejbLoad was invoked. This access type can be further qualified with the following update hints:

- **Exclusive**
  This is a hint that the normal isolation value used for pessimistic update may not be sufficient.

- **No collision**
  This hint indicates that the application will have no row collisions by design and a lesser isolation level may be chosen by the runtime. WebSphere can, of course, make no assurances guaranteeing the data integrity; in fact, row collisions do occur when the no collision hint is specified, and concurrent transactions can overwrite each other’s updates. Use this policy if only one transaction updates at any given time.

- **Weakest lock**
  Same as no collision, but with repeatable read isolation level. Deadlocks can occur when the updating of one entity bean by two transactions is attempted. This is the default type that will be used if no type has been explicitly assigned.

**Optimistic read**
This type is equivalent to pessimistic read, but with a different isolation level.

**Optimistic update**
Locks will not be acquired during ejbLoad; instead, overqualified updates will be used in order to assure data integrity. For a JDBC resource, an overqualified update will use the WHERE part of the UPDATE statement to compare every field of the record to its old value, as in:

```
UPDATE Customer SET field1=newvalue1, field2=newvalue2
WHERE field1=oldvalue1 AND field2=oldvalue2
```
Note that, for records with a large number of fields, this type of comparison can become a rather expensive operation.

So with optimistic update, instead of locking everything, you hope that nobody else will change the data, or only a few columns are changed. This could be the case for your application if changes are highly unlikely and always locking the data would affect scalability. When a conflict does occur, an exception is thrown.

It is important to note that nullable columns are not supported by optimistic concurrency in WebSphere V5. Nullable columns are excluded from overqualified updates so that changes to nullable columns are not detected.

Optimistic concurrency always faces the possibility of getting locked out by another transaction using a stronger locking strategy. However, optimistic concurrency does not suffer from the pessimistic restriction that an entity loaded under read intent cannot be updated.

### 4.3 Typical application flow

Figure 4-3 shows the typical application flow for Web browser clients using either JDBC (from a servlet) or EJB to access application databases in DB2.
Below we review the above figure:

1. A Web client requests a URL in the browser (input page).
2. The request is routed to the Web server over the Internet.
3. The Web server immediately passes the request to the WebSphere plug-in. All requests go to the WebSphere plug-in first.
4. The WebSphere plug-in examines the URL, verifies the list of host name aliases from which it will accept traffic based on the virtual host information, and chooses a server to handle the request.
5. A stream is created. A stream is a connection to the Web container. It is possible to maintain a connection (stream) over a number of requests. The Web container receives the request and, based on the URL, dispatches it to the proper servlet.
6. If the servlet class is not loaded, the dynamic class loader loads the servlet. Servlet \(\text{init}()\), then \(\text{doGet()}\) or \(\text{doPost}()\).
7. JNDI is now used for lookup of either datasources or EJBs required by the servlet.
8. Depending upon whether a datasource is specified or an EJB is requested, the JNDI will direct the servlet:
   - To the corresponding database, and get a connection from its connection pool in the case of a datasource
   - To the corresponding EJB container, which then instantiates the EJB, when an EJB is requested
9. If the EJB requested involves an SQL transaction, it will go back to the JNDI to look up the datasource.
10. The SQL statement will be executed and the data retrieved will be sent back:
    - To the servlet
    - To the EJB
11. Data beans are created and handed off to JSPs in the case of EJBs.
12. Servlet sends data to JSPs.
13. The JSP generates the HTML that is sent back through the WebSphere plug-in to the Web server.
14. The Web server sends the output page (output html) to the browser.
4.4 Topology selection

In Chapter 2, “Overview of WebSphere Application Server V5” on page 19, we have described that WebSphere Application Server provides support for the multi-tier application model. A topology for the integrated system is the layout of its main components such as Web servers, WebSphere Application Servers, and DB2 UDB servers over one or more machines, spread over one or more geographically distributed locations.

4.4.1 Selection criteria

A variety of factors come into play when considering the appropriate topology. The selection criteria typically include a review of your requirements in the following factors:

- Security
- Performance
- Throughput
- Scalability
- Availability
- Maintainability
- Session management

4.4.2 Performance and scalability

Performance involves minimizing the response time for a given transaction load. Although a number of factors relating to application design can affect performance, one or both of the following techniques are commonly used to improve the performance:

- Vertical scaling
  Involves creating additional application server processes on a single physical machine, providing for software/application server fail-over as well as load balancing across multiple JVMs (application server processes). Vertical scaling allows an administrator to profile an existing application server for bottlenecks in performance, and potentially use additional application servers, on the same machine, to get around these performance issues.

- Horizontal scaling
  Involves creating additional application server processes on multiple physical machines to take advantage of the additional processing power available on each machine. This provides hardware fail-over support and allows an administrator to spread the cost of an implementation across multiple physical machines.
4.4.3 Single machine topology

The starting scenario is the configuration where all components reside on the same machine, as shown in Figure 4-4. The Web server routes requests, as appropriate, to the WebSphere Application Server on the same machine for processing.

Some reasons to use a single machine topology are:

- **Maintainability:** Easy to install and maintain.
  
  This configuration is most suitable as a startup configuration in order to evaluate and test the basic functionality of WebSphere and related components. The installation is automated by tools supplied with the WebSphere distribution. This configuration is also the easiest to administer.

- **Performance, security, and availability are not critical goals.**
  
  This may be the case for development, testing, and some intranet environments. We are limited to the resources of a single machine, which are shared by all components.

- **Low cost.**

  Consider the following when you use a single machine topology:
- **Performance: Components' interdependence and competition for resources**
  All components compete for the shared resources (CPU, memory, network, and so on). Since components influence each other, bottlenecks or ill-behaved components can be difficult to identify.

- **Security: No isolation**
  There is no explicit layer of isolation between the components.

- **Availability: Single point of failure**
  This configuration is a single point of failure.

### 4.4.4 Separating the HTTP server

When compared to a configuration where the application server and the HTTP server are collocated on a single physical server, separation of the application server and the HTTP server can be utilized to provide varying degrees of improvement in:

- **Performance**
- **Process isolation**
- **Security**

This configuration is illustrated in Figure 4-5.
The WebSphere V5.0 HTTP plug-in allows the HTTP server to be physically separated from the application server. It uses an XML configuration file (plugin-cfg.xml) containing settings that describe how to handle and pass on requests to the WebSphere Application Server(s).

### 4.4.5 Separating the DB2 UDB server

WebSphere Application Server accesses DB2 UDB Server through DB2 JDBC drivers. As long as the configuration is correct, WebSphere will be able to communicate with a DB2 UDB server located anywhere accessible through TCP/IP. We discuss later on how to configure WebSphere to communicate with a remote DB2 UDB server.

In the simple single machine configuration described in “Single machine topology” on page 90, the application database and WebSphere Application Server reside on the same machine. However, installing the DB2 UDB server on a different machine, creating a two-tier configuration (as illustrated in Figure 4-6), represents a good practice, with several advantages.

![Figure 4-6 Separating the DB2 server](image)

Some reasons to separate the database server are:

- Performance: Less competition for resources
If both the DB2 UDB and WebSphere Application server are placed on the same machine, then you have two programs: The application server and the DB2 UDB server, competing for increasingly scarce resources (CPU and memory). So, in general, we can expect significantly better performance by separating the WebSphere Application server from the DB2 server.

- **Performance: Differentiated tuning**
  
  By separating the servers, we can independently tune the machines that host the database server and the application server to achieve optimal performance for each other. The database server is typically sized and tuned for database performance, which may differ from the optimal configuration for the application server.

  On many UNIX servers, installing the database involves modification of the OS kernel. This database-specific tuning is often detrimental to the performance of application servers located on the same machine.

- **Availability: Use of already established highly available database servers**
  
  Many organizations have invested in high-availability solutions for their database servers, reducing the possibility of the server being a single point of failure in a system.

- **Maintainability: Independent installation/re-configuration**
  
  Components can be re-configured, or even replaced, without affecting the installation of the other component.

Consider the following when using a remote database server:

- **Network access may limit performance**
  
  Depending upon the network hardware and remoteness of the database server, the network response time for communication between WebSphere Application Server and the database server may limit the performance of WebSphere. When collocated on the same server, network response is not an issue.

- **Architectural complexity**
  
  Hosting the database server on a separate machine introduces yet another box that must be administered, maintained, and backed up.

- **Maintainability: Complexity of configuration**
  
  Remote database access requires more complex configuration, setting up clients, and so on.

- **Cost**
  
  The cost of a separate machine for database server may not be justified for the environment in which the WebSphere Application Server will be installed.
4.4.6 Separating the Web container and the EJB container

Depending on how your application is architected, it can be advantageous to physically separate the application server where your servlets run from your EJB application servers. An example topology for this configuration is illustrated in Figure 4-7.

Some reasons to separate the Web container and the EJB container are:

- **Performance**: Less competition for resources
  Performance gains can occur for applications consisting of business objects that are accessed mainly through EJBs. Separating Web containers from EJB containers prevents the servlet load from affecting EJB-based applications. Servlets that have little interaction with EJBs or none at all can also be deployed on a relatively smaller server.

- **Security**
  When running in an environment where two firewalls are employed, you can provide the same level of security for Entity EJBs as is provided for application data.
4.4.7 Vertical scaling

Vertical scaling provides a straightforward mechanism for creating multiple instances of an application server, and hence multiple JVM processes. In the simplest case, one can create many application servers on a single machine, and this single machine also runs the HTTP server process. This configuration is illustrated in Figure 4-8.

Some reasons to use vertical scaling are:

- **Performance:** Better use of the CPU.
  
  An instance of an application server runs in a single Java Virtual Machine (JVM) process. However, the inherent concurrency limitations of a JVM process may prevent it from fully utilizing the processing power of a machine. Creating additional JVM processes provides multiple thread pools, each corresponding to the JVM associated with each application server process. This can enable the application server to make the best possible use of the processing power and increase throughput of the host machine.

- **Availability:** Fail-over support in a cluster.
  
  A vertical scaling topology also provides process isolation and fail-over support within an application server cluster. If one application server instance goes offline, the requests can be redirected to other instances on the machine to process.
Throughput: WebSphere workload management.

Vertical scaling topologies can make use of the WebSphere Application Server workload management facility. The HTTP server plug-in distributes requests to the Web containers and the ORB distributes requests to EJB containers.

Maintainability: Easy-to-administer member application servers.

With the concept of cells and clusters in WebSphere Application Server V5.0, it is easy to administer multiple application servers from a single point.

Maintainability: Vertical scalability can easily be combined with other topologies.

We can implement vertical scaling on more than one machine in the configuration. (IBM WebSphere Application Server Network Deployment V5.0 must be installed on each machine.) We can combine vertical scaling with the other topologies described in this chapter to boost performance and throughput. This assumes, of course, that sufficient CPU and memory are available on the machine.

Cost: Does not require additional machines.

Consider the following when using vertical scaling:

Availability: Machine still a single point of failure.

Single machine vertical scaling topologies have the drawback of introducing the host machine as a single point of failure in the system. However, this can be avoided by using horizontal scaling on multiple machines.

Performance: Scalability limited to a single machine. Scalability is limited to the resources available on a single machine.

4.4.8 Horizontal scaling with clusters

Horizontal scaling exists when the members of an application server cluster are located across multiple physical machines. This lets a single WebSphere application span several machines, yet still present a single logical image. This configuration is illustrated in Figure 4-9 on page 97.
The WebSphere HTTP server plug-in distributes requests to cluster member application servers on the application server nodes.

**Advantages**

Horizontal scaling using clusters has the following advantages:

- **Availability**
  
  Provides the increased throughput of vertical scaling topologies but also provides fail-over support. This topology allows handling of application server process failures and hardware failures without significant interruption to client service.

- **Throughput**
  
  Optimizes the distribution of client requests through mechanisms such as workload management or remote HTTP transport.

**Disadvantages**

Horizontal scaling using clusters has the following disadvantages:

- **Maintainability:** With the concept of cells and clusters in IBM WebSphere Application Server Network Deployment V5.0, it is easy to administer multiple
application servers from a single point. However, there is more installation and maintenance associated with the additional machines.

- Cost: More machines.

4.4.9 Session persistence considerations

If the application maintains state between HTTP requests and we are using vertical or horizontal scaling, then we must consider using an appropriate strategy for session management.

Each application server runs in its own JVM process. To allow a fail-over from one application server to another without logging out users, we need to share the session data between multiple processes. There are two different ways of doing this in IBM WebSphere Application Server Network Deployment V5.

Memory-to-memory session replication

Provides replication of session data between the process memory of different application server JVMs. A Java Message Service (JMS) based publish/subscribe mechanism, called Data Replication Service (DRS), is used to provide assured session replication between the JVM processes. DRS is included with IBM WebSphere Application Server Network Deployment V5 and is automatically started when the JVM of a clustered (and properly configured) application server starts.

Database persistence

Session data is stored in a database shared by all application servers.

Memory-to-memory replication has the following advantages and disadvantages compared to database persistence:

- Memory-to-memory replication is faster by virtue of the high performance messaging implementation used in WebSphere V5.
- No separate database product is required. But normally you will use a database product anyway for your application, so this might not be an inhibitor.
- If you have a memory constraint, using database session persistence rather than memory-to-memory replication might be the better solution. There are also several options to optimize memory-to-memory replication.

Persistent session should be enabled in some scenarios. Please refer to Chapter 8, “DB2 UDB V8 and WAS V5 integrated performance” on page 287, for more details.
### 4.4.10 Topology selection summary

These considerations for topology selection are not mutually exclusive. They can be combined in different ways. Table 4-3 provides a summary of the topology selection considerations.

**Table 4-3 Topology selection criteria**

<table>
<thead>
<tr>
<th>Security</th>
<th>Performance</th>
<th>Throughput</th>
<th>Maintainability</th>
<th>Availability</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single machine</td>
<td>Little isolation between components</td>
<td>Competition for machine resources.</td>
<td>Limited to machine resources.</td>
<td>Ease of installation and maintenance.</td>
<td>Machine is single point of failure.</td>
</tr>
<tr>
<td>Separate DB2 UDB server</td>
<td>Firewall can provide isolation</td>
<td>Separation of loads. Performance usually better than local DB server.</td>
<td>Independent tuning. Must consider network bottleneck.</td>
<td>Use already established DBA procedures. Independent configuration. More administrative overhead.</td>
<td>Introduces single point of failure. Use already established HA servers.</td>
</tr>
<tr>
<td>Separate Web/EJB container</td>
<td>More options for firewall</td>
<td>Typically slower than single JVM.</td>
<td>Clustering can improve throughput.</td>
<td>More administrative overhead.</td>
<td>Introduces single point of failure.</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------</td>
<td>-------------------------------------------</td>
<td>---------------------</td>
<td>---------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Add HTTP server</td>
<td>Distribution of load.</td>
<td>Distribution of connections.</td>
<td>More to install/maintain.</td>
<td>Best in general.</td>
<td>Use load balancers. SSL session ID affinity when using SSL.</td>
</tr>
<tr>
<td>One domain</td>
<td>Ease of maintenance.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple domains</td>
<td>Less lookups and interprocess communication.</td>
<td>Harder to maintain than single domain.</td>
<td></td>
<td>Process hardware and software redundancy.</td>
<td></td>
</tr>
</tbody>
</table>
4.5 How WebSphere Application Server works with DB2 UDB Server

In this section we describe how WebSphere Application Server interacts with DB2 UDB Server. The topics covered are J2EE Connector architecture, resource adapters, and DB2 JDBC drivers.

4.5.1 J2EE Connector architecture

The J2EE Connector architecture (JCA) defines a standard architecture for connecting the J2EE platform to heterogeneous Enterprise Information Systems (EIS), for example, ERP, mainframe transaction processing, database systems, and legacy applications not written in the Java programming language. By defining a set of scalable, secure, and transactional mechanisms, the JCA enables the integration of EISs with application servers and enterprise applications.

WebSphere Application Server V5 provides a complete implementation of the JCA 1.0 specification that is a part of J2EE.

The JCA Resource Adapter is a system-level software driver supplied by EIS vendors or other third-party vendors. It provides the following functionality:

► Provides connectivity between J2EE components (an application server or an application client) and an EIS.

► Plugs into an application server.

► Collaborates with the application server to provide important services such as connection pooling, transaction, and security services. JCA defines the following set of system-level contracts between an application server and EIS:

  – A connection management contract that lets an application server pool connects to an underlying EIS, and lets application components connect to an EIS. This leads to a scalable application environment that can support a large number of clients requiring access to EISs.

  – A transaction management contract between the transaction manager and an EIS that supports transactional access to EIS resource managers. This contract lets an application server use a transaction manager to manage transactions across multiple resource managers. This contract also supports transactions that are managed internally to an EIS resource manager without the necessity of involving an external transaction manager.

  – A security contract that enables a secure access to an EIS. This contract provides support for a secure application environment, which reduces
security threats to the EIS and protects valuable information resources managed by the EIS.

The resource adapter implements the EIS-side of these system-level contracts.

- Implements the Common Client Interface (CCI) for EIS access. The CCI defines a standard client API through which a J2EE component accesses the EIS. This simplifies writing code to connect to an EIS’s data store. The resource adapter provides connectivity between the EIS, the application server, and the enterprise application via the CCI.

- Implements the standard Service Provider Interface (SPI) for integrating the transaction, security, and connection management facilities of an application server (JCA Connection Manager) with those of a transactional resource manager.

A resource adapter is used within the address space of the application server, and multiple resource adapters (that is, one resource adapter per type of EIS) are pluggable into an application server. This capability enables application components deployed on the application server to access the underlying EISs.

WebSphere Application Server provides the relational resource adapter (RRA) implementation. This resource adapter provides data access through JDBC/SQLJ to access the DB2 UDB server. The connection management is based on the J2EE Connector Architecture (JCA) connection management architecture. It provides connection pooling, local transaction, and security support.

Container-managed persistence (CMP) beans data access is managed by the WebSphere Persistence Manager indirectly. The JCA Specification supports Persistence Manager delegation of the data access to the JCA resource adapter without knowing the specific backend store. For the relational database access, Persistence Manager utilizes the relational resource adapter to access the data in the database.

### 4.5.2 JDBC resources

WebSphere resource providers are a class of objects that provide resources needed by running J2EE applications in WebSphere. For example, if an application requires database access through a data source, you would need to install a JDBC data source provider and then configure a data source to be used by your application.
The JDBC API provides a programming interface for data access of relational databases from the Java programming language. The JDBC 2.0 API is comprised of two packages:

- The java.sql package, which is the JDBC 2.0 core API.
- The javax.sql package, which is the JDBC 2.0 Standard Extension API. This package provides data source and connection pooling functionality.

### JDBC providers and data sources

A data source represents a real-world data source, such as a relational database. Once a data source object is registered with a JNDI naming service, an application can retrieve it from the naming service and use it to make a connection to the data source it represents.

Information about the data source and how to locate it, such as its name, the server on which it resides, its port number, and so on, is stored in the form of properties on the DataSource object. This makes an application more portable because it does not need to hard code a driver name, which often includes the name of a particular vendor. It also makes maintaining the code easier because if, for example, the data source is moved to a different server, all that needs to be done is to update the relevant property in the data source. None of the code using that data source needs to be touched.

The connection is usually a pooled connection, that is, once the application closes the connection, the connection is returned to a connection pool, rather than being destroyed.

Data source classes and JDBC drivers are implemented by the database vendor. By configuring a JDBC provider, we are providing information about the set of classes used to implement the data source and the database driver, that is, it provides the environment settings for the DataSource object.

The programming model is as follows:

- An application retrieves a DataSource object from the JNDI naming space.
- After the DataSource object is obtained, the application code calls getConnection() on the data source to get a connection object. The connection is a pooled connection, that is, it is obtained from a pool of connections.
- Once the connection is acquired, the application then sends SQL queries or updates to the database.

WebSphere Application Server V5 provides two types of data sources, each differentiated by how the connections are handled:

- WebSphere Version 4 data source
WebSphere Version 4.0 provided its own JDBC connection manager to handle connection pooling and JDBC access. This support is included with WebSphere Application Server V5 to provide support for J2EE 1.2 applications. If an application chooses to use a Version 4 data source, the application will have the same connection behavior as in WebSphere Version 4 (see Figure 4-10).

![Diagram of connection pooling](image)

**Figure 4-10** Version 4 connection pooling

- **WebSphere Version 5 data source**

In WebSphere Application Server V5, connection pooling is provided by two parts, a JCA Connection Manager and a relational resource adapter. The JCA Connection Manager provides the connection pooling, local transaction, and security supports. The relational resource adapter provides both JDBC wrappers and JCA CCI implementation that allows BMP, JDBC applications, and CMP beans to access the database. Figure 4-11 on page 105 shows the relational resource adapter model.
WebSphere relational resource adapter

WebSphere provides a Persistence Resource Adapter to provide relational persistence services to the EJB beans that are deployed in the J2EE1.3 application, as well as providing database access to BMP and JDBC applications. The Persistence Resource Adapter has two components: The persistence manager, which supports the new EJB 2.0 CMP persistence model; and the relational resource adapter.

The persistence resource adapter code is included in the following Java packages:

- com.ibm.ws.rsadapter.cci - Contains CCI implementation and JDBC wrappers
- com.ibm.ws.rsadapter.spi - Contains SPI implementation
- com.ibm.ws.rsadapter.jdbc - Contains all the JDBC wrappers
- com.ibm.websphere.rsadapter - DataStoreHelper, WSCallerHelper and DataAccessFunctionSet
The EJB 2.0 Persistence Manager utilizes the relational resource adapter to handle the data access to and from the back-end store. The adapter provides relational persistence services to the EJB beans that are deployed in the J2EE1.3 applications. The relational resource adapter implementation is based on the J2EE Connector (JCA) specification and implements the JCA CCI and SPI interfaces.

The relational resource adapter that is available with WebSphere V5 provides JDBC/SQLJ data access to a relational database.

If an application chooses to use a Version 5 data source, the data source will use the JCA connector architecture to get to the relational database. Although there is no difference between the existing WebSphere JDBC support versus the new support in terms of application development, there will be some connection behavior changes because of different architectures.

For an EJB, the sequence is as follows:
1. An EJB performs a JNDI lookup of a data source connection factory and issues a getConnection() request.
2. The connection factory delegates the request to a connection manager.
3. The connection manager looks for an instance of a connection pool in the application server. If no connection pool is available, then the manager uses the ManagedConnectionFactory to create a physical (non-pooled) connection.

So from a JDBC application point of view, there is no difference between using a Version 4 data source or a Version 5 data source. It is the implementation of the data source that changes.

**Data source usage guidelines**

The data source that can be used is determined by whether the application is a J2EE 1.2 or a J2EE 1.3 application, and whether it uses EJB 1.1 modules or EJB 2.0 modules. The guidelines for data source use are:

- **J2EE 1.2 applications**
  - All EJB beans, JDBC applications, or Version 2.2 servlets must use the Version 4 data source.

- **J2EE 1.3 applications**
  - EJB 1.x modules must use the Version 4 data source.
  - EJB 2.0 modules, including CMP Version 2.0 and 1.x, must use the Version 5 data source.
– JDBC applications and Version 2.3 servlets must use the Version 5 data source.

4.5.3 DB2 JDBC providers

According to the JDBC specification, there are four types of JDBC driver architectures:

- **Type 1** - Drivers that implement the JDBC API as a mapping to another data access API, such as ODBC. Drivers of this type are generally dependent on a native library, which limits their portability. The JDBC-ODBC Bridge driver is an example of a Type 1 driver.

- **Type 2** - Drivers that are written partly in the Java programming language and partly in native code. The drivers use a native client library specific to the data source to which they connect. Again, because of the native code, their portability is limited. Notice that a Type 2 has a native component that is part of the driver and is separate from the database access API.

- **Type 3** - Drivers that use a pure Java client and communicate with a middleware server using a database-independent protocol. The middleware server then communicates the client's requests to the data source.

- **Type 4** - Drivers that are pure Java and implement the network protocol for a specific data source. The client connects directly to the data source.

DB2 UDB Version 8 support Type 2, Type 3, and Type 4 JDBC drivers. The JDBC drivers in previous releases were built on top of DB2 CLI (Call Level Interface). DB2 UDB Version 8 Type 2 and Type 3 drivers continue to use the DB2 CLI interface to communicate with DB2 UDB servers (OS/390® and z/OS™, UNIX®, Windows®, Linux, and iSeries™). DB2 UDB Version 8 adds a new DB2 JDBC Universal Driver (Type 2 and Type 4), which uses the Distributed Relational Database Architecture™ (DRDA®) protocol for client/server communications.

**DB2 Legacy CLI-based JDBC application driver (Type 2)**

The DB2 JDBC application (Type 2) driver (Figure 4-12 on page 108) enables Java applications to make calls to DB2 UDB through JDBC. Calls to the JDBC application driver are translated to native methods. The Java applications that use this driver must run on a DB2 UDB client, through which JDBC requests flow to the DB2 UDB server. A DB2 Connect™ Version 8 license/installation is required to access DB2 for OS/390 databases.
The DB2 Legacy CLI-based JDBC application (Type 2) driver is included in the COM.ibm.db2.jdbc.app package.

**JDBC 1 connections**
The implementation classes for establishing a connection to DB2 UDB servers include COM.ibm.db2.jdbc.app.DB2Driver.

**JDBC 2 connections**
The implementation classes for establishing a connection to DB2 UDB servers include:

- COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource
- COM.ibm.db2.jdbc.DB2DataSource
- COM.ibm.db2.jdbc.DB2XADataSource

**IBM DB2 Universal JDBC driver**
Many new features and enhancements have been made to the JDBC drivers in DB2 UDB Version 8. Among these changes, the biggest change is the architectural improvement that shortens the code path between the JDBC driver and DB2 UDB servers. This new IBM JDBC Universal Driver is based on an open distributed protocol, known as Distributed Relational Database Architecture (DRDA), and is compatible with all DB2 UDB server platforms (UNIX, Windows, Linux, z/OS) with appropriate DRDA Application Server (AS) level support, and prerequisite stored procedures.

Features unique to the new IBM DB2 JDBC Universal Driver include:

- Updateable ResultSet support
- Improved security for DB2 authentication
- Improved Java SQL error information
- Programmatic tracing facilities
The new Universal Driver is com.ibm.db2.jcc.DB2Driver, and is contained in db2jcc.jar and sqlj.zip. The file db2jcc.jar contains the JDBC driver and SQLJ runtime modules. The file sqlj.zip contains the reference implementation of the standard SQLJ precompiler.

**JDBC 1 connections**

The DB2 UDB Version 8 server will use a TCP/IP connection, specified in the DB2 DBM configuration file (SVCENAME), to communicate with the new DB2 JDBC Universal Driver.

When obtaining connections from the JDBC 1 driver manager, if the database URL syntax is of the form:

- `jdbc:db2://server:port/database`
- `jdbc:db2://server/database`

Then Type 4 connectivity is selected. Whereas if the URL is of the form:

- `jdbc:db2:database`

Then Type 2 connectivity is selected.

**JDBC 2 connections**

The implementation classes for establishing a connection to DB2 UDB servers include:

- `com.ibm.db2.jcc.DB2SimpleDataSource`
- `com.ibm.db2.jcc.DB2ConnectionPoolDataSource`
- `com.ibm.db2.jcc.DB2XADataSource`

The Universal Driver (com.ibm.db2.jcc, also known as JCC) shares the same Java code for both T2 and T4 connectivity, so the Java files are the same regardless of T2 or T4. The major differences are:

- The Type 2 driver makes its connections via the C Common Client (CCC) code that is used by CLI and the Legacy JDBC drivers (app/net), and does all communication via the CCC. This allows the Type 2 to use all of the communication protocols that CCC supports (shared memory, SNA, Netbios, etc.). It also allows the JCC Type 2 to run in the UDB server to support Java Stored Procedures (JSPs). (The Type 4 can call all types of stored procedures but cannot be used in the server.) If you have a local UDB server the shared memory connection is very fast. When you code just the database alias in the URL the database must be cataloged using the CLP command or Configuration Assistant.
The Type 2 driver requires a DLL/shared library to be installed. The Type 4 driver is a “Pure Java” implementation and most measurements indicate it is faster than the Legacy JDBC drivers but not as fast as the JCC Type 2. The Type 4 driver also supports more security mechanisms than the Type 2 since support for them is in the UDB server but not in the CCC.

For DataSources the major difference is the Property “driverType”. A value of 2 loads the Type 2, and a value of 4 loads the Type 4 (2 is the default). If you code “driverType=2” and do not code “portNumber” and “serverName”, the Type 2 driver will use the “databaseName” as the DB alias to connect to a cataloged database.

**Note:** The Type 2 driver can also make a TCP/IP connection without using the “catalog” if you specify the host name and port number as properties in the URL.

SQLJ support

SQLJ support has been rearchitected in DB2 UDB Version 8. DB2 SQLJ support enables you to build and run SQLJ applets and applications. These Java programs contain embedded SQL statements that are precompiled and bound to a DB2 UDB database.

The IBM Universal driver for JDBC and SQLJ supports JDBC 2 APIs under JRE 1.3, and most of the JDBC 3 APIs under JRE 1.4.

The SQLJ standard has three components: A translator, customizer, and a run-time environment. The translator produces Java code based on the embedded SQL statements within a source SQLJ program. A binary representation of the SQL statements is created in a separate serialized profile.
(.ser file). Static SQL packages are created when the profile is customized using the `db2sqljcustomize` command. SQLJ applications require the `db2jcc.jar` file, and SQLJ program translator also requires the `sqlj.zip` file.

**Note:** SQLJ truly has been greatly enhanced in IBM WebSphere Studio V5.1 products.

SQLJ provides:

- A static package level security model
- A static SQL interface (for example, `SELECT xxx INTO :hv1, :hv2`)
- Increased development productivity as compared to JDBC, especially if an application is being ported from an existing embedded SQL architecture (C, COBOL, etc.)

DB2 Version 8 provides the following SQLJ utilities, as shown in Figure 4-14 on page 112:

- `sqlj`  
  IBM SQLJ Translator. It translates an `.sqlj` source file and creates a serialized profile and a program.

- `db2sqljcustomize`  
  Customizer and online checker. It creates a DB2 customization for the serialized profile, and optionally online-checks SQL statements that can be dynamically prepared, and optionally (by default) binds the DB2 packages for this program.

- `db2sqljbind`  
  Standalone binder. It binds a previously customized SQLJ profile to a database. This utility can be used to deploy SQLJ applications to new DB2 servers.

- `db2sqljprint`  
  Prints contents of a DB2 customized profile.
**DB2 JDBC driver providers**

WebSphere V5 provides the DB2 Universal JDBC driver provider, DB2 Universal JDBC driver provider (XA), DB2 Legacy CLI-based type2 JDBC driver provider, and DB2 Legacy CLI-based type2 JDBC driver provider (XA) for a J2EE application running on WebSphere to communicate with DB2 UDB V8 by utilizing the different JDBC drivers and the data source implementation classes provided by DB2 UDB V8.

The XA JDBC providers create XADataSource, which supports application participation in a single-phase or a global (two-phase) transaction environment. When this data source is involved in a global transaction, the transaction manager provides transaction recovery facilities.

The other JDBC providers create ConnectionPoolDataSource, which supports application participation in a single-phase transaction environment.

### 4.6 DB2 connectivity to z/OS and S/390®

In this section we describe the different ways to establish connectivity among Java applications running on multiplatforms (Windows, Linux UNIX) to access data from DB2 for z/OS and OS/390 subsystem or vice versa.
4.6.1 Type 2 connectivity from a non-z/OS platform

The configuration in Figure 4-15 describes an environment where the DB2 UDB for z/OS and OS/390 system is not local to the JDBC driver, and the WAS system is running on a different machine, as well as on a different platform than the DB2 UDB.

In the upper half of Figure 4-15 we show a configuration where DB2 Connect Server and WAS are on the same machine. In this kind of setup you can eliminate the communication cost between the WAS server and the DB2 Connect Server. Instead of TCP/IP, shared memory can be used to communicate between local applications and DB2 Connect.

If you set up a DB2 Connect server separately from your Web Application Server, as shown in the lower half of Figure 4-15, you can decrease the maintenance workload. You can manage all the connection settings at one place and can monitor transactions running through the DB2 Connect Server. Also connections to the target server can be limited to a small number of DB2 Connect servers.

4.6.2 Type 4 connectivity from a non-z/OS platform

In this case, our application is running on a non-z/OS platform and we are using a Type 4 driver to connect an application to access data from DB2 for z/OS.
system. In the upper half of Figure 4-16 we show a configuration where our Java application is using a Type 4 driver to communicate directly to DB2 UDB for z/OS and OS/390 through DRDA. The Type 4 driver converts JDBC API calls into DB2 API calls using DRDA protocol.

When using the Type 4 driver, you do not need DB2 Connect to access data on a DB2 for z/OS and OS/390 system. In the lower half of Figure 4-16 we use DB2 Connect Server even though we are using a Type 4 driver. So the question arises, why is there a DB2 Connect Server in the figure, if it is not required? The answer is that the DB2 Connect Server provides functionality that is not provided by the Type 4 driver. These functions include sysplex awareness and connection concentration. For a more detailed description of these features, please see Distributed Functions of DB2 for z/OS and OS/390, SG24-6952.

Another question related to the previous section ("Type 2 connectivity from a non-z/OS platform" on page 113) is, why not to use only a Type 4 driver instead of Type 2, in all the scenarios where the Java application is running on another platform?

The reason is that the current Type 4 driver is not a full replacement of the DB2 Connect functionality. The Type 4 driver does not support all the functions that the Type 2 driver does. For example, connection pooling and two-phase commit are currently not supported by the Type 4 driver.
4.6.3 DB2 UDB for z/OS and OS/390 as a DRDA application requester

The configuration shown in Figure 4-17 describes the environment when we have WebSphere Application Server running your Java applications that talk to a Type 2 JDBC driver, which further talks through RRS to a DB2 UDB system that is local to the machine running the WAS. Then this local DB2 diverts all requests to a remote database server using DRDA.

**Important:** If you use the Type-4 driver, since it is implemented as a DRDA Application Requestor, you can access DB2 for z/OS and OS/390 without having to go through DB2 Connect. However, from a licensing point of view, you still need a DB2 Connect licence to be able to connect to a DB2 for z/OS and OS/390 using the Type 4 driver.

The application will connect directly to the remote DB2 UDB system. However, the local DB2 has to exist because its services as a DRDA Application Requester (AR) are needed to be able to connect to the remote DB2 UDB system.

This kind of environment provides a very secure and high-availability environment running WebSphere Application Server systems on zSeries hardware in a parallel sysplex.
4.6.4 Application on z/OS connecting DB2 UDB for Multiplatforms

The DB2 UDB for Windows, UNIX, and Linux provide DRDA Application Server functionality as a standard function. Also DB2 for z/OS and OS/390's DDF also provides DRDA Application Requestor functions.

Figure 4-18 shows the configuration when an application program is running under the z/OS or S/390 environment accessing data form DB2 UDB for Multiplatforms server through DDF. TCP/IP is the only supported network protocol DB2 UDB for Multiplatforms Version 8 serves.

This kind of configuration is very rare but still useful when we need to integrate legacy-based applications with a multiplatorm environment.
Operational setup

In this chapter we discuss the general steps to get DB2 UDB V8 and WebSphere Application Server V5 working together to serve specific business and application requirements. The new DB2 EJB sample application named “AcessEmployee” is introduced in detail to provide you with the basic process of how to fit DB2 UDB V8 and WebSphere Application Server V5 together. In addition, we also discuss the available JDBC drivers shipped with DB2 UDB V8 and the differences between those drivers.

Basically, the following information is covered in this chapter:

- Environment setup (including product installation briefing)
- Steps to create JDBC Providers and DB2 Data Sources
- DB2 EJB sample application installation and testing
- Access to DB2 on z/OS by using DB2 Connect

Also please be aware that this chapter is more operation flavor but not performance flavor, as we think before we talk about the integrated performance tuning. Getting the environment up and running is the basis for further performance tuning.
5.1 Environment setup

This section covers the steps to build a basic environment where the DB2 EJB sample application will be deployed and running. The major products covered here include DB2 UDB V8 and WebSphere Application Server V5, as well as IBM HTTP Server.

5.1.1 Installation planning

Before the actual installation begins, the appropriate topology should be chosen via the thorough consideration for both the business requirements and the available resources. In our case, we choose to use separate machines for the database server, application server, and Web server. Certainly, you could also use the single server topology and put all these products into one machine if the capability of the machine is sufficient to get all those products running together well. Or you could also have a clustering environment to serve more concurrent users and requests.

Before installing the products, you also need to consider the products’ installation prerequisites, such as the disk requirement, memory requirement, group and user accounts requirement, etc. Sometimes the prerequisites vary considerably depending on which edition of the product you choose and on which platform you want to install the product. You could get this kind of information easily from the products’ Web sites or online books. For DB2 UDB V8, to obtain more information about what you need to plan before the installation, please refer to IBM DB2 Universal Database, Quick Beginnings for DB2 Servers, Version 8, GC09-4836. The book can be found at the following URL:

http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/v8pubs.d2w/en_main#V8PDF

Regarding hardware and software requirements for WebSphere Application Server V5 installation, you could refer to the Web page, which can be accessed via the URL as below:


In addition, the section Installing WebSphere Application Server in WebSphere Application Server InfoCenter also provides necessary information to assist you in performing the installation planning task.

There are also some requirements that are related to specific platforms. For example, when running a large WebSphere configuration on the AIX platform, to avoid the excessive paging behavior, which could impact the performance in WebSphere, you probably need to change the file page frame ratio in the kernel.
setting to a lower value as the default setting is relatively too high. Some commands such as `vmstat` and `svmon` (AIX specific) could be used to monitor the runtime resource consumption details. Based on the monitoring result, some commands such as `vmtune` (for AIX 5.1 and previous versions) or `vmo` (for AIX 5.2) could be used to tune those parameters. For WebSphere Application Server V5, there is a special section, *Platform-specific tips for installing and migrating*, in the WebSphere Application Server InfoCenter. You can refer to that for more details.

We also discuss the steps to use WebSphere Application Server V5 to access the DB2 data source residing on the IBM DB2 UDB for z/OS through the DB2 Connect product. We have separate sections to cover this configuration. For a brief overview of DB2 Connect product installation and configuration, please refer to “Using DB2 for z/OS and OS/390 as the Data Source for WAS” on page 127. Regarding using DB2 for z/OS and OS/390 as the data source for our sample application, refer to “Using DB2 for z/OS as the Data Source” on page 163.

### 5.1.2 Products’ installation briefing

In our laboratory environment, we install DB2 UDB V8 ESE, WebSphere Application Server V5.0.2, and IBM HTTP Server (IHS) 1.3.26 on separate machines (the DB2 Runtime Client is also required on the application server machine in order to use type 2 DB2 JDBC Drivers), as shown in the Table 5-1. Please notice that using DB2 Connect to access DB2 on z/OS is covered separately in the section “Using DB2 for z/OS and OS/390 as the Data Source for WAS” on page 127.

<table>
<thead>
<tr>
<th>Usage/purpose</th>
<th>Machine 1</th>
<th>Machine 2</th>
<th>Machine 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine type</td>
<td>pSeries</td>
<td>pSeries</td>
<td>xSeries®</td>
</tr>
<tr>
<td>Operating system</td>
<td>AIX 5L™ 5.1</td>
<td>AIX 5L 5.1</td>
<td>Windows 2000 Server</td>
</tr>
<tr>
<td>Host name</td>
<td>Atlantic</td>
<td>Kanaga</td>
<td>Helium</td>
</tr>
<tr>
<td>IP address</td>
<td>9.1.38.92</td>
<td>9.1.38.90</td>
<td>9.1.38.185</td>
</tr>
<tr>
<td>Products to install</td>
<td>DB2 UDB V8 ESE</td>
<td>WebSphere Application Server V5.0.2 and DB2 UDB V8 Runtime Client</td>
<td>IBM HTTP Server 1.3.26</td>
</tr>
</tbody>
</table>
In addition, the following fix packs or maintenance levels are applied for the operating systems and software products, as Table 5-2 shows. The URLs for downloading fix packs or maintenance levels are also provided in the table.

Table 5-2 Required products and fix packs

<table>
<thead>
<tr>
<th>Category</th>
<th>Product Name</th>
<th>Fix Pack Downloading URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>AIX 5L 5.1 ML5 and Windows 2000 Server with Service Pack 4</td>
<td><a href="https://techsupport.services.ibm.com/server/mlfixes/51/05/00to05.html">https://techsupport.services.ibm.com/server/mlfixes/51/05/00to05.html</a> (AIX) <a href="http://www.microsoft.com">http://www.microsoft.com</a> (Windows)</td>
</tr>
<tr>
<td>Database server</td>
<td>DB2 UDB V8 ESE with Fix Pack 4</td>
<td><a href="http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/download.d2w/report">http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/download.d2w/report</a></td>
</tr>
</tbody>
</table>

If you want to use the WebSphere-embedded messaging as the JMS provider and install it during the WebSphere Application Server installation, there are some specific requirements that you need to fulfill before and after the installation, such as user group and user account preparation, disk space preparation, and so forth. For more details please refer to the section *Installing WebSphere embedded messaging as the JMS provider* in the WebSphere Application Server InfoCenter.

The basic steps for applying the operating system maintenance levels and installing software products as well as their associated fix packs are discussed below:

1. Upgrade AIX Operating Maintenance Level to ML5.

   In our case, as both application server and database server are residing on the AIX 5L platform, we recommend applying the latest maintenance levels on both machines.

   The Web page associated with the URL provided in Table 5-2 for AIX contains the detailed steps to install Maintenance Level 5. You could use `oslevel -r` to verify if you have successfully installed the ML5. The output of this command should be 5100-05.
2. Install DB2 UDB V8 ESE.

The DB2 UDB V8 ESE is installed on the machine Atlantic in our test case. In addition, we also installed DB2 UDB V8 Run-time Client on the machine Kanaga where the WebSphere application server resides. The latest Run-time Client could be obtained via the URL:


You could use the DB2 Setup Wizard (db2setup) to perform the DB2 installation. Regarding detailed steps for DB2 UDB ESE installation, please refer to the Quick Beginnings for DB2 Servers, which can be found via URL:


3. Apply DB2 UDB V8 fix pack.

Before applying the fix pack, please read the fix pack readme file. The file is available in the fix pack installation directory after you uncompress the fix pack archive. The file name for the English version is FixPackReadme.txt. The command installFixPak could help you perform this task. Remember to update the instance by using db2iupdt and rebind packages as required.

Note: DB2 Enterprise Server Edition (ESE) Version 8.1 for UNIX-based operating systems now supports the coexistence of multiple FixPak levels on the same system through the use of “alternate” fix packs. In our case, the regular fix pack 4 is used.

4. Verify the DB2 UDB V8 installation.

You could use db2samp1 command to create the sample database provided by DB2 UDB V8 then try to do some SQL operations against the database to verify if the installation is successful, as shown in Example 5-1.

Example 5-1  DB2 UDB installation verification

$ db2 connect to sample
$ db2 "select * from sales fetch first 3 rows only"

<table>
<thead>
<tr>
<th>SALES_DATE</th>
<th>SALES_PERSON</th>
<th>REGION</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/31/1995</td>
<td>LUCCHESI</td>
<td>Ontario-South</td>
<td>1</td>
</tr>
<tr>
<td>12/31/1995</td>
<td>LEE</td>
<td>Ontario-South</td>
<td>3</td>
</tr>
<tr>
<td>12/31/1995</td>
<td>LEE</td>
<td>Quebec</td>
<td>1</td>
</tr>
</tbody>
</table>

3 record(s) selected.

5. Install WebSphere Application Server V5.
The WebSphere Application Server is installed on machine Kanaga in our case. Before installing the WebSphere Application Server, it is recommended to read through the *Getting Started, IBM WebSphere Application Server, Version 5* online book, which is available in the `docs` subdirectory of the WebSphere Application Server installation directory. You could use the LaunchPad to start the installation. The command to invoke the LaunchPad on AIX is `LaunchPad.sh`, located under the WebSphere Application Server installation directory. Figure 5-1 shows the basic functions that could be achieved via LaunchPad, such as viewing installation guides, starting the installation of the product, etc.

![WebSphere Application Server LaunchPad](image)

*Figure 5-1 Using LaunchPad to install WAS Version 5*

Whether or not to choose installing IBM HTTP Server and Web Server Plug-ins along with the WebSphere Application Server on the same machine depends on your application requirements. In our test case, for more convenience of the testing for some WebSphere Application Server sample applications such as snoop, Technology Samples, etc., the IBM HTTP Server and the Web Server Plug-ins are also installed on the Kanaga machine.

6. Apply Fix Pack 2 for WAS V5.
The file named readme_was50_fp2.html, which can be found under the docs subdirectory of the WebSphere Application Server fix pack installation directory after uncompressing the fix pack archive, contains the fix pack introduction information. There is another file named readme_updateinstaller.txt available under the same subdirectory, which includes detailed instructions about how to apply the fix pack. Basically, you could use the command `updateWizard.sh` to invoke the fix pack installation utility.


The `versionInfo` utility could be used to obtain the current version information. Thereby we can determine if the fix pack is installed successfully. See Example 5-2 on page 124 for details.

**Example 5-2  Using versionInfo.sh to verify the fix pack installation**

```
Kanaga:/usr/WebSphere/AppServer/bin >./versionInfo.sh
WVER0010I: Copyright (c) IBM Corporation 2002; All rights reserved.
WVER0011I: WebSphere Application Server Release 5.0
WVER0012I: VersionInfo reporter version 1.14, dated 5/9/03

IBM WebSphere Application Server Product Installation Status Report

Report at date and time 2003-10-30T15:04:04-08:00

Installation

Product Directory   /usr/WebSphere/AppServer
Version Directory   ${product.dir}/properties/version
DTD Directory       ${version.dir}/dtd
Log Directory       /usr/WebSphere/AppServer/logs/update
Backup Directory    ${version.dir}/backup
TMP Directory       /tmp

Installation Platform

Name           IBM WebSphere Application Server
Version        5.0

Technology List

BASE           installed

Installed Product

Name           IBM WebSphere Application Server
```
Up to now, we have got the WebSphere Application Server base configuration installed on the Kanaga machine. To further verify whether the installation is successful or not, you could try to start the application server by using the `startServer.sh` command, then trying to access the *admin console*, which is generally installed by default, via the URL link `http://kanaga:9090/admin` in an Internet browser available in your testing environment. The login prompt should be displayed in the browser window, as shown in Figure 5-2.

![Figure 5-2  WebSphere Application Server admin console login](image)

You could also verify the installation via the available *Installation Verification Test* (IVT) tool shipped with WebSphere Application Server. It could be invoked by the command `ivt.sh`. A successful verification should return message like that shown in Example 5-3

**Example 5-3  Successful Verification by using ivt.sh**

Kanaga:/usr/WebSphere/AppServer/bin >./ivt.sh
IVTL0095I: defaulting to host kanaga and port 9080
IVTL0010I: Connecting to the WebSphere Application Server kanaga on port: 9080

IVTL0020I: Could not connect to Application Server, waiting for server to start
IVTL0025I: Attempting to start the Application Server
osName = AIX
IVTL0030I: Running /usr/WebSphere/AppServer/bin/startServer.sh server1
>ADMU0116I: Tool information is being logged in file
> /usr/WebSphere/AppServer/logs/server1/startServer.log
>ADMU3100I: Reading configuration for server: server1
If you encounter problems during the verification, “Installation troubleshooting” on page 133 provides some information about how to deal with such problems. In addition, a port conflict issue that leads to a successful startup of WebSphere application server is also discussed in that section.

8. Install IBM HTTP Server 1.3.26 and Web Server Plug-ins.

You could install the IBM HTTP Server from the WebSphere Application Server product CD-ROM labelled “WebSphere Application Server, IBM HTTP Server”. If the Web server and its plug-in running is the only WebSphere Application server component you would like to install on the Web server machine, deselect all options but the IBM HTTP Server and the plug-in for the IBM HTTP Server after choosing the custom installation method. This is the recommended way to install the IBM HTTP Server and its associated plug-in, as the installation wizard could help you to update related configuration files including httpd.conf and plugin-cfg.xml automatically. For more information please refer to the section Preparing to install and configure a Web server in WebSphere Application Server V5 InfoCenter.

In our case, the Web server product IBM HTTP Server is installed on a separate Windows 2000 Server machine, Helium.

In addition, to redirect the HTTP and HTTPS requests to the embedded HTTP Server of the target WebSphere application server, a minor change for the plug-in configuration file plugin-cfg.xml is required. In our case, the configuration file is located under the directory:

C:\Program Files\WebSphere\AppServer\config\cells\n
Edit this file and change the host name property of the Transport entry in the ServerCluster block to the desired host name of the machine where the target application server resides. In our case, as application server is running on the machine Kanaga, so we change that to “kanaga”, as shown in the Example 5-4.

Example 5-4  ServerCluster block of the plug-in configuration file

```xml
<ServerCluster Name="server1_Cluster">
  <Server Name="server1">
    <Transport Hostname="kanaga" Port="9080" Protocol="http"/>
    <Transport Hostname="kanaga" Port="9443" Protocol="https"/>
  </Server>
</ServerCluster>
```

There are several methods available to start the IBM HTTP Server, such as using a service entry in the Windows Services or using a newly created IBM HTTP Server menu item, or from the command line. After starting up the IBM HTTP Server, visit the home page of the Web server via the URL http://helium/, as shown in Figure 5-3.

![IBM HTTP Server Welcome Page](image)

You could also verify whether or not the Web server plug-in is successfully loaded by checking the plug-in log file, which is designated in the plug-in configuration file (generally named plugin-cfg.xml). Messages like Plugins loaded as well as system information, should be written into the log file if the plug-in module is loaded by the IBM HTTP Server.
Moreover, in order to verify if the HTTP or HTTPS request could be redirected to the Web container HTTP Transport listeners of the target application server from the Web server plug-in, you could also try to access the hello servlet via the URL http://helium/hello, as Figure 5-4 shows.

<table>
<thead>
<tr>
<th>Client Type</th>
<th>Data Type Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td>HTML</td>
</tr>
<tr>
<td>Speech</td>
<td>VXML</td>
</tr>
<tr>
<td>Wireless</td>
<td>WML</td>
</tr>
</tbody>
</table>

This servlet demonstrates the ability to develop a Pervasive Computing solution using the WebSphere Application Server. It can be called from any of the following types of clients:

Up to this point, we have completed the installation for all the listed software products, including Web server, application server, and database server, as well as their corresponding fix packs on different machines. This installation is commonly used in our sample application testing scenario. The installation and testing about the sample application is covered in detail in “DB2 EJB sample application setup” on page 153.

5.1.3 Using DB2 for z/OS and OS/390 as the Data Source for WAS

You need a powerful e-business infrastructure that lets you leverage your data, deliver customer values, and maintain your competitive edge in the market. IBM delivers the power you need to keep your e-business competitive without compromising the safety, integrity, and availability of your data.

DB2 UDB for z/OS provides the basic foundation of an e-business infrastructure. It is an obvious choice when it comes a question of robustness, reliability, scalability, and open standards. Also there is a great demand to integrate the data on DB2 UDB for z/OS with applications running on Windows and UNIX workstations. DB2 Connect enables local and remote client applications to manage DB2 databases and host systems using Structured Query Language (SQL), DB2 APIs (Application Programming Interfaces), ODBC (Open Database Connectivity), JDBC (Java Database Connectivity), SQLJ (Embedded SQLJ for Java), or DB2 CLI (Call Level Interface). In addition, DB2 Connect supports Microsoft Windows data interfaces such as ActiveX Data Objects (ADO), Remote Data Objects (RDO), and Object Linking and Embedding (OLE) DB.
We have already discussed in “DB2 connectivity to z/OS and S/390®” on page 112, the most common ways to connect a Java application running on a distributive environment to the DB2 UDB for z/OS subsystem is using the Type 2 driver and Type 4 driver.

When using the Type 4 driver, we do not need DB2 Connect to access data from the DB2 UDB for z/OS system, but from a performance point of view we are demonstrating our examples using DB2 Connect and Type 2 driver. Also the DB2 Connect provides certain functionality that is not provided by the Type 4 driver. These include sysplex awareness and connection concentration. These functions can be very valuable in large installations. For a more detailed description of these features, please see Distributed Functions of DB2 for z/OS and OS/390, SG24-6952.

This section is especially dedicated to providing considerations and recommendations while your applications are running on a distributed environment and using DB2 UDB for z/OS as the data source.

**Specific considerations for DB2 UDB for z/OS**

Before starting work with DB2 Connect, we must ensure that DB2 UDB for z/OS is configured to accept requests from a DB2 Connect client. This section describes the minimum updates required for DB2 Connect client to establish a connection to the DB2 UDB for z/OS.

Following are the steps to configure DB2 UDB for z/OS server to start allowing connections from DB2 Connect clients:

- Configure TCP/IP for DB2 UDB for z/OS.
- Configure DB2 UDB for S/390.

**Configure TCP/IP for z/OS**

Below are the steps required to define connectivity with DB2 UDB for z/OS:

1. TCP/IP communications must be enabled on DB2 Universal Database for z/OS and the partner system.

2. Two TCP/IP port numbers must be assigned. One port is for DRDA and another for re-synchronization for two-phase commit implementation. In this book we are using port number 38050 for database connections, and port number 38051 for re-synchronization.

3. The remote DRDA application server or application requester must use the same port numbers (or service names) as DB2 Universal Database for z/OS.

4. Ensure that the TCP/IP already verified security option is set to YES. Using this option you can specify extended security, which will provide additional diagnostics, such as PASSWORD EXPIRED.
Configure DB2 UDB for z/OS
For the application server to receive distributed database requests over TCP/IP connections, it must be defined to the local TCP/IP subsystem, and have a unique RDB_NAME. Additionally, the DB2 UDB for z/OS Bootstrap Dataset must include the necessary parameters, and you may need to make updates to the DB2 UDB for OS/390 and z/OS Communications Database (CDB).

No CDB updates are required if you will only use inbound database connections, so that if you plan to use DB2 UDB for z/OS only as a server, you do not need to populate the CDB, and default values can be used.

For further details please check with DB2 Connect Connectivity Supplement - db2h1e80.

Specific considerations for DB2 Connect
The IBM DB2 Connect family provides various product offerings like DB2 Connect Personal Edition, DB2 Connect Enterprise Edition, DB2 Connect Application Server Edition and DB2 Connect Unlimited Edition. Before using DB2 Connect you need to understand the different licensing policies, their usability, limitations, and platforms on which they are compatible.

Also, DB2 UDB Enterprise Server Edition includes the DB2 Connect component, but the below-mentioned restriction still applies:

In DB2 Universal Database Enterprise Server Edition (ESE), use of the DB2 Connect component is limited to five registered users per server. This product contains a DB2 Connect component to provide connectivity, DB2 for z/OS, DB2 Server for VSE and VM, and DB2 for iSeries database servers. If additional users need to connect to any of the above-mentioned database servers, a separate DB2 Connect Program license must be acquired. For further details check with DB2 Product Offerings at: http://www-306.ibm.com/software/data/offers/db2.html

Product configuration briefing
You can configure your TCP/IP connection between a DB2 Connect server and DB2 for z/OS subsystem manually through commands or by using the tool Configuration Assistant (CA). Here we are implementing connectivity using the manual option.
To manually configure TCP/IP communications between your DB2 Connect server and a DB2 for z/OS subsystem:

1. Configure TCP/IP on the DB2 Connect server.
   a. Resolve the local host system’s IP address. The DB2 Connect server must know the address of the DB2 for z/OS subsystem to which it is attempting to establish communications. To resolve the IP address, use a text editor to add an entry to the DB2 Connect server’s hosts file for the DB2 for z/OS subsystem system’s host name.
   a. Update the services file. Skip this step if you are planning to catalog a TCP/IP node using a port number (port_number). You need to update the DB2 Connect server’s services file to add the connection service name and port number of the remote host you want to connect to. To update the services file, use a text editor to add the connection service name and port number of the remote host to the DB2 Connect server’s services file.

2. Catalog the TCP/IP node.
   You must add an entry to the DB2 Connect server’s node directory to describe the remote node. This entry specifies the chosen node_name, the host name or IP_address, and the service name or port_number that the client will use to access the remote host.

3. Catalog the DB2 for z/OS subsystem as a Database Connection Service (DCS) database.
   The remote database must be cataloged as a DCS database so that DB2 Connect can provide access to it.

4. Catalog the DCS database.
   Before a client application can access a remote database, the database must be cataloged on the DB2 for z/OS subsystem node and on any DB2 Connect server nodes that will connect to it.

5. Bind utilities and applications to the DB2 for z/OS subsystem.
   After completing the steps to configure the DB2 Connect server to communicate with the DB2 for z/OS subsystem, one must bind the database utilities and DB2 CLI bind files to each database before they can be used. If you need to access multiple databases in a network environment, you must bind the utilities once for each operating system and DB2-version combination. The `bind` command creates a package, which provides all the information that is needed to process specific SQL statements from a single source file. The bind files are grouped together in different .lst files in the `$HOME/sqllib/bnd`, under the installation directory. Each file is specific to a server, for example, `ddcsmvs.lst` is for DB2 for z/OS.

6. Test the connection.

   For our operational environment we are using DB2 Connect Enterprise Edition V8. We are implementing it by installing both DB2 Runtime Client and DB2
Connect EE on a single pSeries server, Kanaga. Also we are assuming that DB2 Connect EE and DB2 UDB for z/OS system are already installed and in place. Here we only give emphasis on the steps related to connectivity. It is always advisable to collect information as described in Table 5-3 before starting the connectivity procedure.

**Table 5-3  DB2 Host environment for DRDA connectivity**

<table>
<thead>
<tr>
<th>DB2 Connect Machine - description</th>
<th>DB2 Connect Machine - value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage/Purpose</td>
<td>Host environment</td>
</tr>
<tr>
<td>Machine Type</td>
<td>zSeries—Sysplex of 16 systems</td>
</tr>
<tr>
<td>Operating System</td>
<td>z/OS</td>
</tr>
<tr>
<td>Host Name</td>
<td>wtscnet.itso.ibm.com®</td>
</tr>
<tr>
<td>Host IP Address</td>
<td>9.12.6.50</td>
</tr>
<tr>
<td>Pre Installed Product</td>
<td>DB2 V8 Subsystem</td>
</tr>
<tr>
<td>DB2 Subsystem Name</td>
<td>wtsc67oe.itso.ibm.com</td>
</tr>
<tr>
<td>DB2 Subsystem IP</td>
<td>9.12.6.42</td>
</tr>
<tr>
<td>DB2 Location Name</td>
<td>DBC8</td>
</tr>
<tr>
<td>DRDA Port</td>
<td>38050</td>
</tr>
<tr>
<td>Resync Port</td>
<td>38051</td>
</tr>
</tbody>
</table>
Figure 5-5 shows an overall operation setup for a DB2 Connect environment and Table 5-3 on page 131 has the details of the parameters required for DB2 UDB for z/iOS connectivity. Example 5-5 shows the commands we used to establish the connectivity using the steps defined previously.

Example 5-5  Cataloging the DB2 UDB z/iOS and S/390 subsystem

Kanaga:/home/db2inst1 >vi /etc/hosts
9.12.6.42  wtsc67oe.itso.ibm.com
"/etc/hosts"  41 lines, 1437 characters

Kanaga:/home/db2inst1 >db2 catalog tcip node wtsc67oe remote
wtsc67oe.itso.ibm.com Server 38050
DB20000I  The CATALOG TCPIP NODE command completed successfully.
DB21056W  Directory changes may not be effective until the directory cache is refreshed.

Kanaga:/home/db2inst1 >db2 catalog db dbc8 at node wtsc67oe authentication dcs

---
5.1.4 Installation troubleshooting

If you failed to install DB2 UDB V8 on the AIX platform, you could refer to `db2setup.log` generated under the /tmp directory during installation, then correct the problem indicated in the `db2setup.log` and retry the installation. If the messages are not complete to identify the cause of the issue, the installation trace information could be generated by using the parameter -t for DB2 Setup Wizard. For more information, please refer to Chapter 17, “b2setup - Install DB2” section, “Additional reference topics” in the book Quick Beginnings for DB2 Servers, SC09-4836.

Similarly, if you encounter problems during installation of WebSphere Application Server V5, a temporary log file named `log.txt` can be found in the /tmp directory on an AIX platform. If the information that is contained in the log.txt file is not sufficient to determine the cause of the problem, then further details about the installation procedure could be captured by using the additional installation logging or getting the stdout and stderr output reported on the console, etc. For detailed parameters of WebSphere Application Server installation utility to gather more information for installation troubleshooting, refer to Chapter 3, Troubleshooting the Installation section, Installing WebSphere Application Server in the book Getting Started, IBM WebSphere Application Server, Version 5.
There is a common port conflict issue that can happen when you run ivt.sh the first time to verify the WebSphere Application Server on the AIX 5L platform. It leads to the failure of starting up the application server, as shown in the Example 5-6.

Example 5-6  Port Conflict during the installation verification test

Kanaga:/usr/WebSphere/AppServer/bin >./ivt.sh
IVTL0035I: Scanning the file
/usr/WebSphere/AppServer/logs/server1/SystemOut.log for errors and warnings
[10/30/03 14:45:42:955 PST] 67c3b127 WsServer   E WSVR0003E: Server server1
failed to start
[10/30/03 14:45:42:969 PST] 67c3b127 WsServer   E WSVR0009E: Error occurred
during startup
[10/30/03 14:45:29:744 PST] 67c3b127 WebContainer E SRVE0146E: Failed to Start
Transport on host , port 9090. The most likely cause is that the port is
already in use. Please ensure that no other applications are using this port
and restart the server. com.ibm.ws.webcontainer.exception.TransportException:
Failed to start transport http: java.net.BindException: The socket name is
already in use.
[10/30/03 14:45:27:482 PST] 67c3b127 FreePool   E J2CA0046E: Method
createManagedConnctionWithMCWrapper caught an exception during creation of the
ManagedConnection for resource JMS$SampleJMSQueueConnectionFactory, throwing
ResourceAllocationException. Original exception:
javax.resource.spi.ResourceAdapterInternalException: javax.resource.spi.ResourceAdapterInternalException: createQueueConnection
failed
IVTL0040I: 5 errors/warnings were detected in the file
/usr/WebSphere/AppServer/logs/server1/SystemOut.log

This is a common problem due to the conflicting port use on port 9090. If you install on an AIX 5.1 system, with maintenance level 2 or later, it is possible that the Web-based system manager, a standard component of AIX systems, already uses port 9090. When starting the server you get information that port 9090 is already in use. You could use the command netstat to see if port 9090 is used by other applications, as Example 5-7 shows.

Example 5-7  The conflicting port 9090 on AIX

Kanaga:/usr/WebSphere/AppServer/bin >netstat -an | grep 9090
tcp4  0 0  *:9090 *:* LISTEN
Kanaga:/usr/WebSphere/AppServer/bin >cat /etc/services | grep wsm
wmsserver      9090/tcp
Here it happens that the service entry for port 9090 could be found in the /etc/services file, and we know that the wsmserver is controlled by the Internet Service Management daemon inetd. If you want to use port 9090 for WebSphere Application Server but not other applications, then you could stop the corresponding application if allowed or change the port usage of the conflicting application. In our case, we stopped the web-based system manager by:

1. Commenting the line associated with \texttt{wsm} in /etc/inetd.conf, which is the configuration file of inetd.

2. Refreshing inetd process to read in the new configuration via the command:
\begin{verbatim}
kill -1 <pid of inetd>
\end{verbatim}

3. Recheck the wsmserver process. If it is still there, use the \texttt{kill} command to remove it out of the process table.

\textbf{Tip}: If you do not know which application process is occupying the desired port number (not just port 9090), and that port number is required by your installation, the \texttt{lsof} utility is very useful for this case. This utility could be used to list which process is using the desired network port or ordinary file and directory, etc. The \texttt{lsof} utility can be downloaded from the URL:
\begin{verbatim}
ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof
\end{verbatim}

Certainly, to give up using port 9090 for WebSphere Application Server is also a method to resolve the conflicting port use. The port assignment entry for transports the HTTP\_TRANSPORT\_ADMIN port in the server.xml file could be changed to a non-conflicting port number to avoid the conflict. In our test case, the file path of server.xml is as follows:

\begin{verbatim}
/usr/WebSphere/AppServer/config/cells/kanaga/nodes/kanaga/servers/server1/server.xml
\end{verbatim}

5.2 Creating and configuring DB2 JDBC Provider

IBM DB2 UDB family products are seamlessly integrated with WebSphere Application Server. As mentioned in “DB2 JDBC providers” on page 107, DB2 UDB V8 supports the legacy JDBC Type 2 driver and the universal JDBC driver that combines Type 2 and Type 4 JDBC implementations. In addition, DB2 UDB V8 also supports a Type 3 driver, although this driver is deprecated.

For more information about DB2 UDB’s support for JDBC and SQLJ, please refer to the DB2 UDB V8 application development support home page for Java at:

\begin{verbatim}
\end{verbatim}
Here we describe some of the major DB2 JDBC providers supported in WebSphere Application Server V5. We also show the supported data source classes and their required properties.

**DB2 Universal JDBC Driver Provider**
The DB2 Universal JDBC Driver is an architecture-neutral JDBC driver for distributed and local DB2 access. Because the Universal Driver architecture is independent of any particular JDBC driver connectivity or target platform, it allows both Java connectivity (Type 4) or Java Native Interface (JNI) based connectivity (Type 2) in a single driver instance to DB2. Starting with Version 5.0.2, the Websphere Application Server product now supports both Type 2 and Type 4 JDBC drivers. To use the Type 4 driver, you must install DB2 Version 8.1 or a later version. To use the Type 2 driver, you must install DB2 Version 8.1 FixPak 2 or a later version.

The DB2 Universal JDBC Driver only supports one-phase transactions. This JDBC driver allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access.

The DB2 Universal JDBC Driver Provider supports one-phase data source:

`com.ibm.db2.jcc.DB2ConnectionPoolDataSource`

The following JDBC driver files are required to use the DB2 Universal JDBC Driver Provider:

- **db2jcc.jar**
  After you install DB2 UDB, you can find this jar file in the sqllib/java directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You should also remember to set the `DB2UNIVERSAL_JDBC_DRIVER_PATH` environment variable to point to the `db2jcc.jar` file if that WebSphere environment variable is used.

- **db2jcc_license_cu.jar**
  This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database on Linux, UNIX and Windows.

- **db2jcc_license_cisuz.jar**
  This is the DB2 Universal JDBC driver license file that allows access to the following databases:
  - DB2 Universal Database for Linux, UNIX and Windows
  - DB2 for iSeries
  - DB2 UDB for z/OS
The required DataStoreHelper class is as below:

com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper

The required properties for the DB2 Universal JDBC Driver Provider include:

- **databaseName**
  
  This is an actual database name if the driverType is set to 4, or a locally cataloged database name if the driverType is set to 2.

- **driverType**
  
  The JDBC connectivity type of a data source. There are two permitted values: 2 and 4. If you want to use Universal JDBC Type 2 driver, set this value to 2. If you want to use Universal JDBC Type 4 driver, set this value to 4.

- **serverName**
  
  The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. Provide a value for this property only if your driverType is set to 4. This property is not required if your driverType is set to 2.

- **portNumber**
  
  The TCP/IP port number where the DRDA server resides. Provide a value for this property only if your driverType is set to 4. This property is not required if your driverType is set to 2.

### DB2 Universal JDBC Driver Provider (XA)

The DB2 Universal JDBC Driver (XA) is an architecture-neutral JDBC driver for distributed and local DB2 access. In WebSphere Application Server Version 5.0.2, this driver only supports Java Native Interface (JNI) based connectivity (Type 2) in a single driver instance to DB2. To use this driver, you must install DB2 Version 8.1 FixPak 2 or a later version. This driver supports two-phase transactions and allows applications to use both JDBC and SQLJ access.

The DB2 Universal JDBC Driver Provider (XA) supports the two-phase data source:

com.ibm.db2.jcc.DB2XADataSource

The required JDBC driver files list and DataStoreHelper class to use the DB2 Universal JDBC Driver Provider (XA) are same as those required by DB2
Universal JDBC Driver Provider, you could refer to “DB2 Universal JDBC Driver Provider” on page 136 for more details.

The required properties for the DB2 Universal JDBC Driver Provider (XA) include:

- **databaseName**
  
  This is a locally cataloged database name.

- **driverType**
  
  The JDBC connectivity type of a data source. The *only* permitted value is 2.

### DB2 legacy CLI-based Type 2 JDBC Driver

The DB2 legacy CLI-based Type 2 JDBC Driver Provider is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers.

DB2 legacy CLI-based Type 2 JDBC Driver supports one-phase data source:

`COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource`

The required driver file to use the DB2 legacy CLI-based Type 2 JDBC Driver is `db2java.zip`. (Note: If you run SQLJ in DB2 UDB V8, `db2jcc.jar` is also required.)

The required DataStoreHelper class, as shown below, is different from the one required by DB2 Universal JDBC Driver Provider:

`com.ibm.websphere.rsadapter.DB2DataStoreHelper`

The databaseName is the only required property. databaseName is the name of the database from which the data source obtains connections.

### DB2 legacy CLI-based Type 2 JDBC Driver (XA)

The DB2 legacy CLI-based Type 2 JDBC Driver (XA) is built on top of DB2 CLI. It uses the DB2 CLI interface to communicate with DB2 UDB servers.

DB2 legacy CLI-based Type 2 JDBC Driver (XA) supports two-phase data source:

`COM.ibm.db2.jdbc.DB2XADatasource`

The required driver files, DataStoreHelper class and the properties are all the same as those required by DB2 legacy CLI-based Type 2 JDBC Driver.

In addition to the DB2 JDBC providers covered above, WebSphere Application Server V5 also support other DB2 JDBC providers such as DB2/390 JDBC...
Providers and DB2 UDB for iSeries JDBC Providers (including non-XA and XA support). For the more complete DB2 JDBC providers supported by WebSphere Application Server V5, please refer to the section Vendor-specific data sources minimum required settings in WebSphere Application Server V5 InfoCenter.

5.2.1 The steps to create and configure DB2 JDBC Providers

In this section the general steps to create and configure DB2 JDBC Providers in the WebSphere Application Server V5 environment are provided. Creating the JDBC Provider is a prerequisite to access a database. For WebSphere Application Server V5, we could create and configure a JDBC provider using the administrative console or using the Java Management Extensions API. In the example shown below, we use the WebSphere Application Server Administrative Console to create the JDBC Providers. The basic steps are shown as below:

1. Log into WebSphere Application Server Administrative Console via the URL http://kanaga:9090/admin, then the Console login page, as shown in Figure 5-2 on page 124, is displayed. As the security is disabled by default, here you could input a user ID without a password, and the user ID does not need to be a user in the local user registry. It is only used to track user-specific changes to configuration data.

2. After logging into the administrative console, click Resources -> JDBC Providers, then choose the Scope of your definition and click New to create a new JDBC Provider, as the Figure 5-6 on page 140 illustrates.
3. Use the drop-down list to select the type of JDBC provider. If your desired JDBC provider is not displayed in the list of supported DB2 JDBC provider types, you could also select the User-Defined JDBC Provider to create your own JDBC Provider. In our case, the existing DB2 Universal JDBC Driver Provider is chosen for demonstration purposes (see Figure 5-7 on page 141).
4. Enter the properties for your DB2 JDBC provider.

   As illustrated in the Figure 5-8 on page 142, you could provide a name for the JDBC provider, specify a list of paths or JAR file names, which together form the location for the resource provider classes in the classpath block. Be aware that classpaths could contain variable (symbolic) names, which you can substitute using a variable map. In our case, the variable DB2UNIVERSAL_JDBC_DRIVER_PATH is used. We need to define it before running an application with the JDBC Provider. Regarding how to manage variables in WebSphere Application Server environment, refer to the next step for more details. Click OK to return to the JDBC providers page, where your new JDBC provider appears in the list.
5. Apply and save your changes. As Figure 5-9 on page 143 displays, we can find that the changes we just made are saved to the file named resources.xml, which is located under

<USER_INSTALL_ROOT>/cells/kanaga/nodes/kanaga subdirectory, where USER_INSTALL_ROOT is the home directory for the WebSphere Application Server V5 installation.
6. Define required WebSphere Variable.

   Click **Environment -> Manage WebSphere Variables** (as shown in Figure 5-10), and scroll down to the desired variable. If it has been there then assign the value for it, or you could click **New** to create a new WebSphere variable if the required variable has not yet been shown in the list.

Up to this point, the DB2 Universal JDBC Provider has been created and ready to create a DB2 Data Source on top of that.
5.3 Creating and configuring DB2 Data Source

An application uses a data source to access the data in the database. A data source is associated with a JDBC provider that supplies the specific JDBC driver implementation class. The data source represents the J2EE Connector Architecture (JCA) connection factory for the relational resource adapter.

You can create multiple data sources associated with the same JDBC provider. Each JDBC provider supports the interfaces defined by Sun Microsystems listed below. These interfaces enable the application to run in a single-phase or two-phase transaction protocol.

- ConnectionPoolDataSource
  A data source that supports application participation in all transactions, including two-phase commit transactions. When this kind of data source is involved in a global transaction, transaction recovery is not provided by the transaction manager. The application is responsible for providing the backup recovery process if multiple resource managers are involved.

- XADataSource
  A data source that supports application participation in a single-phase or a global (two-phase) transaction environment. When this data source is involved in a global transaction, the transaction manager provides transaction recovery.

5.3.1 The steps to create and configure DB2 Data Source

In this section the general steps to create and configure DB2 Data Source in WebSphere Application Server V5 environment are provided. For WebSphere Application Server V5, we could create and configure a DB2 Data Source using the administrative console or using the Java Management Extensions API. In the example shown below, we use the WebSphere Application Server Administrative Console to create the Data Source. The following shows the general steps:

1. Prepare the J2C Authentication Data Entry for the Data Source
   
   The authentication data entry is generally required for database authentication in run time. You could define the user ID and password in the custom properties of a data source. But possibly it is not desirable as the password can be seen by anyone who accesses the resources.xml file. By defining J2C authentication data entry as an alias, obtaining directly visible password from resources.xml can be avoided. In addition, Each alias can be reused by many Data Source.

   To define a new alias for the J2C Authentication Data Entries, choose Security -> JAAS Configuration -> J2C Authentication Data. Click New on
the J2C Authentication Data Entries page then fill in the fields on the resulting page. An example is shown in the Figure 5-11.

2. Choose data source type based on the desired DB2 JDBC Provider

There are two types of data sources available in WebSphere Application Server V5, which represent two types of separate connection manager (CM) architectures:

– Data Sources: This data source uses the JCA standard architecture to provide J2EE 1.3 support. It runs under the JCA connection manager and the relational resource adapter. Applications using this type of data source might behave differently because of the J2EE 1.3 architecture.

– Data Sources (Version 4.0): This data source runs under the CM architecture. Applications using this data source behave as if they were running in Version 4.0.

In our case, we choose Data Sources for J2EE 1.3 support. Click Resources -> JDBC Providers, then click the previously defined JDBC Providers in the list of the resulting page. In our case, it is My DB2 Universal JDBC Provider. Scrolling down to the bottom of the new page then choose Data Sources.
3. Create a new DB2 Data Source

Follow on the previous step. Click **New** to create a new data source in the Data Sources page, then enter properties for your data source, as shown in the Figure 5-12 on page 148.

Please be aware that additional properties for this new data source will not be available to edit until you click **Apply** or **OK**.

The following contains descriptions for the general properties of the data source:

- **Scope**
  Specifies the level to which this resource definition is visible—the cell, node, or server level.

- **Name**
  Specifies the display name for the data source.

- **JNDI Name**
  Specifies the Java Naming and Directory Interface (JNDI) name.
  If you leave this field blank, a JNDI name is generated from the name of the data source. For example, a data source name of markSection generates a JNDI name of jdbc/markSection.

**Note:** After you set this value, save it, and restart the server, you can see this string when you run dumpnamespace.

- **Container managed persistence**
  Specifies if this data source is used for container managed persistence of enterprise beans.
  If the checkbox is selected, a CMP Connector Factory that corresponds to this data source is created for the relational resource adapter.

- **Description**
  Specifies a text description for the resource.

- **Category**
  Specifies a category string if you want to use it to classify or group the resource.

- **Statement Cache Size**
  Specifies the number of free statements that are cached *per connection*.
  The WebSphere Application Server data source optimizes the processing of prepared statements. A prepared statement is a precompiled SQL
statement that is stored in a prepared statement object. This object is used to efficiently execute the given SQL statement multiple times.
Data Source is used by the application to access the data from the database. A data source is created under a JDBC provider which provides the specific JDBC driver implementation class.

**Figure 5-12 Creating New DB2 Data Source**
If the cache is not large enough, entries are discarded to make room for new entries. To determine the largest value for your cache size to avoid any cache discards, add the number of uniquely prepared statements, callable statements (as determined by the SQL string, concurrency, and the scroll type) for each application that uses this data source on a particular server. This value is the maximum number of possible prepared statements that are cached on a given connection over the life of the server. Setting the cache size to this value means you never have cache discards. In general, the more statements your application has, the larger the cache should be. For example, if the application has five SQL statements, set the statement cache size to 5, so that each connection has five statements.

- **Datasource Helper Classname**
  Specifies the datastore helper that is used to perform database specific functions. Accept the default value.

- **Component-managed Authentication Alias**
  This alias is used for database authentication in run time. If your resource authentication (res-auth) is set to Application, set the alias in the Component-managed Authentication Alias.

- **Container-managed Authentication Alias**
  This alias is used for database authentication in run time. If your res-auth is set to Container, set the Container-managed Authentication Alias. In our case, container-managed authentication is used, and previously created alias for J2C Authentication Data Entry is chosen here.

- **Mapping-Configuration Alias**
  The DefaultPrincipalMapping JAAS configuration maps the authentication alias to the userid and password. You may define and use other mapping configurations.

After applying the above changes, then choose or click **Resources -> JDBC Providers -> My DB2 Universal JDBC Provider** (previously created) -> **Data Sources**. The newly created data source is displayed in the resulting page.

4. **Specify additional properties for the new data source**

Now it is ready to specify additional properties for the new data source, such as database name, driver type, server name or TCPIP address and port number, etc. As a follow-up of the above step, click the new DB2 data source name within the list in the Data Sources page, then scroll down to the bottom of the resulting page. The Additional Properties block and Related Items block are displayed, as shown in the Figure 5-13 on page 150.
Here you could specify properties of the connection pool, such as connection timeout, maximum and minimum number of managed connections, and so forth, by clicking Connection Pool. For more information about the meanings of the parameters for a specific data source, please refer to the topic “Connection Pool Setting” in WebSphere Application Server V5 InfoCenter. In our case, we just use the default values for the data source initially.

To specify custom properties of the DB2 data source, click Custom Properties, then you could change those properties in the resulting page as shown in Figure 5-14 on page 151.
In our case, the sample database shipped with DB2 UDB V8 is used. It is located on the machine Atlantic which is an AIX server. The port number is 50000. We use the Type 4 driver of the DB2 Universal JDBC Driver Provider and enabled SQLJ support.

The following contains descriptions for some of the custom properties for the data source:

- **databaseName**
  
  This is a required property. For Type 4 driver, this is an actual database name, but not the locally catalogued database alias. For Type 2 driver, it could be the actual database name if the associated serverName and portNumber where the database resides are also provided. If serverName and portNumber is not available for Type 2 driver, then use a locally catalogued database alias alone for this property is also working.
- **driverType**
  This is a required property. It stands for the JDBC connectivity-type of a data source. If you want to use type 4 driver, set the value to 4. If you want to use type 2 driver, set the value to 2.

- **serverName**
  The TCP/IP address or host name for the DRDA server. If custom property driverType is set to 4, this property is required.

- **portNumber**
  The TCP/IP port number where the DRDA server resides. If custom property driverType is set to 4, this property is required.

- **enableSQLJ**
  This value is used to indicate whether SQLJ operations may be performed with this data source. If enabled, this data source can be used for both JDBC and SQLJ calls. Otherwise, only JDBC usage is permitted.

- **description**
  The description of this datasource.

- **traceLevel**
  The DB2 trace level for logging to the logWriter or trace file. Possible trace levels are:
  - `TRACE_NONE = 0`
  - `TRACE_CONNECTION_CALLS = 1`
  - `TRACE_STATEMENT_CALLS = 2`
  - `TRACE_RESULT_SET_CALLS = 4`
  - `TRACE_DRIVER_CONFIGURATION = 16`
  - `TRACE_CONNECTS = 32`
  - `TRACE_DRDA_FLOWS = 64`
  - `TRACE_RESULT_SET_META_DATA = 128`
  - `TRACE_PARAMETER_META_DATA = 256`
  - `TRACE_DIAGNOSTICS = 512`
  - `TRACE_SQLJ = 1024`
  - `TRACE_ALL = -1`

- **traceFile**
  The trace file to store the trace output.

Besides the properties listed above, there are still a few more other properties that impacts the connection behaviors to the DB2 data source, such as resultSetHoldability to control cursor behavior when committing a transaction; readOnly to control if the connection is read only or not, etc. For more complete descriptions for the DB2 Universal JDBC Driver Provider properties,
refer to the data source custom properties page in the WebSphere Application Server Administrative Console, or visit the following URL:

5. Test the connection to the newly defined data source

After applying and saving the changes made in the above steps, then click or choose Resources -> JDBC Providers -> My DB2 Universal JDBC Provider -> Data Sources. In the resulting page, you could begin to test the connection to the data source by checking the check box preceding to the desired data source name, then click the button Test Connection, as shown in the Figure 5-15.

![Figure 5-15 Test Connection to the Data Source](image)

If the testing is successful, then a window like Figure 5-16 would be displayed. If the connection could not be established successfully, make sure the properties given are correct and working. There is a section “Connectivity scenario” on page 359 talking about the problem troubleshooting for connectivity issues, you could refer to that for more information.

![Figure 5-16 Successful Connection Testing to the Data Source](image)

5.4 DB2 EJB sample application setup

In this section, we provide detailed steps regarding how to get the sample application up and running for demonstration purpose so that you could obtain a
basic understanding how WebSphere Application Server V5 and DB2 UDB V8 work together.

5.4.1 DB2 Enterprise JavaBean (EJB) sample application

There is a DB2 Enterprise JavaBean (EJB) sample application called AccessEmployee shipped with DB2 UDB V8. This program utilizes server-side business components known as EJBs to implement distributed applications using a component-based development model on top of the J2EE platform provided by WebSphere Application Server. The sample application is provided in a form of Enterprise Application Archive (EAR). You can find it under sqllib\samples directory if you have installed the samples. In our test case, the name of the EAR with complete path is as below (on AIX platform):

/home/db2inst1/sqllib/samples/java/Websphere/AccessEmployee.ear or
/usr/opt/db2_08_01/samples/java/Websphere/AccessEmployee.ear

On Windows platform, it can be found in:

<DB2PATH>\samples\java\Websphere\AccessEmployee.ear

Here <DB2PATH> is home directory of DB2 UDB installation.

Basically this DB2 EJB sample application provides four business services using the DB2 sample database:

- Create a new employee
  This service allows users to create a new employee record in the sample database. The methods in AccessEmployeeBean will take the necessary actions to create an employee EJB and its corresponding record in the employee table.

- Query an Employee's record
  This service provides users the ability to retrieve an employee's information from DB2. The servlet, AccessEmpServlet, delegates the user's request to AccessEmployee session bean. The getEmployeeInfo method in AccessEmployee then finds the employee entity bean, retrieves the employee data from the entity bean and sends the data back to the servlet.

- Query Employees by department and salary
  This service invokes the same AccessEmployee session bean. The getEmployeeByDept method will perform a qualified search of Employee records based on their department and a minimum salary. The findBySalaryDept method in the Employee EJB "EmployeeBean.java" uses CMR (Container-managed relationships) and an EJBQL (EJB Query Language) query to return a collection of qualifying Employee EJBs.
Chapter 5. Operational setup

5.4.2 Application installation & resource configuration

Before discussing the detailed installation steps, you could review our testing environment by reference to Table 5-1 on page 119. Basically, besides the products installation which is covered by “Products’ installation briefing” on page 119, we have the following tasks to set up the DB2 EJB Sample application:

- Database preparation
- Application preparation
- Application server preparation
- Application resource preparation
- Application installation
- Sample application testing

The following pages provide the details to carry out these tasks.

Database preparation

As the DB2 UDB database server resides on a machine separated from the machine where WebSphere Application Server resides, in order to use the Type 2 driver of DB2 Universal JDBC Driver Provider in our case, we need to catalog the remote database entry to the local machine (here we refer to kanaga machine where WebSphere Application Server resides as local machine and atlantic machine where DB2 UDB is installed as remote machine). In our scenario, we use the SAMPLE database shipped with DB2 UDB V8. The following example shows you how to catalog the remote database.

**Example 5-8  Remote Database Cataloging**

Kanaga:/ >su - db2inst1
As the Example 5-8 on page 155 shows, the basic steps to catalog a remote database consists of two steps. The first step is node cataloging and it could be skipped if there is matching node entry existing in the local node directory. The second is database cataloging. Here we use the database alias “RSAMPLE” on local machine to reference the “SAMPLE” database on remote machine which could be created by DB2SETUP Wizard or manual creation by using the command `db2sampl` via DB2 instance owner.

Note: It is not a requirement to catalog a local database entry for the remote database. The DB2 Universal JDBC Driver Provider, we could use the Type 2 driver without using a local database catalog entry by providing the combination of database name, server name and the port number.

In addition, as required by the EJB sample application, we set the column `empno` as the primary key for the employee table as below:

```
db2 "alter table employee add constraint emp_pk primary key(empno)"
```

This could be done remotely after establishing connection to the RSAMPLE database alias.

**Application preparation**

There is no DB2 EJB sample application archive on the Kanaga machine in our scenario as we only installed DB2 UDB Runtime Client in Kanaga. To install the sample application, we could copy it from Atlantic machine via ftp. In our sample, we copied it into the commonly used directory for application installation `/usr/WebSphere/AppServer/installableApps`. 
The sample application uses environment entry to reference database schema which by default is `db2admin`. But for our case it is `db2inst1`, so we need to change it before installing the sample application. We use Application Assembly Tool (AAT) that could be invoked by the command `assembly.sh` to modify the environment entry, `AccessEmp/DbSchema`, under the Environment entries for the Session bean AccessEmployee. The value should be `db2inst1` for our case, just as shown in the Figure 5-17. Remember to save the changes after setting the new value.

![Application Assembly Tool](image)

Figure 5-17 Using AAT to application archive properties

**Application server preparation**

Before installing the sample application, we need to start the WebSphere Application Server first, in our sample the server name is `server1`. The following example (Example 5-9) shows how to start the application server by using `startServer.sh` command.

**Example 5-9 Starting the Application Server**

```
Kanaga:/usr/WebSphere/AppServer/bin >./startServer.sh server1
ADMU0116I: Tool information is being logged in file
   /usr/WebSphere/AppServer/logs/server1/startServer.log
```
As the example indicates, when the message like “Server server1 open for e-business; process id is 9326” appears, it means the application server is started successfully. If your application has been started before, you could use `serverStatus.sh` command to verify if the server status is running as shown in Example 5-10.

**Example 5-10  Using serverStatus.sh to verify the Application Server is running**

```
Kanaga:/usr/WebSphere/AppServer/bin >./serverStatus.sh server1
ADMU0116I: Tool information is being logged in file
    /usr/WebSphere/AppServer/logs/server1/serverStatus.log
ADMU0500I: Retrieving server status for server1
ADMU0508I: The Application Server "server1" is STARTED
```

**Application resources preparation**

We use the DB2 Universal JDBC Driver Provider in our case. Regarding how to create JDBC Providers, you could refer to the section “The steps to create and configure DB2 JDBC Providers” on page 139 for more details. In that section, a DB2 Universal JDBC Driver Provider is created, we also use it for our sample application.

In addition to the JDBC Providers creation, we also need to create DB2 Data Sources for the application. In order to access the DB2 database resource, an J2C Authentication Data Entry named AccEmpAlias with the following properties is created in our case:

- **Alias**: kanaga/AccEmpAlias
- **User ID**: db2inst1
- **Password**: password
- **Description**: Demo entry for AccessEmployee

You could refer to the step 1 in the section “The steps to create and configure DB2 Data Source” on page 144 for more details regarding how to create J2C Authentication Data Entry.

Then we could create the DB2 Data Source for our sample application with the following general and custom properties.

**Table 5-4  General and custom Properties Setting for sample application**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>EJBSample</td>
<td>General</td>
</tr>
</tbody>
</table>
For other available properties, just use the default setting. For Custom type properties in Table 5-4 on page 158, you need to apply the changes made in general properties page first, then the custom properties could be modified.

The server may need to be restarted for all these changes to take effect. After the server is restarted, try to test connection and make sure the connection to the newly created DB2 data source could be established successfully. Regarding how to test the connection, refer to the step 5 in the section “The steps to create and configure DB2 Data Source” on page 144.

**Application installation**

The sample application is installed using these steps:

- In the WebSphere Application Server Administrative Console, expand Applications on the left pane.
- Choose Enterprise Applications.
- Click the install button in the resulting pane that is on the right side of the page.
- The “Preparing for the application installation” page is displayed, as shown in the Figure 5-18 on page 160. In our case, as the WebSphere Application Server Administrative Console is invoked on the browser of a client machine, so for the path where the enterprise application archive exists, we choose “Server path”, and the corresponding value is:

/usr/WebSphere/AppServer/installableApps/AccessEmployee.ear

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNDI Name</td>
<td>jdbc/Sample</td>
<td>General</td>
</tr>
<tr>
<td>Container managed persistence</td>
<td>Checked</td>
<td>General</td>
</tr>
<tr>
<td>Component-managed Authentication Alias</td>
<td>kanaga/AccEmpAlias</td>
<td>General</td>
</tr>
<tr>
<td>Container-managed Authentication Alias</td>
<td>kanaga/AccEmpAlias</td>
<td>General</td>
</tr>
<tr>
<td>databaseName</td>
<td>rsample</td>
<td>Custom</td>
</tr>
<tr>
<td>driverType</td>
<td>2</td>
<td>Custom</td>
</tr>
<tr>
<td>serverName</td>
<td>Null</td>
<td>Custom</td>
</tr>
<tr>
<td>portNumber</td>
<td>Null</td>
<td>Custom</td>
</tr>
<tr>
<td>enableSQLJ</td>
<td>true</td>
<td>Custom</td>
</tr>
</tbody>
</table>
Click Next. You see a page with a subtitle You can choose to generate default bindings and mappings. Keep the default settings and click the Next button.

Now you are in Step 1, “Install New Application” page; select “Deploy EJBs”. Next click Next to Step 2.

In Step 2, choose DB2UDB_V81 as the Deploy EJBs Option - DataBase Type, then click Next.

Then just keep the default settings, and keep clicking Next until you reach the Summary step, as the following Figure 5-19 on page 161 shows.
Chapter 5. Operational setup

Figure 5-19 Application Installation Summary Page

- Click Finish to install the sample, then click Save to Master Configuration in the resulting page to finish the installation.

Now AccessEmployee application is ready for the testing.

Sample application testing
Start the application by checking the AccessEmployee application in the Enterprise Applications page, then click Start to start the application.

When the application is successfully started, go to a Web browser and enter the following URL to access the application:


Then the DB2 EJB Sample application could be invoked as the following Figure 5-20 on page 162 shows.
Then you could use the available buttons and the hot link to do different testing such as query employee information, create a new employee, etc.

In the above example, the embedded HTTP server of the application server is used for serving the request from port number 9080 on the machine Kanaga, as indicated by the URL: http://kanaga:9080/AccessEmp/AccessEmployee.html.

In order to use a separate machine as the Web server to access the sample application, we need to add URI entry for the AccessEmployee application into the Web server plug-in file plugin-cfg.xml. For demonstration purpose, we use the machine Helium as the Web server, where the IBM HTTP Server has been installed and the plug-in file has been modified to redirect the HTTP and HTTPS requests to the embedded HTTP Server of the target WebSphere application server, as shown in the step Install IBM HTTP Server and Web Server Plugins of the section “Products’ installation briefing” on page 119. The plug-in file could be found under the directory:

C:\Program Files\WebSphere\AppServer\config\cells\
Edit this file and add the URI entry for the AccessEmployee application as shown in the Example 5-11. You could add the entry at the end of the UriGroup block (before the line </UriGroup>).

Example 5-11  Add URI Entry into plug-in file for AccessEmployee application

```xml
<UriGroup Name="server1_Cluster_URIs">
  <Uri Name="/snoop/**/>
  ...
  <Uri Name="/AccessEmp/**/>
</UriGroup>
<Route ServerCluster="server1_Cluster"
  UriGroup="server1_Cluster_URIs" VirtualHostGroup="default_host"/>
```

Restart IBM HTTP Server on the machine Helium for the modification to take effect, then the application should be reachable via the URL:

```
http://helium/AccessEmp/AccessEmployee.html
```

### 5.4.3 Using DB2 for z/OS as the Data Source

We could also use DB2 for z/OS as the data source of the DB2 EJB sample application through DB2 JDBC Providers. There are several types of DB2 JDBC Providers which support the connection to DB2 on z/OS, such as the legacy DB2 JDBC Provider, DB2/390 JDBC Provider, and DB2 Universal JDBC Driver Provider (including Type 2 and Type 4). The advantages and disadvantages of these two types is covered in previous DB2 Connect or DB2 for z/OS related sections. In this subsection, we take the Type 2 driver of DB2 Universal JDBC Driver Provider as an example to demonstrate how to make WebSphere Application Server V5 and DB2 for z/OS working together. As the intermediate layer, DB2 Connect is used to connect WebSphere Application Server with DB2 for z/OS.

To make use of the DB2 for z/OS as the data source for the DB2 EJB sample application via the Type 2 driver of DB2 Universal JDBC Driver Provider, basically the following steps should be covered:

1. DB2 for z/OS connection preparation
2. Application data preparation on DB2 for z/OS
3. Application preparation
4. Application resource preparation
5. Application installation and testing
As the way to perform most of the above steps has been covered by previous sections, so that here we would just focus on the distinctive parts, for example, data preparation for the application.

1. DB2 for z/OS connection preparation

   Please refer to the section “Using DB2 for z/OS and OS/390 as the Data Source for WAS” on page 127 for the details about how to establish the connection to DB2 for z/OS via DB2 Connect. We use the connection obtained in that step in the following steps, the related database alias is DBC8 in our example.

2. Application data preparation on DB2 for z/OS

   The DB2 EJB sample application uses two tables (EMPLOYEE and DEPAERTMENT) in the sample database which is available for DB2 UDB on distributed platforms. To prepare the data on DB2 for z/OS, you could create tables on DB2 for z/OS via the DDL obtained from the DB2 UDB on distributed platform, then export the data and import into the DB2 for z/OS. In addition, as the entity bean is used in the sample application, so it also requires a primary key on the employee table.

   **Tip:** You could use `db2look` utility to generate the table and index creation scripts. Use Integrated eXchangable Format (IXF) to export the data from the distributed platform then it could be imported into DB2 for z/OS easily.

   You could also use the following scripts, see Example 5-12, to create the tables and the required index on the DB2 for z/OS:

### Example 5-12  Data Preparation Scripts for DB2 EJB Sample Application

```sql
CONNECT TO DBC8 USER DB2WAS USING <password>;

-----------------------------------------------
-- DDL Statements for table "DB2WAS"."DEPARTMENT"
-----------------------------------------------
CREATE TABLE "DB2WAS"."DEPARTMENT"  (
  "DEPTNO" CHAR(3) NOT NULL ,
  "DEPTNAME" VARCHAR(29) NOT NULL ,
  "MGRNO" CHAR(6) ,
  "ADMRDEPT" CHAR(3) NOT NULL ,
  "LOCATION" CHAR(16) )
;

-----------------------------------------------
-- DDL Statements for table "DB2WAS"."EMPLOYEE"
-----------------------------------------------
```
CREATE TABLE "DB2WAS"."EMPLOYEE" (  
"EMPNO" CHAR(6) NOT NULL ,  
"FIRSTNME" VARCHAR(12) NOT NULL ,  
"MIDINIT" CHAR(1) NOT NULL ,  
"LASTNAME" VARCHAR(15) NOT NULL ,  
"WORKDEPT" CHAR(3) ,  
"PHONENO" CHAR(4) ,  
"HIREDATE" DATE ,  
"JOB" CHAR(8) ,  
"EDLEVEL" SMALLINT NOT NULL ,  
"SEX" CHAR(1) ,  
"BIRTHDATE" DATE ,  
"SALARY" DECIMAL(9,2) ,  
"BONUS" DECIMAL(9,2) ,  
"COMM" DECIMAL(9,2) ) ;

-- DDL Statements for primary key on Table "DB2WAS"."EMPLOYEE"

ALTER TABLE "DB2WAS"."EMPLOYEE"  
ADD CONSTRAINT "EMP_PK" PRIMARY KEY  
("EMPNO") ;

3. Application preparation

Change the database schema to your desired value. In our example, it is "DB2WAS". Refer to “Application preparation” on page 156 for details about how to change it via the AAT.

4. Application resource preparation

The J2C Authentication Entry data is also required. In our example, the user ID and password for the testing environment are provided for the alias AccZ0SAlias. Then we also need to create the JDBC provider and data source for the application. Refer to “Application resources preparation” on page 158 for more details about how to create JDBC Provider and Data Sources. In our example, the JDBC Provider previously created in the section “The steps to create and configure DB2 JDBC Providers” on page 139 is used. Similar to the Table 5-4 on page 158, the following properties are provided for the DB2 for z/OS data source creation.

Table 5-5  Properties Setting of DB2 for z/OS data source

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>MFSample</td>
<td>General</td>
</tr>
<tr>
<td>JNDI Name</td>
<td>jdbc/MFSample</td>
<td>General</td>
</tr>
<tr>
<td>Container managed persistence</td>
<td>Checked</td>
<td>General</td>
</tr>
</tbody>
</table>
5. Application installation and testing

If the application AccessEmployee has been existing in your WebSphere Application Server environment, you could remove it before reinstalling the application. To remove the application, first choose the AccessEmployee application in Enterprise Applications page in the WebSphere Application Server Administrative Console, then click **Stop** button to stop the application if its original state is **Started**, after that, click **Uninstall** button to remove it from the WebSphere Application Server installed application list.

Refer to “Application installation” on page 159 for more details about how to install the sample application. Specifically for the DB2 on z/OS data source, some changes need to be made based on the steps provided in “Application installation” on page 159:

- The value of property “Deploy EJBs Option - Database Type” in step 2
  The value DB2UDBOS390_V7 in the drop-down list is chosen. Currently, specific item like DB2UDBOS390_V8 is not yet available in WAS.

- The value of property “JNDI Name” in step 5
  As the default data source mapping for modules containing 2.0 entity beans, here jdbc/MFSample is used in our example.

- The value of property “JNDI Name” in step 8
  This is the step “Map resource references to resources “. Same as step 5, in our example the JNDI name is jdbc/MFSample.

After finishing the DB2 EJB sample application installation, then you could start the application for testing purpose as demonstrated in the “Sample application testing” on page 161. Please be aware that probably a restart of the WebSphere Application Server is required to make the above changes effective, if you have not done during the previous steps.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component-managed Authentication Alias</td>
<td>kanaga/AccZOSAlias</td>
<td>General</td>
</tr>
<tr>
<td>Container-managed Authentication Alias</td>
<td>kanaga/AccZOSAlias</td>
<td>General</td>
</tr>
<tr>
<td>databaseName</td>
<td>DBC8</td>
<td>Custom</td>
</tr>
<tr>
<td>driverType</td>
<td>2</td>
<td>Custom</td>
</tr>
<tr>
<td>serverName</td>
<td>Null</td>
<td>Custom</td>
</tr>
<tr>
<td>portNumber</td>
<td>Null</td>
<td>Custom</td>
</tr>
<tr>
<td>enableSQLJ</td>
<td>false</td>
<td>Custom</td>
</tr>
</tbody>
</table>
Up to this point, you should have acquired the general steps to get DB2 UDB and WebSphere Application Server working together by installing and configuring related products as well as installing and testing the DB2 EJB sample application. Also please be aware that basically this chapter is focusing on operations that covers the general steps to prepare the runtime environment. If you want to obtain more information about the DB2 UDB V8 and WebSphere Application Server V5 integrated performance tuning, please refer to Chapter 8, “DB2 UDB V8 and WAS V5 integrated performance” on page 287 of this redbook.
WebSphere Application Server V5 performance tuning

In this chapter we discuss tuning an existing WebSphere environment for performance and J2EE application best practices for performance. Performance tuning methodology and guidelines are also provided. We also cover the Performance Monitoring Infrastructure (PMI) and performance tuning and monitoring tools found in WebSphere Application Server V5.
6.1 Performance Monitoring Infrastructure

The Performance Monitoring Infrastructure (PMI) is a set of packages and libraries designed to assist with gathering, delivering, processing, and displaying performance data in WebSphere Application Server runtime components. In this section, we introduce the updated Performance Monitoring Infrastructure (PMI) in IBM WebSphere Application Server V5.

The PMI uses a client-server architecture. The server collects performance data from various WebSphere Application Server components. A client retrieves performance data from one or more servers and processes the data.

The PMI components and infrastructure have been extended and updated in WebSphere Application Server V5 to support the new management structure and to comply with the Performance Data Framework of the J2EE Management Specification.

PMI provides several types of interfaces to access performance data. A new JMX API is introduced in this version. Java Management Extensions (JMX) is a framework that provides a standard way of exposing Java resources (application servers, for example) to a system management infrastructure. The JMX framework allows a provider to implement functions, such as listing the configuration settings, and allows users to edit the settings. It also includes a notification layer that can be used by management applications to monitor events such as the startup of an application server.

The servlet and Java client interfaces are still available for compatibility with Versions 3.5.5+ and 4.0+. These PMI interfaces are used to create tools to help monitor and tune performance.

WebSphere Application Server V5 contains Tivoli Performance Viewer, a Java client that displays and monitors performance data.

PMI is composed of components for collecting performance data on the application server side and components for communicating between runtime components and between the clients and servers. The primary PMI components and related Management Beans (MBeans) are illustrated in Figure 6-1 on page 171.
Below we explain the above figure:

1. PMI Client API with its communications implementation, the PMI Collector. The PMI Client API initially contacts the Admin Service of the Deployment Manager (1a) to get a list of nodes, servers and MBeans for the entire cell.

2. PMI service consisting of PMI modules for collecting performance data and methods for instrumenting and retrieving the data from the runtime components. This service contacts the PMI modules upon application server startup, depending on the current configuration.

3. PerfMBean, a JMX management bean used to extract performance data from the PMI modules to the PMI Collector of the PMI Client API.

4. Extensions of the standard JMX MBeans (used for managing components and settings) to support management of performance data. This enables a JMX-based client to retrieve performance data.

The J2EE classes and PMI classes can use either the RMI over IIOP or SOAP protocol to communicate with the Admin Service. In a single-server environment the classes connect to the Admin Service of each individual application server in
order to collect performance data. In a Network Deployment environment, the client can choose to connect to the Deployment Manager first to retrieve a list of nodes, servers and MBeans in the cell. Performance data retrieval is subsequently performed in the same way for the two environments.

### 6.1.1 Performance data organization

Performance Monitoring Infrastructure (PMI) provides server-side monitoring and a client-side API to retrieve performance data. PMI maintains statistical data within the entire WebSphere Application Server domain, including multiple nodes and servers. Each node can contain one or more WebSphere Application Servers. Each server organizes PMI data into modules and submodules.

The Tivoli Performance Viewer, formerly the Resource Analyzer, organizes performance data in a centralized hierarchy of the following objects:

- **Node**
  - A node represents a physical machine in the WebSphere Application Server administrative domain.

- **Server**
  - A server is a functional unit that provides services to clients over a network. No performance data is collected for the server itself.

- **Module**
  - A module represents one of the resource categories for which collected data is reported to the performance viewer. Each module has a configuration file in XML format. This file determines organization and lists a unique identifier for each performance data in the module. Modules include enterprise beans, JDBC connection pools, J2C connection pool, Java Virtual Machine (JVM) runtime module (including Java Virtual Machine Profiler Interface (JVMPI)), servlet session manager, thread pools, transaction manager, Web applications, Object Request Broker (ORB), Workload Management (WLM), Web Services Gateway (WSGW), and dynamic cache.

- **Submodule**
  - A submodule represents a fine granularity of a resource category under the module. For example, ORB thread pool is a submodule of the thread pool category. Submodules can contain other submodules.

- **Counter**
  - A counter is a data type used to hold performance information for analysis. Each resource category (module) has an associated set of counters. The data points within a module are queried and distinguished by the Mbean ObjectNames or PerfDescriptors. Examples of counters include the number of active enterprise beans, the time spent responding to a servlet request and the number of kilobytes of available memory.
The Tivoli Performance Viewer allows users to view and manipulate the data for counters. A particular counter type can appear in several modules. For example, both the servlet and enterprise bean modules have a response time counter. In addition, a counter type can have multiple instances within a module. For example, in Figure 6-2, both the Enterprise beans module and Bean1 have an Avg Method RT counter.

Counters are enabled at the module level and can be enabled or disabled for elements within the module. For example, in Figure 6-2, if the Enterprise beans module is enabled, its Avg Method RT counter is enabled by default. However, you can then disable the Avg Method RT counter even when the rest of the module counters are enabled. You can also, if desired, disable the Avg Method RT counter for Bean1, but the aggregate response time reported for the whole module will no longer include Bean1 data.

Each counter has a specified monitoring level: None, low, medium, high or maximum. If the module is set to a lower monitoring level than required by a particular counter, that counter will not be enabled. Thus, if Bean1 has a medium monitoring level, Gets Found and Num Destroys are enabled because they
require a low monitoring level. However, Avg Method RT is not enabled because it requires a high monitoring level.

Data collection can affect performance of the application server. The impact depends on the number of counters enabled, the type of counters enabled, and the monitoring level set for the counters.

The following PMI modules are available to provide statistical data:

- Enterprise bean module, enterprise bean, methods in a bean
  Data counters for this category report load values, response times, and life cycle activities for enterprise beans. Examples include the average number of active beans and the number of times bean data is loaded or written to the database. Information is provided for enterprise bean methods and the remote interfaces used by an enterprise bean. Examples include the number of times a method is called and the average response time for the method. In addition, the Tivoli Performance Viewer reports information on the size and use of a bean objects cache or enterprise bean object pool. Examples include the number of calls attempting to retrieve an object from a pool and the number of times an object is found available in the pool.

- JDBC connection pools
  Data counters for this category contain usage information about connection pools for a database. Examples include the average size of the connection pool or number of connections, the average number of threads waiting for a connection, the average wait time in milliseconds for a connection, and the average time the connection is in use.

- J2C connection pool
  Data counters for this category contain usage information about the Java 2 Enterprise Edition (J2EE) Connector Architecture that enables enterprise beans to connect and interact with procedural back-end systems, such as Customer Information Control System (CICS) and Information Management System (IMS™). Examples include the number of managed connections or physical connections and the total number of connections or connection handles.

- Java Virtual Machine API (JVM)
  Data counters for this category contain memory used by a process as reported by Java Virtual Machine (JVM) run time. Examples are the total memory available and the amount of free memory for the JVM. JVM run time also includes data from the Java Machine Profiler Interface (JVMPI). This data provides detailed information about the JVM running the application server.

- Servlet session manager
Data counters for this category contain usage information for HTTP sessions. Examples include the total number of accessed sessions, the average amount of time it takes for a session to perform a request, and the average number of concurrently active HTTP sessions.

- **Thread pool**
  Data counters for this category contain information about the thread pools for Object Request Broker (ORB) threads and the Web container pools used to process HTTP requests. Examples include the number of threads created and destroyed, the maximum number of pooled threads allowed, and the average number of active threads in the pool.

- **Java Transaction API (JTA)**
  Data counters for this category contain performance information for the transaction manager. Examples include the average number of active transactions, the average duration of transactions, and the average number of methods per transaction.

- **Web applications, servlet**
  Data counters for this category contain information for the selected server. Examples include the number of loaded servlets, the average response time for completed requests, and the number of requests for the servlet.

- **Object Request Broker (ORB)**
  Data counters for this category contain information for the ORB. Examples include the object reference lookup time, the total number of requests, and the processing time for each interceptor.

- **Web Services Gateway (WSGW)**
  Data counters for this category contain information for WSGW. Examples include the number of synchronous and asynchronous requests and responses.

- **System data**
  Data counters for this category contain information for a machine (node). Examples include the CPU utilization and memory usage. Note that this category is available at node level, which means it is only available at NodeAgent in the multiple servers version.

- **Workload Management (WLM)**
  Data counters for this category contain information for workload management. Examples include the number of requests, the number of updates, and the average response time.
6.1.2 Performance data classification

The Performance Monitoring Infrastructure provides server-side data collection and client-side API to retrieve performance data. Performance data has two components: Static and dynamic.

The static component consists of a name, ID and other descriptive attributes to identify the data. The dynamic component contains information that changes over time, such as the current value of a counter and the time stamp associated with that value.

The PMI data can be one of the following statistical types defined in the JSR-077 specification (Figure 6-3 on page 177):

- CountStatistic
- BoundaryStatistic
- RangeStatistic
- TimeStatistic
- BoundedRangeStatistic

RangeStatistic data contains the current value, as well as lowWaterMark and highWaterMark.

In general, CountStatistic data require a low monitoring level and TimeStatistic data require a medium monitoring level. RangeStatistic and BoundedRangeStatistic require a high monitoring level.

There are a few counters that are exceptions to this rule. The average method response time, the total method calls, and active methods counters require a high monitoring level. The Java Virtual Machine Profiler Interface (JVMPI) counters, SerializableSessObjSize, and data tracked for each individual method (method level data) require a maximum monitoring level.
In previous versions, PMI data was classified with the following types:

- **Numeric**
  Maps to `CountStatistic` in the JSR-077 specification. Holds a single numeric value that can either be a long or a double. This data type is used to keep track of simple numeric data, such as counts.

- **Stat**
  Holds statistical data on a sample space, including the number of elements in the sample set, their sum, and sum of squares. You can obtain the mean, variance, and standard deviation of the mean from this data.

- **Load**
  Maps to the `RangeStatistic` or `BoundedRangeStatistic`, based on the JSR-077 specification. This data type keeps track of a level as a function of time, including the current level, the time that level was reached, and the integral of that level over time. From this data, you can obtain the time-weighted average of that level. For example, this data type is used in the number of active threads and the number of waiters in a queue.

These PMI data types continue to be supported through the PMI API. Statistical data types are supported through both the PMI API and Java Management Extension (JMX) API.

The `TimeStatistic` type keeps tracking many counter samples and then returns the total, count, and average of the samples. An example of this is an average count.
method response time. Given the nature of this statistic type, it is also used to track non-time related counters, like average read and write size. You can always call the `getUnit` method on the data configuration information to learn the unit for the counter.

In order to reduce the monitoring overhead, numeric and stat data are not synchronized. Since these data track the total and average, the extra accuracy is generally not worth the performance cost. Load data is very sensitive; therefore, load counters are always synchronized. In addition, when the monitoring level of a module is set to max, all numeric data are also synchronized to guarantee accurate values.

### 6.1.3 Enabling PMI service

In order to monitor a resource with Tivoli Performance Viewer or any PMI or JMX client, the PMI service of the application server associated with that resource has to be enabled. The PMI service can be enabled from the Performance Monitoring Service configuration pane in the Administrative Console or by using the `wsadmin` command interface.

#### Enabling PMI service using the administrative console

In order to enable the PMI service from the Administrative Console, open the Performance Monitoring Service properties configuration pane by using the following steps.

In the application server:

1. Expand the **Servers** folder from the navigation tree.
2. Click **Application Servers** from the Servers folder.
3. Click the name of your application server (for example, server1) from the list of application servers in the workspace.
4. Click the **Performance Monitoring Service** entry in the Additional Properties pane of the workspace. The Performance Monitoring Service properties configuration pane opens in the workspace.
5. To enable the PMI service of this application server, select the **Startup** check box.

In the node agent:

1. Open the administrative console.
2. Click **System Administration -> NodeAgents** in the console navigation tree.
3. Click **node_agent**.
4. Click **Performance Monitoring Service**.
5. Select the check box **Startup**.

6. (Optional) Select the PMI modules and levels to set the initial specification level field.

7. Click **Apply** or **OK**.

8. Click **Save**.

9. Restart the NodeAgent.

The changes you make will not take affect until you restart the NodeAgent.

**Enabling PMI service through wsadmin**

In order to configure the PMI service of a specific application server, a reference to the PMI service configuration object of that application server is needed. All PMI service configuration objects can be listed using the `wsadmin list PMIService` command (see Example 6-1).

---

**Example 6-1  List PMI service configuration objects**

```plaintext
C:\WebSphere\AppServer\bin>wsadmin
WASX7209I: Connected to process "dmgr" on node net1Manager using SOAP connector; The type of process is: DeploymentManager
WASX7029I: For help, enter: "$Help help"

wsadmin>$AdminConfig list PMIService

(cells/net1Network/nodes/node1/servers/CSVR1:server.xml#PMIService_1)
(cells/net1Network/nodes/node1/servers/CSVR2:server.xml#PMIService_1)
(cells/net1Network/nodes/node1/servers/nodeagent:server.xml#PMIService_1)
(cells/net1Network/nodes/node1/servers/server1:server.xml#PMIService_1)
(cells/net1Network/nodes/net1Manager/servers/dmgr:server.xml#PMIService_1)

wsadmin>
```

Each line of output contains the PMIService configuration ID that can be used for referencing the PMIService component of a specific application server.

To enable performance data monitoring, use the following `wsadmin modify` command with your specific PMI service configuration ID:

```plaintext
wsadmin> $AdminConfig modify \\
(cells/net1Network/nodes/node1/servers/server1: \\
server.xml#PMIService_1) {{enable true}}
```

The configuration needs to be saved before restarting the application server. Use the `wsadmin save` command to save the configuration:

```plaintext
wsadmin> $AdminConfig save
```
To restart the application server, use these `wsadmin` commands (in a single-server environment, do not specify the node in the `startServer` command):

```
wsadmin> $AdminControl stopServer server1
wsadmin> $AdminControl startServer {server1} {net1}
```

To disable performance data collection, use the following `wsadmin modify` command (save the configuration and restart the application server for the change to take effect):

```
wsadmin> $AdminConfig modify \n(cells/net1Network/nodes/node1/servers/server1: \n server.xml#PMIService_1) {{enable false}}
```

### 6.2 Performance monitoring and analysis tools

In this section we introduce the monitoring and tuning tools provided by WebSphere Application Server V5.

This section discusses the following topics:

- Tivoli Performance Viewer (formerly known as the Resource Analyzer)
- Performance monitoring servlet
- PMI Request Metrics
- Dynamic Cache Monitor
- Monitoring the IBM HTTP Server
- Other tools in this area, such as Log Analyzer and Tivoli Performance Advisors

#### 6.2.1 Tivoli Performance Viewer

The Tivoli Performance Viewer (TPV) is a stand-alone runtime performance monitor coming with WebSphere Application Server V5. It provides a GUI console that monitors and analyzes the performance data and is available on Windows and UNIX platforms. It can be used remotely and across platforms. The Tivoli Performance Viewer can record the information it collects and replay it without connecting to WebSphere Application Server.

The tool was formerly known as Resource Analyzer, but was renamed for the WebSphere Application Server V5 release.
Starting Tivoli Performance Viewer

On Windows platforms, to start Tivoli Performance Viewer, do one of the following:

- Select Start -> Programs -> IBM WebSphere -> Application Server v5.0 -> Tivoli Performance Viewer.
- Or use the command line:
  
  ```
  <WAS_HOME>/bin\tperfviewer.bat host_name port_number connector_type
  ```

On UNIX platforms:

Type into a shell:

```
<WAS_HOME>/bin/tperfviewer.sh host_name port_number connector_type
```

For example:

```
tperfviewer.bat localhost 8879 SOAP
```

Connector_type can be either SOAP or RMI. 8879 is the default ND port for the SOAP connector. 9809 is the default ND port for the RMI connector.

**Note:** The default port number varies based on the WebSphere software version in use. That is, the default port for WebSphere Network Deployment is 8879, while the default port for WebSphere Base is 8880. The reason for this is that in a Network Deployment environment, TPV connects to the Deployment Manager host rather than to the individual application server. You can find the default port number by looking for the configured port number using the Administrative Console or in the SystemOut.log file.

Setting performance monitoring level

The monitoring settings determine which counters are enabled. Changes made to the settings from the Tivoli Performance Viewer affect all applications using the Performance Monitoring Infrastructure (PMI) data.

Steps to set the level:

1. Choose the **Data Collection** icon on the Resource Selection panel. This selection provides two options on the Counter Selection panel. Choose the **Current Activity** option to view and change monitoring settings. Alternatively, use **File -> Current Activity** to view the monitoring settings.

2. Set monitoring levels by choosing one of the following options:
   - None: Provides no data collection
   - Standard: Enables data collection for all modules except enterprise bean method level data
– Custom: Allows customized settings for each module
These options apply to an entire application server.

3. (Optional) Fine tune the monitoring level settings.
   a. Click **Specify**. This sets the monitoring level to custom.
   b. Select a monitoring level.
      For each resource, choose a monitoring level of *None, Low, Medium, High* or *Maximum*. The dial icon will change to represent this level.

   **Note:** The instrumentation level is set recursively to all elements below the selected resource. You can override this by setting the levels for children after setting their parents.

4. Click **OK**.
5. Click **Apply**.

### 6.2.2 Enabling JVMI facility

The Java Virtual Machine Profiler Interface (JVMPI) is a facility of the JVM used to enable a more comprehensive performance analysis. This profiling tool enables the collection of information, such as garbage collection data, about the Java Virtual Machine (JVM) that runs the application server or Node Agent.

JVMPI is a two-way function call interface between the JVM and an in-process profiler agent. The JVM notifies the profiler agent of various events, such as heap allocations and thread starts. The profiler agent can activate or inactivate specific event notifications, based on the needs of the profiler.

All JVMPI performance data is collected by the JVM module, but JVMPI needs to be enabled for the module to update its counters. To enable Java Virtual Machine Profiler Interface (JVMPI) data reporting for each individual application server or Node Agent:

1. Open the administrative console.
2. Click **Servers -> Application Servers or System Administration -> Node Agents** in the console navigation tree.
3. Click the application server or node agent for which JVMPI needs to be enabled.
4. Click **Process Definition**.
5. Click the **Java Virtual Machine**.
6. Type `-XrunpmiJvmpiProfiler` in the Generic JVM arguments field.
7. Click **Apply** or **OK**.
8. Click **Save**.
9. Start the application server or node agent, or restart the application server or Node Agent if it is currently running.

### 6.2.3 Monitoring an application

In this section we discuss what Tivoli Performance Viewer (TPV) can do to monitor enterprise applications running on WebSphere Application Server.

Before starting the TPV, the PMI service has to be enabled for the application servers you would like to monitor. See “Enabling PMI service using the administrative console” on page 178 or “Enabling PMI service through wsadmin” on page 179 for a description of how to enable the PMI service.

**Refresh data**

When new modules or instances (JSPs, servlets, enterprise beans) are loaded by WebSphere Application Server while Tivoli Performance Viewer is running, they are not automatically displayed and included in performance data reporting.

Use the following steps to include newly loaded modules or instances:

1. Select the resource category that the module or instance belongs to.
2. Select **File -> Refresh** from the main menu, or click the **Refresh** icon in the toolbar.
3. Set the instrumentation level for this new module or instance as described in “Setting performance monitoring level” on page 181.

**Showing correlated counters**

To analyze the performance data captured by Tivoli Performance Viewer, it is often necessary to correlate counters from multiple modules. To select multiple counters in one chart or table:

1. Hold down the Ctrl key and select the resource categories whose counters you want to see.
2. Select the required counters using the Select column in the counter selection window on the lower left.
For example, in Figure 6-4 you can see that the counters for Created Sessions (from Servlet Session Manager), Num Allocates (from JDBC Connection Pools), and Average Response Time (from Web Applications) have all been selected.

Record a load scenario

All data being reported by the Tivoli Performance Viewer can be saved in a log file. The data is written to the log as serialized Java objects or as an XML document. To start recording data, do the following:

1. Select Logging -> On... from the Tivoli Performance Viewer main menu or click the Logging on/off icon in the toolbar.

2. In the Save log file window, specify the file name (and path), and select Save as type. The Save as type field allows an extension of *.perf (Tivoli Performance Viewer logs) for binary files or *.xml for XML files.

3. Click OK.

To stop the recording of data, select Logging -> Off from the main menu or click the Logging on/off icon in the toolbar.
Replaying a load scenario
Both log file types can be replayed using the Tivoli Performance Viewer. Log files saved in XML format (*.xml) provide the option of analysis using third-party tools, but can get rather large. If size is of concern, it is recommended that you zip the files while they are not needed. To replay a log file, do the following:

1. Select File -> Log from the Tivoli Performance Viewer main menu.
2. Click the Browse... button to open the file browser.
3. In the Open window, locate the name of the file to replay and click the Open button.
4. Click the Play icon from the toolbar or select Setting -> Log Replay -> Play from the main menu.

By default, the data is replayed at the same rate it is collected (written to the log). If data is collected every minute, it is displayed every minute. You can change the speed at which the log is replayed by clicking the FF button (Fast Forward) in the toolbar or by selecting Setting -> Log Replay -> FF from the main menu.

While replaying the log, you can change your resource selections with the resource selection pane. You can also view the data in either of the views available (data or chart) in the data display window.

You can stop and resume the log at any point. However, data cannot be replayed in reverse.

To rewind the log file, click the Rewind button in the toolbar or select Setting -> Log Replay -> Rewind from the main menu.

Clearing values from tables and charts
After stopping a resource, use the Clear Values operation to remove the remaining data from a table or chart. You can then begin populating the table or chart with new data. To clear the values currently displayed, do the following:

1. Select one or more resources in the resource hierarchy.
2. Select Setting -> Clear Buffer from the main menu. Alternatively, click the Clear Buffer icon in the toolbar.

Resetting counters to zero
To reset the start time for calculating aggregate data, do the following:

1. Select one or more resources in the resource hierarchy.
2. Select Setting -> Reset to Zero from the main menu. Alternatively, click the Reset to Zero icon in the toolbar.
The reset operation sets the clock used for reporting aggregate data for counters of the selected performance category. Instead of reporting data from the time the server was started, reporting now begins from the time of the reset action. Not all counters can be reset. If you use the reset operation for a group containing counters that cannot be reset, the reset action has no effect. You can select multiple performance groups and reset them simultaneously.

Viewing and modifying chart data
When selected counters are using measurement units that are not proportionally similar, the scaling factor can be set manually to allow a more meaningful display. The following sections explain how you can manually change the scaling factor for the chart view.

**Scaling the chart display manually**
To manually change the scale of a counter:

1. In the counter selection window, on the lower right, double-click the Scale column for the counter that you want to modify.
2. Type the desired scale value for the counter into the field.

The chart view will be updated immediately to reflect the change in the scaling factor.

The possible values for the Scale field range from 0 to 100, and the relationship is shown in Table 6-1.

<table>
<thead>
<tr>
<th>Scale value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 1</td>
<td>Scaling reduces the value. For example, a scale of 0.5 means that the data points for the variable are reduced to half of their actual values.</td>
</tr>
<tr>
<td>= 1</td>
<td>The value is not scaled. This is the default.</td>
</tr>
<tr>
<td>&gt; 1</td>
<td>Scaling increases the value. For example, a scale of 1.5 means that the data points for the variable are increased to one and one-half times their actual values.</td>
</tr>
</tbody>
</table>

Negative results are displayed as zero (0). This value is reflected only in the View Chart window.

Change display settings
This section looks at some of the Tivoli Performance Viewer display settings:

- Specifying a different refresh rate
- Changing the data view
Changing the buffer size

**Specifying a different refresh rate**
By default, the Tivoli Performance Viewer retrieves data from the administrative server every 10 seconds. To change the rate at which data is retrieved from the server, do the following:

1. Select Setting -> Set Refresh Rate... from the main menu.
2. In the Set Refresh Rate window, type a positive integer representing the number of seconds. The integer must be 1 or greater.
3. Click OK.

**Changing the data view**
The data view mode determines whether counter values represent absolute values, changes in values, or rates of change. The data view mode meanings differ slightly depending on where you are viewing data. To change the data view mode:

1. Select Setting -> View Data As from the main menu.
2. Select from the following choices:
   - **Raw Value**. Displays the absolute values. If the counter represents load data (for example, the average number of connections in a database pool or the average number of live bean objects), the Tivoli Performance Viewer displays the current value followed by the average, for example, 18(avg:5).
   - **Change in Value**. Displays the change in the current value from a previous value.
   - **Rate of Change**. Displays the ratio change/(T1 - T2), where change is the change in the current value from a previous value, T1 is the time when the current value was retrieved, and T2 is the time when the previous value was retrieved.

**Changing the buffer size**
By default, the View Data window displays 40 rows, corresponding to the values of the last 40 data points retrieved from the administrative server. To change the size of the table (number of rows displayed):

1. Select Setting -> Set Buffer Size... from the main menu.
2. In the Set Buffer Size window, specify the number of rows to display.
3. Click OK.
6.2.4 PMI request metrics

PMI request metrics is a facility introduced in WebSphere Application Server V5 for providing response-time information for the major services used by a request as it moves through application servers. A PMI request metrics record is created while a response to a request is being generated. It contains information about the request, such as its type, URI, invoked bean method, client origin, etc., and the time spent responding by major WebSphere services. Metrics are produced at the Web and EJB container level and are measured from the time the request enters the container to the time a response is provided.

The request metrics component is designed to be very lightweight in processing and to have the smallest resource footprint possible. This is very important because the facility is globally enabled and extends to every Web and EJB container in the cell. To minimize process and resource usage, a filter can be applied to the metrics component in order to allow only a subset of incoming requests to be monitored. This filter mechanism not only reduces process usage but also minimizes the amount of metrics data produced.

The request metrics record can be written to a text file (System.out) or sent to an Application Response Measurement (ARM 2.0) compliant agent, or both. Every Web container and EJB container stores the time stamp of all requests entering that specific container and writes a record with response time and other metrics to the System.out JVM log or to an available ARM agent as soon as a response is provided. As requests in a WebSphere environment often fan out to different processes on several physical nodes, request metrics can be scattered across different log files. The metrics can be correlated together to resemble a detailed sequence diagram of request response times.

Request metrics are useful for getting information on overall system performance, and hence give WebSphere administrators and developers a feel for the experience a user might have with the system at a given time. They can also be used to give an indication of what component is causing problems to the overall system performance and needs to be examined more closely; problems being application flaws, bottlenecks, leaks, communications, etc. The indication of problems with a specific component can also be helpful in tuning efforts in order to direct the attention to the correct node and process.

Enabling and configuring PMI request metrics

Request metrics are enabled and configured globally for all application servers in the cell. For a single-server environment, the metrics are enabled and configured for all application servers residing on the local node.

The PMI request metrics component can be enabled, disabled, and configured at runtime, without having to restart each individual application server in the cell.
To enable the PMI request metrics using the Administrative Console select **Troubleshooting -> PMI Request Metrics**. From here you can do the following:

- Enable/disable the request metrics.
- Enable an ARM agent. This implies that you have installed an ARM agent that supports request metrics.
- Specify the level of detail on the request metrics data, as shown in Figure 6-5. A setting of NONE produces no metrics data. HOPS shows the EJB method invoked, requested URI, or JDBC statement executed, along with the response time. PERF_DEBUG and DEBUG give you more detailed information but are more performance intensive.

![Performance Monitoring Request Metrics](image)

**Figure 6-5  PMI request metrics configuration panel**

As soon as request metrics are enabled and a trace level greater than NONE is specified and saved to the WebSphere configuration, trace records are written to the System.out JVM log for all application servers for any incoming Web or EJB request.
PMI request metrics trace record format

The request metric trace record written to the System.out JVM log has the following format:

```
PMRM0003I:
    parent:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
    - current:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
      type=TTT detail=some_detail_information elapsed=nnnn
```

The fields of the trace record are described in Table 6-2. Note that PMRM0003I is the message identifier for trace records as found in the System.out log. Table 6-3 lists the entries of the correlator fields (identical for both parent and current correlator).

**Table 6-2  PMI request metric trace record format**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>Identifier to uniquely represent the upstream request (parent correlator).</td>
</tr>
<tr>
<td>current</td>
<td>Identifier to uniquely represent the current request (current correlator).</td>
</tr>
<tr>
<td>type</td>
<td>Type of request (valid types: URI, EJB or JDBC).</td>
</tr>
<tr>
<td>detail</td>
<td>Detailed request information showing the full URI of the request, the fully qualified EJB method invoked, or the JDBC SQL statement executed.</td>
</tr>
<tr>
<td>elapsed</td>
<td>The total elapsed time for the operation in milliseconds. The time includes time for all child operations performed (metrics records having their parent correlator field equal to this record's current correlator).</td>
</tr>
</tbody>
</table>

The type and detail fields are described as follows:

- **Universal Resource Identifier (URI):** The trace record was generated by a Web component. The URI is the name of the URI used to invoke the request.
- **Enterprise bean (EJB):** The fully qualified package and method name of the enterprise bean.
- **Java Database Connectivity (JDBC):** The values select, update, insert, or delete for prepared statements. For non-prepared statements, the full statement can appear.

**Table 6-3  PMI request metric correlator fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ver</td>
<td>The version of the correlator. For convenience, it is duplicated in both the parent and current correlators.</td>
</tr>
<tr>
<td>ip</td>
<td>The IP address of the node of the application server that generated the correlator.</td>
</tr>
</tbody>
</table>
The trace record format is composed of two correlators: A parent correlator and current correlator. The parent correlator represents the upstream request and the current correlator represents the current operation. If the parent and current correlators are the same, then the record represents an operation that occurred as it entered WebSphere Application Server.

To correlate trace records for a particular request, collect records with a message ID of PMRM0003I from the appropriate server logs. Records are correlated by matching current correlators to parent correlators. The logical tree can be created by connecting the current correlators of parent trace records to the parent correlators of child records. This tree shows the progression of the request across the server cluster.

### 6.2.5 Log Analyzer

The Log Analyzer is a GUI tool that allows the user to view any logs generated with log analyzer trace format, such as the IBM service log file and other traces using this format. It can take one or more service logs or trace logs, merge all the data, and display the entries in sequence. Log Analyzer is shipped with an XML database, the symptom database, which contains entries for common problems, reasons for the errors, and recovery steps. The Log Analyzer compares every error record in the log file to the internal set of known problems in the symptom database and displays all the matches. This allows the user to get error message explanations and information such as why the error occurred and how to recover from it.

### 6.2.6 Other monitoring facilities

In this section we have a brief introduction of other monitoring facilities in WebSphere Application Server V5 that are not discussed in previous sections, such as Dynamic Cache Monitor. For more information please refer to the WebSphere InfoCenter and the HTTP Server InfoCenter.
Dynamic Cache Monitor
J2EE applications have high read/write ratios and can tolerate small degrees of latency in the currency of their data. Therefore the dynamic cache creates an opportunity for significant gains in server response time, throughput, and scalability. You can use the dynamic cache to improve the performance of servlet and JSP files by serving requests from an in-memory cache. Cache entries contain servlet output, results of servlet execution, and metadata.

WebSphere Application Server consolidates several caching activities, including servlets, Web services, and WebSphere commands into one service called the dynamic cache. These caching activities work together to improve application performance, and share many configuration parameters, which are set in an application server's dynamic cache service.

The Dynamic Cache Monitor is an installable Web application that displays simple cache statistics, cache entries, and cache policy information. Install the application CacheMonitor.ear from the install_root/installableApplications directory. Access the Web application using a Web browser and the URL http://<your hostname>:<your port_number>/cachemonitor, where <your port_number> is the administrative port or the default port based on whether you installed the application on the admin host or the default host.

Monitoring IBM HTTP Server
The Windows version of IBM HTTP Server includes Windows performance monitor hooks. This allows you to use the Windows NT® performance monitor or the Windows 2000 system monitor to observe the current state of an active IBM HTTP Server, along with all kinds of other system resources.

For more information, please refer to the IBM HTTP Server InfoCenter:


6.2.7 Developing your own monitoring applications
You can use the Performance Monitoring Infrastructure (PMI) interfaces to develop your own applications to collect and display performance information.

There are three such interfaces: A Java Machine Extension (JMX)-based interface, a PMI client interface, and a servlet interface. All three interfaces return the same underlying data. The JMX interface is accessible through the AdminClient API. The PMI client interface is a Java interface that works with Version 3.5.5 and later. The servlet interface is perhaps the simplest, requiring minimal programming, as the output is XML.
6.3 WebSphere Application Server tuning

The performance of a WebSphere Application Server system depends on various factors, including network, database, memory, application design, and application server configuration. Since these factors vary from installation to installation, each recommendation should be evaluated for applicability in one's own unique situation.

6.3.1 WebSphere Queuing Network

In a typical J2EE application, a client request flows through a Web server, application server, and a database. With WebSphere Application Server, the request flows through a network of queues\(^1\).

These queues represent WebSphere Application Server system resources and should be tuned to achieve optimal performance. These queues include the network, Web server, Web container, EJB container, Object Request Broke(ORB), datasource, and possibly a connection manager to a custom backend system, as shown in Figure 6-6.

---

Figure 6-6  WebSphere queuing network

Each of these WebSphere Application Server resources represents a queue of requests waiting to use that resource. WebSphere Application Server queues are load-dependent resources, and therefore the average service time of a request depends on the number of concurrent clients.

\(^1\) We call them queues from a queueing theory perspective, but in reality they are thread pools.
Queues may either be closed or open:

- A closed queue allows the administrator to limit the maximum number of requests active in that queue.

A closed queue allows system resources to be tightly managed. For example, the Web container's Max Connections setting controls the size of the Web container thread pool. If the average servlet running in the Web container creates 10 MB of objects during each request, then setting Max Connections to 100 would limit the memory consumed by the Web container to approximately 1 GB. Hence, closed queues typically allow the system administrators to manage their applications more effectively and robustly.

In a closed queue, a request can be in one of the two following states:

- Active: In this state, a request is doing work, or is waiting for a response from a downstream queue. For example, an active request in the Web server is either doing work such as retrieving static HTML, or waiting for a request to complete in the Web container.

- Waiting: In this state, the request is waiting to become active. The request will remain in the waiting state until one of the active requests finishes processing and vacates the queue.

An open queue does not allow the administrator to restrict the maximum number of requests active in that queue.

### 6.3.2 Configuring WebSphere Application Server queues

We now outline a methodology for configuring the WebSphere Application Server queues.

There are four general rules for tuning the queues.

- Minimize the number of requests in WebSphere Application Server queues.

In general, you should make requests wait in the network in front of the Web server, rather than waiting in the WebSphere Application Server. This configuration only supports those requests that are ready for processing to enter the queuing network. Specify that the queues furthest upstream or closest to the client are slightly larger, and queues further downstream or furthest from the client are progressively smaller. Figure 6-7 on page 195 shows an example.
Queues in the queuing network become progressively smaller as work flows downstream. When 200 client requests arrive at the Web server, 125 requests remain queued in the network because the Web server is set to handle 75 concurrent clients. As the 75 requests pass from the Web server to the Web container, 25 requests remain queued in the Web server and the remaining 50 are handled by the Web container. This process progresses through the data source until 25 user requests arrive at the final destination, the database server. Because there is work waiting to enter a component at each point upstream, no component in this system must wait for work to arrive. The bulk of the requests wait in the network, outside of the WebSphere Application Server. This type of configuration adds stability, because no component is overloaded. You can then use the Edge Server to direct waiting users to other servers in a WebSphere Application Server cluster.

- Draw throughput curves to determine when the system capabilities are maximized.

You can use a test case that represents the full spirit of the production application by either exercising all meaningful code paths or using the production application. Run a set of experiments to determine when the system capabilities are fully stressed or when it has reached the saturation point. Conduct these tests after most of the bottlenecks are removed from the application. The goal of these tests is to drive CPUs to near 100 percent utilization. For maximum concurrency through the system, start the initial baseline experiment with large queues. For example, start the first experiment with a queue size of 100 at each of the servers in the queuing network: Web server, Web container, and data source. Begin a series of experiments to plot
a throughput curve, increasing the concurrent user load after each experiment. For example, perform experiments with one user, two users, five, 10, 25, 50, 100, 150 and 200 users. After each run, record the throughput requests per second, and response times in seconds per request. The curve resulting from the baseline experiments resembles the typical throughput curve shown in Figure 6-8.

![Figure 6-8 Throughput curve](image)

The WebSphere Application Server throughput is a function of the number of concurrent requests present in the total system. Section A, the light load zone, shows that as the number of concurrent user requests increases, the throughput increases almost linearly with the number of requests. At light loads, concurrent requests face very little congestion within the WebSphere Application Server system queues. At some point, congestion starts to develop and throughput increases at a much lower rate until it reaches a saturation point that represents the maximum throughput value, as determined by some bottleneck in the WebSphere Application Server system. The most manageable type of bottleneck occurs when the WebSphere Application Server machine CPUs become fully utilized because adding CPUs or more powerful CPUs fixes the bottleneck.

In the heavy load zone or Section B, as the concurrent client load increases, throughput remains relatively constant. However, the response time increases proportionally to the user load. That is, if the user load is doubled in the heavy load zone, the response time doubles. At some point, represented by Section C, the buckle zone, one of the system components becomes exhausted. At
this point throughput starts to degrade. For example, the system might enter the buckle zone when the network connections at the Web server exhaust the limits of the network adapter or if the requests exceed operating system limits for file handles.

If the saturation point is reached by driving CPU utilization close to 100 percent, you can move on to the next step. If the saturation CPU occurs before system utilization reaches 100 percent, it is likely that another bottleneck is being aggravated by the application. For example, the application might be creating Java objects causing excessive garbage collection bottlenecks in the Java code.

There are two ways to manage application bottlenecks: Remove the bottleneck or clone the bottleneck. The best way to manage a bottleneck is to remove it. You can use a Java-based application profiler, such as WebSphere Studio Application Developer, Performance Trace Data Visualizer (PTDV), Borland's Optimizeit, JProbe, or Jinsight to examine overall object utilization.

- Decrease queue sizes while moving downstream from the client.

The number of concurrent users at the throughput saturation point represents the maximum concurrency of the application. For example, if the application saturates WebSphere Application Server at 50 users, using 48 users might produce the best combination of throughput and response time. This value is called the Max Application Concurrency value. Max Application Concurrency becomes the preferred value for adjusting the WebSphere Application Server system queues. Remember, it is desirable for most users to wait in the network; therefore, queue sizes should increase when moving downstream farther from the client. For example, given a Max Application Concurrency value of 48, start with system queues at the following values: Web server 75, Web container 50, data source 45. Perform a set of additional experiments adjusting these values slightly higher and lower to find the best settings.

To help determine the number of concurrent users, view the Servlet Engine Thread Pool and Concurrently Active Threads metric in the Tivoli Performance Viewer.

- Adjust queue settings to correspond to access patterns.

In many cases, only a fraction of the requests passing through one queue enter the next queue downstream. In a site with many static pages, a number of requests are fulfilled at the Web server and are not passed to the Web container. In this circumstance, the Web server queue can be significantly larger than the Web container queue. In the previous example, the Web server queue was set to 75, rather than closer to the value of Max Application Concurrency. You can make similar adjustments when different components have different execution times.
For example, in an application that spends 90 percent of its time in a complex servlet and only 10 percent of its time making a short Java database connectivity (JDBC) query, on average 10 percent of the servlets are using database connections at any time, so the database connection queue can be significantly smaller than the Web container queue. Conversely, if the majority of servlet execution time is spent making a complex query to a database, consider increasing the queue values at both the Web container and the data source. Always monitor the CPU and memory utilization for both the WebSphere Application Server and the database servers to verify that the CPU or memory are not saturating.

Let us now look at how to configure the queue network in each layer.

**Web Server**

All Web servers supported by WebSphere Application Server maintain a thread pool to process the incoming HTTP request. Their size can be controlled by the following parameters in conf/httpd.conf:

- **IBM HTTP Server**
  - `MaxClients` for UNIX
  - `ThreadsPerChild` for Windows

- **Microsoft IIS**
  - `MaxPoolThreads`
  - `PoolThreadLimit`

There is a way to monitor how many clients/threads are being used under load. Use IBM HTTP Server server-status. (This choice works on all platforms.) To enable the server-status module, do the following:

1. Edit `httpd.conf` located in the directory `<IBM HTTP Server Home>/conf and remove the comment character “#” from the lines shown in Example 6-2.

   **Example 6-2   Enabling HTTP Server server-status in httpd.conf on Windows**

   ```
   #LoadModule status_module modules/ApacheModuleStatus.dll
   #<Location/server-status>
   #SetHandler server-status
   #</Location>
   ```

2. Save the changes and restart the IBM HTTP server.

3. In a Web browser, go to the URL `http://yourhost/server-status` and click **Reload** to update the status.

4. Alternatively, if the browser supports refresh, go to `http://yourhost/server-status?refresh=5` to refresh every five seconds.
On Windows, there is another way to find out how many threads are being used under load. Use the Windows NT or 2000 Performance Monitor. To open the Performance Monitor, select **Start -> Programs -> Administrative Tools -> Performance Monitor**. In Performance Monitor, select **Edit -> Add to chart**. Then set the following:

- Object: IBM HTTP Server
- Instance: Apache
- Counter: Waiting for connection

To calculate the number of busy threads, subtract the Number Waiting (Windows NT or 2000 Performance Monitor) from the Total Available (ThreadsPerChild).

### Web container

To route servlet requests from the Web server to the Web containers, a transport queue between the Web server plug-in and each Web container is established. The number of client requests accepted by the container is determined by the Web container thread pool. Connection reuse is another factor that influences the number of concurrent threads that are processed by the Web container.

### Thread pool size

The Web container has a pool of threads to process inbound servlet/JSP requests and Web services and can be set in the WebSphere Application Server V5 Admin Console, as shown in Figure 6-9 on page 200.

Tivoli Performance Viewer can help tune the Web container thread pool size settings. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the **Percent Maxed** and **Active Threads** counters of the Web container submodule of the Thread Pools module. If the value of the **Percent Maxed** counter is consistently in the double digits, then the Web container could be a bottleneck and the number of threads should be increased.

On the other hand, if the number of active threads is significantly lower than the number of threads in the pool, consider lowering the thread pool size for a performance gain.
MaxKeepAliveConnections

The MaxKeepAliveConnections parameter specifies the maximum number of concurrent connections to the Web container that are allowed to be kept alive, that is, to be processed in multiple requests. To make a particular transport close connections after a request, you can set MaxKeepAliveConnections to 0 (zero) or you can set KeepAliveEnabled to false on that transport.

The Web server plug-in keeps connections open to the application server as long as it can. However, if the value of this property is too small, performance is negatively impacted because the plug-in has to open a new connection for each request instead of sending multiple requests through one connection. The application server might not accept a new connection under a heavy load if there
are too many sockets in TIME_WAIT state. If all client requests are going through the Web server plug-in and there are many TIME_WAIT state sockets for port 9080, the application server is closing connections prematurely, which decreases performance. The application server closes the connection from the plug-in, or from any client, for any of the following reasons:

- The client request was an HTTP 1.0 request when the Web server plug-in always sends HTTP 1.1 requests.
- The maximum number of concurrent keep-alives was reached. A keep-alive must be obtained only once for the life of a connection, that is, after the first request is completed, but before the second request can be read.
- The maximum number of requests for a connection was reached, preventing denial of service attacks in which a client tries to hold onto a keep-alive connection forever.
- A time out occurred while waiting to read the next request or to read the remainder of the current request.

The value should be at least 90 percent of the maximum number of threads in the Web container thread pool. If it is 100 percent of the maximum number of threads in the Web container thread pool, all the threads could be consumed by keep-alive connections, leaving no threads available to process new connections.

The netstat command utility can help tune the maximum keep-alive connections setting. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the number of connections in the TIME_WAIT state to the application server port. If the count of TIME_WAITs is consistently in the double digits, it might improve performance to raise the maximum keep-alive connections or maximum keep-alive requests parameters. Commands for retrieving the count of TIME_WAITs are shown in Example 6-3. Substitute the port number with the port of the specific application server you want to monitor. Be aware that having both the Web server and application server installed on the same machine would result in a double count of every connection since the TIME_WAIT state is listed from both the client side and server side by netstat.

Example 6-3  Using netstat to determine the time_wait values

On the Windows platform the chain of commands would be:
```
netstat -na | find /i "time_wait" | find /c "9080"
```
On the Unix platform the chain of commands would look like:
```
netstat -na | grep -i time_wait | grep -c 9080
```

MaxKeepAliveRequests

The MaxKeepAliveRequests is the maximum number of requests allowed on a single keep-alive connection. This parameter can help prevent denial of service
attacks when a client tries to hold onto a keep-alive connection. The Web server plug-in keeps connections open to the application server as long as it can, providing optimum performance.

A good starting value for the maximum number of requests allowed is 100. If the application server requests are received from the Web server plug-in only, increase this parameter's value. The netstat utility can be used to tune the value of maximum keep-alive requests as described in “MaxKeepAliveConnections” on page 200. If the number of connections in the TIME_WAIT state is too high, consider raising the maximum keep-alive requests setting.

**EJB container**

In EJB container, the following parameters can be adjusted to improve the performance.

**ORB thread pool size**

Method invocations to enterprise beans are only queued for requests coming from remote clients going through the RMI activity service. An example of such a client is an EJB client running in a separate Java Virtual Machine (another address space) from the enterprise bean. In contrast, no queuing occurs if the EJB client (either a servlet or another enterprise bean) is installed in the same JVM that the EJB method runs on and the same thread of execution as the EJB client.

Remote enterprise beans communicate by using the RMI/IIOP protocol. Method invocations initiated over RMI/IIOP are processed by a server-side ORB. The thread pool acts as a queue for incoming requests. However, if a remote method request is issued and there are no more available threads in the thread pool, a new thread is created. After the method request completes, the thread is destroyed. Therefore, when the ORB is used to process remote method requests, the EJB container is an open queue, due to the use of unbounded threads.

Tivoli Performance Viewer can help tune the ORB thread pool size settings. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the Percent Maxed counter of the Object Request Broker submodule of the Thread Pools module. If the value of this counter is consistently in the double digits, then the ORB could be a bottleneck and the number of threads in the pool should be increased. The degree to which the ORB thread pool value needs to be increased is a function of the number of simultaneous servlets (that is, clients) calling enterprise beans and the duration of each method call. If the method calls are longer or the applications spend a lot of time in the ORB, consider making the ORB thread pool size equal to the Web container size. If the servlet makes only short-lived or quick calls to the ORB,
Servlets can potentially reuse the same ORB thread. In this case, the ORB thread pool can be small, perhaps even one-half of the thread pool size setting of the Web container.

The ORB *Thread pool size* can be configured using the WebSphere Admin Console, as shown in Figure 6-10. As explained above, this pool can grow beyond the specified size depending on the incoming traffic.

![Figure 6-10 ORB thread pool size](image)

**Cache settings (cache size and cleanup interval)**

To determine the cache absolute limit, multiply the number of enterprise beans active in any given transaction by the total number of concurrent transactions expected. Then add the number of active session bean instances. Use the Tivoli Performance Viewer to view bean performance information. The cache settings consist of two parameters: The cache size and the cleanup interval (see Figure 6-11 on page 204).

The *cleanup interval* specifies the interval at which the container attempts to remove unused items from the cache in order to reduce the total number of items to the value of the cache size.
The cache size specifies the number of buckets in the active instance list within the EJB container.

![EJB Cache Settings](image)

**Figure 6-11  EJB cache settings**

**Data sources**

During WebSphere Application Server startup, the database connection pool is created with a zero size, and it grows until it reaches the maximum size, depending on the demand.

Tivoli Performance Viewer provides detailed performance data about each data source pool, as shown in Figure 6-12 on page 205.
Figure 6-12  TPV datasource monitoring

The data includes the number of connections allocated, returned, and the average wait time before a connection is granted. The summary report for the database connection pool provides information about the pool in use and the pool size over time.

Database connection pool size and prepared statement cache tuning is covered in detail in “Connection pooling” on page 288 and “Prepared statement cache” on page 292.

6.3.3 Using Performance Advisors

WebSphere Application Server V5 provides the following Performance Advisors:

- Runtime Performance Advisor
- Performance Advisor in Tivoli Performance Viewer
Runtime Performance Advisor

The Runtime Performance Advisor provides advice to help tune systems for optimal performance and is configured using the WebSphere Application Server administrative console. The Runtime Performance Advisor uses Performance Monitoring Infrastructure (PMI) data to provide recommendations for performance tuning. Running in the JVM of the application server, this advisor periodically checks for inefficient settings, and issues recommendations as standard product warning messages. These recommendations are displayed both as warnings in the administrative console under WebSphere Runtime Messages in the WebSphere Status panel and as text in the application server SystemOut.log file. Enabling the Runtime Performance Advisor has minimal system performance impact.

The following are the instructions on how to use the Runtime Performance Advisor:

1. Enable PMI services in WebSphere Application Server through the administrative console, and enable PMI services in NodeAgent through the administrative console if running WebSphere Application Server Network Deployment.

In order to obtain advice, you must first enable the performance monitoring service through the administrative console and restart the server. If running Network Deployment, you must enable PMI service on both the server and on the node agent and restart the server and node agent. The Runtime Performance Advisor enables the appropriate monitoring counter levels for all enabled advice. If there are specific counters that are not wanted, disable the corresponding advice in the Runtime Performance Advisor Panel, and disable unwanted counters.

2. Enable PMI services in NodeAgent through the administrative console. In order to obtain advice, you must enable PMI service on both the server and on the node agent, and restart the server and node agent.

3. Start the administrative console.

4. Click Servers -> Application Servers in the console navigation tree.

5. Click server_name -> Runtime Performance Advisor Configuration.

6. Click the Configuration tab.

7. Select the Number of Processors.

8. Select the appropriate settings for your system configuration to ensure accurate advice.

9. (Optional) Select the interval.

PMI data is taken over an interval of time and averaged to provide advice. The interval specifies the length of the time over which data is taken for this
advice. Therefore, details within the advice messages appear as averages over this interval.

10. (Optional) Select the Maximum Warning Sequence.

The Maximum Warning Sequence refers to the number of consecutive warnings issued before the threshold is updated. For example, if the maximum warning sequence is set to 3, then the advisor only sends three warnings to indicate that the prepared statement cache is overflowing. After that, a new alert is only issued if the rate of discards exceeds the new threshold setting.

11. Click Apply.

12. Click Save.

13. Click the Runtime tab.

14. Click Restart.

Selecting Restart on the Runtime tab reinitializes the Runtime Performance Advisor using the last configuration information saved to disk.

**Note:** This action also resets the state of the Runtime Performance Advisor. For example, the current warning count is reset to zero for each message.

15. Simulate a production level load.

If you are using the Runtime Performance Advisor in a test environment, or doing any other tuning for performance, simulate a realistic production load for your application. The application should run this load without errors. This simulation includes numbers of concurrent users typical of peak periods, and drives system resources, such as CPU and memory to the levels expected in production. The Runtime Performance Advisor only provides advice when CPU utilization exceeds a sufficiently high level.

16. Select the check box to enable the Runtime Performance Advisor to achieve the best results for performance tuning, when a stable production level load is being applied.

**Note:** Enable the Runtime Performance Advisor once conditions have reached a fully loaded condition to achieve the best results for performance tuning.

17. Click OK.

18. Select Warnings in the administrative console under the WebSphere Runtime Messages in the WebSphere Status panel or look in the
SystemOut.log file, located in the install_root\logs\servername directory to view tuning advice. Some messages are not issued immediately.

19. Update the product configuration for improved performance, based on advice.

Although the performance advisors attempt to distinguish between loaded and idle conditions, misleading advice might be issued if the advisor is enabled while the system is ramping up or down. This result is especially likely when running short tests. Although the advice helps in most configurations, there might be situations where the advice hinders performance. Due to these conditions, advice is not guaranteed. Therefore, test the environment with the updated configuration to ensure it functions and performs well.

**Runtime Performance Advisor output**

After completing the appropriate steps, the Advisor will report recommendations into the SystemOut.log. Example 6-4 shows sample output from the Advisor.

Example 6-4  Sample output from the Runtime Advisor

```
[11/12/03 15:28:53:312 EST] 1b7c0fb7 TraceResponse W TUNE0204W: Performance may be improved by decreasing the number of threads in the ORB thread pool. Try setting the minimum size to 0 and the maximum to 3.
The specific data that triggered this message follows.
0% of the pool is in use.
CPU usage: 100%.
Number of threads in pool: 0.
Number of active threads in the pool: 0.
The advisor will now look for the pool usage % between 0 and 99. Originally, the advisor looked for the pool usage % between 50 and 99.
The acceptable variance is: 81
```

```
[11/12/03 15:28:53:312 EST] 1b7c0fb7 TraceResponse W TUNE0208W: Datasource DefaultDatasource does not seem to be in use. If this is true, please delete the data source. If not, set minConnections to 0, and maxConnections to 3.
The specific data that triggered this message follows.
0% of the pool is in use.
This alert has been issued 1 times in a row. The threshold will be updated to reduce the overhead of the analysis.
The advisor will now look for pool size to be greater than 30 with a pool usage of 0%.
```

**TPV Performance Advisor**

The Performance Advisor in Tivoli Performance Viewer (TPV) provides advice to help tune systems for optimal performance and gives recommendations on inefficient settings by using collected Performance Monitoring Infrastructure (PMI) data. Advice is obtained by selecting the Performance Advisor icon in TPV. The Performance Advisor in TPV provides more extensive advice than the
Runtime Performance Advisor. For example, TPV provides advice on setting the dynamic cache size, setting the JVM heap size, and using the DB2 Performance Configuration Wizard.

If the WebSphere environment is an IBM WebSphere Application Server Network Deployment V5, the only advisor that executes is the TPV Advisor. To utilize the TPV Advisor, direct console access or a remote access tool for UNIX or Windows is required.

The instructions to use TPV Performance Advisor are similar to the instructions to use TPV in “Starting Tivoli Performance Viewer” on page 181 and “Setting performance monitoring level” on page 181.

To review the advisor suggestions, expand the Performance Advisor tree, expand the node that is executing the application server, and highlight the desired application server. Tivoli Performance Viewer polls and retrieves performance data. The TPV Advisor works in conjunction with Tivoli Performance Viewer using the TPV infrastructure. TPV Advisor provides advice based on analysis of the gathered data. If the node being monitored is a production node, it is likely the Advisor will have information available fairly quickly.

Information from the Advisor is categorized into three classifications:

- **Warning**
  
  Warning information is provided to indicate possible areas that the Advisor cannot adequately monitor, or other general configuration problems that may need attention.

- **Config**
  
  Configuration options that could be changed to improve performance that are read from the base XML configurations and not directly related to captured performance data are listed in this category.

- **Alert**
  
  Alert information describes tuning suggestions discovered while monitoring the current load. Alerts make suggestions for changes to existing values as determined by the load currently on the application server.

As the TPV Advisor executes, output is displayed in the Tivoli Performance Viewer application window. Figure 6-13 on page 210 shows a sample of this output. To see the details of a specific advisor message, double-click the message and a new pop-up window will display the exact details of the message.

The box will also describe how to change the parameter, and where it can be located in the Administrative Console.
6.3.4 JVM memory tuning

This section focuses on tuning JVM memory. Enterprise applications written in Java involve complex object relationships and utilize large numbers of objects. Although Java automatically manages memory associated with an object's life cycle, understanding the application's usage patterns for objects is important. In particular, ensure the following:

- The application is not over-utilizing objects.
- The application is not leaking objects (that is, memory).
- The Java heap parameters are set to handle the use of objects.

Understanding the effect of garbage collection is necessary to apply these management techniques.

The garbage collection

Examining Java garbage collection (GC) can give insight into how the application is utilizing memory. Garbage collection is a Java strength. By taking the burden of
memory management away from the application writer, Java applications are more robust than applications written in languages that do not provide garbage collection capability. This robustness applies as long as the application is not abusing objects.

Examining GC gives insights into how the application is utilizing memory. Garbage collection normally consumes anywhere from 5 to 20 percent of the total execution time of a properly functioning application. If not managed, garbage collection can be one of the biggest bottlenecks for an application, especially when running on SMP server machines.

Garbage collection can be used to evaluate application performance health. By monitoring garbage collection during the execution of a fixed workload, users gain insight as to whether the application is over-utilizing objects. Garbage collection can even be used to detect the presence of memory leaks.

Use the garbage collection and heap statistics in Tivoli Performance Viewer (see “Tivoli Performance Viewer” on page 180 and “Monitoring an application” on page 183) to evaluate application performance health. By monitoring garbage collection, memory leaks and overly used objects can be detected.

For this type of investigation, set the minimum and maximum heap sizes to the same value. Choose a representative, repetitive workload that matches production usage as closely as possible, user errors included. To ensure meaningful statistics, run the fixed workload until the state of the application is steady. Reaching this state usually takes several minutes.

**Detecting over-utilization of objects**

To see if the application is overusing objects, look in Tivoli Performance Viewer at the counters for the JVMPI profiler. The average time between garbage collection calls should be five to six times the average duration of a single garbage collection. If not, the application is spending more than 15 percent of its time in garbage collection. Also, look at the numbers of freed, allocated, and moved objects.

If the information indicates a garbage collection bottleneck, there are two ways to clear the bottleneck. The most cost-effective way to optimize the application is to implement object caches and pools. Use a Java profiler to determine which objects to target. If the application cannot be optimized, adding memory, processors, and application clusters might help. Additional memory allows each application server in a cluster to maintain a reasonable heap size. Additional processors allow the cluster members to run in parallel.
Detecting memory leaks
Memory leaks in Java are a dangerous contributor to garbage collection bottlenecks. Memory leaks are more damaging than memory overuse, because a memory leak ultimately leads to system instability. Over time, garbage collection occurs more frequently until finally the heap is exhausted and Java fails with a fatal Out of Memory exception.

Memory leaks occur when an unneeded object has references that are never deleted. This most commonly occurs in collection classes, such as Hash table, because the table itself always has a reference to the object, even after real references have been deleted.

Memory leaks must be fixed. The best way to fix a memory leak is to use a Java profiler that allows you to count the number of object instances. Object counts that exhibit unbounded growth over time indicate a memory leak.

The following considerations apply to memory leaks:

- Long-running test: Memory leak problems manifest only after a period of time; therefore, recognizing memory leaks is related to long-running tests.
- System test: Some memory leak problems occur only when different components of a big project are combined and executed.
- Repetitive test: In many cases, memory leak problems occur by successive repetitions of the same test case. Repetitive tests can be used at the system level or module level.
- Concurrency test: Some memory leak problems can occur only when there are several threads running in the application.

Tivoli Performance Viewer helps to find memory leaks. For best results, repeat experiments with increasing duration, such as 1000, 2000, and 4000-page requests. The Tivoli Performance Viewer graph of used memory should have a sawtooth shape. Each drop on the graph corresponds to a garbage collection. There is a memory leak if one of the following occurs:

- The amount of memory used immediately after each garbage collection increases significantly. The sawtooth pattern will look more like a staircase.
- The sawtooth pattern has an irregular shape.

Also, look at the difference between the number of objects allocated and the number of objects freed. If the gap between the two increases over time, there is a memory leak.
Heap fragmentation
If heap consumption indicates a possible leak during a heavy workload (the application server is consistently near 100 percent CPU utilization), yet the heap appears to recover during a subsequent lighter or near-idle workload, this is an indication of heap fragmentation.

Heap fragmentation can occur when the JVM is able to free sufficient objects to satisfy memory allocation requests during garbage collection cycles, but does not have the time to compact small free memory areas in the heap into larger contiguous spaces. Another form of heap fragmentation occurs when small objects (less than 512 bytes) are freed. The objects are freed, but the storage is not recovered, resulting in memory fragmentation.

Heap fragmentation can be avoided by turning on the -Xcompactgc flag in the JVM advanced settings command line arguments. The -Xcompactgc ensures that each garbage collection cycle eliminates fragmentation, but with a small performance penalty.

Java heap parameters
The Java heap parameters also influence the behavior of garbage collection. Increasing the heap size allows more objects to be created. Because a large heap takes longer to fill, the application runs longer before a garbage collection occurs. However, a larger heap also takes longer to compact and causes garbage collection to take longer.

For performance analysis, the initial and maximum heap sizes should be equal. When tuning a production system where the working set size of the Java application is not understood, a good starting value is to let the initial heap size be 25 percent of the maximum heap size. The JVM will then try to adapt the size of the heap to the working set size of the application.

Run a series of test experiments that vary the Java heap settings. For example, run experiments with 128 MB, 192 MB, 256 MB, and 320 MB. During each experiment, monitor the total memory usage. If the heap is expanded too aggressively, paging can occur. (Use the `vmstat` command or the Windows NT or 2000 Performance Monitor to check for paging.) If paging occurs, reduce the size of the heap or add more memory to the system. When all the runs are finished, compare the following statistics:

- Number of garbage collection calls
- Average duration of a single garbage collection call
- Ratio between the length of a single garbage collection call and the average time between calls
If the application is not over-utilizing objects and has no memory leaks, a state of steady memory utilization is reached. Garbage collection also occurs less frequently and for shorter durations.

6.3.5 Application assembly considerations

Application assembly tools are used to assemble J2EE components and modules into J2EE applications. Generally, this consists of defining application components and their attributes, including enterprise beans, servlets and resource references. Many of these application configuration settings and attributes play an important role in the runtime performance of the deployed application. The most important parameters and advice for finding optimal settings are:

- **Enterprise bean modules**
  - Entity EJBs - Bean cache
  - Entity EJBs (EJB 2.0) - Access intent
  - Entity EJB Method extensions (EJB 1.1)
    - Isolation level
    - Access intent
  - Container transactions

- **Web modules**
  - Web application - Distributable
  - Web application - Reload interval
  - Web application - Reload enabled
  - Web application - Web components - Load on startup

**Enterprise bean modules**

This section explains the enterprise bean module parameters mentioned above, in detail.

**Entity EJBs - Bean cache**

WebSphere Application Server provides significant flexibility in the management of database data with Entity EJBs. The Entity EJBs *Activate at* and *Load at* configuration settings specify how and when to load and cache data from the corresponding database row data of an enterprise bean. These configuration settings provide the capability to specify enterprise bean caching Options A, B or C.

*Option A* provides maximum enterprise bean performance by caching database data outside of the transaction scope. Generally, Option A is only applicable where the EJB container has exclusive access to the given database. Otherwise, data integrity is compromised. *Option B* provides more aggressive caching of
Entity EJB object instances, which can result in improved performance over Option C, but also results in greater memory usage. Option C is the most common real-world configuration for Entity EJBs.

- **Bean cache - Activate at**

  This setting specifies the point at which an enterprise bean is activated and placed in the cache. Removal from the cache and passivation are also governed by this setting. Valid values are Once and Transaction. **Once** indicates that the bean is activated when it is first accessed in the server process, and passivated (and removed from the cache) at the discretion of the container, for example, when the cache becomes full. **Transaction** indicates that the bean is activated at the start of a transaction and passivated (and removed from the cache) at the end of the transaction. The default value is Transaction.

- **Bean cache - Load at**

  This setting specifies when the bean loads its state from the database. The value of this property implies whether the container has exclusive or shared access to the database. Valid values are Activation and Transaction.

  **Activation** indicates the bean is loaded when it is activated and implies that the container has exclusive access to the database. **Transaction** indicates that the bean is loaded at the start of a transaction and implies that the container has shared access to the database. The default is Transaction.

The settings of the Activate at and Load at properties govern which commit options are used.

- **For Option A** (exclusive database access), use Activate at = Once and Load at = Activation.

  This option reduces database input/output by avoiding calls to the ejbLoad function, but serializes all transactions accessing the bean instance. Option A can increase memory usage by maintaining more objects in the cache, but can provide better response time if bean instances are not generally accessed concurrently by multiple transactions.

  **Note:** When using WebSphere Network Deployment and workload management is enabled, Option A cannot be used. You must use settings that result in the use of Options B or C.

- **For Option B** (shared database access), use Activate at = Once and Load at = Transaction.

  Option B can increase memory usage by maintaining more objects in the cache. However, because each transaction creates its own copy of an object, there can be multiple copies of an instance in memory at any given time (one
per transaction), requiring the database be accessed at each transaction. If an enterprise bean contains a significant number of calls to the ejbActivate function, using Option B can be beneficial because the required object is already in the cache. Otherwise, this option does not provide significant benefit over Option A.

- For Option C (shared database access), use Activate at = Transaction and Load at = Transaction.

  This option can reduce memory usage by maintaining fewer objects in the cache. However, there can be multiple copies of an instance in memory at any given time (one per transaction). This option can reduce transaction contention for enterprise bean instances that are accessed concurrently but not updated.

**Note:** The following information refers to EJB 1.1 settings.

**Method extensions - Isolation level**

WebSphere Application Server enterprise bean method extensions provide settings to specify the level of transactional isolation used when accessing data.

Isolation level settings specify various degrees of runtime data integrity provided by the corresponding database. First choose a setting that meets data integrity requirements for the given application.

The valid values are:

- Serializable
- Repeatable read
- Read committed
- Read uncommitted

Isolation level also plays an important role in performance. Higher isolation levels reduce performance by increasing row locking and database overhead while reducing data access concurrency. Various databases provide different behavior with respect to the isolation settings.

The isolation level can be specified at the bean or method level. Therefore, it is possible to configure different isolation level settings for various methods. This is an advantage when some methods require higher isolation than others, and can be used to achieve maximum performance while maintaining integrity requirements. However, isolation cannot change between method calls within a single enterprise bean transaction. A runtime exception will be thrown in this case.
The following section describes the four isolation levels:

- **Serializable**
  
  This level prohibits the following types of reads:
  
  - Dirty reads: A transaction reads a database row containing uncommitted changes from a second transaction.
  
  - Non-repeatable reads: One transaction reads a row, a second transaction changes the same row, and the first transaction re-reads the row and gets a different value.
  
  - Phantom reads: One transaction reads all rows that satisfy an SQL WHERE condition, a second transaction inserts a row that also satisfies the WHERE condition, and the first transaction applies the same WHERE condition and gets the row inserted by the second transaction.

- **Repeatable read**
  
  This level prohibits dirty reads and nonrepeatable reads, but it allows phantom reads.

- **Read committed**
  
  This level prohibits dirty reads, but allows nonrepeatable reads and phantom reads.

- **Read uncommitted**
  
  This level allows dirty reads, nonrepeatable reads, and phantom reads.

The container uses the transaction isolation level attribute as follows:

- **Session beans and entity beans with bean-managed persistence (BMP)**
  
  For each database connection used by the bean, the container sets the transaction isolation level at the start of each transaction unless the bean explicitly sets the isolation level on the connection.

- **Entity beans with container-managed persistence (CMP)**
  
  The container generates database access code that implements the specified isolation level.

**Method extensions - Access intent**

WebSphere Application Server enterprise bean method extensions provide settings to specify individual enterprise bean methods as read-only. This setting denotes whether the method can update entity attribute data (or invoke other methods that can update data in the same transaction).
By default, all enterprise bean methods are assumed to be “update” methods. This results in EJB Entity data always being persisted back to the database at the close of the enterprise bean transaction. Marking enterprise methods that do not update entity attributes as Access Intent Read provides a significant performance improvement by allowing the WebSphere Application Server EJB container to skip the unnecessary database update.

A behavior for “finder” methods for CMP Entity EJBs is available. By default, WebSphere Application Server will invoke a Select for Update query for CMP enterprise bean finder methods such as findByPrimaryKey. This exclusively locks the database row for the duration of the enterprise bean transaction. However, if the enterprise bean finder method has been marked as Access Intent Read, the container will not issue the For Update on the select, resulting in only a read lock on the database row.

For EJB 2.0, please see Chapter 8, “DB2 UDB V8 and WAS V5 integrated performance” on page 287, for more information.

**Web module**
This section explains the parameters that can be set for Web modules.

**Web application - Distributable**
The distributable flag for J2EE Web applications specifies that the Web application is programmed to be deployed in a distributed servlet container. Web applications should be marked as Distributable if, and only if, they will be deployed in a WebSphere Application Server clustered environment.

**Web application - Reload interval**
The reload interval specifies a time interval, in seconds, in which the Web application’s file system is scanned for updated files, such as servlet class files or JSPs.

The Reload interval can be defined at different levels for various application components. Generally, the reload interval specifies the time the application server will wait between checks to see if dependent files have been updated and

---

**Note:** This setting is applicable only for EJB 1.x-compliant beans, that is:

- EJB 1.x compliant entity beans
- Enterprise beans with CMP Version 1.x that are packaged in EJB 2.x-compliant modules.

To specify the access intent for EJB 2.x-compliant beans, select an access intent policy.
need to be reloaded. Checking file system time stamps is an expensive operation and should be reduced. The default is 0 (zero). Setting this to a value of 3 seconds is good for a test environment, because the Web site can be updated without restarting the application server. In production environments, checking a few times a day is a more common setting.

*Web application - Reloading enabled*
This specifies whether file reloading is enabled. The default is false.

*Web application - Web components - Load on startup*
Indicates whether a servlet is to be loaded at the startup of the Web application. The default is false.

Many servlets perform resource allocation and other up-front processing in the servlet init() method. These initialization routines can be costly at runtime. By specifying Load on startup for these servlets, processing takes place when the application server is started. This avoids runtime delays, which can be encountered on a servlet's initial access.

### 6.3.6 Other considerations

Besides the considerations mentioned above, the following considerations also have significant impact on the WebSphere Application Server system.

**Hardware considerations**

Hardware configurations used by WebSphere Application Server will obviously have significant impact on performance.

- Processor speed: Increasing the processor speed often helps throughput and response times once other bottlenecks have been removed where the processor is waiting on events like input/output and application concurrency. In this case, increasing the processor speed often helps throughput and response times.

- System memory: In general, increasing memory to prevent the system from paging memory to disk is likely to improve performance. Allow at least 512 MB memory for each processor. Try adjusting the parameter when the system is paging (and processor utilization is low because of the paging).

- Disk speed and configuration can have a dramatic effect on the performance of application servers that run applications that are heavily dependent on database support, that use extensive messaging, or are processing workflow. Using disk I/O subsystems that are optimized for performance (for example, RAID array) are essential components for optimum application server performance in these environments. It is recommended that you spread the
disk processing across as many disks as possible to avoid contention issues that typically occur with one or two disk systems.

Network considerations
Run network cards and network switches at full duplex. Running at half duplex decreases performance. Verify that the network speed can accommodate the required throughput. Make sure that 100 MB is in use on 10/100 Ethernet networks.

Operation system tuning
To get a good performance, tuning operating system parameters for AIX, Sun Solaris, and Windows NT/2000 is also very important. For more details please refer to the WebSphere InfoCenter. Expand Monitoring and troubleshooting -> Performance -> Tuning Performance, then select Tuning parameter index -> Operating system.

IBM HTTP server
We have discussed tuning the HTTP server from the WebSphere Queue network perspective. In this section we discuss some important HTTP server parameters that are not mentioned in “WebSphere Queuing Network” on page 193.

Access logs
All incoming HTTP requests are logged here. Logging degrades performance because I/O operation overhead causes logs to grow significantly in a short time. To turn logging on or off, edit the IBM HTTP Server httpd.conf file, located in the directory <IBM HTTP Server Home>/conf. Search for a line with the text CustomLog. Comment out this line, then save and close the httpd.conf file. Stop and restart the IBM HTTP Server. By default, logging is enabled, but for better performance it is recommended that you disable the access logs.

MinSpareServers, MaxSpareServers, and StartServers
Pre-allocates and maintains the specified number of processes so that few processes are created and destroyed as the load approaches the specified number of processes (based on MinSpareServers). Specifying similar values reduces the CPU usage for creating and destroying HTTPD processes. Adjust this parameter if the time waiting for IBM HTTP Server to start more servers so that it can handle HTTP requests is not acceptable.

For optimum performance, specify the same value for the MaxSpareServers and the StartServers parameters. If MaxSpareServers is set to less than MinSpareServers, IBM HTTP Server resets MaxSpareServer=MinSpareServer+1. Setting the StartServers too high can cause swapping if memory is not sufficient, degrading performance.
To view or change these values, edit the following directives in the file httpd.conf, located in the directory &lt;IBM HTTP Server Home&gt;/conf:

- MinSpareServers
- MaxSpareServers
- StartServers

**IBM HTTP Server - Linux**

The following instructions are important when running the IBM HTTP Server on Linux.

**MaxRequestsPerChild**

The MaxRequestsPerChild directive sets the limit on the number of requests that an individual child server process handles. After the number of requests reaches the value set for the MaxRequestsPerChild parameter, the child process dies. If there are no known memory leaks with Apache and Apache's libraries, set this value to zero (0).

To change this value, edit the IBM HTTP server file httpd.conf located in the directory &lt;IBM HTTP Server Home&gt;/conf. Change the value of the MaxRequestsPerChild parameter. Save the changes and restart the IBM HTTP server. By default, this value is set to 500.

**Object Request Broker (ORB)**

Several settings are available for controlling internal Object Request Broker (ORB) processing. You can use these to improve application performance in the case of applications containing enterprise beans.

You can change these settings for the default server or any application server configured in the administrative domain from the Administrative Console.

**Pass by value versus Pass by reference**

This is specified in property com.ibm.CORBA.iiop.noLocalCopies. For EJB 1.1 beans, the EJB 1.1 specification states that method calls are to be *Pass by value*. For every remote method call, the parameters are copied onto the stack before the call is made. This can be expensive. The *Pass by reference*, which passes the original object reference without making a copy of the object, can be specified.

For EJB 2.0 beans, interfaces can be local or remote. For local interfaces, method calls are Pass by reference, by default.

If the EJB client and EJB server are installed in the same WebSphere Application Server instance, and the client and server use remote interfaces, specifying Pass by reference can improve performance up to 50 percent.
Please note that Pass by reference helps performance only when non-primitive object types are being passed as parameters. Therefore, data type int and floats are always copied, regardless of the call model.

**Important:** Pass by reference can be dangerous and can lead to unexpected results. If an object reference is modified by the remote method, the change might be seen by the caller.

Use the Administrative Console to set this value:

1. Select **Servers -> Application Servers**.
2. Select the application server you wish to change.
3. Select **ORB Service** from Additional Properties.
4. Select the check box **Pass by Reference**.
5. Click **OK** and **Apply** to save the changes.
6. Stop and restart the application server.

The default is Pass by value for remote interfaces and Pass by reference for EJB 2.0 local interfaces.

If the application server expects a large workload for enterprise bean requests, the ORB configuration is critical. Take note of the following properties.

**com.ibm.CORBA.ServerSocketQueueDepth**

This property corresponds to the length of the TCP/IP stack listen queue and prevents WebSphere Application Server from rejecting requests when there is no space in the listen queue.

If there are many simultaneous clients connecting to the server-side ORB, this parameter can be increased to support the heavy load up to 1000 clients. The default value is 50.

To set the property, follow these steps:

1. Select **Servers -> Application Servers**.
2. Click the application server you want to tune.
5. Enter `-Dcom.ibm.CORBA.ServerSocketQueueDepth=200` in the Generic JVM Properties field.

**Object Request Broker connection cache maximum**

This property is also called **com.ibm.CORBA.MaxOpenConnection** and corresponds to the size of the ORB connection table. The property sets the
standard for the number of simultaneous ORB connections that can be processed.

If there are many simultaneous clients connecting to the server-side ORB, this parameter can be increased to support the heavy load up to 1000 clients. The default value is 240. To change this value:

1. Select **Servers -> Application Servers**.
2. Select the application server you want to tune.
3. Select **ORB Service** under Additional Properties.
4. Update the Connection cache maximum field and click **OK**.
5. Click **Apply** to save the changes then restart the application server.

**ORB thread pool size**
Please see “EJB container” on page 202 for details.

**Dynamic Cache Service**
The Dynamic Cache Service improves performance by caching the output of servlets, commands and JavaServer Pages (JSP) files. WebSphere Application Server consolidates several caching activities, including servlets, Web services, and WebSphere commands into one service called the *dynamic cache*. These caching activities work together to improve application performance, and share many configuration parameters, which are set in an application server's dynamic cache service.

The dynamic cache works within an application server Java Virtual Machine (JVM), intercepting calls to cacheable objects, for example, through a servlet's `service()` method or a command's `execute()` method, and either stores the object's output to or serves the object's content from the dynamic cache. Because J2EE applications have high read/write ratios and can tolerate small degrees of latency in the currency of their data, the dynamic cache can create an opportunity for significant gains in server response time, throughput, and scalability.

See “Dynamic Cache Monitor” on page 192 or the InfoCenter article “Improving performance through the dynamic cache service” for more information.

### 6.3.7 Application best practice for performance

To have a good performance for a WebSphere Application Server system, the J2EE application must perform well. In this section we discuss the best practices when writing a WebSphere Application Server application that includes servlets, JSPs, JDBC connections, and EJBs for performance.
General design considerations
To make a J2EE application perform well, as a prerequisite you must ensure that your application has a good design and architecture.

Using proven patterns
It is important to reuse existing patterns and algorithms where appropriate, rather than “reinventing the wheel”. There are established algorithms and approaches for solving many of the commonly encountered problems in computer science, and any decision to use a custom solution in preference to one of these commonly used solutions should be made with caution.

Reuse the resources
J2EE applications spend much time fetching, creating, or destroying some complex objects or resources. Others may be too expensive for a single application to maintain on a one-to-one basis. Therefore, you should create a limited number of the resource, and share them from a common pool. You can pool many types of resources, from complex objects to connections. J2EE will manage some of these for you. For example, J2EE connection pools can improve performance by an order of magnitude for extreme cases. For others, you will have to create and manage the pool yourself.

Memory
A key factor in the performance of any Java application and hence any WebSphere Application Server application is the use of memory. Unlike other programming languages, Java does not require (or even allow) programmers to explicitly allocate and reclaim memory. The Java Virtual Machine (JVM) runtime environment will allocate memory when a new object is created, and will reclaim the memory once there are no more references to the object. This reduces the amount of coding required, as well as minimizing the potential for memory “leaks” caused by the programmer forgetting to deallocate memory once it is no longer required. Additionally, Java does not allow pointer arithmetic. Memory deallocation is performed by a thread executing in the JVM called the garbage collector (GC). The algorithm used for garbage collection is not specified in the Java Virtual Machine specification, and hence different JVM implementations may use different garbage collection algorithms.

Although Java performs memory management automatically, programmers still need to be aware of the impact of memory management on performance. Creating an object consumes system resources, because the JVM must allocate memory and initialize the object. Similarly, reclaiming memory using the garbage collector also uses resources, particularly CPU time. Garbage collection occurs asynchronously when free memory reaches threshold values, and it cannot be explicitly scheduled programmatically. A call to the System.gc() method will
request that the JVM performs garbage collection. However, this is not guaranteed to happen immediately or within any specified time period.

Hence, the key to minimizing the performance impact of memory management is to minimize memory usage, particularly object creation and destruction. This can be achieved by a number of means:

- **Object creation**
  Do not create objects prematurely if there is a possibility that they will not be needed. For example, if the object is only used in one path of an if statement, then create the object inside that path rather that outside the if statement—lazy initialization. If the same object can be reused inside a loop body, then declare and instantiate it outside the loop rather than inside the loop, to avoid creating and destroying a number of objects of the same class.

- **Object pools**
  If objects of the same class are being repeatedly created and destroyed, it can be beneficial to create an object pool that allows the objects to be reused. Classes whose objects will be used in a pool need an initializer, so that objects obtained from the pool have some known initial state. It is also important to create a well-defined interface to the pool to allow control over how it is used.

- **Appropriate sizing for collections**
  Although the Java runtime environment will dynamically grow the size of collections such as java.util.Vector or Java.util.Hashtable, it is more efficient if they are appropriately sized when created. Each time the collection size is increased, its size is doubled, so when the collection reaches a stable size it is likely that its actual size will be significantly greater than required. The collection only contains references to objects rather than the objects themselves, which minimizes the over allocation of memory due to this behavior.

- **Temporary objects**
  Developers should be aware that some methods such as toString() methods can typically create a large number of temporary objects. Many of the objects may be created in code that you do not write yourself, such as library code that is called by the application.

- **Use of static and final variables**
  When a value is used repeatedly and is known at compile time, it should be declared with the static and final modifiers. This will ensure that it will be substituted for the actual value by the compiler. If a value is used repeatedly but can be determined only at runtime, it can be declared as static and referenced elsewhere to ensure that only one object is created. Note that the scope of static variables is limited to the JVM. Hence if the application is
cloned, care needs to be taken to ensure that static variables used in this way are initialized to the same value in each JVM. A good way of achieving this is the use of a singleton object. For example, an EJB initial context can be cached with a singleton using the following code fragment (Example 6-5).

**Example 6-5  Use of singleton to cache EJB initial context references**

```java
public class EJBHelper {
    private static javax.naming.InitialContext initialContext = null;

    public javax.naming.InitialContext getInitialContext() {
        if (initialContext == null) {
            initialContext = new javax.naming.InitialContext();
        }
        return initialContext;
    }
}
```

- Object references

Although memory does not have to be explicitly deallocated, it is still possible to effectively have “memory leaks” due to references to objects being retained even though they are no longer required. These objects are commonly referred to as loitering objects. Object references should be cleared once they are no longer required, rather than waiting for the reference to be implicitly removed when the variable is out of scope. This allows objects to be reclaimed sooner.

Care should be taken with objects in a collection, particularly if the collection is being used as a type of cache. In this case, some criteria for removing objects from the cache is required to avoid the memory usage constantly growing. Another common source of memory leaks in Java is due to programmers not closing resources such as Java Database Connectivity (JDBC), Java Message Service (JMS), and Java Connector Architecture (JCA) resources when they are no longer required, particularly under error conditions. It is also important that static references be explicitly cleared when no longer required, because static fields will never go out of scope. Since WebSphere Application Server applications typically run for a long time, even a small memory leak can cause the Java Virtual Machine to run out of free memory.

An object that is referenced but no longer required may in turn refer to other objects, so that a single object reference can result in a large tree of objects that cannot be reclaimed. The profiling tools available in IBM WebSphere Studio Application Developer V5 can help to identify memory leaks. Other
tools that can be used for this purpose include Rational Purify®, Sitraka JProbe (by Quest Software), and Borland Optimizelt.

**Vertical clustering**

Most current garbage collection implementations are partially single threaded (during the heap compaction phase). This causes all other program threads to stop, potentially increasing the response times experienced by users of the application. The length of each garbage collection call is dependent on numerous factors, including the heap size and number of objects in the heap. Thus as the heap grows larger, garbage collection times can increase, potentially causing erratic response times depending on whether a garbage collection occurred during a particular interaction with the server. The effect of this can be reduced by using vertical scaling and running multiple copies of the application on the same hardware. This can provide two benefits: First, the JVM for each member of the cluster will only require a smaller heap; and secondly, it is likely that while one JVM is performing garbage collection, the other one will be able to service client requests as the garbage collection cycles of the JVMs are not synchronized in any way. However, any client requests that have been directed by workload management to the JVM (doing garbage collection) will be affected.

**Session management**

In a Web application, state information relating to each client is typically stored in an HTTP session, which is identified by some unique identifier that is associated with an HTTP cookie. In an environment with a single application server, session information can be stored in-memory by WebSphere Application Server V5. However, it is more common to use a clustered environment with multiple application servers to provide scalability and improve fault tolerance.

In this cluster scenario, session information needs to be made available across all of the cluster members. In past versions of WebSphere Application Server, this was achieved using a session persistence database that was available to all clones in a server group. In addition to this, a new mechanism for memory-to-memory replication has been introduced in IBM WebSphere Application Server V5.

In either case, it is beneficial to minimize the amount of data stored in the session. Since the session must be shared, it must be serialized, which also involves serializing all objects that are reachable from the session. Serialization in Java is an expensive operation. If persistent sessions are used, the serialized session data must be stored in the database, which introduces further overhead. Usually the session is stored as a binary data type (BLOB) in the database, which also may introduce overhead. If memory-to-memory replication is used, there is still overhead associated with serializing and transmitting the session data.
In order to reduce the amount of data stored in the session, avoid storing large, complex object graphs in it. In some cases it may be beneficial to store objects in the session, although they can be recreated or retrieved to avoid the overhead of doing so. In these cases, consideration should be given to making these attributes transient. If this is done, you have to ensure that the application code will handle the transient attributes having null values. Alternatively, the readObject() method of the object could be overwritten to recreate the transient data when the object is deserialized.

The session object can be garbage collected after it has been invalidated. This can be done programmatically or after a predefined timeout period during which the session was not accessed. To allow the memory used by the session to be reclaimed as early as possible, it is best to explicitly invalidate the session when finished with it rather than waiting for the timeout. This may require the introduction of logout functionality into the application, and training for the users to make use of this functionality rather than simply closing the browser.

References to the session should always be obtained from the current servlet context as required; they should not be cached by the application. This ensures that the session objects can be reclaimed when the session is invalidated. Another reason for doing this is because a session reference is not guaranteed to be valid outside of the context of a specific server interaction.

Special care must be taken when using HTML frames when each frame is displaying a JSP belonging to a different Web application on the same server. In this case, a session should only be created and accessed by one of the pages. Otherwise, although a session will be created for each page, the same cookie will be used to identify the session. This means that the cookie for each newly created session will overwrite the previous cookie, and only one of the sessions will be accessible. The remaining sessions will be created but will be inaccessible and thus will consume memory until the timeout interval is reached. If the Web application was split into multiple applications in order to improve scalability, consider combining all of the Web applications into a single one, and using clustering to achieve the required scalability.

There are alternatives to the use of the session that may be appropriate in some situations:

- In some cases use of the session can be avoided by using hidden form fields or cookies to store data. Note that there is a 4-KB limit on the total size of all cookies for a particular site. Also be aware that the use of hidden fields increases the page size and the data can be seen by the user when viewing the HTML source.

- Data can be stored directly in a database by the application. By using native data types instead of serialized BLOBs, it is often possible to achieve better performance. It is also possible to read and write only the data that has
changed, rather than the entire data set as is normally the case with BLOBs. The application will need to remove data when it is no longer required (including after a timeout period). This can be implemented by placing an object that implements the HttpSessionBindingListener interface into the session, and placing the clean up code in the valueUnBound() method.

- Entity EJBs can be used to store session data. However, the performance implications of this approach need to be assessed before implementing this type of solution. In particular, this may not perform well if complex data structures need to be stored.

**Synchronization**

The mechanism by which access to shared resources by different threads is controlled is called synchronization. While the synchronization functionality in Java is convenient and easy to use, it can introduce significant performance overhead. When a block of code is synchronized, only a single thread can execute it at any one time. There are two performance impacts of synchronization:

- Managing the monitors, the objects internal to the JVM that are used to control access to synchronized resources. Although they are not explicitly accessible by programmers, there is an overhead due to the management of the monitors by the JVM.

- Reduced concurrency, since threads have to wait for other threads to exit from synchronized blocks of code.

When using synchronization, it is best to use specific lock objects to synchronize on. Synchronizing using the keyword can cause different methods to be unnecessarily synchronized with each other, and hence reduce concurrency. Note that synchronizing on an object has a greater overhead than calling a synchronized method. However, synchronizing the method may result in significantly greater amounts of code being synchronized, again reducing the concurrency. So the trade-off between the synchronization overhead and reduced concurrency needs to be evaluated on a case-by-case basis.

In addition to the explicit use of synchronization in application code, synchronization may be used indirectly, as some of the commonly used core Java functionality uses synchronization. Some particular examples are:

- The Java I/O libraries. It is best to minimize the use of System.out.println() for this reason. Use of a multi-threaded logging library is suggested.

- Some of the Java collection classes, such as java.util.Hashtable and java.util.Vector, are synchronized. If only a single thread is accessing the data (or multiple threads are reading only), the synchronization overhead is unnecessary. Many of the newer collections introduced in Java 1.2, such as
java.util.ArrayList, are not synchronized and may provide better performance. However, care needs to be taken when accessing them from multiple threads.

**Servlets and JavaServer Pages**

Since servlets and JavaServer Pages (JSPs) can include Java code, many of the issues discussed in other sections of this chapter are relevant for JSPs as well. However, there are some particular issues that need to be considered when developing JSPs:

- Minimize the use of the `<jsp:include>` tag, since each included JSP is a separate servlet.
- When a `<jsp:usebean>` tag is encountered and an existing Java bean object with the appropriate name does not exist, a new one is created. This is done by a call to Beans.instantiate(), which is an expensive operation because the JVM checks the file system for a serialized bean. Hence it is recommended that the `<jsp:usebean>` tag only be used to obtain a reference to an existing object, rather than for creating a new object.
- In accordance with the Java 2 Enterprise Edition (J2EE) specification, when executing a JSP, a session object is normally created implicitly if one does not already exist. However, if the session is not required, creation can be avoided by the use of the `<%@ page session="false" %>` directive.
- Avoid the use of the SingleThreadModel for servlets, since it permits only one servlet thread to be executing at any time. Other client requests will be queued, waiting until a servlet to process them becomes available. This reduced concurrency can significantly reduce the throughput, and consequently increase the response times experienced by users. If a servlet has shared variables that need to be protected, it is preferable to do so using synchronization of the relevant accesses. The SingleThreadModel is effectively equivalent to synchronizing the servlets’ entire service() method.
- The `javax.servlet.Servlet.init()` method can be used to perform expensive operations that need to be performed once only, rather than using the `doGet()` or `doPost()` methods of the servlet.

**Logging**

The Java I/O classes use synchronization. Hence `System.out.println()` should not be used for logging purposes. If a lot of output using stdout is generated by an application in a UNIX environment, the overhead can be avoided by redirecting stdout to `/dev/null` in the production environment. However, a better approach is to use a multithreaded logging library such as the WebSphere logging facilities or Log4J (http://jakarta.apache.org/log4j/). In addition to providing better performance due to their multithreaded implementations, these libraries allow logging statements to be defined at a particular level, which can be dynamically changed at runtime. Thus the amount of logging in production environments can
be reduced in comparison to development and test environments without requiring code changes, improving the performance of the application. It is also good practice to guard log statements so that the parameters are not evaluated if the logging level is not on. The use of guard statements is shown in Example 6-6.

**Example 6-6 Use of guard statements for logging**

```java
if (Log.isLogging(Log.WARN))
{
    Log.log(LOG.WARN, "This is a warning");
}
```

**Enterprise JavaBeans**

Enterprise JavaBeans (EJBs) are an important aspect of the J2EE specification, and appropriate use of EJBs can support the development of modular and scalable applications. There are, however, a number of performance considerations that need to be taken into account when using EJBs:

- **Obtaining EJB references**

  Since EJBs are able to be accessed remotely, obtaining a reference to an EJB involves a lookup process. This can take a relatively long time, and hence caching the references obtained can improve performance on subsequent lookup operations. Obtaining an EJB reference typically involves the following steps:

  a. Obtain an InitialContext instance.

  b. Obtain a reference to the home interface of a particular EJB by looking up the EJB via the initial context object and performing a narrow operation.

  c. Obtain a particular EJB instance by executing the appropriate finder or create method from the EJB home instance.

  The calls to obtain an InitialContext instance and lookup of an EJB Home instance (steps a and b) are expensive, and performance can be improved by caching these objects. References to EJB homes may become stale, and hence it is best to wrap accesses to EJB Home methods with code that refreshes stale references.

  Because the WebSphere Application Server provides caching of homes automatically, application-level caching is no longer required. So for WebSphere Application Server V5, it may be a good idea to implement a home factory without caching the EJB homes, but caching the initial context.

- **Remote method calls**

  Since EJBs are intended to be accessible remotely, calls to EJB method are implemented as remote calls even if the EJB exists in a container that shares
the same JVM as the Web container, which introduces some overhead. In some cases, the overhead can be reduced by implementing the approaches outlined below.

- **Session facade**

  One of the key mechanisms to reduce EJB overhead is to minimize the number of remote calls. In general, a servlet should make a single EJB method call to a session bean to perform an operation. If required, this EJB method can call other methods as required to perform more complex operations. All access to Entity EJBs should be performed through stateless session beans. Session beans should be used to implement the business logic, while entity beans should be used for data storage and retrieval.

  The EJB sample application, AccessEmployee, shipped with DB2 V8.1, is an example that utilizes session facade.

- **Using local interface**

  IBM WebSphere Application Server V5 is compliant with the J2EE 1.3 specifications, which mandate support for EJB 2.0. The concept of a local interface has been introduced for EJB 2.0. If local interfaces are used, the overhead of serializing the parameters and transmitting them via Remote Method Invocation/Internet Inter-ORB Protocol (RMI/IIOP) is avoided. Parameters can be passed by reference instead of value, avoiding the need to copy them. In EJB 2.0, an EJB can have a remote interface, local interface, or both, although typically a particular bean will have either a remote or a local interface. Since Entity EJBs are normally accessed from session beans, in most cases they will only need local interfaces.

- **Transaction management**

  In addition to the overhead associated with remote method calls, transaction management provided by the EJB container can also introduce overhead. Session facade can limit the impact of this by reducing the number of transactions. In the deployment descriptor for EJBs, the transaction type can be specified. This can be one of five values:

  - **NotSupported**
  - **Supports**
  - **Required**
  - **Requires New**
  - **Mandatory**

  Set the transaction type to **NotSupported** if no transaction is required.

- **Entity EJBs**

  While entity beans can reduce the amount of coding work required by freeing programmers from having to write code for persisting data, care must be
taken with the use of entity beans to avoid performance problems. As mentioned earlier, we recommend that using session facade to access entity beans, instead of directly by servlets or other clients. The EJB access beans functionality provided in IBM WebSphere Studio Application Developer V5 can also be used to simplify access to EJBs.

In cases where entity beans have deep relationships or significant use of inheritance, care must be taken to ensure that performance is adequate. If performance problems are encountered, they can potentially be addressed by reducing the use of inheritance in the model or by use of bean-managed persistence (BMP) in preference to container-managed persistence (CMP). Another strategy is to avoid turning each entity (table) into a single EJB. In some cases two or more related entities can be represented by a single EJB.

**Important:** The following applies only to EJBs that are compliant with the EJB 1.1 specification. We recommend that new EJBs be developed to be compliant with the EJB 2.0 specification. However, some existing EJBs may only be compliant with EJB 1.1

Entity EJBs are best used for manipulating small amounts of data. Returning large result sets from (default) EJB finders can be inefficient and it is often better to use JDBC directly from a session bean for this.

- **Isolation levels and read-only methods**

  If an entity bean has methods that do not update attributes (getter type methods) they can be specified as read-only methods in the deployment description. This will avoid executing a SELECT for UPDATE/UPDATE pair of SQL statements, and hence will provide a performance improvement. EJBs deployed in WebSphere Application Server can be assigned one of four transaction isolation levels. These are Repeatable Read, Read Committed, Read Uncommitted and Serializable. The default level is Repeatable Read, but performance can be improved by the use of a lower isolation level such as Read Committed.

  **Note:** The following applies only to EJBs that are compliant with the EJB 2.0 specification.

- **Access intent**

  The access intent for each method of an EJB 2.0 CMP entity bean can be specified in the EJB deployment descriptor to provide hints to the EJB container about how the method will be used. The supported access intents are:

  - pessimistic update - Weakest Lock At Load
In general, read-only methods and optimistic locking will provide better performance. However, be careful with the use of optimistic locking since problems may only become apparent under heavy load and hence may not be found in development. For more information please refer to “Enable database persistence” on page 313.

### Stateful session beans

The use of stateful session beans should be minimized (or avoided if possible). Unlike stateless session beans, they are not able to be workload managed or pooled. The EJB container may passivate inactive stateful session beans and activate them if they are referenced later. This involves serializing and deserializing the beans, which can impact the performance. In cases where stateful session beans are required, the occurrence of passivating the beans can be reduced by explicitly removing the beans when they are no longer required, rather than waiting for them to time out. One scenario in which stateful session beans may be useful is the implementation of a cache for application data. This may use JDBC or a BMP entity bean to retrieve the data to be cached.

### EJB caching

The EJB container has three types of caching that can be performed for entity beans between transactions. Selection of the appropriate option requires an understanding of how the entity beans will be used, as there is a trade-off between minimizing database reads and supporting Workload Management (WLM).

- **Option A caching**
  
  With this option, it is assumed that the data represented by the EJB is exclusively accessed by the EJB container and is not accessed by other containers or by some mechanism that is external to the application server. Use of this option can improve performance, since the data does not have to be retrieved again from the database prior to each transaction. However, if option A caching is used, the entity bean cannot be clustered or participate in WLM.

- **Option B caching**
  
  The entity bean will remain in the cache throughout the transaction, but is reloaded at the start of each method call. This introduces the greatest overhead.
– Option C caching

  The entity bean will be reloaded at the beginning of each transaction. This is the default setting, and allows for a client to access a particular bean from any container.

➤ Asynchronous message processing

  Asynchronous Java Message Service (JMS) messages can be processed using message-driven beans (MDBs), which have been introduced in WebSphere Application Server V5. This avoids the need for developing individual server programs to receive and process incoming messages.

**Database access**

For the best practices in database access, please refer to “Application considerations for performance in database access” on page 332 for more information.
Monitoring and tuning of DB2 UDB V8

Performance tuning begins at the design phase. The performance goals of both the system and the application should be specified in the project requirements. The performance requirements should be considered in the design and implemented while the application is being developed. In this chapter we show you the performance considerations and tuning methods of a database and an application starting from the design stage.

We introduce some tools that are useful in monitoring and tuning the system. These tools are all part of DB2 UDB, and there is no need to spend extra money buying these software.

We then discuss database- and application-related performance topics that DBAs and developers should consider while building and developing the system.

Finally we show how to administer a running system. We show you where you can find the information you need to monitor your running system. We describe what you should look at and how to tune it if there is problem.
7.1 Tools for monitoring and tuning

DB2 UDB offers several tools that help you monitor and tune your databases and applications. Before you start tuning your system, you must have an idea of what to tune. Monitoring helps find out what is going on in your system; for example, if you have a long running transaction and need to know what the reason is. One method to handle the problem is to try something and see if it helps. The better method is to monitor the system, pinpoint the problem, and resolve the problem.

Monitoring consumes system resources. Therefore, it is a good idea to do it within a test environment, and not in a production system. If you cannot recreate the problem in a test environment or if the production system and the test system are not comparable, then you should start monitoring in production. But use monitoring only as long as needed and switch it off after you finish your tuning.

The database manager parameter `MON_HEAP_SZ` configures the amount of memory reserved for monitoring within DB2 UDB. The default value is not designed for high monitor activities. Therefore it might be necessary to change the parameter to a higher value if you plan to monitor.

7.1.1 Snapshot monitor

The snapshot monitor gives you the status of your database server for a specific point in time. DB2 UDB offers several monitors that can be turned on or off separately. To view the available monitoring options and the status use the command `DB2 GET MONITOR SWITCHES`. In Figure 7-1, you can see an example output. By default, the monitor switches are turned off except for the monitor for timestamp information that is turned on by default. If a switch is on, you can see from the output of `DB2 GET MONITOR SWITCHES` when the monitor was activated or reset.

Note: Any switch (except DFT_MON_TIMESTAMP) that is turned ON instructs the database manager to collect monitor data related to that switch. Collecting additional monitor data increases database manager overhead that can impact system performance. Turning the dft_mon_timestamp switch OFF becomes important as CPU utilization approaches 100 percent.

The values collected from the monitor output will be accumulated from the moment the monitor switch is activated or reset. The time displayed behind an active monitor shows the timestamp for the monitor that was activated. With the reset command this time will not be updated.
Chapter 7. Monitoring and tuning of DB2 UDB V8

Figure 7-1 Status of monitor switches for snapshot monitoring

To change the status of a monitor, change the switches with the command shown in the Example 7-1.

Example 7-1 Command syntax for UPDATE MONITOR SWITCHES

```
UPDATE MONITOR SWITCHES USING {switch-name {ON | OFF} ...} [AT DBPARTITIONNUM db-partition-number | GLOBAL]
switch-name: BUFFERPOOL, LOCK, SORT, STATEMENT, TABLE, TIMESTAMP, UOW
```

Example 7-2 shows how to turn on the monitor switches that are turned off by default. Any combination is possible with ON or OFF.

Example 7-2 Command to activate snapshot monitoring

```
-- activates all monitor switches except TIMESTAMP that is on by default
UPDATE MONITOR SWITCHES USING BUFFERPOOL ON LOCK ON SORT ON STATEMENT ON TABLE ON UOW ON;
```

The switches for snapshot monitoring are also part of the database manager configuration. In the Default database monitor switches section of the DB2 GET DBM CFG command output, you can see the startup configuration of the monitor switches. Using the command UPDATE DBM CFG USING ..., you can configure the monitor switches to be active from the startup of the server. You should turn on a switch in the configuration file only if you want to collect data starting from the moment the database manager is started. Otherwise, each monitoring application should set its own switches, and the data collected are relative to the time its switches are set.
In Table 7-1 we show you the monitor switches along with the information they provide. The DBM parameters shown in the table are online configurable. No `db2stop` and `db2start` are needed to activate the changes.

### Table 7-1 Data returned by the snapshot monitor

<table>
<thead>
<tr>
<th>Group</th>
<th>Information provided</th>
<th>Monitor Switch</th>
<th>DBM parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bufferpools</td>
<td>Number of reads and writes from/to bufferpool and disk; time taken</td>
<td>BUFFERPOOL</td>
<td>DFT_MON_BUFPOOL</td>
</tr>
<tr>
<td>Locks</td>
<td>Number of locks held; number of deadlocks; what is locked and which lock mode is used</td>
<td>LOCK</td>
<td>DFT_MON_LOCK</td>
</tr>
<tr>
<td>Sorts</td>
<td>Number of heaps used; sort overflows; performance</td>
<td>SORT</td>
<td>DFT_MON_SORT</td>
</tr>
<tr>
<td>SQL statements</td>
<td>Start and stop time</td>
<td>STATEMENT</td>
<td>DFT_MON_STMT</td>
</tr>
<tr>
<td>Tables</td>
<td>Rows read; rows written</td>
<td>TABLE</td>
<td>DFT_MON_TABLE</td>
</tr>
<tr>
<td>Timestamps</td>
<td>Timestamp information</td>
<td>TIMESTAMP</td>
<td>DFT_MON_TIMESTAMP</td>
</tr>
<tr>
<td>Unit of work</td>
<td>Start and end time; completion status</td>
<td>UOW</td>
<td>DFT_MON_UOW</td>
</tr>
</tbody>
</table>

To retrieve the collected information use the DB2 GET SNAPSHOT command. This command is very complex. Example 7-3 shows the complete syntax.

### Example 7-3 Command syntax for get snapshot

```
>>-GET SNAPSHOT FOR--------------------------------------------->

>---++---DATABASE MANAGER++---WRITE TO FILE ---------------+++--->
  | +--DB MANAGER------+                                 |
  | 'DBM--------------'                                 |
  +--ALL--++---DATABASES-----------------------------+   |
  | 'DCS-'                                        |
  +--ALL--++---APPLICATIONS--------------------------+   |
  | 'DCS-'                                        |
  +--ALL BUFFERPOOLS-----------------------------------+   |
  +--APPLICATION++--APPLID--appl-id++               |
  | 'DCS-'                                         |
  +--FCM FOR ALL DBPARTITIONNUMS---------------------+   |
  +--LOCKS FOR APPLICATION++--APPLID--appl-id-------+   |
  | 'AGENTID--appl-handle-'                         |
  +--ALL REMOTE DATABASES-----------------------------+   |
  +--ALL REMOTE APPLICATIONS--------------------------+   |
  '++ALL-------------------+--ON--database-alias---'
  +--DATABASE+++               |
  | 'DCS-' 'DB-------'                |
  +--APPLICATION+++               |
```
The information you can receive from the snapshot is very extensive. As you can see from the options, it is possible to take a snapshot from several points of view, starting from a snapshot for the database manager with a global view to a granular snapshot of dynamic SQL statements. The snapshots can also be taken when the monitor switches are switched off. Some information is collected all of the time, and for those are not collected by default, the values are marked with \textsc{NOT COLLECTED}.

In Example 7-4 and Example 7-5 on page 242 we present example output of snapshots to give you a feeling how they look. We show the outputs of a snapshot for the database manager and a database. For more details we recommend using the snapshot monitor and the online documentation.

\textit{Example 7-4} \hspace{1em} \textbf{Output of a DBM snapshot}

\begin{verbatim}
Database Manager Snapshot

Node name =
Node type = Enterprise Server Edition with local and remote clients
Instance name = db2inst1
Number of database partitions in DB2 instance = 1
Database manager status = Active

Product name = DB2 v8.1.1.24
Service level = s030728 (U488481)

Private Sort heap allocated = 0
Private Sort heap high water mark = 277
Post threshold sorts = 0
Piped sorts requested = 4
Piped sorts accepted = 4

Start Database Manager timestamp = 10-24-2003 10:34:43.403708
Last reset timestamp = 11-05-2003 09:35:14.167372
Snapshot timestamp = 11-05-2003 13:16:11.390812

Remote connections to db manager = 4
Remote connections executing in db manager = 0
\end{verbatim}
Local connections             = 0
Local connections executing in db manager = 0
Active local databases       = 1

High water mark for agents registered = 13
High water mark for agents waiting for a token = 0
Agents registered            = 13
Agents waiting for a token   = 0
Idle agents                  = 8

Committed private Memory (Bytes) = 4947968

Switch list for db partition number 0
Lock Information             (LOCK) = ON  10-24-2003 11:08:45.878709
Sorting Information          (SORT) = ON  11-05-2003 09:33:14.184821
SQL Statement Information    (STATEMENT) = ON  11-05-2003 09:33:14.184815
Table Activity Information   (TABLE) = ON  11-05-2003 09:33:14.184817
Take Timestamp Information   (TIMESTAMP) = ON  10-24-2003 10:34:43.403708
Unit of Work Information     (UOW) = ON  11-05-2003 09:33:14.184812

Agents assigned from pool                            = 355
Agents created from empty pool                       = 15
Agents stolen from another application               = 0
High water mark for coordinating agents              = 13
Max agents overflow                                   = 0
Hash joins after heap threshold exceeded              = 0

Total number of gateway connections                   = 0
Current number of gateway connections                 = 0
Gateway connections waiting for host reply            = 0
Gateway connections waiting for client request        = 0
Gateway connection pool agents stolen                 = 0

Memory usage for database manager:

Memory Pool Type                                      = Database Monitor Heap
Current size (bytes)                                  = 311296
High water mark (bytes)                               = 344064
Maximum size allowed (bytes)                          = 540672

Memory Pool Type                                      = Other Memory
Current size (bytes)                                  = 5423104
High water mark (bytes)                               = 5439488
Maximum size allowed (bytes)                          = 15958016

Example 7-5  Output of a database snapshot

Database Snapshot

Database name                                      = TRADE3DB
Database path                                     = /home/db2inst1/db2inst1/NODE0000/SQL00002/
Input database alias                              = TRADE3DB
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database status</td>
<td>Active</td>
</tr>
<tr>
<td>Catalog database partition number</td>
<td>0</td>
</tr>
<tr>
<td>Catalog network node name</td>
<td></td>
</tr>
<tr>
<td>Operating system running at database server</td>
<td>AIX</td>
</tr>
<tr>
<td>Location of the database</td>
<td>Remote</td>
</tr>
<tr>
<td>First database connect timestamp</td>
<td>10-24-2003 11:08:45.657390</td>
</tr>
<tr>
<td>Last reset timestamp</td>
<td>11-05-2003 09:35:14.167372</td>
</tr>
<tr>
<td>Last backup timestamp</td>
<td></td>
</tr>
<tr>
<td>Snapshot timestamp</td>
<td>11-05-2003 13:51:35.622079</td>
</tr>
<tr>
<td>High water mark for connections</td>
<td>9</td>
</tr>
<tr>
<td>Application connects</td>
<td>4</td>
</tr>
<tr>
<td>Secondary connects total</td>
<td>1</td>
</tr>
<tr>
<td>Applications connected currently</td>
<td>4</td>
</tr>
<tr>
<td>Appls. executing in db manager currently</td>
<td>0</td>
</tr>
<tr>
<td>Agents associated with applications</td>
<td>4</td>
</tr>
<tr>
<td>Maximum agents associated with applications</td>
<td>9</td>
</tr>
<tr>
<td>Maximum coordinating agents</td>
<td>9</td>
</tr>
<tr>
<td>Locks held currently</td>
<td>0</td>
</tr>
<tr>
<td>Lock waits</td>
<td>0</td>
</tr>
<tr>
<td>Time database waited on locks (ms)</td>
<td>0</td>
</tr>
<tr>
<td>Lock list memory in use (Bytes)</td>
<td>3420</td>
</tr>
<tr>
<td>Deadlocks detected</td>
<td>0</td>
</tr>
<tr>
<td>Lock escalations</td>
<td>0</td>
</tr>
<tr>
<td>Exclusive lock escalations</td>
<td>0</td>
</tr>
<tr>
<td>Agents currently waiting on locks</td>
<td>0</td>
</tr>
<tr>
<td>Lock Timeouts</td>
<td>0</td>
</tr>
<tr>
<td>Number of indoubt transactions</td>
<td>0</td>
</tr>
<tr>
<td>Total Private Sort heap allocated</td>
<td>0</td>
</tr>
<tr>
<td>Total Shared Sort heap allocated</td>
<td>0</td>
</tr>
<tr>
<td>Shared Sort heap high water mark</td>
<td>0</td>
</tr>
<tr>
<td>Total sorts</td>
<td>4</td>
</tr>
<tr>
<td>Total sort time (ms)</td>
<td>0</td>
</tr>
<tr>
<td>Sort overflows</td>
<td>0</td>
</tr>
<tr>
<td>Active sorts</td>
<td>0</td>
</tr>
<tr>
<td>Buffer pool data logical reads</td>
<td>31</td>
</tr>
<tr>
<td>Buffer pool data physical reads</td>
<td>3</td>
</tr>
<tr>
<td>Asynchronous pool data page reads</td>
<td>0</td>
</tr>
<tr>
<td>Buffer pool data writes</td>
<td>0</td>
</tr>
<tr>
<td>Asynchronous pool data page writes</td>
<td>0</td>
</tr>
<tr>
<td>Buffer pool index logical reads</td>
<td>2</td>
</tr>
<tr>
<td>Buffer pool index physical reads</td>
<td>0</td>
</tr>
<tr>
<td>Asynchronous pool index page reads</td>
<td>0</td>
</tr>
<tr>
<td>Buffer pool index writes</td>
<td>0</td>
</tr>
<tr>
<td>Asynchronous pool index writes</td>
<td>0</td>
</tr>
<tr>
<td>Total buffer pool read time (ms)</td>
<td>9</td>
</tr>
<tr>
<td>Total buffer pool write time (ms)</td>
<td>0</td>
</tr>
<tr>
<td>Total elapsed asynchronous read time</td>
<td>0</td>
</tr>
<tr>
<td>Total elapsed asynchronous write time</td>
<td>0</td>
</tr>
<tr>
<td>Asynchronous data read requests</td>
<td>0</td>
</tr>
<tr>
<td>Asynchronous index read requests</td>
<td>0</td>
</tr>
<tr>
<td>Asynchronous index read requests</td>
<td>0</td>
</tr>
</tbody>
</table>
LSN Gap cleaner triggers = 0
Dirty page steal cleaner triggers = 0
Dirty page threshold cleaner triggers = 0
Time waited for prefetch (ms) = 5
Unread prefetch pages = 0
Direct reads = 0
Direct writes = 0
Direct read requests = 0
Direct write requests = 0
Direct reads elapsed time (ms) = 0
Direct write elapsed time (ms) = 0
Database files closed = 0
Data pages copied to extended storage = 0
Index pages copied to extended storage = 0
Data pages copied from extended storage = 0
Index pages copied from extended storage = 0

Host execution elapsed time = 0.027284

Commit statements attempted = 161
Rollback statements attempted = 159
Dynamic statements attempted = 293
Static statements attempted = 2
Failed statement operations = 0
Select SQL statements executed = 209
Update/Insert/Delete statements executed = 16
DDL statements executed = 0

Internal automatic rebinds = 0
Internal rows deleted = 0
Internal rows inserted = 0
Internal rows updated = 0
Internal commits = 0
Internal rollbacks = 0
Internal rollbacks due to deadlock = 0

Rows deleted = 1
Rows inserted = 3
Rows updated = 12
Rows selected = 551
Rows read = 707
Binds/precompiles attempted = 0

Log space available to the database (Bytes) = 20400000
Log space used by the database (Bytes) = 0
Maximum secondary log space used (Bytes) = 0
Maximum total log space used (Bytes) = 8393003
Secondary logs allocated currently = 0
Log pages read = 0
Log pages written = 9

Package cache lookups = 225
Package cache inserts = 20
Package cache overflows = 0
Package cache high water mark (Bytes)      = 3186052
Application section lookups                = 293
Application section inserts                = 20
Catalog cache lookups                      = 48
Catalog cache inserts                      = 0
Catalog cache overflows                    = 0
Catalog cache high water mark              = 0

Workspace Information
Shared high water mark                    = 0
Corresponding shared overflows            = 0
Total shared section inserts              = 0
Total shared section lookups              = 0
Private high water mark                   = 838854
Corresponding private overflows           = 0
Total private section inserts             = 20
Total private section lookups             = 205

Number of hash joins                       = 0
Number of hash loops                       = 0
Number of hash join overflows              = 0
Number of small hash join overflows        = 0

Memory usage for database:
Memory Pool Type                           = Backup/Restore/Util Heap
      Current size (bytes)                   = 16384
      High water mark (bytes)                = 16384
      Maximum size allowed (bytes)           = 20660224

Memory Pool Type                           = Package Cache Heap
      Current size (bytes)                   = 4096000
      High water mark (bytes)                = 4505600
      Maximum size allowed (bytes)           = 4294950912

Memory Pool Type                           = Catalog Cache Heap
      Current size (bytes)                   = 638976
      High water mark (bytes)                = 638976
      Maximum size allowed (bytes)           = 4294950912

Memory Pool Type                           = Buffer Pool Heap
      Current size (bytes)                   = 4341760
      High water mark (bytes)                = 4341760
      Maximum size allowed (bytes)           = 4294950912

Memory Pool Type                           = Buffer Pool Heap
      Current size (bytes)                   = 671744
      High water mark (bytes)                = 671744
      Maximum size allowed (bytes)           = 4294950912

Memory Pool Type                           = Buffer Pool Heap
      Current size (bytes)                   = 409600
7.1.2 Event monitor

The event monitor is a tool that allows you to monitor ongoing actions in the database server. It is a good tool for checking problems that are very difficult to handle using the snapshot monitor. One example is to find the reason for a deadlock situation. A deadlock event monitor waits for a deadlock to occur; when one does, it collects information about the applications involved and the locks in contention. Monitoring a deadlock situation with a snapshot monitor could be difficult. DB2 will roll back any transactions that are part of the deadlock situation except one that is allowed to finish the transaction. The timeframe for this action is very short; therefore, a tool like Event monitor, that records actions within a period of time, is better than one that collects the state in a point of time.

To create an event monitor, use the CREATE EVENT MONITOR SQL statement. Event monitors collect event data only when they are active. To activate or deactivate an event monitor, use the SET EVENT MONITOR STATE SQL statement. The status of an event monitor (whether it is active or inactive) can be determined by the SQL function EVENT_MON_STATE.
Another way to create and control event monitors is by using the Control Center. Figure 7-2 shows a capture of the Control Center with the event monitors view open. This is a good way to see if the monitors are running or not.

![Control Center with event monitors](image)

Figure 7-2 Event monitors from within Control Center

With DB2 UDB V8, you can store the monitor output in tables. Other options are to use files or named pipes. Be careful because event monitors gather a large amount of data. You should reserve enough space in the output directory or use tables to store the data. Using tables allows you to define exactly which data elements you want. Data that is not useful can be discarded. Once the data is stored in tables you can use SQL to retrieve the information. This is a very powerful and flexible way.

The SQL command you need to create an event monitor from outside of the Control Center is shown in Example 7-6.

Example 7-6 Syntax of CREATE EVENT MONITOR statement

```sql
CREATE EVENT MONITOR--event-monitor-name--FOR---------------->
```

`V
-------------------+-------------------
| >--------+DATABASE

Chapter 7. Monitoring and tuning of DB2 UDB V8 247`
<table>
<thead>
<tr>
<th>TABLES</th>
<th>DEADLOCKS</th>
<th>TABLESPACES</th>
<th>BUFFERPOOLS</th>
<th>CONNECTIONS</th>
<th>STATEMENTS</th>
<th>TRANSACTIONS</th>
</tr>
</thead>
</table>

**WRITE TO**

- TABLE
  - Table Options
    - PIPE
    - FILE

**MANUALSTART**

**AUTOSTART**

**LOCAL**

**ON DBPARTITIONNUM**

**Event Condition**

- **AND** | **OR**

  - **NOT**
    - AUTH_ID
    - APPL_ID
    - APPL_NAME

**Table Options**

- **EVMTARGET**
  - **-BLOCKED**
  - **BUFFERSIZE**

**targetTableInfo**

**-MANUALSTART**

**-AUTOSTART**

**-LOCAL**

**-ON DBPARTITIONNUM**

**-GLOBAL**
From the Control Center, the event monitors will be created using a wizard. Within the Control Center you can only see those event monitors that are defined to write to tables. To create an event monitor that stores the data in tables use this command:

```sql
DB2 CREATE EVENT MONITOR dlmon FOR eventtype WRITE TO TABLE
```

If you do not specify a table name, DB2 generates names for you. To find the table names DB2 generated use the following command:

```sql
DB2 SELECT * FROM SYSCAT.EVENTTABLES WHERE EVMONNAME = 'dlmon'
```

Or use DB2 LIST TABLES and look for tables like:

- connheader_dlmon
- conn_dlmon
- deadlock_dlmon
- dlconn_dlmon
- dllock_dlmon
- control_dlmon

Once you have chosen the table names you can start analyzing what the event monitor has collected. For example, if you want to look for the long-running statements, use:

```sql
SELECT * FROM STMT_TEST WHERE START_TIME + 60 SECOND < STOP_TIME
```

If you chose to use a file to store the output of the event monitor you can analyze the results with the `db2evmon` command. This method is also useful but not so powerful like the one using SQL to analyze the collected data.
7.1.3 Explain utilities

When DB2 UDB is requested to retrieve data, it uses an optimizer to determine which access plan is the most efficient one. It is possible to see the access plan that DB2 UDB created and to view how expensive the retrieval is.

The optimizer uses several inputs to calculate the best access plan. Information like the speed and count of CPUs and disks is important. DB2 calculates the CPU speed during the installation and stores it in the database manager configuration parameter CPUSPEED. This value should not be changed unless you are modeling a different hardware environment. If you change the CPUs in a machine, you can set the value to -1 to have DB2 compute the new CPU speed. The access plans in your development environment can differ from the ones you have in the productive environment. So if you deploy your database to a productive system, it is necessary to look at the access plans again. Access plans can differ due to speed and number of CPUs and disks changed.

By default DB2 UDB assumes that you have very slow disks and calculates the access plan with that assumption. To have the best plan you need to tell DB2 the characteristics of your disks. For detailed information look at Chapter 4 of the Table space impact on query optimization in Administration Guide: Performance, SC09-4821.

Other very important information that the optimizer uses is the statistics of the tables. It is necessary to keep this information up to date and to rebind your applications once the statistics are re-collected.

It is possible to configure how intensivly DB2 UDB should search for the best access plan. You can choose between 10 optimization classes, where 0 means minimal optimization and 9 stands for using all available optimization techniques. By default DB2 UDB uses the optimization class 5 configured with the database configuration parameter DFT_QUERYOPT. You can change this default value or use DB2 SET CURRENT QUERY OPTIMIZATION = value when accessing the data with statements like:

```
SELECT PKGNAME, PKGSHEMA FROM SYSCAT.PACKAGES WHERE QUERYOPT = CURRENT QUERY OPTIMIZATION.
```

The db2exfmt and Visual Explain utilities require the DB2 Explain tables to be built for each user who is executing the explain commands. The db2exfmt utility will dynamically explain static SQL and does not require the DB2 Explain tables. A variation of the db2exfmt utility, dynexfmt, allows dynamic SQL to be explained without using the DB2 Explain tables. Further details on the db2exfmt and dynexfmt utilities can be found by using the online help available within the DB2 Command Line Processor, by entering db2exfmt -h or dynexfmt -h. These two explain utilities can be useful if the Explain tables are not created or if a user needs to run an explain but does not own a DB2 Explain table schema.
The easiest way to create an access plan is from the Command Center by choosing the menu Interactive -> Create Access Plan or using the icon . The explain tables will be created automatically if not still present. A message will inform you that the explain tables have been created. Another option to create explain tables is to run the EXPLAIN.DDL script that is located under sqlib\misc.

For viewing the access plan you have several options. The graphical version can be seen in the Command Center when choosing the menu Interactive -> Create Access Plan. Figure 7-3 shows a simple example for the output for the statement:

```
SELECT * FROM ORDEREJB WHERE ORDERSTATUS = 'closed'.
```

![Visual explain](image)

*Figure 7-3  Visual explain*

The db2exfmt utility will format the data contained in the DB2 Explain tables in a textual report that explains the access plans selected for each SQL statement that is contained in the DB2 Explain tables.

For more details we recommend the following DB2 UDB manuals:
- Command Reference, SC09-4828
- Administration Guide: Performance, SC09-4821

### 7.1.4 DB2 Diagnostic Log (DB2DIAG.LOG)

DB2 UDB provides a diagnostic capturing mechanism. It writes information into a file called DB2DIAG.LOG. By default, this file is written to:

- On UNIX platforms
  
  `$INSTHOME/sqlib/db2dump` directory
Where INSTHOME is the database instance home directory.

- On Intel platforms:
  - If the DB2INSTPROF environment variable is not set:
    x:\Program Files\IBM\SQLLIB\%DB2INSTANCE%
  - If DB2INSTPROF is set:
    x:\%DB2INSTPROF%\%DB2INSTANCE%

Where x:\SQLLIB is the drive reference and directory specified in the DB2PATH registry variable. DB2INSTANCE is the name of the instance owner and DB2INSTPROF is the name of the instance profile directory.

In most situations, the default paths are adequate and need not be changed.

In DB2DIGA.LOG, you can find information like values of parameters that need to be increased, or errors that occurred, or lock escalations, and more. The value of the database manager parameter DIAGLEVEL defines what level of diagnostic information will be captured. The levels are:

- 0: No diagnostic data captured
- 1: Severe errors only
- 2: All errors
- 3: All errors and warning (default)
- 4: All errors, warnings, and informational messages

In general level 3 is sufficient. In a developing environment it is sometimes helpful to change to level 4 to have all the details captured. With level 4 you can, for example, see what Java environment is used by DB2. The IBM Support also needs this information. If possible use a level 4 diaglevel when you need support. The management of the DB2DIGA.LOG file is by the administrator, manually. This file can be deleted without impacting the DB2 UDB system. DB2 UDB will recreate the file if it is not exited.

### 7.1.5 Health Center/Memory Visualizer

The Health Center is a new feature of DB2 UDB Version 8. The Health Monitor gathers information about the health of the system using new interfaces that do not impose a performance penalty. It allows you to monitor the state and utilization of many parts of the database manager and databases. It automatically evaluates a set of health indicators. If the Health Monitor detects an abnormal state for an object, the Health Monitor will raise an alert. The health state of the system is presented using traffic light-like icons. Green indicates the state is normal. If you find a yellow or red state, you need to tune the system. You can set up the Health Center to inform you by e-mail if an alert occurs. You also
can set up what should be done in a case of an alert so that the system can heal itself.

The Memory Visualizer is a part of the Health Center. With the Memory Visualizer you can monitor the memory usage like sort heaps, buffer pools, and caches. It displays the memory allocation for a particular instance and all of its databases. Because many of the memory configuration parameters are dynamic, you can change these parameters and monitor if your change has the expected result.

### 7.1.6 Design Advisor

Indexes play an important part in tuning the performance. An index is good when retrieving data because data access is, in general, much faster through an index than scanning an entire table to find matching rows. But having indexes is also an overhead. An index needs storage, and every insert and delete needs a write operation on the table and also on the index. If an update changes the value of an index field, an update on the index is also necessary. So indexes are important but not every index is helpful.

The Design Advisor is a wizard that can be started from the Control Center or from the command line using `db2advis`. The Design Advisor can help you design and define suitable indexes for your data. It can:

- Find the best indexes for a problem query.
- Find the best indexes for the set of queries that define in a workload, subject to resource limits that are optionally applied.
- Test an index or materialized query table on a workload without having to create the index or materialized query table.

### 7.1.7 Configuration Advisor

The Configuration Advisor is another helpful tool for DBAs to evaluate and tune the database system. This graphical tool takes you step-by-step to gather the information about your system requirements. Once the process is completed, the Configuration Advisor provides you with the recommended database configuration parameter values for an optimal performance. These values can be applied immediately or later by Task Center or via the command line processor.

In Figure 7-4 we show the recommendations screen of the Configuration Advisor.
7.2 Application tuning

Tuning the DB2 UDB server is very important to achieve the performance goal. Well-designed databases and a performance-optimized application play a major role in reaching the performance tuning goal. From our experiences, very often the application or the database design are the reasons for a performance problem. In this section we discuss, from a performance-tuning point of view, some major topics and the best practices for database design and what to keep in mind when developing an application. Performance tuning is an on-going task. These guidelines are also applicable for database and application tuning once the application is in production.

7.2.1 Database design

Before you start the database design, you need to understand application requirements and performance goals. The database logical and physical design, database server system resource needed will be based on the application requirement and performance goal. If you can request the system you need, examine the performance requirement carefully and order hardware with enough...
capacity. If you are given a system, check if the system has enough capacity to meet the performance goal.

Often the database server resides on a separate server. In this case, you can use all the resources available. You will have to share the resources with the application if you have a single server containing the application and database server. The recommendation is to have a separate server for DB2 UDB when having an environment including WebSphere application server.

**Gather the information**

To gather the application requirements for setting up a database, you should look for the following items:

- How many users will connect to DB2 UDB? This number may be different from the number of real users when using connection pooling. The more connections you have the more memory is necessary.

- How many instances/databases will be on the system? The configuration is different when more than one instances/databases are on the server. So you have to think about how many instances to create and where to put the databases.

- How big is the volume of the data?

Consider the following resources for DB2 UDB database server:

- CPU: How many and what speed?
  It is better to have more than one CPU to allow parallel processing. The speed of the CPU is necessary for the optimizer. During installation, DB2 UDB will measure the speed. Any change afterwards need to be configured in the DBM CFG.

- Memory: How much is available or required (if one machine, how much available for DB2)?
  You need to know how much memory is available when configuring the database server.

- Disks: What kind of storage and how many disks?
  How the data is stored is important and only one disk will slow down the system because I/O will become the bottleneck. Many disks allow parallel processing.

You should consider the performance from the very beginning on both logical and physical database design. There is a lot of literature available that covers the database design theory. We cannot discuss them one by one but we provide information here to give you a basic understanding of this topic.
Logical database design

DB2 UDB is a relational database server. A relation can be displayed as a table with two dimensions. The first dimension is the fields of the table, while the second dimension is the values. Each field has a data type. The data types of DB2 UDB are described in “Data types” on page 67.

Data types

Data types should be defined to minimize disk storage consumption, and any unnecessary processing, such as data type transformations. The following are some of the main recommendations as they apply to the choice of data types required by the application:

- Use SMALLINT rather than INTEGER if SMALLINT can contain the size of the number.
- Use the DATE and TIME data types rather than CHAR for data/time data.
- Use NOT NULL for columns wherever possible.
- Choose data types that can be processed most efficiently.
- Use CHAR rather than VARCHAR for any narrow columns (for example, those that are less than 50 characters wide).
- Choose VARCHAR instead of LONG VARCHAR when the row size is less than 32 K.
- Use FLOAT rather than DECIMAL if exact precision is not required.
- Use IDENTITY columns where appropriate.

Normalization

An important thing in designing tables is to use normalization. There are several level of normalization to guarantee consistency of the data and to reduce redundant data. Figure 7-5 shows how tables get normalized and denormalized. In general normalization means more tables with less columns, and denormalization vice versa.
The level most often used is called the third normal form. For performance reasons, sometimes it makes sense to have some data redundancy to speed up access, especially in Data Warehouses where only read accesses occur. Redundant data has the following characteristics:

- More disk space is needed.
- More network traffic and memory usage when inserting data.
- Data can become inconsistent because of failure during insert or update.
- When redundant data gets updated all occurrences need an update.
- Can be faster for accessing data because less optimization is needed and data can be accessed with less read operations.

Once the tables are normalized, you should consider defining constraints and primary keys to let DB2 UDB take care of data consistency. The constraint is a relation put in place between two tables and is one of the good practices in database design. We do not discuss this topic here because it is not a performance factor.

**Index**

Indexes are important for a good performance and we will cover them in more detail during the system tuning discussion. The need of indexes can be seen when you designed the application. In general this task is performed by the DBA. Some considerations should be taken into account during the design:

- Define primary keys and unique keys, wherever possible, by using the CREATE UNIQUE INDEX statement. That helps by avoiding some sorts.
To improve join performance with a multiple-column index, if you have more than one choice for the first key column, use the column most often specified with the “=” predicate or the column with the greatest number of distinct values as the first key.

To improve data retrieval, add INCLUDE columns to unique indexes. Good candidates are columns that:

- Are accessed frequently and therefore would benefit from index-only access
- Are not required to limit the range of index scans
- Do not affect the ordering or uniqueness of the index key

The creation of indexes should be done by a DBA. The DBA can monitor the system to see if an index is necessary. DBA also has to decide where to put the index (when using DMS) and what buffer pool should be used. A good teaming between developer and DBA is the best way to find a solution with a good performance.

**Physical database design**

After specifying the logical database design the DBA has to think about the placement of the data. This includes understanding the files that will be created to support and manage the database, understanding how much space will be required to store the data, and determining how to use the table spaces that are required to store your data.

The data placement is also one of the performance considerations. Just think about a database that stores all the data on one physical disk. You can understand that there will be no chance to allow parallel processing and I/O will become the bottleneck in the system.

**Buffer pool**

The buffer pool is the memory area where DB2 UDB caches the data that are requested, updated or inserted by client applications. The reason for buffer pools is better performance, because the access to memory is much faster than to a disk. Therefore it is good for performance if you can keep as many as you can the data needed by the client applications in the memory. If the data requested by a client is not in the buffer pool it will be retrieved from disk into the buffer pool. If the buffer pool has no more space left, the data currently retrieved from disk will push some data out of the buffer pool.

Buffer pools are assigned to table spaces with a single buffer pool supporting one or more table spaces. This flexibility allows the database designer to group data table spaces and indexes table spaces into separate buffer pools (only with DMS possible, not with SMS), assign small lookup tables to a small buffer pool to
keep the pages memory resident, maintain a separate buffer pool for the system catalog tables, etc. These techniques can minimize disk I/O by allowing better reuse of data from the buffer pool(s), which provides another mechanism to improve overall DB2 system performance.

Since the space for a buffer pool is allocated at database startup at the full amount of memory and pages cannot be shared between buffer pools, it is usually a good idea to start with a small amount of buffer pools. Upon further analysis and monitoring additional buffer pools may be added, but initially you should keep the number from one to three.

If the database server is “small,” a single buffer pool is usually the most efficient choice. Large systems with applications utilizing dynamic SQL that may do table scans intermixed with index access usually benefit from multiple buffer pools by separating the indexes and data for frequently accessed tables. This configuration prevents a table scan from “flushing” the index pages from the buffer pool to make room for additional data pages. Since most table accesses are via an index, this will allow the index data to remain in the buffer pool (since it is sharing space with other index pages only, not data pages).

Fewer shared buffer pools allow for some margin of error in sizing, when multiple table spaces are sharing a buffer pool. If one table space is less used than initially thought, and another is more used, the overall buffer pool impact is easier to balance. Single table space buffer pools could waste server memory with little or no performance benefit to the database, if that table space is not frequently read by the application.

**Buffer pool data page management**

Pages in the buffer pool are either in use, dirty, or clean:

- In-use pages are currently being read or updated. They can be read, but not updated, by other agents.
- Dirty pages contain data that has been changed but has not yet been written to disk.
- After a changed page is written to disk, it is clean but remains in the buffer pool until its space is needed for new pages. Clean pages can also be migrated to an associated extended storage cache, if one is defined.

When the percentage of space occupied by changed pages in the buffer pool exceeds the value specified by the `CHNGPGS_THRESH` configuration parameter, page-cleaner agents begin to write clean buffer pages to disk.

**Page-cleaner agents**

Page-cleaner agents perform I/O as background processes and allow applications to run faster because their agents can perform actual transaction
work. Page-cleaner agents are sometimes referred to as asynchronous page cleaners or asynchronous buffer writers because they are not coordinated with the work of other agents and work only when required.

To improve performance in update-intensive workloads, configure more page-cleaner agents. Performance improves if more page-cleaner agents are available to write dirty pages to disk. This is also true when there are many data-page or index-page writes in relation to the number of asynchronous data-page or index-page writes.

**Table spaces**

For the physical design you have to decide whether to use DMS or SMS table spaces. The arguments for SMS are the ease of administration and that only the space needed becomes allocated. The administration is easier because you only need to tell DB2 UDB in which directory to store the data. The only time you need to worry about space for a SMS table space is when the device is full. Especially if the database temporarily needs a high amount of disk space, it makes sense to use SMS because the space is released after completion. With DMS table spaces the space is preallocated and will not increase automatically.

From the performance point of view we recommend DMS table spaces using raw devices. With this option DB2 UDB will handle the storage itself without the operating system and is able to store table data in contiguous pages on the disk. The operating system cannot assure this and the data could be fragmented, resulting in worse performance during read and write operations.

With DMS table spaces it is possible to separate the index data and long data from the regular table data. Using this feature can result in better performance. If the index data and the regular data get stored in separated table spaces it is, for example, possible to assign a separate buffer pool for the index data to have the index resident in memory.

By default DB2 uses three SMS table spaces. One is to store the system catalog, one is for temporarily needed space (during a sort, for example), and one is for the user data. The output of DB2 LIST TABLESPACES is shown in Example 7-7.

### Example 7-7  Default table spaces

<table>
<thead>
<tr>
<th>Tablespace for Current Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tablespace ID</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>Detailed explanation:</td>
</tr>
</tbody>
</table>

260  DB2 UDB V8 and WebSphere V5 Performance Tuning and Operations Guide
Two key factors must be considered when designing SMS table spaces:

- **Containers for the table space:** When specifying the number of containers for the table space, it is very important to identify all the containers required up front, since containers cannot be added or deleted after the SMS table space has been created. Each container used for an SMS table space identifies an absolute or relative directory name. Each of these directories can be located on a different file system (or physical disk).

- **Extent size for the table space:** The extent size can only be specified when the table space is created. Because it cannot be changed later, it is important to select an appropriate value for the extent size, as described in “Tablespaces and container” on page 61.

To calculate the optimal size for the table space extent, you should calculate the row length of the tables within the table space and determine how many rows will fit on each data page (remember there is a maximum of 255 rows per page) and how rows will fill up the selected extent size (number of rows per page x extent size). Then the update/insert activity of the tables must be estimated to determine how often an extent will be filled. For example, if you assume the default extent size of 32 pages, and a particular table row length allows one hundred (100) rows per page, it would take 3200 rows of data to fill an extent. If this particular table had a transaction rate of 6000 rows per hour, it would take roughly one half-hour to fill the extent. Therefore, the default extent size of 32 would be adequate for this table. Generally, the default extent size of 32 has been shown to be the optimal extent size through numerous DB2 UDB benchmark tests. The only time that the extent size should be set differently is if the extent size is too small and will fill up every several minutes. This would cause the database manager to allocate an additional extent every few minutes, and since there is some overhead involved in the extent allocation process, this could impact overall database performance.
When choosing an extent size, if it is difficult to determine which of two extent sizes is best (for example, 16 and 32), it is better to choose the larger extent size. If an extent size is too large, the worst scenario is that part of the last extent for the table is unused (that is, the table is very small, just 40 rows of 50 bytes each). If an extent size is too small, the database manager will have to allocate new extents frequently, which could cause a performance impact.

Using DMS table spaces makes it necessary to think about sizing because you have to pre-allocate the space for DMS table spaces. From the Control Center you can access, for example, a table and right click it. From the pop-up menu chose **ESTIMATE SIZE...** to estimate the size of the table and the indexes by entering the expected row count. That helps you to decide how much space is needed. It is also helpful for choosing the right page size.

By default, the page size is 4 KB. Each page (regardless of page size) contains 68 bytes of overhead for the database manager. This leaves 4028 bytes to hold user data (or rows), although no row on a 4-KB page can exceed 4005 bytes in length. A row will not span multiple pages. You can have a maximum of 500 columns when using a 4-KB page size.

If you have rows that need more space, DB2 UDB offers the ability to create table spaces with page sizes of 8 KB, 16 KB or 32 KB. The maximum row sizes for these page sizes are:

- 8 KB: Up to 8101 bytes
- 16 KB: Up to 26293 bytes
- 32 KB: Up to 32677 bytes

A buffer pool assigned to a table space must have the same page size. During installation a small buffer pool of each page size is created that cannot be seen and is not big enough for supporting a table space. It is possible to use the same buffer pool for each table space, but when you have tables that are more frequently accessed than others it may be useful to use a separate buffer pool to have much of the data resident in the memory. Or if you have a large table that is not as frequently accessed you can assign a small buffer pool to the table space where the table is located because it is no problem when this data is not present in the memory.

If you have only one large table in a tablespace it might be useful to use a bigger page size for the tablespace because the bigger the page size the more data a table can contain. When using small and large tables within one tablespace use the smallest page size fitting your large tables. For small tables use a 4-KB page size to not waste memory, as described in the following note.
Refer to Chapter 5, “Physical Database Design”, in the manual *IBM DB2 Universal Database Administration Guide: Planning*, SC09-4822, for a complete description.

As you can see from Example 7-7 on page 260, DB2 UDB creates, by default, a system temporary table space. This table space is used to store internal temporary data required during SQL operations such as sorting, reorganizing tables, creating indexes, and joining tables. Although you can create any number of system temporary table spaces, it is recommended that you create only one, using the page size that the majority of your tables use.

User temporary table spaces are used to store declared global temporary tables that store application temporary data. User temporary table spaces are not created by default at database creation time.

Separating an index from the data makes sense if you have a table with many columns that is frequently accessed and has many rows. In this case the table space for the indexes can have a separate buffer pool to make the most of the index pages resident in memory. Think about the feature of putting additional columns on the index to avoid accessing the table.

Figure 7-6 shows an example of how table space can be arranged.
Table spaces are a logical view while a container is a physical storage device. Each table space has at least one container and a container can belong only to one table space. When using more than one container per table space, DB2 UDB uses a striping mechanism to distribute the data across the containers.

If you have several disks then create the containers for a tablespace on different disks to allow parallel processing. We recommend using striped devices like RAID systems, because a RAID system is already striped. It is recommended to use only one container per tablespace. If you use a RAID system you have to set the registration variable DB_PARALLEL_IO=* to allow parallel data access from all tablespaces. The size of an extent should be a multiple of the RAID stripe size. The prefetch size should be a multiple of the extent size and a multiple of the RAID stripe size multiplied by the count of devices.
For the physical design it is also important to think about the location of the log files. Because any change to the data is written to the log files, the number of I/Os on these files is high. It is a good practice to have a different disk on which to put the logs. If you have to use any disk for your data, use the disk with the lowest I/O for the log files.

The path to the log files is stored in the database configuration. Use the command DB2 GET DB CFG and look for PATH TO LOGFILE. To change the path, update the **NEWLOGPATH** parameter in the database configuration. The new setting does not become the value of logpath until both of the following occur:

- The database is in a consistent state, as indicated by the **DATABASE_CONSISTENT** parameter from the output of DB2 GET DB CFG.
- All users are disconnected from the database.

When the first new connection is made to the database, the database manager will move the logs to the new location specified by logpath.

A set of system catalog tables is created and maintained for each database. These tables contain information about the definitions of the database objects (for example, tables, views, indexes, and packages), and security information about the type of access that users have to these objects. These tables are stored in the **SYSCATSPACE** table space.

**Note:** The catalog tables are flagrantly accessed and it is recommended to use a separate disk or a disk with less activity for the SYSCATSPACE.

### 7.2.2 SQL tuning

The next step is implementing the application. SQL is the language to speak with relational databases like DB2 UDB. It is used to select, insert and update the data. DB2 UDB uses an SQL compiler to produce an access plan. Figure 7-7 shows the steps for the SQL compiler.

Like every compiler the SQL compiler checks the syntax and the semantic first. If one of the checks fails an error will be returned to the application. If these checks are passed the compiler tries to rewrite the statement if necessary. The compiler uses the global semantics stored in the query graph model to transform the query into a form that can be optimized more easily and stores the result in the query graph model.

The next step is to optimize the query. How intensive the optimizations will be is configurable. More details are in “Explain utilities” on page 250.
With DB2 UDB you have the ability to use static SQL or dynamic SQL. The difference is that static SQL uses a precalculated access plan. This allows faster access because there is no need to optimize the query every time. So if you look at Figure 7-7 you see a process that needs to be done every time a dynamic SQL statement requests data from the server. For static SQL statements this step is done only once at creation time. But you need to rebind the code every time the...
conditions change and the optimizer might find another access plan. Do a rebind after:

- The package is marked as invalid.
- An index was created or removed.
- The statistics of the tables used in the statement became updated.
- A fix pack was installed.

**Efficient SELECT statements**

Also, the optimizer is able to optimize the select statement. You have to take care of the following points that the optimizer could not do for you:

- Specify only the columns needed. Unnecessary columns will waste CPU cycles and slow down the network traffic.
- Avoid selecting more rows than needed by specifying all applicable predicates in the query, to reduce the answer set to a minimum. Do not discard rows in your application code, let DB2 do it for you.
- When the number of rows you need is significantly less than the total number of rows that might be returned, specify the OPTIMIZE FOR clause.
- Use the FOR READ ONLY or FOR FETCH ONLY clause to avoid exclusive lock and to take advantage of row blocking.
- Use the FOR UPDATE OF clause for update cursors to prevent deadlocks.
- Avoid numeric data type conversion whenever possible.
- Use operations like DISTINCT and ORDER BY only if needed. Sorting is very CPU intensive. If the order of the result set is insignificant, do not order it.
- Use COUNT(*) FROM only when needed.
- Use WHERE COL1 IN (1,2,5) instead of WHERE COL1 = 1 OR COL1 = 2 OR COL1 = 5.
- Do not use user-defined functions in join conditions. If you need a function in a join condition then create a view with an additional field including the UDF and join the view with the other table.

It is a good practice to use the same code page on both the client and the server site because the conversion of data types slows down the application. Character conversion occurs in the following circumstances:

- When a client or application runs in a code page that is different from the code page of the database that it accesses.

  The conversion occurs on the database server machine that receives the data. If the database server receives the data, character conversion is from the application code page to the database code page. If the application
machine receives the data, conversion is from the database code page to the application code page.

- When a client or application that imports or loads a file runs in a code page different from the file being imported or loaded.

### 7.2.3 Stored procedures

Stored Procedures (SPs) are programs running in the database environment. SPs can be written in SQL, C and Java and help to reduce the network traffic. In any case, procedures are typically written to contain multiple SQL data manipulation language (DML) statements as well as procedure logic constructs such as loops and if/else statements. Therefore, stored procedures are conceptually similar to “small” applications, providing useful, database-intensive business service to multiple applications. These applications, which typically are remote from DBMS itself, invoke the procedure with a single call statement.

Stored procedures can require parameters for input (IN), output (OUT), or input and output (INOUT). They may return one or more sets of results.

Keep in mind that calling a SP has an overhead and it should be clear that it makes no sense to have a single statement inside the SP. The benefit arises if the network traffic reduces significantly. This can be the case if the SP includes several SQL statements, especially when the result set of the SP is smaller than the result set that the SP itself has to handle, or if the SP has loops including SQL statements.

**FENCED vs. NOT FENCED - Security vs. performance**

A stored procedure can run in two modes, FENCED and NOT FENCED. A SP running in a fenced mode uses a different address space than the database server, while a not fenced SP runs in the same process as the database manager. We recommend using FENCED SPs only because an error in a NOT FENCED SP can accidentally or maliciously corrupt the database or damage the database control structure.

The benefit of a NOT FENCED SP is the better performance. So for performance reasons it might be useful to use a SP NOT FENCED, but again it is a high risk. Use the SP as a FENCED one first and later, if you know that the SP runs stably and has no errors, create the SP in a NOT FENCED mode.

**Java routines**

For Java routines running on UNIX platforms, scalability may be an issue if NOT FENCED is specified. This is due to the nature of the DB2 UNIX process model, which is one process per agent. As a result, each invocation of a NOT FENCED Java routine will require its own JVM. This can result in poor scalability, because
JVMs have a large memory footprint. Many invocations of NOT FENCED routines on a UNIX-based DB2 server will use a significant portion of system memory.

This is not the case for Java routines running on Windows NT and Windows 2000, where each DB2 agent is represented by a thread in a process shared with other DB2 agent threads. This model is scalable, as a single JVM is shared among all the DB2 agent threads in the process.

If you intend to run a Java routine with large memory requirements, it is recommended that you register it as FENCED NOT THREADSAFE. For FENCED THREADSAFE Java routine invocations, DB2 attempts to choose a threaded Java fenced mode process with a Java heap that is large enough to run the routine. Failure to isolate large heap consumers in their own process may result in out of Java heap errors in multi-threaded Java db2fmp processes. If your Java routine does not fall into this category, FENCED routines will run better in thread safe mode where they can share a small number of JVMs.

**C/C++ routines**

C or C++ routines are generally faster than Java routines, but are more prone to errors, memory corruption, and crashing. For these reasons, the ability to perform memory operations makes C or C++ routines risky candidates for THREADSAFE or NOT FENCED mode registration. These risks can be mitigated by adhering to programming practices for secure routines (see the topic “Security Considerations for Routines” in the DB2 Information Center), and thoroughly testing your routine.

**SQL-bodied routines**

SQL-bodied routines are also generally faster than Java routines, and usually share comparable performance with C routines. SQL routines always run in NOT FENCED mode, providing a further performance benefit over external routines.

**Note:** For creating SQL Stored Procedures you need a C-Compiler. For further details have a look at *Cross-Platform DB2 Procedures: Building and Debugging*.

### 7.2.4 Declared global temporary tables

The DECLARE GLOBAL TEMPORARY TABLE statement defines a temporary table for the current session. The benefit of temporary tables is that their description does not appear in the system catalog and that any action on them is not logged. A temporary table consists only as long as the session exists that created it. So if the session terminates the temporary is gone.
For some circumstances temporary tables are very helpful and in some case they can become very big. With DB2 UDB Version 8 it is now possible to optimize the access to the data in temporary tables because it is now possible to create indexes and update the statistics on them. With these options the temporary tables are now much more powerful than in former versions.

So think about a very large temporary table in your system. Every time you need some data from that table a tablescan on this temporary table is necessary. With an index and with statistical data it is now possible to have the same performance as with regular tables.

7.2.5 Concurrency

When designing applications it is important to think about concurrency. There is a need to balance between the requirement of the application for consistency when accessing data and the impact of performance when the isolation level is set too high.

Generally, in a production environment, DBA is the one that binds the system utilities like CLI, etc. Therefore, DBA will set the default isolation level for applications that use those utilities. In order to set a proper default isolation level, DBA should understand the applications running on the server. A good communication between DBA and the application designer/programmer is necessary. More important is that developer should always explicitly set the isolation best suit for the application.

Isolation level

DB2 UDB offers four isolation levels: Repeatable read (RR), read stability (RS), cursor stability (CS) and uncommitted read (UR). Each of these isolation levels allows the user and application to control the number and duration of read (Share) locks held within a unit of work. By setting the appropriate isolation level based on a particular application’s requirement, lock resources can be minimized and user concurrency (application’s tendency to multiple users) can be increased as much as possible for a particular application. Each of the isolation levels and their definitions, along with a brief example, is shown in the Table 7-2 on page 271 (highest to lowest locking impact). The example is based on the following information:

- A table (EMPLOYEES) that contains a column LAST_NAME with the following values:

  ARNOLD, BLACK, BROWN, BRUEGGEMAN, BRUTY, HANCOCK, HURT, JAYNE, JONES, LAW, MORGAN, NOBLES, NOVACEK, OATES, STAUSKAS, TOBIAS

  The table has no index on LAST_NAME.
The SQL query being executed is:

```sql
SELECT LAST_NAME, EMPLOYEE_NUMBER
FROM A.EMPLOYEES
WHERE LAST_NAME LIKE 'B%'
```

<table>
<thead>
<tr>
<th>Isolation level</th>
<th>Brief description</th>
<th>Example that illustrates impact on Share (read) locking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeated Read (RR)</td>
<td>Repeated Read means that the same query can be executed multiple times within the same unit-of-work and the results of the query will be identical every time (repeatable). A Share lock will be held on each row and examined during the execution of a SQL statement (for table scans, this would encompass each row within the table).</td>
<td>A Share lock would be held on every row within the table for the entire unit-of-work, effectively preventing updates (X locks) from occurring until this SQL statement is committed.</td>
</tr>
<tr>
<td>Read Stability (RS)</td>
<td>Read Stability means that the same query will return the same base set of information each time the query is executed in a unit-of-work, but additional rows may be included with each subsequent execution. A Share lock will be held on every row that satisfies the result set of the query (that is, is part of the answer),</td>
<td>A Share lock would be held on the rows containing BLACK, BROWN, BRUEGGEMAN and BRUTY, but all other rows could be available for update and read.</td>
</tr>
<tr>
<td>Cursor Stability (CS)</td>
<td>Cursor Stability means that the data is repeatable on the current position of the cursor only. Previously read rows or to-be-read rows will allow updates and/or reads. A Share lock will be held on the current position of the cursor only.</td>
<td>A Share lock would be held on the rows containing BLACH, BROWN, BRUEGGEMAN and BRUTY, but only when the cursor was positioned on that row. After the cursor is repositioned, the row is eligible for read and/or update by another user.</td>
</tr>
<tr>
<td>Uncommitted Read (UR)</td>
<td>Uncommitted Read means that no Share locks are held on any row by this query, and updates can take place as well. Only super exclusive (Z) locks prevent UR queries. No Shared locks are held at all.</td>
<td>No locks would be held on any rows of the table and the data could be in the process of being changed as it is read.</td>
</tr>
</tbody>
</table>

In looking at the isolation levels and their descriptions, it is easy to see that using an unnecessary isolation level (too high a locking impact) will impact performance of a database environment that supports both database read and write transactions. A good rule to go by in selecting the isolation level is to use the lowest locking level that will support the application requirements. For some
reporting applications, if the answer set does not have to be exact (but rather an approximation), UR might be the proper isolation level. There are actually very few instances where the RR isolation level is required, and the application requirements should be verified to confirm that RR isolation is necessary (due to the potential for lock waits and lock escalation).

**Locking options**

DB2 UDB offers options in the area of locking to reduce DB2 UDB resource consumption for the lock list memory area and reduce the chance of lock escalation for the entire database or an individual database agent. The *LOCKSIZE* of the table can be changed from the default of row level locking to table level locking via the ALTER TABLE statement. This will allow read-only tables to hold less DB2 UDB lock resources (as Share locks behave the same at the table or row level) due to the fact that one lock (at the table level) is all that will be required on the table. Depending on the isolation level setting, this can reduce the memory consumption within the lock list considerably.

Another locking option to perform essentially the same function of reducing the memory consumption within the lock list focuses on the application level by providing the LOCK TABLE statement. This statement can be issued by an application process or user to lock a table in either Share or Exclusive (X) mode. The use of this option allows locks to be reduced for a single application task or process, and will prevent other applications or tasks from accessing the table at all (through the use of Exclusive (X) mode).

Careful consideration should be given to the database usage requirements and application design before either of these locking options should be utilized. Since the lock granularity is at the table level, this type of locking strategy can have a major impact on overall database performance, if it is utilized in an inappropriate situation.

### 7.3 System tuning

Although the performance considerations have been put in place as much as we can during the application development, once the whole system is up and running we need to monitor and adjust the system or application to ensure the optimal performance is reached.

In “DB2 architecture overview” on page 49 you can find the architecture to understand how DB2 UDB is designed. In this section we show you how to tune a running DB2 UDB system. We go through the key performance areas and describe what to look for and how to set parameters to fit the requirements for your system.
7.3.1 Tuning the buffer pools

When creating a buffer pool (DB2 CREATE BUFFERPOOL) you have to specify the size of a buffer pool. The size specifies the amount of pages to use. When using -1, the DB parameter BUFFPAGE is used to specify the size of the buffer pool. To change the size use DB2 ALTER BUFFERPOOL, when the size of the buffer pool is -1 then change the BUFFPAGE.

**Note:** The DB parameter BUFFPAGE controls the size of any buffer pool with the size set to -1. If you want different sizes you have to set the size using the command CREATE BUFFERPOOL or ALTER BUFFERPOOL.

In prior DB2 UDB versions it was necessary to increase the DBHEAP parameter when using more space for the buffer pool. With version 8 nearly all buffer pool memory, including page descriptors, buffer pool descriptors, and the hash tables, comes out of the database shared-memory set and is sized automatically.

To determine how well your buffer pool is designed, have a look using the snapshot monitor. The monitor switch for BUFFERPOOLS must be set to **ON**. Check the status with command the DB2 GET MONITOR SWITCHES. If the switch for BUFFERPOOL is **OFF**, use the following command to turn it on:

```
DB2 UPDATE MONITOR SWITCHES USING BUFFERPOOL ON
```

Or use the following to set the DBM parameter **DFT_MON_BUFPOOL** to **ON**.

```
DB2 UPDATE DBM CFG USING DFT_MON_BUFPOOL ON
```

When the system is running for a while with the monitor switch for buffer pools set to on, then use DB2 GET SNAPSHOT FOR ALL BUFFERPOOLS to see the current status of the buffer pools. In Example 7-8 we show output of a snapshot for buffer pools. As you can see, the systems have the standard buffer pool with the standard size (for UNIX systems) of 1000 4-K pages.

**Example 7-8  Sample output for a snapshot of a buffer pool**

```
$ db2 get snapshot for all bufferpools

Bufferpool Snapshot

  Bufferpool name                      = IBMDEFAULTBP
  Database name                        = TRADE3DB
  Database path                        = /home/db2inst1/db2inst1/NODE0000/SQL00002/
  Input database alias                 =
  Snapshot timestamp                   = 11-17-2003 16:30:46.883397

  Buffer pool data logical reads       = 12823
  Buffer pool data physical reads      = 279
  Buffer pool data writes              = 0
```
Buffer pool index logical reads = 222
Buffer pool index physical reads = 68
Total buffer pool read time (ms) = 775
Total buffer pool write time (ms) = 0
Asynchronous pool data page reads = 88
Asynchronous pool data page writes = 0
Buffer pool index writes = 0
Asynchronous pool index page reads = 0
Asynchronous pool index page writes = 0
Total elapsed asynchronous read time = 22
Total elapsed asynchronous write time = 0
Asynchronous data read requests = 5
Asynchronous index read requests = 0
Direct reads = 17210
Direct writes = 0
Direct read requests = 7545
Direct write requests = 0
Direct reads elapsed time (ms) = 3564
Direct write elapsed time (ms) = 0
Database files closed = 0
Data pages copied to extended storage = 0
Index pages copied to extended storage = 0
Data pages copied from extended storage = 0
Index pages copied from extended storage = 0
Unread prefetch pages = 0
Vectored I/Os = 5
Pages from vectored I/Os = 88
Block I/Os = 0
Pages from block I/Os = 0
Physical page maps = 0
Node number = 0
Tablespaces using bufferpool = 5
Alter bufferpool information:
  Pages left to remove = 0
  Current size = 1000
  Post-alter size = 1000

From the output of DB2 GET SNAPSHOT FOR ALL BUFFERPOOLS you can determine how good the hit ratio of your buffer pools is. The more logical reads and the less physical reads the buffer pool has, the better the ratio will be. To determine the ratio for the buffer pool and the index pool use the following formulas:

\[
\text{BufferPoolHitRatio} = \frac{\Sigma \text{LogicalReads} - \Sigma \text{PhysicalReads}}{\Sigma \text{LogicalReads}} \times 100
\]
In our example the overall buffer pool ratio is 97.82, and this means that the buffer pool is big enough. The hit ratio for indexes is below 70. The small value of the indexes hit ratio show that the indexes are in less use. This could mean that the system is in the point of diminishing returns for a over sized buffer pool. With this ration we could try to reduce the buffer pool size to set memory free to improve the system performance.

In general, a good value for the hit ratio is a value more than 90 percent for both buffer pool hit ratio and index pool hit ratio. When the value is smaller, then increase the buffer pool size and continue monitoring. But be sure not to allocate so much memory that the operating systems starts paging. In AIX use `lsp -s` or `topas` to monitor the memory usage and the paging activities.

To monitor the memory of the database use the DB2 Memory Visualizer.

### 7.3.2 Table management

The optimizer uses the table statistics to calculate the best access plan. The statistics will become out-of-date invalid because of insert, update, and delete operations performed on tables. It is necessary to update the statistics from time to time using the `runstats` command. After `runstats` is performed on a table, it is necessary to rebind the applications that access the table(s). When an application with static SQL is bound to the database the access plan gets calculated using the current statistics. This access plan is updated as the application is rebound due to the statistics change, or an index is created.

**Table reorganization**

If your SQL codes slows down and this performance problem does not improved after running `runstats` and rebinding the applications, the SQL code performance might be helped by reorganizing the tables. When inserting new data into tables, it is often not possible to place them in a physical sequence that is the same as the logical sequence defined by the index (unless you use clustered indexes). When accessing the data, many more operations are necessary than with having the data on sequential pages.

**Note:** With DB2 UDB Version 8, reorg becomes an online feature. The reorg can be paused and restarted. The in-place reorg operation reorganizes only a small portion of data and therefore only a small amount of additional space is necessary and only a few pages are locked for updates at a time.
For more details on how to determine if an REORG is necessary, search DB2 Information Center or DB2 Online Help for the REORGCHK command at:

http://publib.boulder.ibm.com/infocenter/db2help/index.jsp

After table is reorganized, you should run the update statistic by running the runstats, then re-bind the static applications. To reduce the need for reorganizing a table, perform these tasks after you create the table:

1. Alter table to add PCTFREE.
2. Create clustering index with PCTFREE on index.
3. Sort the data.
4. Load the data.

After you have performed these tasks, the table with its clustering index and the setting of PCTFREE on the table helps preserve the original sorted order. If enough space is allowed in table pages, new data will be inserted into the pages to maintain the clustering characteristics of the index. As more data are inserted and the pages of the table become full, records are appended to the end of the table so that the table gradually becomes un-clustered.

### 7.3.3 Index management

The most important thing for performance in the database design is index. When designing the index on tables, you need to understand how the tables will be accessed by the application. First determine what queries will be used on the database and then analyze the statements by yourself or use the Design Advisor provided by DB2 UDB.

Once the data are loaded into the tables, run the runstats utility to update the statistics on the tables so the optimizer can utilize the indexes. For frequently updated tables, you should run the runstats regularly to update the table statistics to ensure that the best access plan will be used in your queries.

If you notice that there are long-running queries in an existing system, you should use the explain tool to analyze the access path the optimizer created. If the table scan is used, it could be an indication that an index is needed. But in some situations it is better to use a table scan than an index if the table has only a few rows or the query needs most of the rows as a result set. Therefore, we recommend using the Design Advisor because you can virtually test if a new index will help to achieve a better performance.

Be careful that if you see that the access plan uses a table scan and you have already had an index that fits, then it may be that the statistics of the table are not up to date. Use the runstats utility to update the statistics and let DB2 UDB
create a new access plan. If you are testing with static SQL, do not forget to rebind your application to generate a new access plan.

Each index entry contains a search-key value and a pointer to the row containing that value. If you specify the ALLOW REVERSE SCANS parameter in the CREATE INDEX statement, the values can be searched in both ascending and descending order. It is therefore possible to bracket the search, given the right predicate. An index can also be used to obtain rows in an ordered sequence, eliminating the need for the database manager to sort the rows after they are read from the table.

Consider the following suggestions for using and managing indexes:

- Specify parallelism at index creation.
  When you create indexes on large tables hosted by an SMP machine, consider setting the DBM configuration parameter INTRA_PARALLEL to YES (1) or SYSTEM (-1) to take advantage of parallel processing for performance improvements. Multiple processors can be used to scan and sort data.

- Specify separate table spaces for indexes.
  Indexes can be stored in a different table space from the table data. This can allow for more efficient use of disk storage by reducing the movement of read/write heads during index access. You can also create index table spaces on faster physical devices. In addition, you can assign the index table space to a different buffer pool, which might keep the index pages in the buffer longer because they do not compete with table data pages.

- Ensure the degree of clustering.
  If your SQL statement requires ordering, such as ORDER BY, GROUP BY, and DISTINCT, even though an index might satisfy the ordering, the optimizer might not choose the index if clustering is poor or the table is so small that it is cheaper to scan the table and sort the answer set in memory.

  After you create a clustering index, perform a REORG TABLE in classic mode, which creates a perfectly organized index. In general, a table can only be clustered on one index.

- Use volatile tables for tables that vary widely in size.
  A volatile table is a table that might vary in size at run time from empty to very large. For this kind of table, in which the cardinality varies greatly, the optimizer might generate an access plan that favors a table scan instead of an index scan.

  Declaring a table “volatile” using the ALTER TABLE...VOLATILE statement allows the optimizer to use an index scan on the volatile table.
Agents

There are some database manager configuration parameters that influence the number of agents created and the way they are managed.

- **MAXAGENTS**
  The number of agents that can be working at any one time. This value applies to the total number of agents that are working on all applications, including coordinator agents, subagents, inactive agents, and idle agents.

- **NUM_POOLAGENTS**
  The total number of agents, including active agents and agents in the agent pool, that are kept available in the system. The default value for this parameter is half the number specified for maxagents.

- **NUM_INITAGENTS**
  When the database manager is started, a pool of worker agents is created based on this value. This speeds up performance for initial queries. The worker agents all begin as idle agents.

- **MAX_CONNECTIONS**
  Specifies the maximum number of connections allowed to the database manager system on each partition.

- **MAX_COORDAGENTS**
  For partitioned database environments and environments with intra-partition parallelism enabled when the connection coordinator is enabled. This value limits the number of coordinating agents.

- **MAXCAGENTS**
  This value controls the number of tokens permitted by the database manager. For each database transaction (unit of work) that occurs when a client is connected to a database, a coordinating agent must obtain permission to process the transaction from the database manager. This permission is called a processing token. The database manager permits only agents that have a processing token to execute a unit of work against a database. If a token is not available, the agent must wait until one is available to process the transaction.

---

**Note:** In DB2 UDB V8.1 and later, all new indexes are created as type-2 indexes. The one exception is when you add an index on a table that already has type-1 indexes. In this case only, the new index will also be a type-1 index. To find out what type of index exists for a table, execute the INSPECT command. To convert type-1 indexes to type-2 indexes, execute the REORG INDEXES command. To see the advantages of the new index, look at Chapter 8, “Operational performance,” in *DB2 UDB Administration Guide: Performance* - Document Number, SC09-4821.
This parameter can be useful in an environment in which peak usage requirements exceed system resources for memory, CPU, and disk. For example, in such an environment, paging might cause performance degradation for peak load periods. You can use this parameter to control the load and avoid performance degradation, although it can affect either concurrency or wait time, or both.

The snapshot monitor provides some information about the activity of the agents. When taking a snapshot of the database manager and `grep` for agent you will see information like that shown in Example 7-9.

**Example 7-9  Agent information from the snapshot monitor**

```
db2 get snapshot for dbm  | grep -i agent

High water mark for agents registered = 16
High water mark for agents waiting for a token = 0
Agents registered = 16
Agents waiting for a token = 0
Idle agents = 10
Agents assigned from pool = 92
Agents created from empty pool = 18
Agents stolen from another application = 0
High water mark for coordinating agents = 16
Max agents overflow = 0
Gateway connection pool agents stolen = 0
```

### 7.3.4 Prefetcher

How the prefetcher works is described in “Key performance-related areas” on page 58. Prefetching can be configured using the `NUM_IOSERVERS` database parameter. Use as many prefetchers as agents allow DB2 to perform the prefetching requests in parallel.

Use the snapshot monitor for buffer pools and calculate the difference of *Buffer pool data physical reads* and *Asyncornous pool data page reads* to see how well the prefetchers are working. It is better to define too many than too little prefetchers. If you specify extra I/O servers, these servers are not used, and performance does not suffer. Each I/O server process is numbered. The database manager always uses the lowest numbered process, so some of the upper numbered processes might never be used.

Configuring enough I/O servers with the `NUM_IOSERVERS` configuration parameter can greatly enhance the performance of queries for which prefetching of data can be used. To maximize the opportunity for parallel I/O, set `NUM_IOSERVERS` to at least the number of physical disks in the database.
7.3.5 Cleaner

To improve performance in update-intensive workloads, configure more page-cleaner agents. Performance improves if more page-cleaner agents are available to write dirty pages to disk. This is also true when there are many data-page or index-page writes in relation to the number of asynchronous data-page or index-page writes.

Consider the following factors when setting the value for DB parameter NUM_IOCLEANERS:

- Application type
  - If it is a query-only database that will not have updates, set this parameter to be zero (0). The exception would be if the query workload results in many TEMP tables being created (you can determine this by using the explain utility).
  - If transactions are run against the database, set this parameter to be between one and the number of physical storage devices used for the database.

- Workload
  Environments with high update transaction rates may require more page cleaners to be configured.

- Buffer pool sizes
  Environments with large buffer pools may also require more page cleaners to be configured.

You may use the database system monitor to help you tune this configuration parameter using information from the event monitor about write activity from a buffer pool:

- The parameter can be reduced if both of the following conditions are true:
  - pool_data_writes is approximately equal to pool_async_data_writes
  - pool_index_writes is approximately equal to pool_async_index_writes.

- The parameter should be increased if either of the following conditions are true:
  - pool_data_writes is much greater than pool_async_data_writes
  - pool_index_writes is much greater than pool_async_index_writes.

7.3.6 Sort heap

The sort heap is a piece of memory where DB2 UDB stores data during a sort. When the sort heap is big enough the sort can be done in one operation called piped sort. If the sort heap is not big enough to store the whole data for the sort
then a temporary table is created in the buffer pool and sort will be divided in several pieces, which is more time consuming. So try to avoid sorts that do not fit into the sort heap. The application developer has to verify if the sort is really necessary or if he can perform the search query without a order by clause. The DBA should monitor if the sort heap is big enough and if not he needs to increase the heap size when a better performance is necessary and more memory is available.

Be careful when increasing the sort heap. Because the heap is part of the agent memory, for each connection the space is allocated and that multiples the memory required by the number of agents.

To find out if you have a sort performance problem, look at the total CPU time spent on sorting compared to the time spent for the whole application using, for example, the snapshot monitor for database and database manager.

If total sort time is a large portion of CPU time for the application, then look at the following values, which are also shown by default:

- Percentage of overflowed sorts
  This variable (on the performance details view of the Snapshot Monitor) shows the percentage of sorts that overflowed. If the percentage of overflowed sorts is high, increase the `SORTHEAP`. To find out if there were any post threshold sorts, use the Snapshot Monitor.

- Post threshold sorts
  If post threshold sorts are high, it indicates that more sort memory is requested than defined with the `SHEAPTHRES` parameter. So any sort following after this value that is reached receives less sort memory than defined by the `SORTHEAP` parameter. To avoid this situation increase `SHEAPTHRES` and/or decrease `SORTHEAP`.

In general, overall sort memory available across the instance (`SHEAPTHRES`) should be as large as possible without causing excessive paging. Although a sort can be performed entirely in sort memory, this might cause excessive page swapping. In this case, you lose the advantage of a large sort heap. For this reason, you should use an operating system monitor to track changes in system paging whenever you adjust the sorting configuration parameters. Also note that in a piped sort, the sort heap is not freed until the application closes the cursor associated with that sort. A piped sort can continue to use up memory until the cursor is closed.

### 7.3.7 Locking

An indicator for problems is lock time-outs or long respond times reported from the users. The database parameter `LOCKTIMEOUT` specifies how long an
application should wait for a resource that is locked by another application. The value -1 defines an infinite wait time. Any positive values specifies the wait time in seconds. The LOCKTIMEOUT has nothing to do with deadlocks. DB2 recognizes deadlocks using separate process. Setting LOCKTIMEOUT to infinite waiting will cause the application to wait forever, but will not result in a deadlock situation.

**Monitoring locks**

Use the snapshot monitor described in “Snapshot monitor” on page 238 to look for lock time-out and deadlock situations. The locks currently held can be monitored with the command:

`DB2 GET SNAPSHOT FOR LOCKS ON database name`

In Example 7-10 is the first part of the snapshot output of our sample database TRADE3DB. This part shows the current lock situation on the database. You can see that currently three locks are held and that no other application is waiting for any of the locked objects.

**Example 7-10  Snapshot for locks (database part)**

<table>
<thead>
<tr>
<th>Database Lock Snapshot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database name          = TRADE3DB</td>
</tr>
<tr>
<td>Database path          = /home/db2inst1/db2inst1/NODE0000/SQL00002/</td>
</tr>
<tr>
<td>Input database alias   = TRADE3DB</td>
</tr>
<tr>
<td>Locks held             = 3</td>
</tr>
<tr>
<td>Applications currently connected = 3</td>
</tr>
<tr>
<td>Agents currently waiting on locks = 0</td>
</tr>
<tr>
<td>Snapshot timestamp     = 11-24-2003 15:08:46.134255</td>
</tr>
</tbody>
</table>

From the same output, you can see which application is currently locking one or more objects in the database. Example 7-11 is the output shown the application that held the three locks. The status of the application is UOW Waiting. DB2 UDB is waiting for the user to do something. The output of the snapshot shows all connected applications, so you have a chance to see which application is waiting for a lock.

**Example 7-11  Snapshot for locks (application part)**

<table>
<thead>
<tr>
<th>Application handle = 18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application ID         = G9012775.MA0D.017404230648</td>
</tr>
<tr>
<td>Sequence number        = 0018</td>
</tr>
<tr>
<td>Application name       = javaw.exe</td>
</tr>
<tr>
<td>CONNECT Authorization ID = DB2INST1</td>
</tr>
<tr>
<td>Application status     = UOW Waiting</td>
</tr>
<tr>
<td>Status change time     = 11-24-2003 15:08:43.201674</td>
</tr>
<tr>
<td>Application code page  = 1208</td>
</tr>
<tr>
<td>Locks held             = 3</td>
</tr>
<tr>
<td>Total wait time (ms)   = 0</td>
</tr>
</tbody>
</table>
The list of currently held locks follows the list of connected applications as shown in Example 7-12. From the output you can see what object is locked and in what mode.

**Example 7-12  Snapshot for locks (list of locks)**

<table>
<thead>
<tr>
<th>List Of Locks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock Name                     = 0x00000000100000010001940056</td>
</tr>
<tr>
<td>Lock Attributes                = 0x00000000</td>
</tr>
<tr>
<td>Release Flags                  = 0x40000000</td>
</tr>
<tr>
<td>Lock Count                     = 1</td>
</tr>
<tr>
<td>Hold Count                     = 0</td>
</tr>
<tr>
<td>Lock Object Name               = 0</td>
</tr>
<tr>
<td>Object Type                    = Internal V Lock</td>
</tr>
<tr>
<td>Mode                           = S</td>
</tr>
</tbody>
</table>

| Lock Name                     = 0xA6B2A69FA4A17C7D9175505041 |
| Lock Attributes                = 0x00000000 |
| Release Flags                  = 0x40000000 |
| Lock Count                     = 1 |
| Hold Count                     = 0 |
| Lock Object Name               = 0 |
| Object Type                    = Internal P Lock |
| Mode                           = S |

| Lock Name                     = 0x00030008000000000000000054 |
| Lock Attributes                = 0x00000000 |
| Release Flags                  = 0x00000001 |
| Lock Count                     = 1 |
| Hold Count                     = 1 |
| Lock Object Name               = 8 |
| Object Type                    = Table |
| Tablespace Name                = TRADEDAT |
| Table Schema                   = DB2INST1 |
| Table Name                     = ACCOUNTEJB |
| Mode                           = IN |

If there are a lot of locks held or if DB2 UDB finds a lot of deadlock situations, it indicates that something with the application is wrong. From the database side, you can check if there is a long running transaction that causes the problem. Try to tune the system to make this transaction faster. On the application side check if the isolation level is needed and try to have short transaction or to commit as often as possible to reduce locking. When changing the isolation level, both DBA and application developer should work together to experiment with different isolation levels.

**Lock escalation**

Some error situations occur because of lock escalation used by DB2 UDB to reduce locks. There are two parameters in the database configuration responsible for lock escalation. With LOCKLIST you define the amount of
memory used for the locks. \textit{MAXLOCKS} parameter indicates the percentage an application can hold of the available locks. If an application requests more locks than specified by this parameter then lock escalation occurs.

The database manager determines which locks to escalate by looking through the lock list for the application and finding the table with the most row locks. If after replacing these with a single table lock, the MAXLOCKS value is no longer exceeded, lock escalation will stop. If not, it will continue until the percentage of the lock list held is below the value of MAXLOCKS. The MAXLOCKS parameter multiplied by the MAXAPPLS parameter cannot be less than 100.

As a recommendation set \textit{MAXLOCKS} to \(2 \times \frac{100}{\text{MAXAPPLS}}\).

Lock escalation can result in deadlock and time-out situations. Look into the \textit{DB2DIAG.LOG} file and search for lock escalation, especially when deadlocks or time-outs are reported.

### 7.3.8 Logging

The database parameter \textit{LOGBUFSZ} allows you to specify the amount of the database heap (defined by the \textit{DBHEAP} database parameter) to use as a buffer for log records before writing these records to disk. The log records are written to disk when one of the following occurs:

- A transaction commits or a group of transactions commit, as defined by the \textit{MINCOMMIT} configuration parameter.
- The log buffer is full.
- As a result of some other internal database manager event.

This parameter must also be less than or equal to the \textit{DBHEAP} database parameter. Buffering the log records will result in a more efficient logging file I/O because the log records will be written to disk less frequently and more log records will be written at each time.

Increase the size of this buffer area if there is considerable read activity on a dedicated log disk, or there is high disk utilization. When increasing the value of this parameter, you should also consider the \textit{DBHEAP} parameter since the log buffer area uses space controlled by the \textit{DBHEAP} parameter.

You may use the snapshot monitor for the application to determine how much of the log buffer space is used for a particular transaction (or unit of work). Refer to the \textit{unit of work log space used} monitor element.
7.3.9 Tablespace

When using DMS tablespace it is necessary to check if there is enough space available in the tablespaces. Use the snapshot monitor for tablespaces and look for the available and used pages in the containers. With the command `ALTER TABLESPACE name_of_tablespace ADD CONTAINER...` you can add containers to a tablespace.

**Note:** With DB2 UDB V8 it is now possible to drop containers (if there is enough space in the other containers to store the data of the container you want to drop), reduce the size of containers, and add containers without rebalancing the data.
In this chapter, we provide a description of the components that have important performance impact in a WebSphere Application Server and DB2 UDB integrated environment, and offer some best-practices guidelines for their usage.

In Chapter 2, Chapter 3, Chapter 4, and Chapter 6, we provided an overview of WebSphere Application Server and DB2 UDB, and described their key performance components, indicators, best practices, methodology and tools for achieving optimal performance. In this chapter, we focus on those key components specific to the interface between WebSphere Application Server and DB2 that can impact the performance of WebSphere Application Server and DB2 UDB environment.

The topics covered include:

- Connection pool
- Prepared statement cache
- Session database
- Enterprise Java Beans
8.1 WebSphere data sources

In previous chapters, we have discussed WebSphere data sources. In this section, we provide more detailed information on two important WebSphere components in data source, connection pooling and statement cache, which have a critical impact on performance of a WebSphere Application Server and DB2 UDB system.

8.1.1 Connection pooling

Each time an application attempts to access a database, it must first connect to that database, before it can issue queries against it. A database connection incurs overhead, as it requires resources to create the connection, maintain it, and then release it when it is no longer required.

Database connection overhead can be particularly high for Web-based applications for the following reasons:

- Web users connect and disconnect more frequently.
- User interactions are typically shorter, with more effort often spent connecting to the database, and disconnecting from it, than performing the actual user requests.
- Web requests tend to be unpredictable both in terms of volume and frequency, which can place severe demands on database connections.

To address this problem, WebSphere has provided a connection pooling feature based on the JDBC 2.0 Optional Package API specification. For a basic understanding of the JDBC 2.0 Core API and the JDBC 2.0 Optional Package API, refer to:


Connection pooling is a mechanism whereby a system administrator can define a pool of connections for a single data source that may be reused by multiple users, without each one of them incurring the overhead of connecting and disconnecting from the database. Connection pooling can improve the response time of any application that requires connections, especially Web-based applications.
When a user makes a request over the Web to a WebSphere application, it is very common that the application needs to access DB2 server before sending a response back. These user requests do not incur the overhead of creating a new connection, because the data source might locate and use an existing connection from the pool of connections. When the request is satisfied and the response is returned to the user, the connection is returned to the connection pool for reuse.

Here again, the overhead of a disconnect is avoided. Each user request incurs a fraction of the cost of connection or disconnection. After the initial resources are used to produce the connections in the pool, additional overhead is insignificant because the existing connections are reused. Such reuse can have significant performance benefits since the cost of database connect and disconnect is amortized over multiple users.

**WebSphere exceptions**
These refer to WebSphere monitors' specific errors thrown by the database. A set of these errors is mapped to WebSphere Application Server specific exceptions.

All enterprise bean container-managed persistence (CMP) beans under the EJB 2.0 Specification receive a standard F when an operation fails.

JDBC applications receive a standard SQLException if any JDBC operation fails.

The product provides special exceptions for its relational resource adapter (RRA), to indicate that the connection currently held is no longer valid. The ConnectionWaitTimeoutException indicates that the application timed out trying to get a connection. The StaleConnectionException indicates that the connection is no longer valid.

**ConnectionWaitTimeout**
The ConnectionWaitTimeout exception indicates that the application has waited for the number of seconds specified by the connection timeout setting and has not received a connection. This situation can occur when the pool is at maximum size and all of the connections are in use by other applications for the duration of the wait. In addition, there are no connections currently in use that the application can share because either the connection properties do not match, or the connection is in a different transaction.

When using a Version 4.0 data source, the ConnectionWaitTimeout throws an exception whose class is com.ibm.ejs.cm.pool.ConnectionWaitTimeoutException.
For connection factories, the ConnectionWaitTimeout throws a ResourceException whose class is com.ibm.websphere.ce.j2c.ConnectionWaitTimeoutException.

Finally, Version 5.0 data sources throw an SQLException subclass called com.ibm.websphere.ce.cm.ConnectionWaitTimeoutException.

**Stale connections**

WebSphere V5 provides a special subclass of java.sql.SQLException when using connection pooling to access a relational database. This com.ibm.websphere.ce.cm.StaleConnectionException subclass exists in both a WebSphere 4.0 data source and in the new data source using the relational resource adapter. It is used to indicate that the connection currently held is no longer valid. This situation can occur for many reasons, including the following:

- The application tries to get a connection and fails, as when the database is not started.
- A connection is no longer usable because of a database failure. When an application tries to use a previously obtained connection, the connection is no longer valid. In this case, all connections currently in use by the application can get this error when they try to use the connection.
- The connection is orphaned (because the application had not used it in at most two times the value of the unused timeout setting) and the application tries to use the orphaned connection. This case applies only to Version 4.0 data sources.
- The application tries to use a JDBC resource, such as a statement, obtained on a stale connection.
- A connection is closed by the Version 4.0 data source auto connection cleanup and is no longer usable. Auto connection cleanup is the standard mode in which connection management operates. This mode indicates that at the end of a transaction, the transaction manager closes all connections enlisted in that transaction. This enables the transaction manager to ensure that connections are not held for excessive periods of time and that the pool does not reach its maximum number of connections prematurely.

One ramification of having the transaction manager close the connections and return the connection to the free pool after a transaction ends, is that an application cannot obtain a connection in one transaction and try to use it in another transaction. If the application tries this, a StaleConnectionException is thrown because the connection is already closed.

In the case of trying to use an orphaned connection or a connection cleaned up by auto connection cleanup, a StaleConnectionException indicates that the application has attempted to use a connection already returned to the connection
pool. It does not indicate an actual problem with the connection. However, other cases of a StaleConnectionException indicate that the connection to the database has gone bad, or stale. Once a connection has gone stale, you cannot recover it, and you must completely close the connection rather than returning it to the pool.

**Detecting stale connections**
When a connection to the database becomes stale, operations on that connection result in an SQLException from the JDBC driver. Because an SQLException is a rather generic exception, it contains state and error code values that you can use to determine the meaning of the exception. However, the meanings of these states and error codes vary depending on the database vendor. The connection pooling runtime module maintains a mapping of which SQL state and error codes indicate a StaleConnectionException for each database vendor supported. When the connection pooling runtime module catches any SQLException, it checks to see if this SQLException is considered a StaleConnectionException for the database server in use.

**Recovering from stale connections**
Recovering from stale connections is a joint effort between the application server run time and the application developer. From an application server perspective, the connection pool is purged based on its PurgePolicy setting. For instructions about setting PurgePolicy, please refer to the topic “Connection Pool Settings” in WebSphere Application Server V5 InfoCenter.

Explicitly catching a StaleConnectionException is not required in an application. Because applications are already required to catch java.sql. SQLException, and StaleConnectionException extends SQLException. StaleConnectionException can be thrown from any method that is declared to throw SQLException, and is caught automatically in the general catch-block. However, explicitly catching StaleConnectionException makes it possible for an application to recover from bad connections. When application code catches StaleConnectionException, it should take explicit steps to handle the exception.

**Error mapping in DataStoreHelper**
Error mapping is necessary because various database vendors can provide different SQL errors and codes that might mean the same things. For example, the StaleConnectionException has different codes in different databases. The DB2 SQLCODEs of 1015, 1034, 1036 and so on, indicate that the connection is no longer available because of a temporary database problem.

To provide portability for applications, WebSphere Application Server provides a DataStoreHelper interface to enable mapping of these codes to the WebSphere
Application Server exceptions. The following code segment illustrates how to add two error codes into the error map.

**Example 8-1  DataStoreHelper**

```java
public class NewDSHelper extends GenericDataStoreHelper
{
    public NewDSHelper()
    {
        super(null);
        java.util.Hashtable myErrorMap = null;
        myErrorMap = new java.util.Hashtable(2);
        myErrorMap.put(new Integer(-803), myDuplicateKeyException.class);
        myErrorMap.put(new Integer(-1015), myStaleConnectionException.class);
        myErrorMap.put("S1000", MyTableNotFoundException.class);
        setUserDefinedMap(myErrorMap);
        ...
    }
}
```

### 8.1.2 Prepared statement cache

Applications may access a database using JDBC via any of the three options shown in Figure 8-1 on page 293.
Below we explain statements.

- A **statement** is a class that can execute a SQL string passed into it. When the SQL statement is executed, two high-level phases occur: The statement must be “prepared”, and then it can be “executed”. During the prepare phase, DB2 will parse the SQL text and perform the steps necessary to put the query into a form DB2 can understand. Then during the execution phase the query can actually be performed. The prepare phase can consume a noticeable amount of time and CPU resources.

- A **PreparedStatement**\(^1\) refines a statement by adding substitution parameters, and by separating the SQL compilation process from the execution of the statement.

This allows applications to prepare the statement once, and then reuse it multiple times with different distinct values in the parameter markers.

Applications that repeatedly execute the same SQL statement across multiple transactions can save a significant amount of processing time and network traffic by:

- Associating each such statement with its own statement handle.

---

\(^1\) A prepared statement is a pre-compiled SQL statement that is stored in a prepared statement object.
b. Preparing these statements once at the beginning of the application.

c. Then executing the statements as many times as is needed throughout the application.

By holding onto the statement handle, application programmers can reuse prepared statements across units-of-work (UOW).

The *global dynamic statement cache* is a memory area on the server that is used to store the most popular access plans for prepared SQL statements. Before each statement is prepared, the server automatically searches this cache to see if the access plan has already been created for the exact SQL statement (by this application or any other application or client). If so, the server does not need to generate a new access plan, but will use the one in the cache instead.
**Note:** Note the following:

- For DB2 Universal JDBC Driver Provider, a custom property of a data source named deferPrepares is provided as a performance directive that affects the semantics of the input data type conversion capability of the driver. By default the Universal driver defers prepare requests. For example, a logical call to java.sql.Connection.prepareStatement() will not result in a physical prepare on the DB2 server until a logical request to execute the statement is issued, that is, java.sql.PreparedStatement.execute().

If deferPrepares is enabled, the internal server prepare requests are deferred until execute time. This allows the prepare and execute to be piggybacked as a single request to the server, thereby reducing network delays for Type 4 connectivity. The deferral of prepare means that the driver works without the benefit of a described parameter or result set meta data. So undescribed input data is sent “as is” to the server without any data type cross-conversion of the inputs.

Support for cross conversion of input types is an extension to JDBC, which does not require such support, but is supported by the Universal driver. Therefore, if a statement execution fails when deferPrepares is enabled, the execution is retried by the driver with described input. This statement retry logic is internal to the driver, and is seamless to the application. However, if an application enables deferPrepares, it is strongly recommended that input data types match the database column types as required by the JDBC specification.

This means that if deferPrepares is enabled, then setter calls such as PreparedStatement.setShort() should match the type of the underlying column type (for example, SMALLINT) for optimum performance. If deferPrepares is disabled, then the driver requests describe information when the statement is logically prepared and therefore has described input parameter meta data available, and cross conversion of input types is supported for all PreparedStatement setter methods without requiring internal driver statement retry logic.

- For DB2 Legacy CLI-based Type 2 JDBC Driver, DB2 CLI provides an option, DEFERREDPREPARE, which defers the sending of the PREPARE request until the corresponding execute request is issued. The two requests are then combined into one command/reply flow (instead of two) to minimize network flow and to improve performance.

- Deferred prepare is the default and must be explicitly turned off, if required.
A CallableStatement takes away the SQL compilation process entirely by executing a SQL stored procedure\(^2\).

WebSphere Application Server provides the administrator with an option of specifying a prepared statement cache. Similar to a prepared statement in the database, this cache stores the PreparedStatement object of previously prepared statements on a connection basis.

The statement cache contains PreparedStatement objects of the most recently executed statements on a per connection basis, as shown in Figure 8-2. It describes a data source configured with a statement cache size of 10 statements, and a maximum of three concurrent connections.

![Figure 8-2 PreparedStatement cache: An example](image)

In Figure 8-2 the application runs five SQL statements (two selects, one delete, one insert, and one update). The connections have already been created, and many SQL statements have been executed. There are three prepared statements cached for Connection 1 and 2. Connection 3 has four statements cached. Because statements are compiled into prepared statements as they are used, the prepared statement cache reflects the database usage patterns of the application.

\(^2\) A stored procedure is a prepared sequence of statements that resides on the DBMS.
A PreparedStatement object, representing a given SQL statement, can appear multiple times in the prepared statement cache. In particular, it can appear once for every connection in the connection pool. In Figure 8-2 on page 296, statements 1 and 2 appear three times—one for each connection. Statement 3 does not appear for connection 3, and Statements 4 and 5 only appear for connection 3. Hence, it might take a little longer to execute Statements 4 and 5 if they occur on Connections 1 and 2 because of the need to recompile them for those connections. A better alternative for this example would be to set the prepared statement cache size to 15 statements, to allow for each of the 3 connections to cache their 5 prepared statements.

The value of the statement cache can be set in a data source configuration panel in the WebSphere V5 Administrative Console. To disable the statement cache, set the Statement Cache Size value to 0.

The actions performed by WebSphere Application Server when the statement cache is defined with non-zero and zero values is described here.

**Non-zero statement cache size**

The following actions are performed by WebSphere Application Server when an application issues a prepare statement against a connection:

- It does a look aside in the statement cache for this connection to see if a PreparedStatement object already exists for this statement.
  - If such an object exists, it returns this object to the application
  - If such an object does not exist, WebSphere Application Server requests a prepare against the database for this statement, and returns the PreparedStatement object to the application.

**Note:** A WebSphere Application Server request to prepare a statement does not necessarily cause the database to execute the prepare statement. DB2 UDB will first perform a look aside in its own global dynamic statement cache for this statement, before deciding to execute the prepare statement and return the PreparedStatement object to WebSphere Application Server.

**Note:** DB2 UDB will prepare a statement if it is not found in its global dynamic statement cache, even if the PreparedStatement object for this statement is still available in WebSphere Application Server’s statement cache.

---

3 With Deferred Prepare (which is the default), this prepare request is only sent to the database with the next execute request for this PreparedStatement object.
This operation is repeated for each prepare statement.

- When the application performs a statement close() or commits, it puts this PreparedStatement object in the statement cache. If this action would cause the cache size limit to be exceeded, then it discards an existing PreparedStatement object in the cache using a first-in-first-out (FIFO) basis, to make room for this one. The PrepStmt Cache Discards field in Figure 8-3 on page 299, which counts the number of discards from the statement cache, increments after a discard.

**Note:** The statement cache holds a pool of PreparedStatement objects, each of which is associated with a statement handle that is tied to a particular connection.

WebSphere Application Server allocates a separate statement handle for each open cursor in the application, and moves the information related to this statement handle to the statement cache at commit time. Think of the statement cache as a pool of open statement handles for reuse by subsequent application units-of-work (UOWs) that use this same connection.
Zero statement cache size

When there is no statement cache, WebSphere Application Server will execute a request for a prepare against the database for every application prepare statement. Here again, a separate statement handle is allocated by WebSphere Application Server for each open cursor. However, at commit, the statement handles are destroyed, since there is no statement cache. Subsequent applications that reuse this connection will require WebSphere Application Server to set up the appropriate structures to issue the prepare request against the database.
The following is an overview of how DB2 handles prepared statements.

In DB2, each SQL statement is cached at a database level, and can be shared among different applications, unlike WebSphere Application Server's prepared statement cache. Static SQL statements are shared among applications using the same package, while dynamic SQL statements are shared among applications using the same compilation environment, and the exact same statement text.

Once a dynamic SQL statement has been created and cached, it can be reused over multiple units of work without the need to prepare the statement again.

The size of the DB2 global statement cache is defined via the `PCKCACHESZ` database configuration parameter. This cache is allocated in DB2 out of the Database Global Memory when the database is initialized, and freed when the database is shut down. It is used for caching both static and dynamic SQL statements.

Caching of packages allows the database manager to reduce its internal overhead by eliminating the need to access the system catalogs when reloading a package; or, in the case of dynamic SQL, eliminating the need for compilation. Sections are kept in the package cache until one of the following occurs:

- The database is shut down.
- The package or dynamic SQL statement is invalidated.
- The cache runs out of space.

**Note:** DB2 will automatically recompile the statement as required if environment or data object changes occur, such as an object being created or dropped, running `runstats`, etc.

Given our understanding of the default deferred prepare statement option, and DB2’s algorithm for the global dynamic statement cache, performance benefits of the statement cache is unlikely to stem from a reduction in network flows, or an elimination of an SQL prepare by DB2.

**Note:** The elimination of the creation of a prepare request, and the creation and destruction of statement handle structures on the WebSphere Application Server and DB2 side, may contribute to measurable performance degradation depending upon the proportion of this activity to the overall processing of the application.
This caching of the section for a static or dynamic SQL statement can improve performance, when the same statement is used multiple times by applications connected to a database, including WebSphere Application Server applications. This is particularly beneficial in a transaction processing application.

There must be sufficient memory allocated in the cache to hold all sections of the SQL statements currently being executed. If there is more space allocated than currently needed, then sections are cached. These sections can simply be executed the next time they are needed without having to load or compile them.

**Note:** The limit specified by the PCKCACHESZ parameter is a soft limit. This limit may be exceeded, if required, provided memory is still available in the database shared set.

Use the pkg_cache_size_top monitor element to determine the largest that the package cache has grown, and the pkg_cache_num_overflows monitor element to determine how many times the limit specified by the pckcachesz parameter has been exceeded. Refer to DB2 product documentation for more details.

### 8.1.3 Monitoring WebSphere application on DB2 UDB server

In a typical scenario, applications running on WebSphere Application Server that access data on DB2 UDB servers through a data source are indistinguishable to the DB2 server.

For example, Figure 8-4 shows the output from the DISPLAY THREAD(*) command on DB2 UDB for z/OS and OS/390 before we change the environment to set additional information about the incoming client. The application name is labeled as java.exe and the user ID and workstation defaults to the user ID and workstation that was used to get the database connection. This might not be accurate for all scenarios where further modular information about the application or the module currently running needs to be set.

![Figure 8-4 Monitor WebSphere application from DB2 for zOS](attachment:image)

Client connections on a DB2 UDB for UNIX, Windows and Linux can be seen through a DB2 LIST APPLICATIONS command, as shown in Example 8-2 on page 302.
Example 8-2   Monitor WebSphere application from DB2 UDB

```bash
> db2 list applications
Auth Id Application Name      Appl. Handle   Application Id                DB       # of Agents
------- ----------- ------- ------------------------------ -------- ------
DB2V81   java.exe     195     *LOCAL.db2v81.07B0C0003412   TRADE3DB   1
DB2V81   java.exe     169     G91A59AF.017A.0BA7C0210130   TRADE3DB   1
```

To solve this problem, DB2 provides a native API (SQLESETI) that can be used to set additional information about the clients from the application. This API can be accessed from its C interface, or the JDBC programming interface. WebSphere provides an integrated feature in data source using this API to allow users to pass more client information to DB2 UDB server from WebSphere. This information could be the application name, user ID, workstation name, or an accounting string.

To use the integrated feature, you can set the client information in the data source configuration panel from the WebSphere V5 administrative console. In the custom properties panel, set any of the following properties (Figure 8-5):

- ClientApplName
- ClientWrkstnName
- ClientUserid
- ClientAcctStr

![Figure 8-5   Set client application properties](image_url)

After setting the properties, restart WebSphere Application Server. Then we can see the effort of the configuration in DB2 UDB monitoring.
On the server, turn on DB2 monitor switches and issue the `db2 snapshot` command (see Example 8-3).

**Example 8-3  DB2 UDB Sanpshot command**

```
db2 update monitor switches using
    table on bufferpool on uow on sort on lock on statement on
    db2 get snapshot for applications on <dbname>
```

In the snapshot, you should see content similar to the following:

```
...  ...
TP Monitor Client User ID:           WebSphere-Userid
TP Monitor Client Workstation Name:  WebSphere-Wrkstn
TP Monitor Client Application Name:  WebSphere-AppName
TP Monitor Client Accounting String: WebSphere-AccStr
...  ...
```

This information can be used to identify WebSphere Application Server connections.

Figure 8-6 shows the output of a DISPLAY THREAD(*) command running on DB2 UDB for OS/390 and z/OS server after the client information has been set. Notice that the application name, user ID and workstation name have been set to the values we specified earlier.

8.1.4 Tuning WebSphere DataSources

Tuning WebSphere DataSources is a very critical part of tuning the WebSphere Queue network mentioned in Chapter 6, “WebSphere Application Server V5 performance tuning” on page 169.

There are two settings to be concerned with for determining data source queues:

- Connection pool size
- Prepared statement cache size
Connection pool size
When accessing any database, the initial database connection is an expensive operation. WebSphere Application Server supports JDBC 2.0 Standard Extension APIs to provide support for connection pooling and connection reuse. The connection pool is used for direct JDBC calls within the application, as well as for enterprise beans using the database.

Tivoli Performance Viewer can help find the optimal size for the connection pool. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the Pool Size, Percent Used and Concurrent Waiters counters of the data source entry under the JDBC Connection Pools module. The optimal value for the pool size is that which reduces the values for these monitored counters. If Percent Used is consistently low, consider decreasing the number of connections in the pool.

Better performance is generally achieved if the value for the connection pool size is set lower than the value for the Max Connections in the Web container. Lower settings for the connection pool size (10-30 connections) typically perform better.

Figure 8-7 TVP monitors connection pooling
than higher (more than 100) settings. On UNIX platforms, a separate DB2
process is created for each connection. These processes quickly affect
performance on systems with low memory, causing errors.

Each entity bean transaction requires a connection to the database specifically to
handle the transaction. Be sure to take this into account when calculating the
number of data source connections.

Instructions on how to set the connection pool size are provided in “Creating and
configuring DB2 Data Source” on page 144.

Deadlock can occur if the application requires more than one concurrent
connection per thread, and the database connection pool is not large enough for
the number of threads. Suppose each of the application threads requires two
concurrent database connections and the number of threads is equal to the
maximum connection pool size. Deadlock can occur when both of the following
are true:

- Each thread has its first database connection, and all are in use.
- Each thread is waiting for a second database connection, and none would
  become available since all threads are blocked.

To prevent the deadlock in this case, the value set for the database connection
pool must be at least one higher, one of the waiting threads to complete its
second database connection and free up to allow database connections.

To avoid deadlock, code the application to use, at most, one connection per
thread. If the application is coded to require C concurrent database connections
per thread, the connection pool must support at least the following number of
connections, where T is the maximum number of threads.

\[ T \times (C - 1) + 1 \]

The connection pool settings are directly related to the number of connections
that the DB2 UDB server is configured to support. If the maximum number of
connections in the pool is raised, and the corresponding settings in the DB2 UDB
server are not raised, the application fails and SQL exception errors are
displayed in the stderr.log file.

**DB2 MAX_COORDAGENTS and MAXAGENTS settings**

In DB2 UDB V8, some parameters need to be adjusted to make DB2 UDB
perform well with WebSphere. *MAX_COORDAGENTS* defines the maximum
number of physical DB2 server agents. When configuring the connection pool
size of a data source for DB2, confirm the *MAX_COORDAGENTS* setting in DB2
server is high enough to handle the maximum number of connections for all the
data sources. If you are planning to use multiple cluster members, set the
The data source optimizes the processing of prepared statements to help make SQL statements process faster. It is important to configure the cache size of the data source to gain optimal statement execution efficiency. A prepared statement is a precompiled SQL statement that is stored in a prepared statement object.

This object is used to efficiently execute the given SQL statement multiple times. If the JDBC driver specified in the data source supports precompilation, the creation of the prepared statement will send the statement to the database for precompilation. Some drivers might not support precompilation and the prepared statement might not be sent until the prepared statement is executed.

If the cache is not large enough, useful entries will be discarded to make room for new entries. In general, the more prepared statements your application has, the larger the cache should be. For example, if the application has five SQL statements, set the prepared statement cache size to 5, so that each connection has five statements.

Tivoli Performance Viewer (Figure 8-8 on page 307) can help tune this setting to minimize cache discards. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the PrepStmt Cache Discard counter of the JDBC Connection Pools module. The optimal value for the statement cache size is the setting used to get either a value of zero or the lowest value for PrepStmt Cache Discards.
Figure 8-8  Using TPV tuning statement cache

As with the connection pool size, the statement cache size setting requires resources at the database server. Specifying too large of a cache could have an impact on database server performance. It is highly recommended that you consult your database administrator for determining the best setting for the prepared statement cache size.

8.1.5  Best practices

Best practices for using WebSphere data sources are categorized as being application related, and system related.
Application-related best practices
The following application programming techniques are considered to be best practices for achieving optimal performance.

- Use connection sharing.

  Connections can be shared by the same user (user name and password), but only within the same transaction. This is known as connection sharing, and should be used only on a single thread. For example, when a servlet starts a user transaction and gets a connection, it does some database operations and then calls a CMP EJB, which also needs a connection. The connection is then shared between the servlet and the EJB on a single thread.

  Connections should never be shared across threads. It is possible to see the same connection on multiple threads at the same time, but this is an error state, and is caused by poor programming practices; for example, when a servlet init() method performs the JNDI lookup for a connection, and also gets its own connection. Each time the service() method is called, the same connection gets used, but these calls come in on different threads within the JVM. This can cause problems like time-outs and StaleConnectionException errors.

- Do not declare connections as static objects.

  If connection objects are declared as static, then it is possible for the same connection to be shared by different threads at the same time, resulting in problems with connection pooling and database access. A connection should always be obtained and released within the method that requires it.

- Do not declare connection objects as instance variables.

  In a servlet, all variables declared as instance variables act as if they are class variables. For example, suppose a servlet is defined with an instance variable:

  ```java
  Connection conn = null;
  ```

  This variable acts as if it is static. This implies that all instances of the servlet would use the same connection object. This is because a single servlet instance can be used to serve multiple Web requests in different threads.

- Open one connection at a time.

  In general, an application should open only one connection to the database at a time. If two getConnection() calls with the same parameters are issued in the same global transaction, only a single connection is allocated. You are allowed to open only one single-phase-commit connection within a global transaction.

---

4 This is a static method, which is what enables multiple threads to use the same connection.
However, if the application is not running in a global transaction (as is the case with most servlets), two `getConnection()` calls result in two separate connections. This utilizes more resources than necessary, and causes your connection pool to fill twice as fast, often resulting in `ConnectionWaitTimeout` exceptions.

If the application requires multiple simultaneous connections, close each connection as soon as it is no longer required, to free that connection up for another user.

- **Always close objects.**

  It is very important that `ResultSet`, `Statement`, `PreparedStatement`, and `Connection` objects get closed properly in the application.

  If connections are not closed properly, users may experience long waits for connections to time out, and delay return of the connection to the free pool. Unclosed `ResultSet`, `Statement`, or `PreparedStatement` objects unnecessarily hold resources at the database as well.

  To ensure that these objects are closed in both correct execution and exception or error states, always close `ResultSet`, `Statement`, `PreparedStatement`, and `Connection` objects in the `finally` section of a `try/catch` block.

  WebSphere Application Server will try to clean up JDBC resources on a connection after it has been closed. However, this behavior should not be relied upon, especially if the application might be migrated to another platform in the future, thus requiring code rewrite.

- **Do not close connections in a finalize method.**

  If an application relies on the method `finalize()` to close a connection or other JDBC resource, the connection is not closed until the object that obtained it is garbage collected, because that is when the `finalize()` method is called.

  Databases can quickly run out of the memory required to store the information about all of the JDBC resources currently open. In addition, the pool can quickly run out of connections to service other requests.

- **Set fetch size using JDBC.**

  Use `setFetchsize()` to set the fetch size that match DB2’s recording blocking parameter (RQRIOBLK) to optimize SELECT query’s performance.

**System-related best practices**

Routine monitoring of the connection pool is critical to adjusting the various parameters for achieving optimal performance. The Tivoli Performance Viewer is the primary tool for monitoring connection pool usage. Appropriate monitoring levels have to be set in order to view desired information.
While the default monitoring level is *none*, we recommend that you choose an appropriate monitoring level based on your organization’s needs, and the overheads associated with it for your particular environment. The administrator should also occasionally perform exception monitoring with a monitoring level of Maximum during bursts of peak activity to obtain adequate information about the system under stress, in order to effectively tune connection pool parameters.

The following considerations apply to tuning the various connection pool parameters.

- **Minimum pool size**: A correct minimum value for the pool can be determined by examining the applications that are using the pool. If it is determined, for example, that at least four connections are needed at any point in time, the minimum number of connections should be set to 4 to ensure that all requests can be fulfilled without connection wait timeout exceptions.

  At off-peak times, the pool shrinks back to this minimum number of connections. A good rule of thumb is to keep this number as small as possible to avoid holding connections unnecessarily open.

- **Maximum pool size**: The maximum number of connections that the connection pool can hold open to the database. The pool holds this maximum number of connections open to the database at peak times. At off-peak times, the pool shrinks back to the minimum number of connections.

  The best practice is to ensure that only one connection is required on a thread at any time. This avoids possible deadlocks when the pool is at maximum capacity and no connections are left to fulfill a connection request. Therefore, with one connection per thread, the maximum pool size can be set to the maximum number of threads. When using servlets, this can be determined by looking at the *MaxConnections* property in the Servlet Engine.

  The Avg Wait Times (ms), Faults, Percent Used and Percent Maxed fields in Figure 8-7 on page 304 can help you decide whether the maximum pool size ought to be increased or decreased.

- **Connection timeout**: If applications are often catching ConnectionWaitTimeoutException, this usually means one of two things: Either the connection timeout property is set too low, or the connection pool is always at maximum capacity, and cannot find a free connection for the application to use:

  - If the exception is being caused by the connection timeout value being set too low, the solution is to set it to a higher value.
  - If the exception is caused by too few connections in the pool, the maximum pool size setting needs to be investigated.
The Faults field in Figure 8-7 on page 304, when analyzed in conjunction with the Percent Used and Percent Maxed fields, can provide guidance on the setting of this parameter.

## 8.2 Persistent session

By default, WebSphere places session objects in memory. However, the administrator has the option of enabling persistent session management, which instructs WebSphere to place session objects in a persistent store. Administrators should enable persistent session management when:

- The user's session data must be recovered by another cluster member after a cluster member in a cluster fails or is shut down.
- The user's session data is too valuable to lose through unexpected failure at the WebSphere node.
- The administrator desires better control of the session cache memory footprint. By sending cache overflow to a persistent session store, the administrator controls the number of sessions allowed in memory at any given time.

There are two ways to configure session persistence in WebSphere V5. In addition to the traditional database persistence, there is a new feature that provides memory-to-memory session state replication using WebSphere internal messaging; see Figure 8-9 on page 312.
The memory-to-memory session state replication is a new feature of WebSphere Application Server V5 that enables sessions to be shared among application servers without using a database. Using this method, sessions are stored in the memory of an application server, providing the same functionality as a database for session persistence. Separate threads handle this functionality within an existing application server process.

Note: The internal messaging system is not associated with the embedded WebSphere JMS provider. Instead the code is provided as part of the core WebSphere Application Server libraries. As such, the JMS Server does not need to be started.

All information stored in a persistent session store must be serialized (that is, implement the java.io.Serializable interface). As a result, all of the objects held by a session must implement java.io.Serializable if the session needs to be stored in a persistent session store.

In general, consider making all objects held by a session serialized, even if immediate plans do not call for the use of persistent session management. If the
Web site grows, and persistent session management becomes necessary, the transition between local and persistent management occurs transparently to the application if the sessions only hold serialized objects. If not, a switch to persistent session management requires coding changes to make the session contents serialized.

However, the HttpSession can also contain the following J2EE objects, which are not serializable:

- javax.ejb.EJBObject
- javax.ejb.EJBHome
- javax.naming.Context
- javax.transaction.UserTransaction

The WebSphere session manager works around the problem of serializing these objects.

Persistent session management does not impact the session API, and Web applications require no API changes to support persistent session management. However, as mentioned above, applications storing un-serializable objects in their sessions require modification before switching to persistent session management.

If database persistence is used, using multi-row sessions becomes important if the size of the session object exceeds the size for a row, as permitted by the WebSphere session manager. If the administrator requests multi-row session support, the WebSphere session manager breaks the session data across multiple rows as needed. This allows WebSphere to support large session objects. Also, this provides a more efficient mechanism for storing and retrieving session contents under certain circumstances. See "Single vs. multi-row schemas in database persistence" on page 322. Using a cache lets the session manager maintain a cache of most recently used sessions in memory. Retrieving a user session from the cache eliminates a more expensive retrieval from the persistent store. The session manager uses a “least recently used” scheme for removing objects from the cache. Session data is stored to the persistent store based on the write frequency and write option selected.

In this section, we only discuss the session database persistence using DB2 UDB V8 in detail.

### 8.2.1 Enable database persistence

Before enabling database persistence from the WebSphere V5 administration console, the following tasks should be done:

- Create a session database in the DB2 server to store session data.
Create a DB2 non-XA JDBC provider. Instructions refer to Chapter 5, “Operational setup” on page 117.

Create a data source for the session database. In the following instructions, it is assumed that the data source JNDI name is jdbc/Sessions.

The following steps shows how to enable database persistence at the application server level. In WebSphere V5, session management settings can also be performed at the enterprise application level and the Web application level.

In order to enable database persistence, repeat the following steps for each application server.

1. Click Servers -> Application Servers.
2. Select the server.
3. Click Web Container in the Additional Properties table.
4. Click Session Management.
5. Click Distributed Environment Settings.
6. Select Database and click Database.
7. Enter the database information:
   – Enter the data source JNDI name.
   – Enter the user ID and password to be used to access the database. You will need to confirm the password.
   – If you are using DB2 and you anticipate requiring row sizes greater than 4 KB, select the appropriate value from the DB2 row size pull-down.
   – If the DB2 row size is other than 4 KB, you are required to enter the name of tablespace.
8. Click **OK**.

After you have updated each server, save the configuration changes, synchronize them with the servers, and restart the application servers.

### 8.2.2 Session management tuning

Performance tuning for session management consists of defining the following:

- How often session data is written (write frequency settings)
- How much data is written (write contents settings)
- When the invalid sessions are cleaned up (session cleanup settings)
Write frequency settings
You can select from three different settings that determine how often session data is written to the persistent data store:

- **End of servlet service**: If the session data has changed, it will be written to the persistent store after the servlet finishes processing an HTTP request.
- **Manual update**: The session data will be written to the persistent store when the sync() method is called on the IBMSession object.
- **Time-based**: The session data will be written to the persistent store based on the specified write interval value.

**Note**: The last access time attribute is updated each time the session is accessed by the servlet or JSP, whether the session is changed or not. This is done to make sure the session does not time out.

- If you choose the end of servlet service option, each servlet or JSP access will result in a corresponding persistent store update of the last access time.
- If you select the manual update option, the update of the last access time in persistent store occurs on sync() call or at later time.
- If you use time-based updates, the changes are accumulated and written in a single transaction. This can significantly reduce the amount of I/O to the persistent store.

Let us consider an example where the Web browser accesses the application once every 10 seconds:

- In End of servlet service mode, the session would get written out every 10 seconds.
- In Manual update mode, the session gets written out whenever the servlet issues IBMSession.sync(). It is the responsibility of the servlet writer to use the IBMSession interface instead of the HttpSession Interface and the servlets/JSPs must be updated to issue the sync().
- In Time-based mode, the servlet or JSP need not use the IBMSession class nor issue IBMSession.sync(). If the write interval is set to 120 seconds, then the session data gets written out at most every 120 seconds.

**End of servlet service**
When the write frequency is set to the end of servlet service option, WebSphere writes the session data to the persistent store at the completion of the HttpServlet.service() method call. Exactly what is written depends on the write content settings.
**Manual update**

In manual update mode, the session manager only sends changes to the persistent data store if the application explicitly requests a save of the session information.

Manual update mode requires that an application developer use the IBMSession class for managing sessions. When the application invokes the sync() method, the session manager writes the modified session data and last access time to the persistent store. The session data that is written out to the persistent store is controlled by the write contents option selected. Example 8-4 is a sample code.

If the servlet or JSP terminates without invoking the sync() method, the session manager saves the contents of the session object into the session cache (if caching is enabled), but does not update the modified session data in the session database. The session manager will only update the last access time in the persistent store asynchronously, at a later time.

**Note:** Manual updates use an IBM extension to HttpSession that is not part of the Servlet 2.3 API.

**Example 8-4   Use IBMSession to hold the session information**

```java
public void service (HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException
{
  // Use the IBMSession to hold the session information
  // We need the IBMSession object because it has the manual update
  // method sync()
  com.ibm.websphere.servlet.session.IBMSession session =
  (com.ibm.websphere.servlet.session.IBMSession)req.getSession(true);
  Integer value = 1;
  //Update the in-memory session stored in the cache
  session.putValue("MyManualCount.COUNTER", value);
  //The servlet saves the session to the persistent store
  session.sync();
}
```

This interface gives the Web application developer additional control of when (and if) session objects go to the persistent data store. If the application does not invoke the sync() method, and manual update mode is specified, the session updates goes only to the local session cache, not the persistent data store. Web developers use this interface to reduce unnecessary writes to the session database, and thereby to improve overall application performance.

All servlets in the Web application server must perform their own session management in manual update mode.
**Time-based writes to the session database**

Using the time-based write option will write session data to the persistent store at a defined write interval. The reasons for implementing time-based write lies in the changes introduced with the Servlet 2.2 API. The Servlet 2.2 specification introduced two key concepts:

- It limits the scope of a session to a single Web application.
- It both explicitly prohibits concurrent access to an HttpSession from separate Web applications but allows for concurrent access within a given JVM.

Because of these changes, WebSphere provides the session affinity mechanism that assures us that an HTTP request is routed to the Web application handling its HttpSession. This assurance still holds in a WLM environment when using persistent HttpSession. This means that the necessity to immediately write the session data to the persistent store can now be relaxed somewhat in these environments (as well as non-clustered environments), since the persistent store is now really only used for failover and session cache full scenarios.

With this in mind, it is now possible to gain potential performance improvements by reducing the frequency of persistent store writes.

Time-based writes requires session affinity for session data integrity.

The following details apply to time-based writes:

- The expiration of the write interval does not necessitate a write to the persistent store unless the session has been touched (that is, `getAttribute/setAttribute/removeAttribute` was called) since the last write.
- If a session write interval has expired and the session has only been retrieved (that is, `request.getSession()` was called since the last write) then the last access time will be written to the persistent store regardless of the write contents setting.
- If a session write interval has expired and the session properties have been either accessed or modified since the last write then the session properties will be written out in addition to the last access time. Which session properties get written out is dependent on the write contents settings.
- Time-based write allows the servlet or JSP to issue `IBMSession.sync()` to force the write of session data to the database.
- If the time between session servlet requests (for a particular session) is greater than the write interval, then the session effectively gets written out after each service method invocation.
- The session cache should be large enough to hold all of the active sessions. Failure to do this will result in extra persistent store writes, since the receipt of a new session request may result in writing out the oldest cached session to
the persistent store. Or to put it another way, if the session manager has to
remove the least recently used HttpSession from the cache during a full
cache scenario, the session manager will write out that HttpSession (per the
Write contents settings) upon removal from the cache.

- The session invalidation time must be at least twice the write interval to
  ensure that a session does not inadvertently get invalidated prior to getting
  written to the persistent store.

- A newly created session will always get written to the persistent store at the
  end of the service method.

**Write content settings**

These options control what is written. Please refer to “What is written to the
persistent session database” on page 324 before selecting one of the options,
since there are several factors to take into account. The options available are:

- Only update attributes: Only updated attributes are written to the persistent
  store.

- All session attributes: All attribute are written to the persistent store.

**Session cleanup settings**

WebSphere allows the administrator to defer to off hours the clearing of
invalidated sessions (sessions that are no longer in use and timed out) from the
persistent store. For more information see “Invalidating sessions” on page 327.

This can be done either once or twice a day. The fields available are:

- First time of day (0-23): The first hour during which the invalidated persistent
  sessions will be cleared from the persistent store. This value must be a
  positive integer between 0 and 23.

- Second time of day (0-23): The second hour during which the invalidated
  persistent sessions will be cleared from the persistent store. This value must
  be a positive integer between 0 and 23.

- The property, Specify distributed sessions cleanup schedule, is required to be
  selected to enable this option.

- Also consider using schedule invalidation for intranet-style applications that
  have a somewhat fixed number of users wanting the same HTTP session for
  the whole business day.

The property, Specify distributed sessions cleanup schedule, is required to be
selected to enable this option.

Also consider using schedule invalidation for intranet-style applications that have
a somewhat fixed number of users wanting the same HTTP session for the whole
business day.
Configuration

The session management tuning parameters can be set by selecting a pre-defined tuning level or by specifically specifying each parameter. To specify the performance settings for session management:

1. Select **Servers -> Application Servers** and click the application server.
2. Click **Web Container**.
3. Click **Session Management**.
4. Click **Distributed Environment Settings**.
5. Select from the predefined tuning levels or click **Custom Tuning Parameters**. A screen, as shown in Figure 8-11, will be presented.

![Configuration Screen](image)

*Figure 8-11  Session management tuning levels*
If you want to set each tuning parameter explicitly, select **Custom Settings**. Figure 8-12 is the screen for setting the parameter.

![Custom Tuning Parameters](image)

**Figure 8-12  Session management tuning parameters**

### 8.2.3 Using larger DB2 page size for database persistence

WebSphere persistent session supports 4 KB, 8 KB, 16 KB, and 32 KB DB2 page sizes, and hence can have larger varchar for bit data columns of 8 KB, 16 KB, or 32 KB. Using this performance feature, we see faster persistence for HttpSession of sizes of 4 KB to 31 KB.

To enable this feature, we must perform the following steps:

1. Create a new DB2 UDB buffer pool that supports the desired page size.
2. Create a table space with which the newly created buffer pool is associated.
3. Select the desired page size from the WebSphere Application Server's Session Manager (DB2 row size).

4. Type in the new table space name in the WebSphere Application Server's Session Manager (Table space name).

5. If the SESSIONS table already exists, drop it from the DB2 database:

   DB2 connect to session
   DB2 drop table sessions

6. Create a new DB2 buffer pool and tablespace, specifying the same page size (8 KB, 16 KB or 32 KB) for both, and assign the new buffer pool to this tablespace. The following are simple steps for creating an 8 KB page:

   DB2 connect to session
   DB2 CREATE BUFFERPOOL sessionBP SIZE 1000 PAGESIZE 8K
   DB2 connect reset
   DB2 connect to session
   DB2 CREATE TABLESPACE sessionTS PAGESIZE 8K MANAGED BY SYSTEM USING ('C:\DB2\NODE0000\SQL00005\sessionTS.0') BUFFERPOOL sessionBP
   DB2 connect reset

7. Configure the correct tablespace name and page size (DB2 row size) in the session management database configuration (Figure 8-10 on page 315). Restart WebSphere. On startup, the session manager creates a new SESSIONS table based on the page size and tablespace name specified.

8.2.4 Single vs. multi-row schemas in database persistence

When using the single-row schema, each user session maps to a single database row. This is WebSphere's default configuration for persistent session management. With this setup, there are hard limits to the amount of user-defined, application-specific data that WebSphere Application Server can access.

When using the multi-row schema, each user session maps to multiple database rows. In a multi-row schema, each session attribute maps to a database row. In addition to allowing larger session records, using a multi-row schema can yield performance benefits in certain usage scenarios, such as when larger amounts of data are stored in the session but only small amounts are specifically accessed during a given servlet processing of an HTTP request. In such a scenario, reducing both data retrieved and the serialization overhead for data the application does not use is beneficial to performance.

It should be stressed that switching between multi-row and single-row is not a trivial proposition.
Switching from single-row to multi-row schema

To switch from single-row to multi-row schema for sessions:

1. Modify the session manager properties to switch from single to multi-row schema. You need to select the Use Multi row schema on the Database setting of the Session Manager window, shown in Figure 8-10 on page 315.

2. Manually drop the database table or delete all the rows in the session database table.
   
   To drop the table:
   
   a. Determine which data source the session manager is using. This is set in the session management distributed settings window. See “Enable database persistence” on page 313.
   
   b. Look up the database name in the data source settings. You will need to find the JDBC provider, then the data source. The database name is in the custom settings.
   
   c. Use the database facilities to connect to the database and drop it.

3. Restart the application server or cluster.

To verify which option is better for your application’s needs, you can configure single-row usage to one database and multi-row usage to another database. Then you monitor the performance by switching the data source.

Table 8-1 Single vs. multi-row schemas

<table>
<thead>
<tr>
<th>Programming issue</th>
<th>Application scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reasons to use single-row</td>
<td>You can read/write all values with just one record read/write.</td>
</tr>
<tr>
<td></td>
<td>This takes up less space in a database, because you are guaranteed that each</td>
</tr>
<tr>
<td></td>
<td>session is only one record long.</td>
</tr>
<tr>
<td>Reasons not to use single-row</td>
<td>2 MB limit of stored data per session; that</td>
</tr>
<tr>
<td></td>
<td>is, the sum of sizes of all session attributes</td>
</tr>
<tr>
<td></td>
<td>is limited to 2 MB.</td>
</tr>
</tbody>
</table>
WebSphere supports two modes for writing session contents to the persistent store:

- Only updated attributes. Write only the HttpSession properties that have been updated via setAttribute() and removeAttribute().
- All session attributes. Write all the HttpSession properties to the database.

When a new session is initially created (with either of the above two options) the entire session is written, including any Java objects bound to the session. When using database persistence, the behavior for subsequent servlet or JSP requests for this session varies depending on whether the single-row or multi-row database mode is in use.

- In single-row mode:
  - Only updated attributes: If any session attribute has been updated (via setAttribute or removeAttribute), then all of the objects bound to the session will be written to the database.
  - All session attributes: All bound session attributes will be written to the database.

### 8.2.5 What is written to the persistent session database

<table>
<thead>
<tr>
<th>Programming issue</th>
<th>Application scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reasons to use multi-row</td>
<td>The application can store an unlimited amount of data; that is, you are limited only by the size of the database and a 2 MB-per-record limit (so the size of each session attribute can be 2 MB). The application can read individual fields instead of the whole record. When large amounts of data are stored in the session but only small amounts are specifically accessed during a given servlet's processing of an HTTP request, multi-row sessions can improve performance by avoiding unneeded Java object serialization.</td>
</tr>
<tr>
<td>Reasons not to use multi-row</td>
<td>If data is small in size, you probably do not want the extra overhead of multiple row reads when everything could be stored in one row.</td>
</tr>
</tbody>
</table>
In multi-row mode:
- Only updated attributes: Only the session attributes that were specified via `setAttribute` or `removeAttribute` will be written to the database.
- All session attributes: All of the session attributes that reside in the cache will be written to the database. If the session has never left the cache, then this should contain all of the session attributes.

By using the All session attributes mode, servlets and JSPs can change Java objects that are attributes of the HttpSession without having to call `setAttribute()` on the HttpSession for that Java object in order for the changes to be reflected in the database.

Adding the All session attributes mode provides some flexibility to the application programmer and protects against possible side effects of moving from local sessions to persistent sessions.

However, using All session attributes mode can potentially increase activity and be a performance drain. Individual customers will have to evaluate the pros and cons for their installation. It should be noted that the combination of All session attributes mode with time-based write could greatly reduce the performance penalty and essentially give you the best of both worlds.

As shown in Example 8-5 and Example 8-6, the initial session creation contains a `setAttribute` but subsequent requests for that session do not need to use `setAttribute`. The initial session creation contains a `setAttribute`, but subsequent requests for that session do not need to use `setAttribute`.

**Example 8-5  Servlet initially creates a session object**

```java
HttpSession sess = request.getSession(true);
myClass myObject = new myClass();
myObject.someInt = 1;
sess.setAttribute("myObject", myObject); // Bind object to the session
```

**Example 8-6  Subsequently modifying the session object**

```java
HttpSession sess = request.getSession(false);
myObject = sess.getAttribute("myObject"); // get bound session object
myObject.someInt++; // change the session object
// setAttribute() not needed with write "All session attributes" specified
```

**HttpSession set/getAttribute action summary**
Table 8-2 on page 326 summarizes the action of the HttpSession setAttribute and removeAttribute methods for various combinations of the row type, write contents, and write frequency session persistence options.
<table>
<thead>
<tr>
<th>Row type</th>
<th>Write contents</th>
<th>Write frequency</th>
<th>Action for setAttribute</th>
<th>Action for remove-Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-row</td>
<td>Only updated attributes</td>
<td>End of servlet service/sync() call with manual update</td>
<td>If any of the session data has changed, then write all of this session's data from cache.</td>
<td>If any of the session data has changed, then write all of this session's data from cache.</td>
</tr>
<tr>
<td>Single-row</td>
<td>Only updated attributes</td>
<td>Time-based</td>
<td>If any of the session data has changed, then write all of this session's data from cache.</td>
<td>If any of the session data has changed, then write all of this session's data from cache.</td>
</tr>
<tr>
<td>Single-row</td>
<td>All session attributes</td>
<td>End of servlet service/sync() call with manual update</td>
<td>Always write all of this session's data from cache.</td>
<td>Always write all of this session's data from cache.</td>
</tr>
<tr>
<td>Single-row</td>
<td>All session attributes</td>
<td>Time-based</td>
<td>Always write all of this session's data from cache.</td>
<td>Always write all of this session's data from cache.</td>
</tr>
<tr>
<td>Multi-row</td>
<td>Only updated attributes</td>
<td>End of servlet service/sync() call with manual update</td>
<td>Write only thread-specific data that has changed.</td>
<td>Delete only thread-specific data that has been removed.</td>
</tr>
<tr>
<td>Multi-row</td>
<td>Only updated attributes</td>
<td>Time-based</td>
<td>Write thread-specific data that has changed for all threads using this session.</td>
<td>Delete thread-specific data that has been removed for all threads using this session.</td>
</tr>
</tbody>
</table>
Multi-row mode has the notion of thread-specific data. Thread-specific data is defined as session data that was added or removed while executing under this thread. If End of servlet service or Manual update modes are used and Only updated attributes is enabled, then only the thread-specific data is written to the database.

### 8.2.6 Invalidating sessions

Sessions should be invalidated when the user no longer needs the session object (for example, the user has logged off the site). Invalidating a session removes it from the session cache, as well as from the persistent store.

WebSphere offers three methods for invalidation session objects:

- Programantically, by calling the invalidate() method on the session object. If the session object is accessed by multiple threads in a Web application, take care that none of the threads still have references to the session object.

- An invalidator thread scans for timed-out sessions every n seconds, where n is configurable from the administrative console. The session timeout setting is found in the session management configuration for the application server Web container.
For persistent sessions, the administrator can specify times when the scan will be run. This feature has the following benefits when used with persistent session:

- Persistent store scans can be scheduled during periods that normally have low demand. This avoids slowing down online applications due to contention in the persistent store.
- When this setting is used with the End of servlet service write frequency option, WebSphere does not have to write out the last access time with every HTTP request. This is because WebSphere does not have to synchronize the invalidator thread's deletion with the HTTP request access.

The session cleanup schedule setting is found in the session management settings for the Web container. You will find it with the custom tuning properties for distributed environments.

### 8.2.7 Session performance best practices

This section includes some considerations for developing and administering scalable, high-performance Web applications using WebSphere Application Server session support.

#### Session size

Large session objects pose several problems for a Web application. If the site uses session caching, large sessions reduce the memory available in the WebSphere instance for other tasks, such as application execution.

For a database persistent session using DB2, if session objects are larger than 32 k (see “Using larger DB2 page size for database persistence” on page 321), then the data are not cached in DB2 buffer pool. This will have an impact on the session performance. For example, assume that a given application stores 1 MB of information per user session object. If 100 users arrive over the course of 30 minutes, and assume the session timeout remains at 30 minutes, the application server instance must allocate 100 MB just to accommodate the newly arrived users in the session cache. Note that this number does not include previously allocated sessions that have not timed out yet. The actual memory required by the session cache could be considerably higher than 100 MB.

\[
1 \text{ MB per user session} \times 100 \text{ users} = 100 \text{ MB}
\]

Web developers and administrators have several options for improving the performance of session management:

- Reduce the size of the session object.
Configure the size of the session cache.
Add additional instances.
Invalidate unneeded sessions.
Increase the memory available.
Reduce the session timeout interval.

Reduce session object size
Web developers/application designers must carefully consider the information kept by the session object and not make the session objects too large.

Using larger DB2 page size
See “Using larger DB2 page size for database persistence” on page 321 to find out how WebSphere V5 can provide faster persistence of larger session objects when using DB2.

Session cache size
The session manager allows administrators to change the session cache size to alter the cache’s memory footprint. By default, the session cache holds 1000 session objects. By lowering the number of session objects in the cache, the administrator reduces the memory required by the cache.

However, if the user’s session is not in the cache, WebSphere must retrieve it from either the overflow cache (for local caching) or the session database (for persistent sessions). If the session manager must retrieve persistent sessions frequently, the retrievals may impact overall application performance.

Create additional application server instances
WebSphere also gives the administrator the option of creating additional application server instances. Creating additional instances spreads the demand for memory across more JVMs, thus reducing the memory burden on any particular instance. Depending on the memory and CPU capacity of the machines involved, the administrator may add additional instances within the same machine. Alternatively, the administrator may add additional machines to form a hardware cluster, and spread the instances across this cluster.

Invalidate unneeded sessions
If the user no longer needs the session object, for example, when the user has logged out of the site, it should be invalidated. Invalidating a session removes it from the session cache, as well as from the session database. For more information see “Invalidating sessions” on page 327.

Increase available memory
WebSphere allows the administrator to increase an application server’s heap size. By default, WebSphere allocates 256 MB as the maximum heap size.
Increasing this value allows the instance to obtain more memory from the system, and thus hold a larger session cache.

**Session timeout interval**

By default, each user receives a 30-minute interval between requests before the session manager invalidates the user's session. Not every site requires a session timeout interval this generous. By reducing this interval to match the requirements of the average site user, the session manager purges the session from the cache (and the persistent store, if enabled) more quickly.

Avoid setting this parameter too low and frustrating users. The administrator must take into account a reasonable time for an average user to interact with the site (read returned data, fill out forms, and so on) when setting the interval. Also, the interval must represent any increased response time during peak times on the site (such as heavy trading days on a brokerage site, for example).

In some cases where the persistent store contains a large number of session entries, frequent execution of the timeout scanner reduces overall performance. In these cases, avoid setting the session timeout so low it triggers frequent, expensive scans of the persistent store for timed-out sessions. Alternatively, the schedule-based invalidation should be considered where scans for invalid object can be deferred to a time that normally has low demand. See “Invalidating sessions” on page 327 for more information.

**Consider using multi-row session in database persistence**

When a session contains multiple objects accessed by different servlets or JSPs in the same Web application, multi-row session support provides a mechanism for improving performance. See “Single vs. multi-row schemas in database persistence” on page 322 for more information.

Even with multi-row session support, WebSphere applications perform best if the overall contents of the session objects remain small. Large values in session objects require more time to retrieve from the persistent session database, generate more network traffic in transit, and occupy more space in the session cache after retrieval.

Multi-row session support provides a good compromise for Web applications requiring larger sessions. However, single-row persistent session management remains the best choice for Web applications with small session objects. Single-row persistent session management requires less storage in the database, and requires fewer database interactions to retrieve a session's contents (all of the values in the session are written or read in one operation). This keeps the session object's memory footprint small, as well as reduces the network traffic between WebSphere and the persistent session database.
Managing your session database connection pool
When using persistent session management, the session manager interacts with the defined database through a WebSphere Application Server data source. Each data source controls a set of database connections known as a connection pool. By default, the data source opens a pool of no more than 10 connections. The maximum pool size represents the number of simultaneous accesses to the persistent session database available to the session manager.

For high-volume Web sites, the default settings for the persistent session data source may not be sufficient. If the number of concurrent session database accesses exceeds the connection pool size, the data source queues the excess requests until a connection becomes available. Data source queueing can impact the overall performance of the Web application (sometimes dramatically).

See “Tuning WebSphere DataSources” on page 303 to find how to tune the connection pool.

Session database tuning
While the session manager implementation in WebSphere provides for a number of parameters that can be tuned to improve performance of applications that utilize HTTP sessions, maximizing performance will require tuning the underlying session persistence table. WebSphere V5 provides a “first step” in this regard by creating an index for the sessions table when creating the table. The index is comprised of the session ID, the property ID (for multi-row sessions), and the Web application name.

While DB2 UDB provides a great deal of capability in tuning at the table or tablespace level, creation of a separate database or instance will afford the most flexibility in tuning. Proper tuning of the instance/database can improve the performance.

While the specifics will vary depending on the database and operating system in use, in general the database should be tuned and configured as appropriate for a database that experiences a great deal of I/O. The DBA should monitor and tune the database buffer pools, database log size, and write frequency. Additionally, maximizing performance will require striping the database/instance across multiple disk drives and disk controllers, and utilizing any hardware or OS buffering that is available in order to reduce disk contention.
8.3 Enterprise JavaBeans

In this section, we discuss EJB performance considerations as they apply to accessing a DB2 database.

8.3.1 EJB performance considerations

From a database perspective, the following considerations apply.

- EJB 2.0 module
  
  In general, read-only methods and optimistic locking will provide better performance. However, be careful with the use of optimistic locking since problems may only become apparent under heavy load and hence may not be found in development.

  A common mistake is just using the default setting. This might cause performance issues.

- EJB 1.1 module
  
  - Ensure that the Isolation level attributes setting selection is based on the minimal semantic requirements of the application. In most cases, this corresponds to Read Committed on the Java side, which translates to Cursor Stability on the DB2 side.

  - Ensure that the Access Intent is set to Read for all EJB read only methods.

8.4 Application considerations for performance in database access

In this section, we discuss some best practices on database access.

- Using connection pooling
  
  The Java Database Connectivity (JDBC) API provides a vendor-independent mechanism to access relational databases from Java. However, obtaining and closing a connection to a database can be a relatively expensive exercise, so the concept of connection pools has been introduced. When a database operation is to be performed, a connection can be obtained from the pool, which contains a defined number of connections to the database that have already been established. When the connection is closed, it is returned to the pool and made available for reuse. Using connection pooling can significantly reduce the overhead of obtaining a database connection. However, the connection pool is accessed via a data source. References to the data source are obtained by performing a lookup via the Java Naming and Directory Interface (JNDI). This lookup is an expensive operation, so it is good practice to perform the lookup once and cache the result for reuse. Example 8-7 on
page 333 from the EJB sample shipped with DB2 UDB V8.1 shows how to use connection pooling.

Example 8-7  Using connection pooling and PreparedStatement

```java
public DataSource getDataSource() throws SQLException {
    private static DataSource ds = null;

    if (ds == null) {
        try {
            //Create the initial naming context.
            InitialContext ctx = new InitialContext();

            //Perform a naming service lookup to get a DataSource object.
            //The single DataSource object is a "factory" used by all
            //requests to get an individual connection for each request.
            ds = (DataSource) ctx.lookup("java:comp/env/AccessEmp/DataSource");
        }
        catch (NamingException e) {
            throw new EJBException(e);
        }

        return ds;
    }
    else {
        return ds;
    }
}

public ArrayList getEmpNoList() {
    ResultSet rs = null;
    PreparedStatement ps = null;
    ArrayList list = new ArrayList();
    DataSource ds = null;
    Connection con = null;

    try {
        ds = getDataSource();
        con = ds.getConnection();
        String schema = getDbSchema();
```
StringBuffer queryBuf = new StringBuffer("Select EMPNO, FIRSTNME from ");

queryBuf.append(schema);
queryBuf.append(".EMPLOYEE");

ps = con.prepareStatement(queryBuf.toString());
ps.executeQuery();
rs = ps.getResultSet();

...
minimized by using a single statement that returns result set or multiple result sets.
Integrated troubleshooting

This chapter describes some common methods to look into problems that are possibly related to DB2 UDB and/or WebSphere Application Server, including the introduction for diagnostic log files and traces.

Further, we provide a few real-world problem scenarios and methodology to diagnose them and resolve the issues.

Topics covered includes:

- Problem determination methodology
- Dialogistic information collection and analysis
- Scenarios
9.1 Problem determination methodology

The problem determination and performance tuning are closely related topics. It is important to understand the difference between problem determination and performance tuning.

Problem determination is the process of determining the cause of a problem, while performance tuning is the process of changing/adjusting a system or an application to perform in a more efficient way.

We should follow performance tuning routines/guidelines throughout the software development life cycle, right from the beginning of understanding the system requirements, analysis, design, coding and implementation. Irrespective of all the guidelines, no system is perfect. Even though you have designed a perfect system, with the passage of time requirements start changing. Workload on a system increases, and a time comes when applications start giving poor response. This is the point when we initiate the diagnostics process to find out the reasons of performance degradation and take appropriate measures against them.

Basically there are no fixed rules or best guidelines for problem determination. It all depends upon the user experience, kind of environment, and expectations from the system.

Sometimes fixing one problem may lead you into ten new problems, so it is always better to have an understanding of the overall architectural design of the system so that before applying any change you can draw a picture of all the other possible areas that are going to be effected by your corrective measures.

Here we describe the procedure for identifying the problem, analyzing the problem, finding the root cause of the problem, and then taking corrective measures to solve it.

Figure 9-1 on page 339 explains the steps of the problem-determination methodology that we are going to follow in our scenarios.
Below we review the above figure.

1. Try to listen and understand the problem. No action is fruitful without understanding the pains of the end user. Apart from the regular routine-based performance tuning process, generally we come to know about a problem by one of the following ways:
   a. Users starts complaining about poor response times, that connections are not stable, request time-outs, etc.
   b. Alert messages from performance monitoring/diagnostic tools.
   c. System resource monitoring tools start giving threshold indications like page swapping in main memory, CPU consumption, page hits, etc.
   d. Sometimes expectation mismatch occurs when applications are not producing desired results.

2. Once you have a clear understanding of the problem it can be easily translated into objectives to meet. This step is very important because most of the times we start with different objectives and attain something else, which
was actually never desired. Also, in this step you can set an expectation level of the end user. It helps to give a more realistic picture.

3. It is always advisable to document all the steps.

4. You should always have a fallback procedure. If something wrong happened you must be ready with undo scripts.

5. Analyze the problem and areas that are related to it. It’s very much possible to have more than one factor involved in the tuning of a particular problem, or we can have multiple reasons for the same problem.

6. Identify the root cause of the problem. This will help you to isolate the main problem from the rest. The idea of problem determination in a DB2 UDB and WebSphere environment is to determine where the actual problem lies.

7. Once you have identified the main root cause of the problem, apply your best practices and compare the results with the stated objectives in step 2.

8. Problem determination and performance tuning is an iterative process. Just keep on improving things as much as possible. Make sure you are documenting all the procedures you have applied and your fallback scripts are in place.

9.2 Diagnostic information collection and analysis

Diagnostic information collection is fundamental for a problem determination and problem source identification (PD/PSI) process, especially when it is difficult to identify what kind of the problem it is by the superficial phenomena. Generally you could obtain the first-hand information from the diagnostic log files which are commonly available in a mature software products such as DB2 UDB and WebSphere Application Server. If the information from the diagnostic log files is still not sufficient to find out the root cause to resolve the problem, then further information for debugging purposes could be required. For example, taking a trace to obtain the details about the application execution procedure, dumping the stack trace back information to find out the failing function, etc.

Besides the diagnostic information that could be gathered through the existing mechanism built in the products, we could also add user-generated diagnostic information into some applications for a specific intention. There are often times that this kind of supplemental diagnostic information helps a lot to position the problematic code snippet, even in the failing line of the program source code. This kind of technique is most commonly used during the application development stage, and also is adopted popularly in the run-time production environment for many considerate software applications.
Diagnostic information collection is most useful for identifying then resolving functionality issues, but could also do good for performance issue investigation in many situations. For example, you could use diagnostic information to identify which step or steps are contributing the most time for a time-consuming application, then focus on the specific step or steps to find out the underlying reasons of the long execution time, and make adjustments accordingly.

In this section we introduce the available log files for both DB2 UDB and WebSphere Application Server V5. In addition, we also introduce the way to activate the trace function that is built in the products. Furthermore, we also discuss some general methods to analyze the information obtained by the above steps, thereby giving you some clues to of the diagnostic information to resolve the problem.

Note: Operating System (OS) diagnostics also often help for problem determination and performance bottlenecks identification. We do not cover details about how to analyze OS diagnostics in this section. For more information about OS diagnostics, please refer to OS-specific documents.

9.2.1 DB2 UDB V8 diagnostic information collection and analysis

DB2 UDB provides comprehensive information and a variety of methods to assist you in problem determination and root cause identification. You could obtain the problem basic information from the returning code reported by DB2 UDB for your SQL operations or administrative commands, and acquire further details from the diagnostic log files, then taking possibly appropriate actions to resolve the problem.

You can start your investigation from the reported SQLCODE, as DB2 UDB informational messages are always returned in the form of CCCnnnnnS. The CCC identifies the DB2 component returning the message, the nnnnn is a four or five digit error code, and the S is a severity indicator.

The following are some of the SQL component identifiers that you might encounter when using DB2 UDB.

- SQL: Database Manager messages
- DB2: Command Line Processor messages
- CLI: Call Level Interface messages
- DBA: Control Center and Database Administration Utility messages
- SQJ: Embedded SQLJ in Java messages
The easiest way to get more details regarding the DB2 return code is to use the DB2 command, as below:

Kanaga:/home/db2inst1 >db2 "? SQL0289"

In general, the output of the above command would return information including the basic meaning of the DB2 return code, the explanation about why the code comes out, and the recommended user response if it is not just an informational return code.

Besides the investigation on the returning code, DB2 UDB also has the built-in First-failure data capture (FFDC) mechanism which is very helpful for the problem troubleshooting. FFDC is a general term applied to the set of diagnostic information that DB2 captures automatically when errors occur. This information reduces the need to reproduce errors to get diagnostic information. The DIAGPATH parameter, specified in the database manager configuration, gives the fully qualified path to the FFDC storage directory. The DIAGLEVEL and NOTIFYLEVEL configuration parameters control the detail of information you receive in the logs.

The information captured by FFDC includes the following.

**Administration Notification Logs**

When significant events occur, DB2 writes information to the administration notification log. The information is intended for use by database and system administrators. Many notification messages provide additional information to supplement the SQLCODE that is provided. The type of event and the level of detail of the information gathered are determined by the NOTIFYLEVEL configuration parameter.

**db2diag.log**

Diagnostic information about errors is recorded in this text log file. This information is used for problem determination and is intended for DB2 customer support. The level of detail of the information is determined by the DIAGLEVEL configuration parameter.

**Dump files**

For some error conditions, extra information is logged in external binary files named after the failing process ID. These files are intended for use by DB2 customer support.

**Trap files**

The database manager generates a trap file if a DB2 process receives a signal or exception (raised by the operating system as a result of a system event) that is
recognized by the DB2 signal handler. A trap file is generated in the DB2 diagnostic directory.

**Core files (UNIX only)**

When DB2 terminates abnormally, the operating system generates a core file. The core file is a binary file that contains information similar to the DB2 trap files. Core files may also contain the entire memory image of the terminated process.

**Utilizing Messages files:** Some DB2 utilities like BIND, LOAD, EXPORT, and IMPORT provide an option to dump out a messages file to a user-defined location. These files contain useful information to report the progress, success or failure of the utility that was run. You should take advantage of generating these files to ensure you are obtaining the most possible information in case of a problem.

Within the above files, the diagnostic log file `db2diag.log` is the most important and most often used for problem determination. Trap files, dump files and core files are generally required by DB2 support staff under complicated problem troubleshooting situations. If you are interested in the greater details about these files, such as the naming convention and how to generate these files manually, refer to the DB2 Information Center, which is reachable at the URL below:

http://publib.boulder.ibm.com/infocenter/db2help/index.jsp

DB2 UDB V8 also provides a very useful tool named `db2support` to help you collect diagnostic information such as the files described above, configuration files, DB2 product version information, operating system information, and so forth. We discuss this utility in more detail later in this section.

In addition to the FFDC information listed above, DB2 UDB also provides a trace facility that allows you to obtain further details about the process runtime information for debugging purposes.

The following provides more information about the `db2diag.log` file, `db2support` utility and DB2 traces.

**DB2 diagnostic log file db2diag.log**

This file is commonly used in complex problem determination scenarios. You can find this file in the DB2 diagnostic directory, defined by the DIAGPATH parameter in the database manager configuration. By default the directory is defined as follows:

UNIX: $HOME/sqlib/db2dump
Here $HOME is the DB2 instance owner's home directory. For example, if your instance owner is db2inst1 and its home directory is /home/db2inst1, the default location for the db2diag.log should be /home/db2inst1/sqlib/db2dump.

Windows: <INSTALL PATH>\SQLLIB\<DB2INSTANCE>

Here INSTALL PATH represents the directory where DB2 is installed. For example, a default db2diag.log for an instance owner of DB2 on Windows would exist in C:\Program Files\SQLLIB\DB2\db2diag.log.

The database manager configuration also controls how much information is logged to the db2diag.log through the use of the diagnostic level, or DIAGLEVEL parameter. Valid values can range from 0 to 4, as shown below:

- 0 - No messages
- 1 - Severe error messages
- 2 - Only error messages
- 3 - All error and warning messages (default)
- 4 - All error, warning, informational, and internal diagnostic messages

The default diagnostic level of 3 is usually sufficient for problem determination. Setting it to 4 may cause performance issues due to the large amount of data recorded into the file. You should adjust the setting according to the problem you encounter, the amount of data you require to investigate, and the runtime environment where the problem occurs.

The following shows you an example of db2diag.log. For demonstration purposes, we try to create a table space on a non-existent device on AIX platform as below:

db2 "create tablespace aaa managed by database using(device '/dev/raaa' 50M)"

Then the following error messages are written to the db2diag.log, as shown in the Example 9-1.

Example 9-1   DB2 diagnostic log file db2diag.log
2003-11-17-14.24.04.887990   Instance:db2inst1   Node:000
PID:49060(db2agent (SAMPLE) 0)   TID:1   Appid:*LOCAL.db2inst1.0ADF97222248
oper system services   sqloopenp   Probe:20   Database:SAMPLE
errno:
0x2FF189D0 : 0x0000000D
....

PID:49060 TID:1 Node:000 Title: Path/Filename
/dev/raaa
DIA8701C Access denied for resource "", operating system return code was "". ZRC=0x840F0001

Error acquiring container 0 (/dev/raaa) for tbsp 3. Rc = 840F0001

From the above diagnostic log file snippet, it is not difficult to find out the access denial to the device is the reason that creating tablespace failed. It also contains information about when the message was generated, the instance name, partition number, process name and ID, thread ID, application ID, related DB2 UDB components, function and internal probe, database name, and so forth.

**Diagnostic information collection utility db2support**

When it comes to collecting information for a DB2 problem, the most important DB2 utility you need to run is **db2support**. This utility is designed to automatically collect all related DB2 UDB and system diagnostic information available (including information described in previous pages). It has an optional interactive “Question and Answer” session available to help collect information for problems that you may want additional assistance investigating. Using **db2support** avoids possible user errors, as you do not need to manually type commands such as `get dbm cfg` or `list history all for <db_name>`. Also, you do not require instructions on what commands to run or what files to collect, which makes information gathering for problem determination quicker.

This command has been in the DB2 product on Linux, OS/2®, Windows, and UNIX, since version 7 Fix Pack 4, and is continually being enhanced. Executing **db2support -h** brings up the complete list of possible options you can run the utility with. The following basic invocation is usually sufficient for collecting most of the information required to debug a problem (note that if the -c option is used the utility will establish a connection to the database):

```
db2support <output path> -d <database name> -c -g -s
```

If further information is required, you need to review which extra options could be used to help. The output is conveniently collected and stored in a compressed ZIP archive, **db2support.zip**, so it can be transferred and extracted easily on any system. Example 9-2 on page 346 shows you an example of using **db2support** utility.
Example 9-2 Using db2support to collect diagnostic information

Kanaga:/home/db2inst1 >mkdir log
Kanaga:/home/db2inst1 >db2support ./log -d sample -c -g -s

This program generates information about a DB2 server, including information about its configuration and system environment. The output of this program will be stored in a file named 'db2support.zip', located in the directory specified on the application command line. If you are experiencing problems with DB2, it may help if this program is run while the problem is occurring.

... Output file is "/home/db2inst1/log/db2support.zip"
Time and date of this collection: "Mon Nov 17 14:58:06 PST 2003 PST"
... 
... 
db2support is now complete.
An archive file has been produced: "db2support.zip"

Please be aware that the information provided above is incomplete and more details are removed as it is just for demonstration purpose.

DB2 traces

If the first failure data captured is insufficient to diagnose the problem, and if the problem you are experiencing is recurring or reproducible, then taking DB2 traces sometimes allows you to capture additional information.

In general, as there is additional processing incurred by activating the trace and the amount of information gathered by a trace grows rapidly, the process of performing a trace has a global effect on the behavior of a DB2 instance. The degree of performance degradation is dependent on the type of problem and on how many resources are being used to gather the trace information. When you take the trace, capture only the error situation and avoid any other activities whenever possible, if the trace facility supports that. When taking a trace, use the smallest scenario possible to reproduce a problem, as it could also reduce the impact to the performance.

DB2 UDB V8 provides different trace solutions for different problem situations. Some of the available DB2 traces are listed below:

- db2trc
  The db2trc facility lets you trace DB2 internal events and records information about operations, dumps the trace data to a file, and formats the information into a readable form.
Typically you will use the trace facility only when directed by DB2 Customer Support or by your technical support representative.

- **GUI trace**
  It is helpful for GUI tools problem determination. For example, you could use `db2cctrc` command to do a trace for the DB2 Control Center.

- **db2drdat**
  This allows the user to capture the DRDA data stream exchanged between a DRDA Application Requestor (AR) and the DB2 UDB DRDA Application Server (AS). Although this tool is most often used for problem determination, by determining how many sends and receives are required to execute an application, it can also be used for performance tuning in a client/server environment.

- **CLI trace**
  The DB2 CLI and ODBC drivers offer comprehensive tracing facilities. By default, these facilities are disabled and use no additional computing resources. When enabled, the trace facilities generate one or more text log files whenever an application accesses the appropriate driver. This trace facility is also helpful when using DB2 Legacy JDBC Drivers, as the CLI layer is also involved in that case.

- **asntrc**
  This trace facility assists you in the troubleshooting of replication related problems. It logs program flow information from Capture, Apply, and Replication Alert Monitor programs.

- **JDBC Trace**
  DB2 JDBC Drivers offer comprehensive tracing facilities, and it has been continually enhanced, especially with the introduction of the DB2 Universal JDBC Driver. The activation method for the DB2 Legacy JDBC Driver and DB2 Universal JDBC Driver is different. For DB2 Legacy JDBC Driver, as the CLI layer is involved, it could be activated by updating CLI configuration, via the UPDATE CLI CFG command or edit `db2cli.ini` file directly. Example 9-3 shows you an example of using the JDBC trace for the legacy JDBC driver.

**Example 9-3   Using JDBC Trace for the Legacy JDBC Driver**

```
Kanaga:/home/db2inst1 > db2 update cli cfg for section common using JDBCTrace 1
JDBCTracePathName /home/db2inst1/jdbc/trc JDBCFlush 1
Kanaga:/home/db2inst1 >db2 get cli cfg for section common
```

```
Section: common
--------------------------------------------------
JDBCFlush=1
JDBCTracePathName=/home/db2inst1/jdbc/trc
```
/* Running your java application which uses the legacy DB2 JDBC Driver, the sample application shown below is TbRead which is shipped with DB2 UDB V8, it could be found under $HOME/sqlib/samples/java directory. */

Kanaga:/home/db2inst1/jdbc > java TbRead
Kanaga:/home/db2inst1/jdbc > cd trc
Kanaga:/home/db2inst1/jdbc/trc > ls -l
  total 1289
  -rw-r--r--  1 db2inst1 db2grp1        8269 Nov 19 09:30 44182_1_Finalizer.trc
  -rw-r--r--  1 db2inst1 db2grp1      648987 Nov 19 09:30 44182_1_main.trc

Kanaga:/home/db2inst1/jdbc/trc > head -n 40 44182_1_main.trc
  | Loaded db2jdbc from java.library.path
  | DB2Driver: JDBC 2.0, BuildLevel: s031027
jdbc.app.DB2Driver <- DB2Driver() [Time Elapsed = 0.0020] (2003-11-19 09:30:28.297)

DB2Driver - connect(jdbc:db2:sample)

jdbc.app.DB2Connection -> connect( sample, info, DB2Driver: JDBC 2.0 s031027, 0, false ) (2003-11-19 09:30:28.3)
  | 10: conArg =
  | 10: connectionHandle = 1

  | 10: Connection handle = 1
jdbc.app.DB2Connection <- setAutoCommit2() returns 0 [Time Elapsed = 0.039] (2003-11-19 09:30:28.955)

  | | jdbc.app.DB2Statement <- checkResultSetType() [Time Elapsed = 0.0] (2003-11-19 09:30:28.956)
  | | 10: Peak statements = 1
  | | 10: Statement Handle = 1:1
  | jdbc.app.DB2Statement <- DB2Statement() [Time Elapsed = 0.0] (2003-11-19 09:30:28.956)
| 10: Statement Handle = 1:1
| jdbc.app.DB2Statement <- getStatementType() returns STMT_TYPE_QUERY (24)
[Time Elapsed = 0.0010] (2003-11-19 09:30:29.025)
| | 10: StatementHandle  = 1:1
| | 10: SQLExecDirect - returnCode  = 0
| | 10: rowCount = 0
| jdbc.app.DB2Statement <- execute2() [Time Elapsed = 0.013] (2003-11-19 09:30:29.038)
| jdbc.app.DB2Statement -> getResultSet() (2003-11-19 09:30:29.038)
| | 10: Statement Handle = 1:1
| | jdbc.app.DB2ResultSetTrace -> DB2ResultSet( stmt, nCols=0) (2003-11-19 09:30:29.047)
| | | 10: numCols = 2
| | jdbc.app.DB2ResultSetTrace <- DB2ResultSet() [Time Elapsed = 0.0] (2003-11-19 09:30:29.047)
| | jdbc.app.DB2ResultSetTrace -> DB2ResultSetTrace( stmt,0 ) (2003-11-19 9:30:29.47)
...

From the above example, it is not difficult to find out the details about the driver you are using, the time spent to obtain the connection, the SQL statement executed in the context, etc. If you encounter problems when using the Legacy DB2 JDBC Driver, the JDBC trace is generally very helpful to point out where the problem exists.

Due to the importance of DB2 Universal JDBC Driver, we discuss it in a separate subsection as below.

**DB2 Universal JDBC Driver JDBC Trace**

Before jumping into the JDBC trace, please make sure that your DB2 Universal JDBC Driver is correctly installed. Regarding how to install the DB2 Universal JDBC Driver, please refer to the topic *Installing the DB2 Universal JDBC Driver in DB2 Information Center* via the URL below:

http://publib.boulder.ibm.com/infocenter/db2help/index.jsp
In addition, please also make sure the right data source class is chosen for your runtime environment. The DB2 Universal JDBC Driver provides the following DataSource implementations:

- **com.ibm.db2.jcc.DB2SimpleDataSource**
  This implementation does not support connection pooling. You can use this implementation with Universal Type 2 Driver or Universal Type 4 Driver.

- **com.ibm.db2.jcc.DB2DataSource**
  This implementation supports connection pooling. You can use this implementation only with Universal Type 2 Driver. With this implementation, connection pooling is handled internally and is transparent to the application.

- **com.ibm.db2.jcc.DB2ConnectionPoolDataSource**
  This implementation supports connection pooling. You can use this implementation with both Universal Type 2 Driver and Universal Type 4 Driver, but XA is not supported. It is the factory for PooledConnection objects. An object that implements this interface will typically be registered with a naming service that is based on the Java Naming and Directory Interface (JNDI). With this implementation, you must manage the connection pooling yourself, either by writing your own code or by using a tool such as WebSphere Application Server.

- **com.ibm.db2.jcc.DB2XADDataSource**
  This implementation supports distributed transactions and connection pooling. You can use this implementation with Universal Type 2 Driver and XA support, but be aware that Universal Type 4 Driver is not supported by this implementation. With this implementation, you must manage the distributed transactions and connection pooling yourself, either by writing your own code or by using a tool such as WebSphere Application Server.

**Note:** The com.ibm.db2.jcc.DB2BaseDataSource class is the abstract data source parent class for all the DB2 DataSource implementations of the DB2 Universal JDBC Driver discussed above.

You can also use the DriverManager to get the connection of the DB2 Universal JDBC Driver. The related class is com.ibm.db2.jcc.DB2Driver. Using DriverManager to connect to a data source reduces portability because the application must identify a specific JDBC driver class name and driver URL. The driver class name and driver URL are specific to a JDBC vendor and driver implementation. If your applications need to be portable among data sources, it is highly recommended to use of the DataSource interface.
Start trace when using DataSource interface to get the connection

If the DataSource interface is used to connect to a data source, use one of the following methods to start the trace:

- Method 1: Invoke the `DB2BaseDataSource.setTraceLevel` method to set the type of tracing that you need. The default trace level is TRACE_ALL. Then invoke the `DB2BaseDataSource.setJccLogWriter` method to specify the trace destination and turn the trace on.

- Method 2: Invoke the `javax.sql.DataSource.setLogWriter` method to turn the trace on. With this method, TRACE_ALL is the only available trace level.

After a connection is established, you can turn the trace off or back on, change the trace destination, or change the trace level with the `DB2Connection.setJccLogWriter` method. To turn the trace off, 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 resides must be writable. If the file already exists, the driver overwrites it.

Start trace for using DriverManager interface to get the connection

If the DriverManager interface is used to connect to a data source, use the following method to start the trace: Invoke the `DriverManager.getConnection` method with the traceLevel property set in the `info` parameter or `url` parameter for the type of tracing that you need. The default trace level is TRACE_ALL. Then invoke the `DriverManager.setLogWriter` method to specify the trace destination and turn the trace on.

There is an example available in the section *Example of tracing under the DB2 Universal JDBC Driver* in DB2 Information Center. This example shows you how to program to take a JDBC trace when using the DB2 Universal JDBC Driver.

When using the DataSource interface in the WebSphere Application Server V5 environment, it is very convenient and easy for you to configure the trace properties in the WebSphere Application Server Administrative Console. The trace properties are part of the custom properties for the DB2 data source. Regarding how to configure custom properties for DB2 data source in WebSphere Application Server Administrative Console, refer to “The steps to
create and configure DB2 Data Source” on page 144. The following shows you the related parameters to configure the DB2 Universal JDBC Driver trace.

- **traceFile**

  Specifies the name of a file into which the DB2 Universal JDBC Driver 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.

- **traceLevel**

  Specifies what to trace. The data type of this property is int. You can specify one or more of the following traces with the traceLevel property.

  - TRACE_NONE = 0
  - TRACE_CONNECTION_CALLS = 1
  - TRACE_STATEMENT_CALLS = 2
  - TRACE_RESULT_SET_CALLS = 4
  - TRACE_DRIVER_CONFIGURATION = 16
  - TRACE_CONNECTS = 32
  - TRACE_DRDA_FLOWS = 64
  - TRACE_RESULT_SET_META_DATA = 128
  - TRACE_PARAMETER_META_DATA = 256
  - TRACE_DIAGNOSTICS = 512
  - TRACE_SQLJ = 1024
  - TRACE_ALL = -1
  - TRACE_XA_CALLS (Universal Type 2 Driver for DB2 UDB for Linux, UNIX and Windows only)

After configuring the trace properties for the DB2 data source, you could try to test the connection to the DB2 data source in the WebSphere Application Server Administrative Console. Check the file that is set by traceFile property. You could find the content corresponding to the levels set by traceLevel is written into that file. For a detailed tracing example, which is using DB2 Universal JDBC Driver trace in the WebSphere Application Server V5 environment, please refer to “Connectivity scenario” on page 359.

If you want to acquire more information about other kinds of traces available in DB2 UDB V8, such as db2trc, CLI trace, and so forth, DB2 Information Center is a very good resource for that.
Besides the log files and traces investigation for the problem determination, SQL statement access plan analysis is also very useful to identify performance bottlenecks. For example, you could use event monitor to find out the typical or resource consuming SQL statements, then use SQL Explain utilities to do further analysis against specific SQL statements. Regarding more information about DB2 SQL Explain Facilities, refer to the Chapter 7, “SQL Explain facility,” in *Administration Guide: Performance, IBM DB2 Universal Database, Version 8, SC09-4821*.

### 9.2.2 WAS V5 diagnostic information collection and analysis

Generally a typical J2EE application runtime environment consists of multiple tiers such as Web server, application server, and database servers, also including the communication between different tiers by using different components such as Web server plug-in, resource adapter, etc. As the J2EE runtime platform, WebSphere Application Server troubleshooting or problem determination encompasses a wide range of tasks that might need to be performed at any tier or any component in the runtime environment. To assist you in identifying in which component or tier the problem exists, WebSphere Application Server provides a variety of diagnostic logs and tools to make the problem determination easier.

**Note:** As the IBM HTTP Server (IHS) commonly coexists with the WebSphere Application Server, we also discuss diagnostic information analysis about the IHS in this section.

As a starting point to resolve functional problems, we begin this section with components availability verification, so that we could gain a basic understanding if all the related tiers or components are functioning normally. After that we introduce the basic diagnostic information available in the WebSphere Application Server environment and some of the useful tools for problem determination.

**Components availability verification**

It is possible that the problem that you are studying is not derived from application server or its containers. For example, when you make an access request to an entity EJB, if the supporting database resources are not ready at that moment, your access request would fail. Or you might find that the page you requested could not be found when your Web server plug-in is not successfully loaded with the Web server process. Make sure that all the related tiers or components functioning normally is the basis for the deeper level problem troubleshooting.
Table 9-1 gives you some simple sample methods to verify the components availability. Please be aware that it is possible that the methods provided here are not working in your environment if you have changed the default configuration, or make no certain that the component is working normally. For both cases, you could have environment-specific methods to verify the component availability.

Table 9-1  Simple methods for component availability verification

<table>
<thead>
<tr>
<th>Component</th>
<th>Function briefing</th>
<th>Simple methods for availability verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Balancer</td>
<td>Providing intelligent workload dispatching between backend Web servers</td>
<td>Ping related IP address and cluster address.</td>
</tr>
<tr>
<td>IBM HTTP Server</td>
<td>Taking the role of Web server</td>
<td>Access Web Server with default configuration on <a href="HTTP://ipaddress">HTTP://ipaddress</a> or <a href="HTTP://localhost">HTTP://localhost</a> if local.</td>
</tr>
<tr>
<td>Web Server Plug-In for WebSphere Application Server</td>
<td>Routing requests from Web server to application server</td>
<td>Check plug-in log file, http_plugin.log by default.</td>
</tr>
<tr>
<td>WebSphere Application Server</td>
<td>Providing runtime platform for J2EE applications</td>
<td>Check log files and traces; more details cover later.</td>
</tr>
<tr>
<td>Web Container</td>
<td>Providing runtime environment for application Web module, part of WAS</td>
<td>Check HTTP Transport port availability, by default, 9080. Use netstat to see if it is in the listening status.</td>
</tr>
<tr>
<td>EJB Container</td>
<td>Providing runtime environment for application EJB module, part of WAS</td>
<td>Use simplified Java client to access EJB.</td>
</tr>
<tr>
<td>DB2 UDB</td>
<td>Taking the role as database</td>
<td>Use DB2 Command Line Processor to access related tables.</td>
</tr>
</tbody>
</table>

**Diagnostic information analysis**

WebSphere Application Server provides comprehensive diagnostic information to assist you in problem determination. For example, the console messages available in the WebSphere Status pane within the WebSphere Application Server Administrative Console could provide you the runtime messages and help you determine the WebSphere configuration problems. WebSphere Application Server also provides you with general purpose logs, such as the JVM logs, the process logs and the IBM service logs. In addition, in order to obtain more details
about the component runtime information, the diagnostic trace is also available for your use. You could configure these log files and trace in the WebSphere Application Server Administrative Console, as shown in Figure 9-2.

Figure 9-2  Logging and tracing configuration in WAS Administrative Console

Furthermore, there also exists some log files for IBM HTTP Server and the Web server plug-in, which can help you determine if the problem is related to IHS or the Web server plug-in.

The following provides more details about these logs and trace.

**JVM logs**

The JVM logs are created by redirecting the System.out and System.err streams of the JVM to independent log files. WebSphere Application Server writes formatted messages to the System.out stream. In addition, applications and other code can write to these streams using the print() and println() methods defined by the streams. Some JDK built-ins such as the printStackTrace() method on the Throwable class can also write to these streams. Typically, the System.out log is used to monitor the health of the running application server. The System.out log can also be used for problem determination, but it is recommended to use the IBM Service log and the advanced capabilities of the Log Analyzer instead. The System.err log contains exception stack trace information that is useful when performing problem analysis.

Since each application server represents a JVM, there is one set of JVM logs for each application server and all of its applications, located by default in the
installation_root/logs/server_name directory. In the case of a WebSphere Application Server Network Deployment configuration, JVM logs are also created for the deployment manager and each node manager, since they also represent JVMs. The default setting for the JVM logs is listed below:

System.out Stream: ${SERVER_LOG_ROOT}/SystemOut.log
System.err Stream: ${SERVER_LOG_ROOT}/SystemErr.log

See the WebSphere Variables page in the WebSphere Application Server Administrative Console for the definition of SERVER_LOG_ROOT. By default, it is the directory installation_root/logs/server_name.

**Process logs**
The process logs are created by redirecting the stdout and stderr streams of the process to independent log files. Native code, including the Java Virtual Machine (JVM) itself writes to these files. As a general rule, WebSphere Application Server does not write to these files. However, these logs can contain information relating to problems in native code or diagnostic information written by the JVM.

As with JVM logs, there is a set of process logs for each application server, since each JVM is an operating system process, and in the case of a WebSphere Application Server Network Deployment configuration, there is also a set of process logs for the deployment manager and each node manager. The default setting for the process logs is listed below:

Stdout File Name: ${SERVER_LOG_ROOT}/native_stdout.log
Stderr File Name: ${SERVER_LOG_ROOT}/native_stderr.log

You could refer to the JVM logs above for the SERVER_LOG_ROOT definition.

**IBM Service logs**
The IBM service logs contain both the WebSphere Application Server messages that are written to the System.out stream and some special messages that contain extended service information that is important when analyzing complex problems. There is one service log for all WebSphere Application Server JVMs on a node, including all application servers. The IBM Service log is maintained in a binary format and requires a special tool to view. This viewer, the Log Analyzer, provides additional diagnostic capabilities. The default setting for the IBM service logs is listed below:

File Name: ${LOG_ROOT}/activity.log

See the WebSphere Variables page in the WebSphere Application Server Administrative Console for the definition of LOG_ROOT. By default, it is the directory installation_root/logs.
Diagnostic trace

You could use trace to obtain detailed information about the execution of WebSphere Application Server components, including application servers, clients, and other processes in the environment. Trace files show the time and sequence of methods called by WebSphere Application Server base classes, and you can use these files to pinpoint the failure. The default setting of the trace output is stored in `${SERVER_LOG_ROOT}/trace.log`. Trace details could be found in this trace file after the trace is activated. You could refer to the JVM logs above for the SERVER_LOG_ROOT definition.

Attention: Tracing is very demanding on system resources. Remember to turn the trace off once you have finished the trace task.

IBM HTTP Server logs

IBM HTTP Server (IHS) maintains log files to help you monitor fulfilled requests and encountered errors. You could use IHS Administration Server to configure the setting log files or changing httpd.conf directly if you are familiar with that. By default, the log files of IHS on the Windows platform can be found at:

Access Log File: `<IHS_INSTALLATION_ROOT>/logs/access.log`  
Error Log File: `<IHS_INSTALLATION_ROOT>/logs/error.log`

For Linux and UNIX-based platforms:

Access Log File: `<IHS_INSTALLATION_ROOT>/logs/access_log`  
Error Log File: `<IHS_INSTALLATION_ROOT>/logs/error_log`

Here IHS_INSTALLATION_ROOT represents where the IHS is installed.

Web Server Plug-in logs

If you are having problems with the HTTP plug-in component (the component that sends requests from your HTTP server to the WebSphere Application Server), you could find some clues by reviewing the plug-in log file. By default, it is located in install_dir/logs/http_plugin.log. Try to look up any error or warning messages in the message table for possible cause of the problem. You can also change the LogLevel for the plug-in log to a higher level, for example, Trace, to obtain further details about the plug-in operation. This could be configured in the file installation_root/config/plugin-cfg.xml (you could check HttpServer/conf/httpd.conf for the real plug-in configuration file location).

Besides the log files and traces discussed above, WebSphere Application Server V5 also provides a variety of other log files. For example, the StartServer.log and StopServer.log located under `<WAS_INSTALL_ROOT>/logs/<server_name>` (here the variable WAS_INSTALL_ROOT represents the home directory where
WAS is installed) could help you acquire information about when the application server is started or stopped, and if the starting or stopping activity is successful. Another example, the First Failure Data Capture (FFDC) tool preserves the information generated from a processing failure and returns control to the affected engines. The captured data by FFDC is saved under the directory WAS_INSTALL_ROOT>/logs/ffdc and it is intended primarily for use by IBM service personnel. For a more complete description about WebSphere Application Server message logs and trace, please refer the section Monitoring and Troubleshooting in the WebSphere Application Server InfoCenter.

Using troubleshooting tools
There are a number of troubleshooting tools bundled with the WebSphere Application Server product. These tools are designed to help you isolate the source of problems. Some of these tools are discussed below.

Log Analyzer
The Log Analyzer takes one or more service or activity logs, merges all of the data, and displays the entries. Based on its symptom database, the tool analyzes and interprets the event or error conditions in the log entries to help you diagnose problems. Log Analyzer has a special feature enabling it to download the latest symptom database from the IBM Web site. Besides using Log Analyzer to view the service or activity logs, WebSphere Application Server diagnostic trace output could also be dumped in the Log Analyzer format, then you could use the Log Analyzer to help you analyze the trace. The Log Analyzer could be invoked by the command waslogbr.bat on Windows systems or waslogbr on UNIX systems.

In case of the absence of a graphical interface to use the Log Analyzer, the service or activity logs could be viewed via an alternate tool, showlog. This utility could help you dump the service or activity log into a file or stdout in text format.

Collector
The Collector tool gathers information about your WebSphere Application Server installation and packages it in a Java archive (JAR) file that assists you in determining and analyzing the problem. You can also send it to IBM Customer Support for further help when requested. Information in the JAR file includes logs, property files, configuration files, operating system and Java data, and the presence and level of each software prerequisite.

For Linux and UNIX-based platforms, the collector tool could be invoked by the collector.sh command. For Windows platforms, the corresponding command is collector.bat.
WebSphere Application Server products include an enhancement to the Collector tool beginning with Version 5.0.2, known as the collector summary option. Run the Collector tool with the -Summary option to produce a lightweight text file. You can use the collector summary option to retrieve basic configuration and prerequisite software level information when starting a conversation with IBM Support.

In addition to the tools introduced above, there are still a variety of tools available in WebSphere Application Server to help you get more information for problem analysis and performance monitoring and tuning. For example, you could use the name space dump utility (dumpNameSpace) to dump the contents of a name space accessed through a name server, or use Tivoli Performance Viewer to monitor the current application server running status, etc. For more information, you could refer to the WebSphere Application Server InfoCenter.

9.3 Problem determination scenarios

As we have already covered the basic tuning guidelines for DB2 UDB and WebSphere in previous chapters, here we are going to apply those skills and methodologies to solve various problems using scenarios.

9.3.1 Connectivity scenario

In Chapter 5, “Operational setup” on page 117, we demonstrate the WebSphere Application Server and DB2 UDB system setup using the DB2 EJB Sample application AccessEmployee. We install and configure WebSphere Application Server in the Kanaga machine. The database server is in Atlantic, which has DB2 UDB V8 running and Sample database created. We create the DB2 data source named jdbc/Sample, which uses the DB2 Universal JDBC Driver Type 2 Driver to connect to the sample database. During the setup for the DB2 data source, we associated the DB2 data source with a J2C Authentication Data, which represents the combination of the user db2inst1 and its corresponding password. For more details about the above description refer to “The steps to create and configure DB2 Data Source” on page 144.

The db2inst1 user is the instance owner, also a member of the system administrator group of the instance. This user has the upmost authority of the instance, and could do almost anything against the instance. Based on the security consideration, you are recommended to not use the system administrator authority in your database client application, but use a general user with the appropriate authority.

In this subsection, a new user named db2itso is created. This user does not belong to any system groups such as SYSADM, SYSCTRL and SYSMaint of
the DB2 UDB instance. We use this user for the AccessEmployee application to connect to the sample database to replace the original db2inst1 user. The basic steps are as below:

1. Create new user and grant appropriate authority.
2. Create new J2C Authentication Data and use it for AccessEmployee application.
3. Test and troubleshoot runtime issues.

The following shows you the details of the above steps.

**Create new user and grant appropriate authority**

In our example, we create new user db2itso and grant appropriate database access authority using the following steps:

1. Use ksh via the SMIT utility on AIX to create a db2itso user ID belongs to the group db2ugrp.
2. Add the following line to the .profile of db2itso user to initialize the DB2 environment:
   
   ```
   . /home/db2inst1/sqllib/db2profile
   ```
3. Use the db2inst1 user to revoke the database connect authority from the public group.
4. Grant the database connect authority to the db2itso user.
5. Establish the connection to the SAMPLE database by using the db2itso user, as shown in the Example 9-4.

**Example 9-4  New user creating and authority changing**

```
$ db2 connect to sample
$ db2 "revoke connect on database from public"
$ db2 "grant connect on database to db2itso"
$ db2 "select char(grantor,10) grantor,char(grantee,10) grantee, dbadmauth, connectauth, granteetype, securityadmauth from sysibm.sysdbauth"

<table>
<thead>
<tr>
<th>GRANTOR</th>
<th>GRANTEE</th>
<th>DBADMAUTH</th>
<th>CONNECTAUTH</th>
<th>GRANTEETYPE</th>
<th>SECURITYADMAUTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSIBM</td>
<td>DB2INST1</td>
<td>Y</td>
<td>Y</td>
<td>U</td>
<td>N</td>
</tr>
<tr>
<td>SYSIBM</td>
<td>PUBLIC</td>
<td>N</td>
<td>N</td>
<td>G</td>
<td>N</td>
</tr>
<tr>
<td>DB2INST1</td>
<td>PUBLIC</td>
<td>N</td>
<td>N</td>
<td>G</td>
<td>N</td>
</tr>
<tr>
<td>DB2INST1</td>
<td>DB2ITSO</td>
<td>N</td>
<td>Y</td>
<td>U</td>
<td>N</td>
</tr>
</tbody>
</table>
```

4 record(s) selected.

```$ db2 "grant select,insert,update,delete on table db2inst1.employee to user db2itso"
DB20000I  The SQL command completed successfully.```
$ db2 "grant select,insert,update,delete on table db2inst1.department to user db2itso"
DB20000I  The SQL command completed successfully.
$ db2 connect to sample user db2itso
Enter current password for db2itso:

Database Connection Information

Database server        = DB2/6000 8.1.3
SQL authorization ID   = DB2ITSO
Local database alias   = SAMPLE

Create and use new J2C Authentication Data
We create the new J2C Authentication Data with the properties shown in Figure 9-3. The steps for J2C Authentication Data creation are the same as the steps described in step 1 in “The steps to create and configure DB2 Data Source” on page 144.

![Figure 9-3 Creating New J2C Authentication Data for AccessEmployee](image)

Then we modify the property “Container-managed Authentication Alias” for the existing data source jdbc/Sample to utilize the new J2C entry:

1. From the WebSphere Application Server Administrative Console, choose Resource -> JDBC Providers -> My DB2 Universal JDBC Provider -> Data Sources.
2. Click **EJBSample**, then modify the property “Container-managed Authentication Alias” to “kanaga/ITSODB2AccessEmp”.

Be aware that the server may need to be restarted for these changes to take effect. After restarting the WebSphere application server server1, try to test the
connection to jdbc/Sample using the steps described in step 5 of “The steps to create and configure DB2 Data Source” on page 144. The connection should be established successfully at this point.

**Testing and troubleshooting runtime issues**

There is no need to reinstall the AccessEmployee application as we only changed the J2C Authentication Data. We could begin the testing via accessing the following URL:


On the resulting page, try to click **Display Employee List**, Employee List is displayed in another page successfully. But when clicking Query Employee button or Submit Query button, it would fail with error messages like below.

*Example 9-5  Error messages from query employee running*

```
Error: Cannot retrieve the employee's info with emp No.: 000010

StackTrace: RemoteException occurred in server thread; nested exception is:
com.ibm.websphere.csi.CSITransactionRolledbackException: Global tx rolled back
```

The major difference between Display Employee List and other buttons is that Display Employee List does not use the EJB, whereas other buttons use EJB to interact with the data. From the error messages above, it is difficult to tell what is the underlying reason for the problem. Let us begin the troubleshooting from the System.out stream log. In our testing it is:

/usr/WebSphere/AppServer/logs/server1/SystemOut.log

Within the SystemOut.log, you could find the following snippets, as shown in Example 9-6.

*Example 9-6  SystemOut.log Investigation*

**PART 1:**

```
************** Start Display Current Environment **************
WebSphere Platform 5.0 [BASE 5.0.2 ptf2M0325.01] running with process name
kanaga\kanaga\server1 and process id 51268
Host Operating System is AIX, version 5.1
Java version = J2RE 1.3.1 IBM AIX build ca131-20030618 (JIT enabled: jitm), Java Compiler =
jitm, Java VM name = Classic VM
was.install.root = /usr/WebSphere/AppServer
user.install.root = /usr/WebSphere/AppServer
Java Home = /usr/WebSphere/AppServer/java/jre
ws.ext.dirs =
/usr/WebSphere/AppServer/java/lib:/usr/WebSphere/AppServer/classes:/usr/WebSphere/AppServer/clas
```
sses:/usr/WebSphere/AppServer/lib:/usr/WebSphere/AppServer/lib/ext:/usr/WebSphere/AppServer/web
/help:/usr/WebSphere/AppServer/deploytool/itp/plugins/com.ibm.etools.ejbdeploy/runtime
Classpath =
/usr/WebSphere/AppServer/properties:/usr/WebSphere/AppServer/properties:/usr/WebSphere/AppServe
r/lib/bootstrap.jar:/usr/WebSphere/AppServer/lib/j2ee.jar:/usr/WebSphere/AppServer/lib/lmproxy.ja
...

PART 2:
[11/24/03 21:03:04:391 PST] 641cdbcf EJBContainerI I WSVR0037I: Starting EJB jar:
AccessEmpEjb.jar
AccessEmpWeb.war.
[Servlet.LOG]: JSP 1.2 Processor: init
[Servlet.LOG]: SimpleFileServlet: init
[Servlet.LOG]: InvokerServlet: init
[11/24/03 21:03:07:238 PST] 641cdbcf ApplicationMg A WSVR0221I: Application started:
AccessEmployee

PART 3:
[11/24/03 21:03:07:995 PST] 641cdbcf HttpTransport A SRVE0171I: Transport http is listening on
port 9,080.
[11/24/03 21:03:09:206 PST] 641cdbcf HttpTransport A SRVE0171I: Transport https is listening on
port 9,443.
[11/24/03 21:03:09:225 PST] 641cdbcf HttpTransport A SRVE0171I: Transport http is listening on
port 9,090.
[11/24/03 21:03:09:369 PST] 641cdbcf HttpTransport A SRVE0171I: Transport https is listening on
port 9,043.
2809
e-business
...

PART 4:
[Servlet.LOG]: AccessEmpServlet: init
[11/24/03 21:04:36:595 PST] 1e98dbc3 WSRdbDataSour u Database version is
SQL08014
[11/24/03 21:04:36:595 PST] 1e98dbc3 WSRdbDataSour u JDBC Driver version is
1.5.54

PART 5:
or execute an Interaction. com.ibm.ws.rsadapter.cci.WSInteractionImpl@62ddbca
[11/24/03 21:04:37:281 PST] 1e98dbc3 DataAccessReq E PMGR0000E: Call stack:
com.ibm.ws.ejbpm.PersistenceManagerException: PMGR6022E: Error using adapter to create or execute an Interaction.
   at com.ibm.ws.rsadapter.cci.WSInteractionImpl.executeOneRowFBPK(Unknown Source)
   at com.ibm.ws.ejbpm.dataaccess.DataAccessRequestImpl.executeOneRowFBPK(Unknown Source)
   at com.ibm.ws.ejbpm.beanextensions.ConcreteBeanStatefulInstanceExtensionImpl.fetchRecordFromDataStore(Unknown Source)
   at com.ibm.ws.ejbpm.beanextensions.ConcreteBeanStatefulInstanceExtensionImpl.getRecordForLoad(Unknown Source)
   at com.ibm.ws.ejbpm.beanextensions.ConcreteBeanStatefulInstanceExtensionImpl.executeLoad(Unknown Source)
   at com.ibm.ws.ejbpm.beanextensions.CBReadyState.ejbLoad(Unknown Source)
   at com.ibm.db2int.ejb.ConcreteEmployee_52940fe3.ejbLoad(Unknown Source)
   at com.ibm.ejs.container.ContainerManaged2_0BeanO.loadForEnlist(ContainerManaged2_0BeanO.java:331)
   at com.ibm.ejs.container.EntityBeanO.enlist(EntityBeanO.java:718)
   at com.ibm.ejs.container.activator.Activator.activateBean(Activator.java:516)
   at com.ibm.ejs.container.EJSHome.activateBean_Common(EJSHome.java:1234)
   at com.ibm.ejs.container.EJSHome.activateBean_Local(EJSHome.java:1150)
   at com.ibm.db2int.ejb.EJSCMPEmployeeHomeBean_52940fe3.findByPrimaryKey(Unknown Source)
   at com.ibm.ejb._AccessEmployee_Stub.getEmployeeInfo(_AccessEmployee_Stub.java:258)
   at client.AccessEmpServlet.service(AccessEmpServlet.java:195)
   at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)

PART 6:
---- Begin backtrace for nested exception
com.ibm.db2.jcc.a.SQLException: "DB2ITSO.EMPLOYEE" is an undefined name.
   at com.ibm.db2.jcc.a.cq.e(cq.java:1482)
   at com.ibm.db2.jcc.a.cq.a(cq.java:1092)
   at com.ibm.db2.jcc.c.bc.g(bc.java:138)
   at com.ibm.db2.jcc.c.bc.a(bc.java:42)
   at com.ibm.db2.jcc.c.q.a(q.java:31)
   at com.ibm.db2.jcc.c.bo.g(bo.java:145)
   at com.ibm.db2.jcc.a.cq.k(cq.java:1072)
   at com.ibm.db2.jcc.a.cr.Z(cr.java:1495)
   at com.ibm.db2.jcc.a.cr.d(cr.java:1918)
   at com.ibm.db2.jcc.a.cr.Q(cr.java:408)
Some explanations for different parts which are highlighted in the above example are provided as below:

- **PART 1** shows you current environment information such as WAS version, Operating System information, CLASSPATH setting, etc.
- **PART 2** shows you that the AccessEmployee application has been successfully started.
- **PART 3** shows you that the application server is started successfully.
- **PART 4** shows you that the AccessEmpServlet servlet is triggered by clicking the Query Employee button.
- **PART 5** shows you that an error happens when running the application. The call stack is provided. In the call stack you can find that the problem happens when fetching records from the data store that is related to EJB operations.
- **PART 6** shows you the back trace of the nested exception, the SqlException. In this part you can find clues as to why the application running failed. It is due to "DB2ITSO.EMPLOYEE" is an undefined name.
- **PART 7** shows you some information from the Nested Throwables, including the SQLCODE, which is returned by DB2 UDB.

At this point we know the reason the application running failed, as we use the new J2C Authentication Data associated with the user db2itso to access the DB2 data source, but the table name in the application for EJB access is not a fully qualified name with the form "schema.tablename". It is defaulted to DB2ITSO.<Table_Name>, for example, DB2ITSO.EMPLOYEE - an inexistent
table. When access to this inexistent table, error is returned from DB2 to WAS, then WAS to the application.

There other methods also exist to find out the cause of the failed application running. For example, you could find clues in the WAS FFDC logs, get detailed information from the activity log, or increase the DB2 UDB diagnostic level so that you also could find information about the inexistent table, etc.

There is a custom property, named currentSchema, of the DB2 data source that uses DB2 Universal JDBC Driver. This property identifies the default schema name used to qualify unqualified database object references where applicable in dynamically prepared SQL statements. Unless currentSchema is used, the default schema name is the authorization ID of the current session user.

We try to resolve this problem by modifying the currentSchema property. In WebSphere Application Server Administrative Console, select Resources -> JDBC Providers -> My DB2 Universal JDBC Provider -> Data Sources, then choose the EJBSample entry which is related to the JNDI name “jdbc/Sample”. Select Custom Properties and set the value of currentSchema to db2inst1.

Note: Changing currentSchema to “db2inst1” does not resolve the problem. But because it is an “obvious” way to resolve the problem and is commonly used when people see the problem the first time, so we take this action here for demonstration purposes. The correct resolution is provided later on in this subsection.

After applying the change to currentSchema above, we restart the application server for the changes to take effect. We then redo the testing, but the previous problem is still there. We need to check the SystemOut.log again to see what happens after changing the currentSchema property. Some snippets of SystemOut.log are extracted as shown in the following example.

Example 9-7 SystemOut.log investigation after changing currentSchema property

[11/24/03 21:18:10:801 PST] 26251959 DataAccessReq E PMGR0000E: Call stack:
com.ibm.ws.ejbpersistance.utilpm.PersistenceManagerException: PMGR6022E: Error using adapter to create or execute an Interaction. com.ibm.ws.rsadapter.cci.WSInteractionImpl@47c35950
   at com.ibm.ws.ejbpersistance.dataaccess.DataAccessRequestImpl.executeOneRowFBPK(Unknown Source)
   ... Begin backtrace for nested exception
com.ibm.db2.jcc.a.SqlException: "db2inst1.EMPLOYEE" is an undefined name.
   ... Begin backtrace for Nested Throwables
com.ibm.db2.jcc.a.SqlException: An error occurred during implicit system action type "2". Information returned for the error includes SQLCODE "-204", SQLSTATE "42704" and message tokens "db2inst1.EMPLOYEE".

From Example 9-7 on page 366 you can find that the currentSchema modification works, as the table name shown in the SystemOut.log has been changed to "db2inst1.Employee". Perhaps you would get confused if you try to test the SQL statement "select * from db2inst1.EMPLOYEE" from a DB2 Command Line Processor (CLP), as the statement could be run without any problem. But why does the error message say that the "db2inst1.EMPLOYEE" is an undefined name? To obtain more information about the problem nature, the trace of DB2 Universal JDBC Driver could be utilized.

Similar to what we just did to change the currentSchema property, in the same Custom Properties page we could change the following properties to activate the JDBC trace:

- traceLevel: -1
- traceFile: /usr/WebSphere/AppServer/logs/trace.out

After applying the changes made above and restarting the application server, then we redo the testing and check the trace output file trace.out, as shown in the example below:

**Example 9-8  JDBC Trace investigation**

**PART 1:**

```java
[ibm][db2][jcc][Time:1069738090654][Thread:Servlet.Engine.Transports : 0][DB2ConnectionPoolDataSource@2abfd88c] getPooledConnection () called
```

```java
[ibm][db2][jcc] BEGIN TRACE_DRIVER_CONFIGURATION
```

```java
[ibm][db2][jcc] Driver: IBM DB2 JDBC Universal Driver Architecture 1.5.54
```

```java
[ibm][db2][jcc] User's account name = root
```

```java
[ibm][db2][jcc] User's home directory = /
```

```java
[ibm][db2][jcc] User's current working directory = /usr/WebSphere/AppServer
```

```java
[ibm][db2][jcc] END TRACE_DRIVER_CONFIGURATION
```

**PART 2:**

```java
[ibm][db2][jcc] BEGIN TRACE_CONNECTS
```

```java
[ibm][db2][jcc] Using properties: { password=******, dataSourceName=null, traceDirectory=null, portNumber=50000, fullyMaterializeLobData=true, currentSQLID=null, securityMechanism=3, clientAccountingInformation=null, cliSchema=null, resultSetHoldability=2, serverName=null, currentPackageSet=null, loginTimeout=0, planName=null, traceDirectory=null, traceFile=/usr/WebSphere/AppServer/logs/trace.out, currentFunctionPath=null, kerberosServerPrincipal=null, retrieveMessagesFromServerOnGetMessage=true, cursorSensitivity=0, currentPackagePath=null, currentSchema=db2inst1, clientUser=null, keepDynamic=0, driverType=2, clientApplicationInformation=null, description=null, readOnly=false, clientWorkstation=null,
```

```java
```

Chapter 9. Integrated troubleshooting  367
pkList=null, deferPrepares=true, jdbcCollection=NULLID, databaseName=rsample, traceLevel=-1, traceFileAppend=false, user=db2itso

...
PART 5: [ibm][db2][jcc] BEGIN TRACE_DIAGNOSTICS

[[ibm][db2][jcc][SQLException@5a20988c] java.sql.SQLException
[[ibm][db2][jcc][SQLException@5a20988c][Sqlca@5d91d88c] DB2 SQLCA from server

Chapter 9. Integrated troubleshooting 369
Some explanations for the different parts, which are highlighted in the above example, are provided as below:

- **PART 1** contains some environment information, such as the IBM DB2 JDBC Universal Driver version, Operating System information, JVM information, CLASSPATH setting, and so on. Only part of the information is shown in our example.

- **PART 2** shows you the current setting of properties for the DB2 data source. There are some properties highlighted in the example that you probably are interested in, for example, currentSchema=db2inst1, and you also could find
that the function setCurrentSchema() is called to make the currentSchema property change.

- **PART 3** indicates that a connection has been made to the sample database (see the message “Successfully connected to server jdbc:db2:rsample”). Messages also shown here include the SQL statement that is executed on the DB2 side.

- **PART 4** shows you the send buffer content. From the ASCII translation of the send buffer, you could find that the corresponding command that is used to make the currentSchema change:

  SET CURRENT SCHEMA = 'db2inst1'

  The statement above is the problem source, as the schema name is case sensitive when it is enclosed in the single quotes. The problem could be easily reproduced within the DB2 CLP, as shown in the example below.

  **Example 9-9  Reproducing problem using similar SQL statements**

  ```
  atlantic / # su - db2itso
  $ db2 connect to sample user db2itso using ibmdb2
  $ db2 "set current schema = 'db2inst1'"
  DB20000I  The SQL command completed successfully.
  $ db2 "select * from employee"
  SQL0204N  "db2inst1.EMPLOYEE" is an undefined name. SQLSTATE=42704
  ```

  As the current schema value is enclosed by the single quotes, it would not be converted to upper case when running inside DB2. But the fully qualified table object name for the EMPLOYEE table is “DB2INST1.EMPLOYEE”, not “db2inst1.EMPLOYEE”. This is the reason why the error message says “db2inst1.EMPLOYEE” is an undefined name.

  **Note:** The reason why you could run the SQL statement “select * from db2inst1.EMPLOYEE” successfully in the DB2CLP is that the DB2 engine helps you to convert the lower case schema name to an upper case schema name automatically, as there are no quotation marks surrounding the schema name.

- **PART 5** shows you the trace diagnostics, including the output for most tokens of the SQL Communications Area (SQLCA), stack traceback for the function calls, etc.

Now we know that the root cause of the failed application running is due to the lower case schema name. We just need to change the schema name to upper case, that is, DB2INST1, in the Custom Properties configuration page for the DB2 data source in WebSphere Application Server Administrative Console. After applying the change and restarting the application server, all the services
provided by the AccessEmployee application, including the “Query Employee” service, can be run successfully.

Up to this point, we have successfully resolved the problem by investigating both WebSphere Application Server logs and DB2 Universal JDBC Driver traces. The steps discussed above could be applied to most problems that involve both WAS and DB2 UDB. Also please remember to turn off the trace once the problem troubleshooting work is done.

There are also many other methods to resolve this problem. For example, you could create aliases for related tables from the DB2 UDB side so that you could access DB2INST1.<table_name> via the alias DB2ITSO.<table_name>, or you could modify the schema for the mapping between EJB and RDB, etc. As the purpose of this scenario is just to show you some basic methodologies that you can utilize for a general problem troubleshooting situation, so we only discuss the method of changing currentSchema property of the data source to resolve the problem. If you encounter a similar “wrong schema” problem in your environment, you can have your own method that is most appropriate for your runtime environment.

9.3.2 Concurrency scenario

Performance degradation due to inappropriate concurrency settings of an application or database is a very common problem. During our lab environment test we have also found a similar problem and this scenario is dedicated to resolve that issue. The root cause of the problem demonstrated here is one where the default lock settings of the TRADE3DB database are creating a significant locking contention on the DB2 UDB server, which results in poor performance/response time.

We are using a TRADE3 application for our scenarios which is the third generation WebSphere end-to-end benchmark and performance sample application. TRADE3 application is developed using the WebSphere Application Developer (WSAD) development tool. This includes development of EJBs, servlets and visual development of JSPs and Java clients.

**Note:** Refer to the Appendix A, “Trade3 application” on page 401, for a complete understanding of the TRADE3 application and the architecture used.

**Simulate an environment**

In order to simulate this environment we are using the TRADE3 application. We applied a workload through Web Performance Tools (WPT) application stress, which simulates an environment where 40 users log in and make a request for the stock quotes, and then repeatedly request the same stock quotes again and
again. This simulates a high volume trading application where multiple customers perform stock quote requests. Due to applied stress number of user start complaining about poor response time, so a task is assigned to a performance problem determination team to look into this matter and come up with the root cause and an appropriate solution for the same.

Table 9-2 is the detail of hardware and software infrastructure used by TRADE3 application in our lab environment. The DB2 UDB database we created is call TRADE3DB.

Table 9-2  Infrastructure for concurrency scenario

<table>
<thead>
<tr>
<th><strong>Hardware/Software</strong></th>
<th><strong>Configuration</strong></th>
</tr>
</thead>
</table>
| WebSphere Application Server 5.0.2 | Hostname: HELIUM  
IP: 9.1.38.185  
x-Series with Windows 2000 Server  
4 GB RAM  
Dual CPU: Intel Pentium® III CPU family  
1133 MHz  
Two 16 GB HDD  
Application: TRADE3 |
| DB2 UDB 8.1.3 | Hostname: GALLIUM  
IP: 9.1.38.184  
x-Series with Windows 2000 Server  
4 GB RAM  
Dual CPU: Intel Pentium III CPU family  
1133 MHz  
Two 16 GB HDD  
Database: TRADE3DB |
| Web Performance Tools (WPT) | Hostname: LOCHNESS  
IP: 9.1.38.167  
x-Series with Windows 2000  
2 GB RAM  
Intel Pentium 4 CPU 2.40 GHz  
37.2 GB HDD  
WPT application: Stress |

**Analyze the problem and determine the root cause**

As we are working in our controlled environment, so we have ignored the real-world possibilities because it is almost impossible to cover all of them within the scope of this book. We are diagnosing some of the key areas to depict this performance degradation problem. A few of the most common areas are:

- Memory constraints
- Disk I/O contention
Let us start analyzing them one by one to drill down the root cause of the problem.

Memory constraints

In order to verify memory usage, CPU consumption, and disk I/O statistics, we use Performance Monitoring tools of the Windows perfmon and Windows Task Manager. We check these statistics both on WebSphere Application Server (HELIUM) and DB2 UDB server (GALLIUM) by capturing the information through the perfmon tool and Task Manager. Below are the steps to configure these tools:

a. To start the perfmon, click Start -> Run. Enter perfmon, or enter Control Panel. Select Administrative Tools -> Performance.

b. Select Performance Logs and Alerts from the left pane of the window.

c. Right-click Counter logs and select New Log Settings.

d. Give any name (for example, Memory Overview) and click OK, as shown in Figure 9-4.

Figure 9-4   perfmon settings for Memory Overview
e. Click **ADD** to add a performance counter. Select the counters from the list and click **ADD**, as shown in Figure 9-5.

![Figure 9-5 prefmon settings for Memory Overview](image)

f. Close the windows. You will be sent back to the initial screen and click **OK** to close the window.

g. Double-click **Counter Logs**. If the newly created log is not already running, (check the green icon as shown in Figure 9-6 on page 376) then right-click the log and select **Start**.
h. Select **System Monitor** from the tree, then select the **View Log File Data** to see the results.

The rest of the activity we track from the Windows Task Manager to make it more informative and appropriate to our settings.

a. Open the Windows Task Manager and click the **Processes** tab. Chose the View menu and click **Select columns** and finally check the check boxes for the desired options, as shown in Figure 9-7 on page 377.
Figure 9-7  Task Manager settings

b. Select the appropriate columns for the Task Manager and click **OK**.
We considered whether system swapping due to potential memory constraints was the cause of the problem. The statistics shown in Figure 9-8 show that plenty of free memory is available for use, so it is not a memory problem. Let us continue the analysis with the rest of the key areas.

- **Disk I/O contention**

We checked the disk I/O activity through the Task Manager. We have not found much contention on the I/O activity, as shown in Figure 9-9 on page 379, on the GALLIUM server.
We thought that the possible cause of the poor response time at peak periods could be due to disk I/O contention, but we found that the disk I/O rate is within an acceptable range. The output of Figure 9-9 clearly shows that there is not an issue with I/O contention, so we must continue with rest of the possibilities.

CPU resources

In order to check the fate of CPU consumption let us verify it with the Task Manager.
Figure 9-10  CPU consumption for concurrency scenario

The above statistics in CPU column of the output shown in Figure 9-10 clearly indicate that the percentage of CPU consumption is under moderate limits, so CPU is not a problem in this case. We continue our diagnostic process with WebSphere Application Server and DB2 UDB.

- Application server/application issues

While looking at the problem through the performance analyzing tool provided by WebSphere - Tivoli Performance Viewer we found that response time is too high, as shown in Figure 9-11 on page 381.
We checked with WAS logs but could not find much there. The final area left to be explored is the DB2 UDB server, so let us examine the database activity.

**Database configuration/SQL**

We analyzed the DB2 diagnostic file, DB2DIAG.LOG, and found a high number of lock escalation, especially against database table DB2ADMIN.HOLDINGEJB to attain a intent share lock. Example 9-10 shows the few messages thrown in DB2DIAG.LOG file.

**Example 9-10  Information from the DB2 diagnostic log DB2DIAG.LOG**

2003-11-25-18.17.01.968001  Instance:DB2  Node:000
data management  sqldEscalateLocks Probe:3  Database:TRADE3DB

ADM5502W  The escalation of "109" locks on table "DB2ADMIN.HOLDINGEJB" to lock intent "S" was successful.

The diagnostic level parameter DIAGLEVEL default value of 3 is appropriate for routine monitoring. We have changed this value to 4, which is the highest
level of diagnostics. This is because the routine monitoring level does not provide us with the information required to perform proper problem diagnosis. Then we verified this fact with snapshot for the TRADE3 database, as shown in Example 9-11.

Example 9-11  Snapshot of TRADE3DB for concurrency scenario

```
Example 9-11  Snapshot of TRADE3DB for concurrency scenario

db2 get snapshot for database on trade3db
```

```
Database Snapshot

Database name              = TRADE3DB
Database path              = C:\DB2\NODE0000\SQL
Input database alias       = TRADE3DB
Database status            = Active
Catalog database partition number = 0
Catalog network node name  =
Operating system running at database server= NT
Location of the database   = Local
First database connect timestamp = 11-25-2003 18:13:34
Last reset timestamp      =
Last backup timestamp      =
Snapshot timestamp         = 11-25-2003 18:19:20

Locks held currently       = 123
Lock waits                 = 270
Lock list memory in use (Bytes) = 19188
Deadlocks detected        = 0
Lock escalations           = 747
Exclusive lock escalations = 0
Agents currently waiting on locks = 1
Lock Timeouts              = 5
Number of indoubt transactions = 0

Bold values in Example 9-11 also verify the same fact of the lock escalation. In order to further confirm the above cause we checked it through DB2 Command Line Process output, as shown in Figure 9-12 on page 383.
Apart from the above indications, we checked our system Event Viewer Application log and found the message as shown in Figure 9-13 on page 384.
All of the above facts lead to a DB2 concurrency issue, so the root cause of the problem is inappropriate settings of locking parameters.

**Apply best practices**

Referring to the results shown in Example 9-11 on page 382, snapshot clearly points to of DB2 lock contention as follows:

- **270 Lock Waits**
  This corresponds to an application having to wait for another user to release a lock on a required resource before it can continue to execute.

- **747 Lock Escalations**
  According to this factor the total number of locks held by the application exceeded the value specified in LOCKLIST parameter, and therefore an escalation of the locks took place. Such an escalation has the potential to significantly impact concurrency among users requesting incompatible locks on shared resources. Exclusive lock escalations are the most restrictive of such locks, essentially single threading execution of applications contending for the same resource.
We issued the following command to determine the TRADE3DB database configuration parameters relating to locking and output, as shown in Example 9-12.

**Example 9-12  Database configuration setting of TRADE3DB**

```sql
db2 get db cfg for trade3db
#Trimmed results of db config

Percent. of lock lists per application (MAXLOCKS) = 10
Lock timeout (sec) (LOCKTIMEOUT) = -1
```

The LOCKLIST parameter value can be calculated by examining the memory requirements of each lock, the maximum number of concurrent connections, and the average number of concurrent locks held by an application. Each initial lock on a DB2 object consumes 72 bytes of memory, with each additional lock on the same object consuming 36 bytes. The LOCKLIST parameter value is specified in 4-KB blocks, and may be computed as follows:

\[
\text{LOCKLIST} = \text{Maximum Conn App} \times \text{Avg no of locks} \times \frac{72}{4096}
\]

The MAXLOCKS defines the percentage of LOCKLIST memory that can be assigned to single application at a given time before lock escalation occurs.

The considerations when setting maxlocks is to use it in conjunction with the size of the lock list (LOCKLIST). The actual limit of the number of locks held by an application before lock escalation occurs is:

\[
\text{MAXLOCKS} \times \text{LOCKLIST} \times \frac{4096}{(100 \times 36)}
\]

Where 4,096 is the number of bytes in a page, 100 is the largest percentage value allowed for maxlocks, and 36 is the number of bytes per lock. If you know that one of your applications requires 1000 locks, and you do not want lock escalation to occur, then you should choose values for maxlocks and locklist in this formula so that the result is greater than 1000.

The LOCKTIMEOUT parameter specifies the number of seconds that an application waits to obtain a lock. This helps avoid deadlocks for applications. In a transaction processing (OLTP) environment, you can use an initial starting value of 20 to 30 seconds.

If we see the values of MAXLOCKS, LOCKTIMEOUT database parameters of TRADE3DB, as shown in Example 9-12, they are not optimally tuned. Let us change these parameters as shown in Example 9-13 on page 386.
Example 9-13 Update the configuration

db2 update db cfg using MAXLOCKS 75
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.

db2 update db cfg using LOCKTIMEOUT 20
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.
SQL1363W One or more of the parameters submitted for immediate modification were not changed dynamically. For these configuration parameters, all applications must disconnect from this database before the changes become effective.

db2stop force
11-25-2003 19:04:26 0 0 SQL1064N DB2STOP processing was successful. SQL1064N DB2STOP processing was successful.

db2start
11-25-2003 19:04:41 0 0 SQL1063N DB2START processing was successful. SQL1063N DB2START processing was successful.

db2 get db cfg for trade3db

#Trimmed results of db config
Percent. of lock lists per application (MAXLOCKS) = 75
Lock timeout (sec) (LOCKTIMEOUT) = 20

After changing these parameters we again applied the workload through the WPT stress application and examined the DB2DIAG.LOG file, and there are no further messages of lock escalation. In order to confirm it further, let us have a look at the snapshot shown in Example 9-14.

Example 9-14 Snapshot results after resolving concurrency problem

db2 get snapshot for database on trade3db
#Trimmed results of db config

Database Snapshot

<table>
<thead>
<tr>
<th>Database name</th>
<th>TRADE3DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database path</td>
<td>C:\DB2\NODE0000\SQL</td>
</tr>
<tr>
<td>Input database alias</td>
<td>TRADE3DB</td>
</tr>
<tr>
<td>Database status</td>
<td>Active</td>
</tr>
</tbody>
</table>

Locks held currently = 0
Lock waits = 0
Lock list memory in use (Bytes) = 9000
Deadlocks detected = 0
Lock escalations = 0
Exclusive lock escalations = 0
Agents currently waiting on locks = 0
Lock Timeouts = 5
Number of indoubt transactions = 0

This confirms that now we do not have concurrency problems.

9.3.3 High I/O consumption scenario

Performance tuning is an iterative process, and you encounter the situation when you just resolved the one problem and stuck with the next one. So after resolving the problem of concurrency we again simulated the environment via the TRADE3 application by putting a stress through the stress application of WPT. We still found the performance issues, and in order to eradicate the problem we once again started our problem determination and tuning process.

Analyze the problem and determine the root cause

As discussed in previous scenarios, we are evaluating only the key areas to determine a problem. Also here we are eliminating the scope of the concurrency problem, as it is already resolved in previous scenario. Below are the areas which we are going to explore:

- Memory constraints
- CPU resources
- Disk I/O contention
- Application server/application issues
- Database configuration/SQL

Let us start analyzing them one by one to drill down to the root cause of the problem.

- Memory constraints
  We checked both DB2 UDB and WAS servers. None of them is showing any memory issues. We have confirmed this with the help of the prefmon tool of Windows.

- CPU resources
  In order to check the fate of CPU consumption we have verified with the Task Manager. The statistics of CPU indicate that the percentage of CPU consumption is under moderate limits, so CPU is not a problem in this case.

- Disk I/O contention
  The output of the statistics shown in Figure 9-14 on page 388 clearly indicate that there is high I/O activity is happening at the database server, which is probably causing poor performance of the system.
Application server/application-related issues

We checked the O/S resources and processes related to Websphere there is no such indication of contention among resources. Also, the previous diagnosis shows that some high I/O activity is happening at the database server. So we continue with the DB2 UDB server investigation.

Database configuration/SQL

Below are the messages thrown by DB2 in the diagnostic file indicating that there is a BUFFERPOOL-related problem (Example 9-15).

Example 9-15 db2diag.log file to evaluate high CPU consumption scenario

ADM607II The maximum number of pinned pages allowed concurrently was reached in buffer pool "IBMDEFAULTBP" (ID "1"). As a result performance may not be optimal. Increasing the SORTHEAP database configuration parameter may reduce the chances of this condition occurring in the future.
Example 9-16  Snapshot to evaluate high CPU consumption scenario

update dbm configuration using DFT_MON_BUFPOOL ON

db2  get snapshot for bufferpools on tarde3db
#Trimmed results of db config

Buffer pool data logical reads                      = 91411
Buffer pool data physical reads                    = 181
Asynchronous pool data page reads                  = 9
Buffer pool data writes                            = 245
Asynchronous pool data page writes                 = 78
Buffer pool index logical reads                    = 151390
Buffer pool index physical reads                   = 203
Asynchronous pool index page reads                 = 1
Buffer pool index writes                           = 246
Asynchronous pool index page writes                = 61
Total buffer pool read time (ms)                   = 9
Total buffer pool write time (ms)                  = 20943
Total elapsed asynchronous read time               = 0
Total elapsed asynchronous write time              = 14968
Asynchronous data read requests                    = 4
Asynchronous index read requests                   = 1
LSN Gap cleaner triggers                           = 8
Dirty page steal cleaner triggers                  = 0
Dirty page threshold cleaner triggers              = 0
Time waited for prefetch (ms)                      = 2
Unread prefetch pages                              = 0
Direct reads                                       = 528
Direct writes                                      = 136
Direct read requests                               = 103
Direct write requests                              = 36
Direct reads elapsed time (ms)                     = 20
Direct write elapsed time (ms)                     = 323
Database files closed                               = 0
Data pages copied to extended storage              = 0
Index pages copied to extended storage             = 0
Data pages copied from extended storage            = 0
Index pages copied from extended storage           = 0
The above analysis of the `db2diag.log` file in Example 9-15 on page 388 and `snapshot` in Example 9-16 on page 389 clearly indicate that the root cause of the problem is inadequate sizing of the bufferpool. The size of the default bufferpool `IBMDEFAULTBP` of the TRADE3DB database is creating a problem. This configuration of DB2 UDB results in a very high rate of I/O on the system when we are applying a workload through the TRADE3 application.

**Apply best practices**

A buffer pool is a memory used to cache table and index data pages when they are being read from disk or being modified. The buffer pool improves database system performance by allowing data to be accessed from memory instead of from disk. Accessing it from the memory is much faster than disk access. The less often the database manager needs to read from or written to a disk, the better the performance.

As pages are read and modified, they accumulate in the database buffer pool. When a page is read in, it is read into a buffer pool page. If the buffer pool is full of modified pages, one of these modified pages must be written out to the disk before the new page can be read in. When the size of the bufferpool is small, a lot of disk I/O happens, which leads to performance degradation.

In the snapshot output shown in Example 9-17, look for the following *logical reads* and *physical reads* so that you can calculate the buffer pool hit ratio, which can help you tune your buffer pools.

### Example 9-17 Buffer pool values

| Buffer pool data logical reads | = 91411 |
| Buffer pool data physical reads | = 181 |
| Buffer pool data writes | = 245 |
| Buffer pool index logical reads | = 151390 |
| Buffer pool index physical reads | = 203 |

The buffer pool hit ratio indicates the percentage of time the database manager needs to not load a page from disk to fulfill the request, as the page is already in the buffer pool. The higher the buffer pool hit ratio, the lower the frequency of disk I/O.

The above calculation shows all of the pages that have been cached by the buffer pool. Ideally this ratio should be 95 percent or above. So in order to increase the buffer pool hit ratio, increase the buffer pool size.

Take the formula mentioned in “System tuning” on page 272 while calculating the size of the bufferpool and review the results using the snapshot information of...
your environment. See the *DB2 UDB Administration Guide - Performance, SC09-4821*, for more information about bufferpool tuning.

We have identified the root cause of the problem as too small of the size of the bufferpool. Let us increase the size of the buffer pool to remove this problem using command shown in Example 9-18.

**Example 9-18  Bufferpool settings**

```
db2 alter bufferpool ibmdefaultbp immediate size 2000
DB20000I  The SQL command completed successfully.

db2stop force
11-26-2003 12:04:26     0   0   SQL1064N  DB2STOP processing was successful.
SQL1064N  DB2STOP processing was successful.

db2start
11-26-2003 12:04:41     0   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.

db2 get snapshot for database manager
```

After changing these parameters shown in Example 9-18, we again applied the workload through the WPT stress application and examined the DB2DIAG.LOG file. Now there is no longer a buffer pool problem.

### 9.3.4 High CPU utilization scenario

This scenario describes the problem of performance degradation due to high CPU consumption on both the DB2 UDB and WAS server. There can be many reasons of high consumption of CPU. It may be a system problem, DB2 UDB configuration problem, WAS configuration it or might be inefficient query design. So we are going to solve this high CPU consumption by following our problem determination methodology.

**Simulate an environment**

We have simulated this environment on DB2 UDB server by changing few configuration parameters, as shown in Example 9-19. We are using the TRADE3 application and the TRADE3DB database with Web Performance Tools (WPT) application *stress*. The details of hardware and software infrastructure used for this environment are the same as provided in the previous scenario.

**Example 9-19  Database configuration change for high CPU consumption scenario**
su - db2inst1

db2 update dbm cfg using intra_parallel yes

DB20000I  The UPDATE DATABASE MANAGER CONFIGURATION command completed successfully.
SQL1362W  One or more of the parameters submitted for immediate modification were not changed dynamically. Client changes will not be effective until the next time the application is started or the TERMINATE command has been issued. Server changes will not be effective until the next DB2START command.

db2 update dbm cfg using num_poolagents 50 maxagents 500

DB20000I  The UPDATE DATABASE MANAGER CONFIGURATION command completed successfully.
SQL1362W  One or more of the parameters submitted for immediate modification were not changed dynamically. Client changes will not be effective until the next time the application is started or the TERMINATE command has been issued. Server changes will not be effective until the next DB2START command.

db2 update dbm cfg using DIAGLEVEL 4

DB20000I  The UPDATE DATABASE MANAGER CONFIGURATION command completed successfully.

db2stop force

11-21-2003 19:35:19     0   0   SQL1064N  DB2STOP processing was successful.
SQL1064N  DB2STOP processing was successful.

db2start

11-21-2003 19:36:19     0   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.

db2 update db cfg for sample using dft_degree 16

In Example 9-19 on page 391 we have changed a few parameters to replicate the environment of high CPU consumption, especially database agents. This configuration of DB2 UDB results in a very high rate of agent and process creation on the system when appropriate workload is applied by the TRADE3 application, which results in high CPU consumption.

**Analyze the problem and determine the root cause**

As we are working in our controlled environment, so we are diagnosing some of the key areas to depict this performance degradation problem, as shown below:

- Memory constraints
Let us start analyzing them one by one to drill down to the root cause of the problem.

- Memory constraints
  
  In order to verify this concerned area we used the perfmon and Task Manger tools of Windows. The statistics shown by the tools do not lead to a memory problem. It seems to be some other problem. Let us continue the analysis with the rest of the areas.

- Disk I/O contention
  
  We have ruled out the possibility of I/O contention, as we have already resolved this issue in a previous scenario. We must continue on with the rest of the possibilities.

- CPU resources
  
  Both DB2 UDB and WAS server are showing high CPU consumption. The statistics of CPU consumption of Helium - WAS server, shown in Figure 9-15 on page 394, clearly indicate that CPU is being used a lot.
Let us further drill down to find out which process is consuming a lot of processing power of the CPU on Helium. We have checked the configuration of Websphere Application Server through the Tivoli Performance Viewer tool. Thread pool enables components of the server to reuse threads to eliminate the need to create new threads at run time, as creating new threads expends time and resources. Figure 9-16 on page 395 clearly indicates that there is a high number of thread creation and destruction going on, and at the same time the average number of concurrently active threads is very low. This may be the cause of high CPU consumption.
Figure 9-16  Tivoli Performance Viewer for High CPU consumption scenario output

We analyzed the Web container thread pool settings through the admin console (as shown in Figure 9-17), which can be a possible reason for this problem.

Figure 9-17  Webcontainer thread pool settings
Let us evaluate these DB2 UDB settings to resolve this problem. Below in Example 9-20 are the results of the DB2 diagnostic file DB2DIAG.LOG.

Example 9-20  db2diag.log file to evaluate high CPU consumption scenario

2003-11-26-16.36.52.312000  Instance:DB2  Node:000
PID:2524(db2syscs.exe)  TID:3340  Appid:G90126B9.IE0A.015246232717
base sys utilities sqleGetAgentFromPool Probe:35  Database:TRADE3DB

Attemping to STEAL an agent

2003-11-26-16.36.52.375000  Instance:DB2  Node:000
PID:2524(db2syscs.exe)  TID:3340  Appid:G90126B9.IE0A.015246232717
base sys utilities sqleGetAgentFromPool Probe:35  Database:TRADE3DB

Attemping to STEAL an agent

2003-11-26-16.36.52.453000  Instance:DB2  Node:000
PID:2524(db2syscs.exe)  TID:3340  Appid:G90126B9.IE0A.015246232717
base sys utilities sqleGetAgentFromPool Probe:35  Database:TRADE3DB

Attemping to STEAL an agent

The database manager configuration parameter for agent pool size (num_poolagents) affects the total number of subagents that can be kept associated with applications. If the pool size is too small and the pool is full, a subagent disassociates itself from the application it is working on and terminates. Because subagents must be constantly created and re-associated to applications, performance suffers.

In addition, if the value of num_poolagents is too small, one application may fill the pool with associated subagents. Then when another application requires a new subagent and has no subagents in its associated agent pool, it will steal subagents from the agent pools of other applications. This situation is costly, and causes poor performance.

Example 9-21  Snapshot to evaluate high CPU consumption scenario

update dbm configuration using DFT_MON_STMT ON

db2 get snapshot for database manager

Database Manager Snapshot

Node name =
Node type = Enterprise Server Edition with local and remote clients
Instance name = DB2
Number of database partitions in DB2 instance = 1
Database manager status = Active
Product name = DB2 v8.1.3.132
Service level = s030728 (WR21324)

Private Sort heap allocated = 0
Private Sort heap high water mark = 48
Post threshold sorts = Not Collected
Piped sorts requested = 128
Piped sorts accepted = 128

Start Database Manager timestamp = 11-26-2003 16:43:37.000031
Last reset timestamp =
Snapshot timestamp = 11-26-2003 16:51:42.462675

Remote connections to db manager = 14
Remote connections executing in db manager = 14
Local connections = 0
Local connections executing in db manager = 0
Active local databases = 1

High water mark for agents registered = 125
High water mark for agents waiting for a token = 0
Agents registered = 125
Agents waiting for a token = 0
Idle agents = 0

Committed private Memory (Bytes) = 71647232

Switch list for db partition number 0
Buffer Pool Activity Information (BUFFERPOOL) = ON 11-26-2003 16:43:37.000031
Lock Information (LOCK) = ON 11-26-2003 16:50:19.273637
Sorting Information (SORT) = OFF
SQL Statement Information (STATEMENT) = ON 11-26-2003 16:43:37.000031
Table Activity Information (TABLE) = OFF
Take Timestamp Information (TIMESTAMP) = ON 11-26-2003 16:43:37.000031
Unit of Work Information (UOW) = OFF

Agents assigned from pool = 446
Agents created from empty pool = 124
Agents stolen from another application = 1113
High water mark for coordinating agents = 15
Max agents overflow = 0
Hash joins after heap threshold exceeded = 0

The above analysis of db2diag.log file and snapshot in Example 9-21 on page 396 clearly indicate that there is a problem of stealing agents among applications, which eventually creates extra cost to the system. So the root
cause of the problem is inappropriate configuration settings of DB2 agents and Websphere Web container thread pool settings.

Apply best practices
Let us first tune the Websphere Application Server (Helium). As we have seen during the investigation of the Helium, which Tivoli Performance Viewer is showing a high number of thread creation and destruction, which may be the cause of high CPU consumption.

Take the help of Tivoli Performance Viewer to tune the Web container thread pool size settings. Apply the workload with the typical number of incoming client requests, fixed number of iterations, and use a standard configuration settings. Check the Percent Maxed and Active Threads counters of the Web container submodule of the Thread Pools module.

Simple rules are if the value of the Percent Maxed counter is consistently in the double digits, then the Web container could be a bottleneck and the number of threads should be increased.

If the number of active threads are significantly lower than the number of threads in the pool, this indicates a need to lower the thread pool size for a performance gain.

Figure 9-18 shows that we have changed the value of *Maximum Size* parameter to 40 threads. This must enhance the average of active threads, which helps to decrease the more often creation and destruction of the threads.

Figure 9-18   Thread pool settings of Web Container

Now let us rectify the DB2 UDB server problem of stealing agents, which we have found during the diagnostic process.
The ideal settings of agents for a server with a non-partitioned database and local and remote clients is:

\[
\text{maxagents}/50 \times \text{max_querydegree} \text{ or maxagents} - \text{max_coordagents.}
\]

Otherwise, a -1 value can be chosen for NUM_POOLAGENTS setting, which is a calculated value.

-1 stands for calculated value = \text{maxagents} - \text{num_initagents}.

If the ratio of agents created due to empty agent pool versus agents assigned from the pool is very high, it may indicate that the NUM_POOLAGENTS configuration parameter should be increased; otherwise, set the values of NUM_POOLAGENTS and MAXAGENTS to an appropriate ratio.

A low ratio suggests that NUM_POOLAGENTS is set too high, and that some of the agents in the pool are rarely used and are wasting system resources. A high ratio can indicate that the overall workload for this node is too high. You can adjust the workload by lowering the maximum number of coordinating agents specified by the MAXAGENTS configuration parameter.

See the DB2 UDB Administration Guide - Performance, SC09-4821, for more details.

Let us change the configuration parameters, as shown in Example 9-22, to resolve this problem.

**Example 9-22  Num_PoolAgents configuration**

Change agent parameters and show results

```
db2 update dbm cfg using num_poolagents -1
DB20000I  The UPDATE DATABASE MANAGER CONFIGURATION command completed successfully.
SQL1362W  One or more of the parameters submitted for immediate modification were not changed dynamically. Client changes will not be effective until the next time the application is started or the TERMINATE command has been issued. Server changes will not be effective until the next DB2START command.

db2stop force
11-21-2003 19:35:19     0   0   SQL1064N  DB2STOP processing was successful.
SQL1064N  DB2STOP processing was successful.

db2start
11-21-2003 19:36:19     0   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.

db2 update db cfg for trade3db using dft_degree -1
```
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.

db2 get snapshot for database manager

After applying the changes at DB2 UDB server and WAS server we again checked the resources. Now there is no problem of high CPU consumption and the problem has been resolved.
This appendix provides the basic introduction and step-by-step procedure to set up an environment for a *Trade3* application and how to use the Web Performance Tool application *stress* to put an appropriate workload to simulate an environment. This appendix includes:

- Introduction to Trade3 application
- Setup and deploy the Trade3 application
- Use a stress application
Introduction

Trade3 is the third generation of the WebSphere end-to-end benchmark and performance sample application. The Trade3 benchmark models an online stock brokerage application and has been re-designed and developed to cover WebSphere's significantly expanding programming model. This provides a real-world workload driving WebSphere's implementation of J2EE 1.3 and Web Services, including key WebSphere performance components and features.

Trade3's design spans J2EE 1.3, including EJB 2.0 component architecture, Message Driven beans, transactions (1-phase, 2-phase commit) and Web Services (SOAP, WSDL, UDDI). Trade3 also highlights key WebSphere performance components such as DynaCache, WebSphere Edge Server and Web Services.

Note: Register yourself and download the Trade3 application from http://www-3.ibm.com/software/webservers/appserv/benchmark3.html. Also for more understanding and in depth architecture, refer to the documentation that comes with this download.

Trade3 application deployment

Following are the steps to configure and install the Trade3 application in the Windows environment:

1. You must start the application server before the application can be installed.
   
   cd %WasHome%\bin
   stopserver server1
   startserver server1

2. Prepare for installation.
   a. Download and unzip the Trade3 install package.
   b. Open a command window to perform the install and change to the t3install directory.
   c. Run the WAS setupCmdLine script to set up your shell environment:
      <WAS_HOME>\bin\setupCmdLine.bat
   d. Start the WebSphere 5.0 server.
      %WAS_HOME%\bin\startServer server1

3. Create the DB2 database for Trade3 and modify DB2 for Read Stability. In our example, the database created is TRAD3DB, and the tables are created from
the Table.ddl file provided by the Trade3 kit. The bind command binds the necessary packages to the database.

a. Set up a DB2 command window using the following method:
   
   Start Menu -> Run -> db2cmd

b. Execute the following DB2 commands from the Trade3 install directory using the DB2 command window opened in the previous step:
   
   db2 create db trade3db  
db2 connect to trade3db  
db2 -tvf DB2/Table.ddl  
db2 disconnect all  
db2set DB2_RR_TO_RS=yes  
db2 update db config for trade3db using logfilsiz 1000  
db2 update db cfg for trade3db using maxappls 100  
db2 stop force  
db2 start

c. Execute the following commands to bind DB2 packages for the trade3db.
   
   db2 connect to trade3db  
cd <sqllib>/bnd  
db2 bind @db2cli.lst blocking all grant public

As in most of the cases, the database and applications are running on separate servers, so if your system is running in a multi-tier environment then you must catalog the database server - TRADE3DB database to Websphere Application Server.

4. Install, configure and start the Trade3 application and resources. Create the Trade3 resources from the scripts. The forward slash is used to specify the location of the DB2 database driver db2java.zip. The WebSphere Ant tool is used to install Trade3 from the Ant script. Make sure that during each step, there is no error in the run output.

a. Install Trade3 JDBC/JMS resources:
   
   %WAS_HOME%/bin/ws_ant -f Trade3.xml install Resources

b. Install the Trade3 application:
   
   %WAS_HOME%/bin/ws_ant -f Trade3.xml install App

c. Restart WebSphere to pick up the newly created resources and Trade3 application:
   
   %WAS_HOME%/bin/stopServer server1
   %WAS_HOME%/bin/startServer server1

5. Populate the Trade3 database and go trade.
   
   – Open the default page of the Trade3 application (see Figure A-1 on page 404):
http://localhost:9080/trade

Figure A-1  Home page of Trade3

d. Click **Configuration**, then click **(Re-)populate Trade database**. This link populates or re-populates the Trade database with dummy users like uid:0, uid:1, ... and stocks as s:0, s:1, .... By default, the Trade3 script creates 500 users and 1000 stocks. The database is then populated with a new set of test data. See Figure A-2 on page 405.
Figure A-2  Populate the database

e. Run the following db2 command to update DB2 statistics, as shown in Figure A-3 on page 406. This must be done after database population for performance research.

```
db2 connect to trade3db
db2 reorgchk update statistics
```
f. Go trade! You have successfully installed the Trade3 application (Figure A-4 on page 407).
Appendix A. Trade3 application

Figure A-4   Go Trade

Web Performance Tool

The Web Performance Tool (WPT) provides a powerful performance testing mechanism. Here we briefly demonstrate how to use and configure this testing tool with the Trade3 application.

Installation of the WPT tool is just a one-click task. No configuration settings are required; simply install the application. Once WPT installation is complete, run the below commands to simulate your requirement.

```
    cd %WPTHome%\bin
    stress -config ..\config\trade3.acf -host %Hostname%:9080
```

The trade3.acf is a WPT script file that is used to describe the various parameters, like No of connections, Think Time, etc., for the simulation of your environment. The stress application picks the configuration values defined in trade3.acf and produces an equivalent workload.

Example A-1 on page 408 shows a sample configuration file. You can find more information about the configuration file shown in the user documentation. The HTTP port 9080 is used to address the embedded Web server inside
WebSphere, but for a real world environment, it would be more appropriate to use the external IBM HTTP Server running on port 80.

Example: A-1  Trade.acf configuration file

```
# Global Setting information
# Threads - the number of threads to run against the target server
# TotalPageRequests - the number of page requests to make
# - can consist of multiple requests
# ErrorLog - the file that errors will be logged to
# StatServerPort - the port that the stat server will listen for
# RandomEarlyClosePercentage - percentage of requests that will be closed early
#

global_settings

#   Clients                             100
Clients                             60
ThreadsPerClient                    1
TotalPageRequests                   0
Rampup
ThinkMinimum 0
ThinkMaximum 0
#   ErrorLog                            error.log
CookieCache                         on
#   PurgeCookieCache                    on
CookieIgnoreDefaultExpires         on
RequestTimeout                      30 seconds
end_global_settings

stats_definition
    RequestStatistics on
    PageStatistics    on
#   PageLog           page.log
#   RequestLog        request.log
SummaryLog        summary.log
StatsLog          Trade3stats.log
StatsLogInterval     7200
end_stats_definition

header_definition   header0
    Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
    Accept-Language: en-us
    Accept-Encoding: gzip, deflate
    User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
```
Host: <<HOST>>
Connection: Close
end_header_definition

request_definition request0
  Header header0
  Method GET
  URI /trade/scenario
  Protocol HTTP/1.0
  ResultCode 200
end_request_definition

start_page page0
  Request request0
end_page_definition
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 413. Note that some of the documents referenced here may be available in softcopy only.

- *DB2 UDB/WebSphere Performance Tuning Guide*, SG24-6417
- *Database Performance Tuning on AIX*, SG24-5511

Other publications

These publications are also relevant as further information sources:

- *IBM DB2 Universal Database Command Reference*, SC09-4828
- *IBM DB2 Universal Database What’s New*, SC09-4848
- *IBM DB2 Universal Database Administration Guide: Planning*, SC09-4822
- *IBM DB2 Universal Database Administration Guide: Implementation*, SC09-4820
- *IBM DB2 Universal Database Administration Guide: Performance*, SC09-4821
- *IBM DB2 Universal Database Data Movement Utilities Guide and Reference*, SC09-4830
- *IBM DB2 Universal Database Data Recovery and High Availability Guide and Reference*, SC09-4831
- *IBM DB2 Universal Database Federated Systems Guide*, GC27-1224
- *IBM DB2 Universal Database Guide to GUI Tools for Administration and Development*, SC09-4851
- *IBM DB2 Universal Database SQL Reference, Volume 1*, SC09-4844
- *IBM DB2 Universal Database SQL Reference, Volume 2*, SC09-4845
Online resources

These Web sites and URLs are also relevant as further information sources:

- Database and Data Management home page
  http://www.ibm.com/software/data/
- DB2 Universal Database home page
  http://www.ibm.com/software/data/db2/udb/
- DB2 Technical Support
- DB2 online library
  http://www.ibm.com/db2/library
- DB2 for Windows
  http://www.ibm.com/db2/windows
- IBM WebSphere Application Server
  http://www.ibm.com/software/webservers/appserv
- IBM WebSphere InfoCenter
- IBM WebSphere Software Platform
  http://www.ibm.com/websphere
- IBM High Volume Web Site Team
- Sun’s Java 2 Platform, Enterprise Edition site
  http://java.sun.com/j2ee
- Sun’s Java Technology Products and APIs site
  http://java.sun.com/products/

**How to get IBM Redbooks**

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

**Help from IBM**

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services
Index

Symbols
.ear 83
.profile 360
.sqlj 111

A
AAT 39
access 81
access Intent 332
access intent 83–84, 233
access plan 250, 275
access type 85
activate 246
Activate at 214
Active 194
Active Threads 199
adapter 102, 104, 290
AdminClient API 192
administered network 25
administration model 25
administrator 311
advice 206
agent 13, 278
alert 209
algorithm 224
Ant 403
applet 23
application 24
Application Assembly Tool 38
approach 80
architecture 20, 107, 170
asntrc 347
assembly 24
Asynchronous Java Message Service 235
asynchronous message processing 235
asynchronous buffer writers 260
asynchronous page cleaners 260
attribute 322
autocommit() 334
Automation 4
ava Servlet Specification 76
availability 10
availability 89, 354

B
batch update 334
bean-managed 82
bean-managed persistence 233
benefit 322
binary 227, 342
binder 111
BLOB 227
block 334
BMP 233
bottleneck 197
buffer pool 258, 273, 390
built-in 342
business process 5
byte 385

C
C Common Client 109
C++ 269
cache 306
cache size 204
CacheMonitor.ear 192
CallableStatement 296
callback method 82
capacity 255
capture 342
catalog table 265
CCI 102
cell 25
CICS 81
class 250
classification 209
cleanup 290
cleanup interval 203
client 107, 170
ClientApplName 302
client-server 170
close 298
clone 197
cluster 96, 305
clustering 277

Avg Wait Times 310
CMP  78, 102, 233
CMR  79
COBOL  111
code page  267
collection increment  85
collection scope  85
Collector  358
collector.bat  358
collector.sh  358
com.ibm.db2.jcc.DB2Driver  109
COM.ibm.db2.jdbc.app  108
com.ibm.websphere.rsadapter  105
com.ibm.ws.rsadapter.cci  105
com.ibm.ws.rsadapter.jdbc  105
com.ibm.ws.rsadapter.spi  105
commit  114, 299
Common Client Interface  102
Compatibility Test Suite  20
complex  224
complexity  6
comply  170
component  354
component-based  77
concurrency  195, 270, 372
concurrency control  84
concurrency test  212
concurrent  196, 385
config  209
Configuration Advisor  253
connection  103, 385
connection management  101
connection pool  12, 103, 304–305
  Connection Timeout  310
  Maximum Pool Size  310
  Minimum Pool Size  310
connection pooling  114, 288, 350
connection sharing  308
connectivity  77
consistency  257
constraint  257, 373
consumption  339
container  23, 84, 261
container-managed  79
Container-managed persistence  102
container-managed persistence  233
contention  373
context  228, 349
contract  79, 101
cookies  76
core file  343
cost-benefit  11
count  250
CPU  11, 255, 379
criteria  89
CS  86
CTS  20
currentSchema  367
cursor stability  86, 270
curve  195
customizer  110

D
Data Replication Service  98
data source  12, 193
data type  256
database  305
datasource  88
DataStoreHelper  291
DB2 UDB configuration parameter
  BUFFPAGE  273
  CHNGPGS_THRESH  259
  CPUSPEED  250
  DFT_MON_BUFPOOL  273
  DFT_MON_TIMESTAMP  238
  DFT_QUERYOPT  250
  DIAGLEVEL  252, 342
  DIAGPATH  342
  INTRA_PARALLEL  277
  LOCKLIST  385
  LOCKSIZE  272
  LOGBUFSZ  284
  MAX_CONNECTIONS  278
  MAX_COORDAGENTS  278, 306
  MAXAGENTS  278
  MAXCAGENTS  278
  MAXLOCKS  284
  MON_HEAP_SZ  238
  NOTIFYLEVEL  342
  NUM_INITAGENTS  278
  NUM_IOSERVERS  279
  NUM_POOLAGENTS  278
  PCKCACHESZ  300
  SHEAPTHRES  281
  SORTHEAP  13, 281
db2advis  253
db2cctrc  347
DB2DIAG.LOG 251
db2drdat 347
db2evmon 249
db2efmt 250
db2expin 250
db2jcc.jar 109
db2sqljbind 111
db2sqljcustomize 111
db2sqljprint 111
db2support 343, 345
db2trc 346
deactivate 246
dead-ahead hint 85
deadlock 246, 305
default 119
deferPrepares 295
degradation 300
deploy 250
Deployer 22
deployment 6, 232
deployment descriptor 24
descriptor 232
Design Advisor 253, 276
diaglevel 252
dimension 256
dirty reads 217
disk 11, 255
DISPLAY THREAD 303
distributed system 8
DLL 110
DMS 258
domain objects 81
downstream 194
drawback 80
DRDA 108
driver 92, 107, 113
driverType 110, 152
DRS 98
dump 359
dumpNameSpace 39
dynamic 241
dynamic cache 223
Dynamic Cache Monitor 192
dynamic SQL 266, 300
dynexpin 250

E
e-business 2

EIS 101
EISs 102
EJB 77, 88, 94
EJB container 12, 88, 193, 202
EJB QL 80
EJB type
   Entity beans 81
   message-driven bean 82
   Session bean 80
ejbLoad 82
ejbStore 82
elimination 300
enableSQLJ 152
Encina 81
enterprise 202
Enterprise Information System 101
Enterprise JavaBeans 23
entity 78, 82
entity bean
   bean-managed persistence bean 81
environment 372
error 252
error mapping 291
escalation 283
Event Monitor 13
event monitor 246
exception 291
exceptions 289
extent size 261

F
factory 106
fail-over 89, 95
failure 96
Faults 310
FENCED 268
FFDC 342
FIFO 298
finalize() 309
finder 79
first-in-first-out 298
flexibility 258
formula 274
fragmentation 213
framework 170
frequency 288, 316
function 196
functionality 79
G

garbage collection 210
garbage collector 224
getConnection() 103, 308
global 112, 241, 297
global dynamic statement cach 300
global dynamic statement cache 294
g graphical user interface 34
Growable Thread Pool 200
GUI 34
guideline 338

H

handle 293
Health Center 252
Health Monitor 252
heap fragmentation 213
heavy 196
high-availability 93
high-volume 331
horizontal scaling 89, 96
host name 130
HTTP 91
httpd.conf 198
HttpServletRequest 316
HttpSession 76, 325
HttpSessions 318

I

I/O server 279
IBMDEFAULTBP 390
IBMSession 316
IDENTITY 256
life-cycle method 82
IHS 119, 357
IIOP 12
inbound 199
index 253, 257
indicator 252
init() 308
in-memory cache 192
instability 212
installation 118
instance 13
instance variable 308
integrated information 6
Integration 4
integrity 216, 318

interfaces 9
Internet InterORB Protocol 12
Interval 206
intra-partition 278
invalidator 327
investigation 341
IP_address 130
isolation 91
Isolation level 83, 332
isolation level 83, 216, 270
issue 206
iteration 199
iterations 304

J

J2EE 9, 19, 170, 193
J2EE Connector architecture 101
J2EE object
javax.ejb.EJBHome 313
javax.ejb.EJBObject 313
javax.naming.Context 313
javax.transaction.UserTransaction 313
Java 22, 110
Java 2 platform Enterprise Edition 20
Java Database Connectivity 22
Java IDL 22
Java Interface Definition Language 22
Java Message Service 22, 98
Java Naming and Directory Interface 332
Java servlet specification 76
Java Transaction API 22
Java Virtual Machine 77, 95
Java virtual machine 12
java.io.Serializable 312
java.sql 103, 291
JavaBeans 77
JavaMail API 22
JavaServer Pages files 23
javax.sql 103
JCA 101
JDBC 22, 92
JDBC driver 108
JMS 22, 98, 235
JMX 170
JNDI 36, 88, 332
JSP 24, 109
JTA 22
JTS 22
JVM 12, 77, 95

K
kernel 118
ksh 360

L
layer 170
lazy initialization 225
legacy 101
life-cycle 80
load 196
Load at 214
Load Balancer 354
load balancing 89
local 84
lock 272, 281, 283
Log Analyzer 180, 355
logical read 274
logWriter 351
loitering objects 226
long-running test 212
lsps 275

M
magnitude 224
maintainability 89
management 170
Management Beans 170
manual 317
many-to-many 79
Max Connections setting 194
MAX_COORDAGENTS 305
MaxClients 198
MaxConnections 310
MaxPoolThreads 198
MaxRequestsPerChild 221
MBeans 170
MDBs 235
mechanism 83, 318
memory 11, 224, 253, 255, 301
memory leak 212
Memory Visualizer 253
memory-to-memory 36, 311
menu 262
message 251
message consume 82
message-driven beans 235
messaging 312
method 80, 351
methodology 194, 391
middleware 107
modifiers 225
module 24
monitor 273
multiple 300
multi-tier 20

N
named pipe 247
Netbios 109
netstat 201
network 9, 11, 116
node agent 178
node_name 130
non-XA 314
normalization 256
NOT NULL 256
notification 170
notification log 342
notion 327

O
object 228
object pool 225
object reference 226
Object Request Broker 221
object request broker 12
objectifying the data 81
ODBC 107
off-peak 310
On Demand 2
one-to-one 79
online 241
open standard 7
optimistic read 234
optimistic update 234
optimization 7, 250
optimizer 250, 275
option 241, 300, 345
ORB 12, 96, 193, 202, 221, 223
Out of Memory exception 212
overflow 281
overhead 238, 253, 288
P
package 111
page 261
  clean 259
  dirty 259
  in-use 259
page-cleaner 259, 280
panel 206
parallelism 277–278
parameter 223
parameter markers 293
pass by reference 221
pass by value 221
PCKCACHESZ 301
PCTFREE 276
Percent Maxed 199, 310
Percent Maxed counter 202
Percent Used 310
permon 374, 393
performance 89, 170, 224
Performance Advisors 205
Performance Monitor 213
permission 278
persistence 79, 82, 105
persistent 311
persistent session 227, 312
pessimistic read 234
pessimistic update 233
phantom reads 217
physical design 254
physical read 274
physical storage device. 264
pkg_cache_num_overflows 301
pkg_cache_size_top 301
plug-in 92, 353
plugin-cfg.xml 92
PMI 170
policies 84
policy 84, 192
PoolThreadLimit 198
portability 291
portNumber 152
prefetcher 279
prepared statement 293
PreparedStatement 309, 334
preparedStatement 293
PreparedStatement object 296–297
PrepStmt Cache Discards 306
prerequisites 118
primary key 257
primary key. 81
problem determination 340
problem source identification 340
processing token 278
property 103, 110, 222, 367
  ClientAcctStr 302
  ClientUserid 302
  ClientWrkstnName 302
protocol 78, 114, 116
provisioning 7
publish 98
pull-down 314
Q
query 253
queue 193
queues
  closed 194
  open 194
R
rate 379
Rate of Change 187
ratio 274
Raw Value 187
read stability 86, 270
read-ahead 84
rebind 275
recommendation 206
Redbooks Web site 413
  Contact us xv
relational resource adapter 102
relationship 79
release 288
remote 84, 115
removeAttribute() 324
repetitive test 212
request 299
resource adapter prefetch 85
Resource Analyzer 180
ResourceException 290
response 288
response time 10
ResultSet 309
retrieve 247
return on Investment 10
reuse 299
RMI/IIOP 202
rollback 246
RR 85
RS 86
run-anywhere 22
runstats 275
runtime 291

S
sawtooth pattern 212
scalability 22, 89, 268
scenario 227, 372
schema 80
  multi-row 322
  single-row 322
scope 373
security 89, 101, 111
segment 292
self-configuring 9
self-managing 9
self-optimizing 9
Serialization 227
server 305
server-centric 20
serverName 152
servers 8
server-weighting 33
Service Provider Interface 102
service() 308
servlet 23, 94, 170
session 13, 227
session ID 76
session management 89
session manager 331
setAttribute() 324
severe 252
Shared Object 110
showlog 41, 358
single-row 322
SMALLINT 295
SMS 258
SNA 109
snapshot 238, 273
sort 253
sort heap 280
sort memory 13
SP 268
specification 20, 170, 221
speed 250
SPI 102
SQL 81, 111, 241, 265
SQLCODE 341
SQLSETI 302
SQLException 291
SqlException, 365
SQLJ 110
sqlj.zip 109
SSL session identifier 76
stale 291
stale connection 291
StaleConnectionException 289–290
startServer 157
state 252
stateful 234
stateful session 80
stateful session EJB 81
stateless session 80
stateless session EJB 80
statement 293, 309
statement cache 288
  Deferred Prepare 297
  PrepStmt Cache Discards 298
Statement Cache Size 297
static 111
static content 24
static object 308
static SQL 266
stderr 356
stdout 356
storage 8
stored Procedure 268
stream 88
stress 391
stripe 264
structure 80
subclass 290
subscribe 98
-Summary 359
svmon 119
swap file 11
switch 238, 273
sync() 316–317
Synchronization 229
synchronized method 229
syntax 80
sysplex 114
System Administrato 22
system test 212
System.err 355
System.gc() 224
System.out 355
SystemOut.log 206

table 256
temporary 263
text 359
thread 269, 305
thread pool 199
Thread Pool Configuration 200
Thread pool size 203
Thread Pools module 202
threads 305
THREADSAFE 269
thread-specific 327
ThreadsPerChild 198
throughput 10, 89, 95, 195
TIME_WAIT 201
Time-based 318
time-based 316, 318
Tivoli Performance Viewer 309
Tool Provider 22
topas 275
topology 89, 99, 118
toString() 225
total cost of ownership 8
TPV 208
trace 346
TRACE_ALL 351
traceFile 367
traceLevel 152, 367
trade3.acf 407
transaction 308
transaction management 101, 232
transaction-processing 81
translator 110–111
transport queue 199
trap file 342
two-phase 112, 114
Type 2 136
Type 4 136

unique key 257
unit-of-work 270
units-of-work 294, 298
UNIX 305
UOws 298
update 317
UR 86
URL rewriting 76
utility 359
utilization 197, 252

value 297
variables 226
vendor 80
Vertical clones 100
vertical scaling 89, 95
virtualization 4, 8
vmo 119
vmstat 119, 213
vmtune 119
volatile 277
volume 288

W
Waiting 194
warning 209
WAS Admin Console 199
WAS exception
ConnectionWaitTimeout 289, 309
ConnectionWaitTimeoutException 289
java.sql.SQLException 290
SQLException 291
StaleConnectionException 290
waslogbr 358
waslogbr.bat 358
WASPostUpgrade 38
WASPreUpgrade 38
Web container 76
web container 12, 88, 199
Web Performance Tools 372
WebSphere Admin Console 203
WebSphere plug-in 88
WebSphere Queuing Network 193
wizard 253
WLM 234
workload 113, 199, 253, 260
Workload Management 234

U
uncommitted read 86, 270
WPT 372
wrapper 104
write-once 22
wsadmin 178

X
-Xcompactg 213
XML 92
XML-SOAP 39

Z
z/OS 26, 112, 119
zero 297
zone 196
zSeries 26
This IBM Redbook discusses the integrated environment of DB2 UDB and WebSphere Application Server (WAS), including design considerations, best practices, operation, monitoring, and performance tuning.

We provide an overview of the architecture and main components of both WebSphere Application Server V5.0.2 and DB2 UDB V8. We introduce their key application and system performance tuning parameters.

We discuss the general steps to get DB2 UDB V8 and WAS V5 working together. We also discuss the available JDBC drivers shipped with DB2 UDB V8 and their differences. The steps to set up DB2 for z/OS as the data source using DB2 Connect is also included.

We describe the performance tuning tools, methodology, guidelines, and the application performance best practices for DB2 UDB, WAS, and the DB2 UDB/WAS integrated environment. Finally, we present some common methods of diagnosing the problems related to DB2 UDB and/or WebSphere Application Server. We provide some problem scenarios where you will learn the problem determination methodology and mechanisms to resolve such problems.