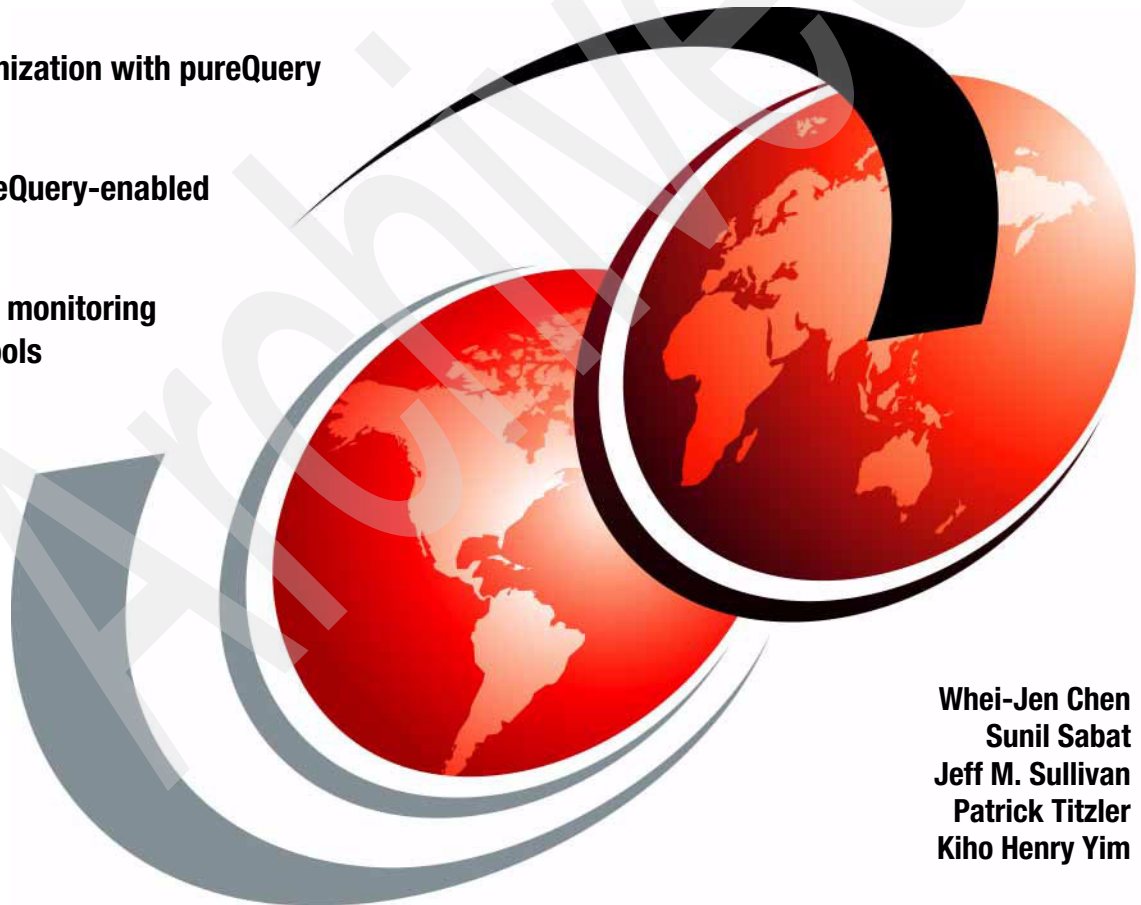


Using Integrated Data Management To Meet Service Level Objectives

Client optimization with pureQuery

Deploy pureQuery-enabled application

End-to-end monitoring with IBM tools



Whei-Jen Chen
Sunil Sabat
Jeff M. Sullivan
Patrick Titzler
Kiho Henry Yim



International Technical Support Organization

Using Integrated Data Management To Meet Service Level Objectives

November 2009

Archived

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

First Edition (November 2009)

This edition applies to IBM Optim Developer Studio Version 2.1 and Version 2.2.

© Copyright International Business Machines Corporation 2009. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
Preface	ix
The team who wrote this book	x
Acknowledgment	xi
Become a published author	xii
Comments welcome	xii
Chapter 1. Data management life cycle	1
1.1 The third question: Why did this happen?	2
1.1.1 Why answering “why?” is so hard	2
1.1.2 Management within the larger context	4
1.2 Integrated data management	5
1.2.1 Integrated data management portfolio	6
1.2.2 Portfolio integration	9
1.2.3 Heterogeneous flexibility	10
1.3 DBA challenges in modern application environments	11
1.3.1 Inability to identify poorly performing applications	11
1.3.2 Lack of insight into source code for database impact analysis and SQL tuning	11
1.3.3 Security risks inherent with dynamic SQL	12
1.3.4 Inadequate metrics for capacity planning and prioritization	13
1.4 How pureQuery addresses those challenges	13
1.4.1 What is pureQuery and how does it help?	13
1.5 The Great Outdoor Company	15
1.5.1 The products used in this book	16
1.5.2 Integrated data management in action	17
Chapter 2. Optimizing existing Java applications	19
2.1 Scenario description	20
2.1.1 Background on DB2 bind process	20
2.2 Application optimization considerations	22
2.2.1 Improving data access performance	22
2.2.2 Improving application manageability	25
2.2.3 Enhancing security	32
2.3 Monitoring with Tivoli OMEGAMON XE for DB2, DB2 Query Monitor, and DB2 Optimization Expert	33
2.3.1 Correlating SQL with its originating application	34

2.3.2	Dynamic and static SQL performance comparison	38
2.3.3	Using OE to further improve SQL performance	39
2.4	Optimizing an existing Java application using pureQuery	51
2.4.1	PureQuery client optimization process overview	51
2.4.2	Setting up pureQuery	52
2.4.3	Capturing SQL from a Java application	66
2.4.4	Configuring the captured SQL metadata	73
2.4.5	Binding the captured and configured SQL	79
2.4.6	Configuring the Runtime to execute SQL statically	82
2.4.7	Enabling client optimization using command line utilities	84
2.4.8	Application maintenance	87
2.5	Evaluation	88
Chapter 3. Managing pureQuery application deployment		89
3.1	Scenario description	90
3.2	Deployment with pureQuery	90
3.2.1	Common deployment process	90
3.2.2	pureQuery dynamic SQL application deployment	91
3.2.3	pureQuery static SQL application deployment	92
3.3	Deployment planning	92
3.3.1	Identifying deployment artifacts	92
3.3.2	Identifying software prerequisites	94
3.3.3	Identifying target system requirements	94
3.3.4	Identifying post deployment tasks	95
3.4	Installing the pureQuery Runtime	96
3.5	Package deployment using pureQuery bind utility	97
3.5.1	Process using pureQuery StaticBinder	98
3.5.2	The scenario usage of pureQuery StaticBinder	100
3.6	Configuring the pureQuery Runtime to process SQL statically	109
3.7	pureQuery application maintenance	111
3.7.1	Application changes	113
3.7.2	Software dependencies	116
3.7.3	Schema changes	117
3.7.4	Access path changes	117
3.7.5	Maintaining packages using Data Studio Developer	117
3.8	Evaluation	118
Chapter 4. Managing enterprise systems with PE/EI and ITCAM		119
4.1	The need for enterprise management	120
4.2	Monitoring with PE/EI	121
4.2.1	Performance Expert Extended Insight Feature	122
4.2.2	Performance Expert	128
4.3	Monitoring with ITCAM	143

4.3.1 ITCAM introduction	143
4.3.2 Configuring ITCAM for WebSphere for application monitoring	144
4.3.3 Application monitoring using ITCAM	146
Chapter 5. Optim Development Studio and Optim pureQuery Runtime: A closer look at version 2.2	153
5.1 Scenario description	154
5.2 Improving database access performance for Java applications	155
5.2.1 SQL literal substitution	155
5.2.2 Improve query performance using SQL replacement	163
5.2.3 Apply SQL development best practices and prevent rogue queries	171
5.3 Improving database access security for Java applications	179
5.3.1 Preventing SQL injection	179
Related publications	183
Other publications	183
Online resources	185
How to get Redbooks	185
Help from IBM	186
Index	187

Archived

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

Trademarks

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

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

CICS®
DB2®
developerWorks®
IBM®
IMS™
Informix®
InfoSphere™

OMEGAMON®
Optim™
OS/390®
Rational®
Redbooks®
Redpaper™
Redbooks (logo) ®

System z®
Tivoli®
VTAM®
WebSphere®
z/OS®

The following terms are trademarks of other companies:

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

Hibernate, and the Shadowman logo are trademarks or registered trademarks of Red Hat, Inc. in the U.S. and other countries.

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

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

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

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

Preface

In this IBM® Redbooks® publication, we learn how The Great Outdoor Company, a fictional retailer of outdoor products, uses IBM Integrated Data Management solutions (in particular Optim™ solutions and other data management tools) to optimize data management for performance, productivity, and manageability, allowing them to get the most from their data assets and to ensure they can meet service level objectives (SLOs). We discuss common organizational situations, which can be complex, and how organizational and infrastructure complexities can negatively affect overall performance and productivity. We discuss how an integrated data management approach can help bridge these silos and gaps to provide end-to-end data and data application life cycle support.

Although the vision of integrated data management extends beyond IBM data servers, this book focuses on DB2® for z/OS® and DB2 for Linux®, UNIX®, and Windows® as the underlying data servers to illustrate the benefits and capabilities of an integrated data management approach. To illustrate, we focus on how to optimize an enterprise Java™ application using the Optim Development Studio (formerly Data Studio Developer) and pureQuery Runtime to improve performance, manageability, and security. We use DB2 for z/OS data management tools to see the results of that optimization. We describe the features that developers and DBAs can use to maximize the manageability of their Java applications using the pureQuery data access platform.

We discuss the practical aspects of managing the deployment and maintenance of pureQuery applications in the context of the usual enterprise application considerations such as application changes, software dependencies, and database object changes.

Because an integrated data management approach extends to the problem-solving aspect of the data life cycle, we also discuss how DB2 Performance Expert, DB2 Performance Expert Extended Insight Feature (PE/EI), and IBM Tivoli® Composite Application Manager can help you pinpoint problems in complex application environments.

The team who wrote this book

This book 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, and an IBM Certified IT Specialist.



Sunil Sabat is a member in the DB2 Advanced Support team. In his current role, he helps customers solve their technical problems. He has had various positions in IBM for 10 years. Mostly, he has worked with external partners selling and enabling them on information management technology. Sunil has presented papers at various IBM and external conferences. Before joining IBM, Sunil worked at Intel®, Cadence, I2, and other start-up companies.

Sunil is IBM, Microsoft®, SNIA, and CompTIA certified professional. He is a Master Certified IT Specialist. He holds Master's degrees in Computer Science, computer engineering, and business administration. His interests include toastmasters, technology, volunteering, and traveling.



Jeff M. Sullivan is a Software Enabling Architect in the IBM Software Group. In over 29 years, he has been involved in most facets of IT, from operations and telecommunications, to programming and database administration and software installation. His expertise is DB2 performance tuning and installation, DB2 and IMS™ Database Administration, and finding ways to work smarter and faster. He can be reached at jeffsull@us.ibm.com.



Patrick Titzler is a member of the Optim Technical Enablement team in the IBM Software group, focusing on optimization of Java database applications with IBM Optim Solutions. He has many years of development experience, having contributed to various business intelligence and data warehousing products supporting DB2 for LUW and DB2 for z/OS. Patrick holds a Master's degree in Computer Science from the University of Rostock, Germany.



Kiho Henry Yim is in the Technical Sales Support team working with DB2 z/OS tools in US East Region. He has held various IT positions after joining IBM in 1976 as a Systems Engineer. His experiences include working as a DBA, developer, architect, and system programmer in z/OS DB2 tools, SAP® database server on DB2,

IDMS, ADABAS, IMS, and CICS®. He was also a contributing writer to *Building the Operational Data Store on DB2 UDB Using IBM Data Replication, WebSphere MQ Family, and DB2 Warehouse Manager*, SG24-6513. He is an IBM certified IT specialist and holds an Electrical Engineering degree from Seoul National University, Korea.

Acknowledgment

The authors thank Bradon L. Waters for his contribution in written content.

Brandon Waters is an Advisory Software IT Specialist in the Information Management brand of IBM Software Group. Focusing on pre and post technical sales support for US Federal Agencies, his expertise is in with DB2 and database tools, data warehouse architecture, data modeling, and more recently IBM Integrated Data Management suite. Brandon holds a masters of Electrical Engineering from Virginia Tech in Blacksburg, Virginia.

The authors express their deep gratitude for the help they received from **Kathryn Zeidenstein** from the IBM Software Group at IBM Silicon Valley Laboratory.

The authors thank the pureQuery Runtime and Optim Development Studio teams for their guidance in producing this book.

The authors also thank the following people for their contributions to this project:

Jeffrey Cauhape
Jaijeet Chakravorty
Holly Hayes
Deb Jenson
Gaurav Shukla
Manoj K Sardana
Christopher M. Farrar
Bill Bireley
Anshul Dawra
IBM Software Group

Paul Wirth
IBM Sales and Distribution

Sadish Kumar
IBM Global Services

Emma Jacob
Rich Conway
International Technical Support Organization, San Jose Center

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

To obtain more information about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

- ▶ Use the online **Contact us** review Redbooks form found at:
ibm.com/redbooks
- ▶ Send your comments in an e-mail to:
redbooks@us.ibm.com
- ▶ Mail your comments to:
IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Data management life cycle

This chapter describes the theme and setting for this IBM Redbooks publication.

This chapter begins by describing a typical organization that needs to monitor and tune the performance of an existing infrastructure to meet service level objectives. Meeting service level objectives requires that an organization have the ability to measure the service they are delivering against their objectives, to understand when they are not meeting them, to perform problem triage when things are not working as they need, to tune the systems for adequate performance, and, hopefully, to improve practices to prevent future recurrences.

For any organization to perform these tasks, the information technology (IT) professionals require effective tooling to assist them in their execution of their duties. It is not uncommon for IT staff to be asked to do more work with fewer resources. At the risk of sounding cliché, this is forcing IT organizations to find ways to work smarter and not just harder. Working smarter requires additional information and expertise be made available to the IT staff. Thus, one can see the need for fully-integrated tooling to manage systems and applications from inception to retirement.

In this chapter, we discuss organizational complexities (and their effects on performance) commonly experienced by IT organizations, the tooling to help with end-to-end full life cycle support, and integrated data management (IDM) as an approach to addressing these complexities. We also introduce pureQuery, IBM technology that provides a fundamental underpinning for much of the value in the integrated data management solutions described in this book.

1.1 The third question: Why did this happen?

It is Monday morning and the database systems support staff start arriving at the office. After the usual pleasantries, each settles into their respective cubicles to start their day. The on-call phone is passed to the next on-call person during a brief turnover meeting, followed by a review of what happened over the weekend.

As they dig deeper into their problem ticket research, group members generally follow a pattern of three questions:

- ▶ “What happened?”
- ▶ “Who or what group did this?”
- ▶ “Why did this happen?”

The database group, collectively, received a number of problem tickets and starts the arduous task of researching root cause for each ticket to answer the third question:

- ▶ Could this problem be an anomaly or is this a new steady state? Did we have a spike in activity that forced the database system into an unusual condition where it was unable to meet performance objectives?
- ▶ Was this problem caused by the introduction of new work to the system? Did the application team not perform the required due-diligence necessary before they rolled in these changes into production?
- ▶ Is this a new system? How did this new system slip past our change management controls we have in place? Perhaps we do not have an adequate feedback loop for changes to the system.

The third question is the key question to answer because we ultimately want to answer a fourth question: “How can we prevent this from happening again?”.

1.1.1 Why answering “why?” is so hard

Answering these questions and handling problems as they occur are a critical part of IT responsibilities. By definition, we have a problem when a service level objective (SLO) or service level agreement (SLA) is not met. A server failover is not a problem in and of itself. Rather, it is as a result of a possible problematic event. That event could have been prevented had the system or application been installed with SLOs and SLAs in mind.

An IT shop can choose to ignore the problem or to perform a quick-fix solution. While choosing to ignore problems is a legitimate business model (prioritizing and handling those items that are most impactful), it can end up being an

expensive one. Quick-fixes, or workarounds, might get the organization functioning quickly by handling the issue tactically, but frequently do not address the root cause of the issue, leaving the door open for repeat performances. These tactical approaches are often the only approaches, and if used as the primary model for problem resolution, can irreparably hurt the business.

Organizations need to also think strategically about how to prevent the problem from reoccurring, but thinking strategically requires knowledge of why the problem occurred, which is in and of itself a challenge. The organizations that struggle to learn why something happened tend to have the following characteristics:

- ▶ Lack sufficient expertise among the staff.
- ▶ Lack availability of appropriate service metrics.
- ▶ Lack an effective monitoring solution.

There are a variety of reasons why organizations lack sufficient expertise. Perhaps there can only be a single person that knows the application and knows the intricate details of its usage. Or there could be high turnover in the shop, a small staff doubling up on tasks, or a lack of documentation. Training budgets are tight or possibly nonexistent.

As a result of a thin staff or insufficient training, critical data is not gathered. In many cases, service metrics are not gathered due to inflexible infrastructure policies, which might not allow certain tooling to be installed. There could also be a lingering perception from earlier days that tools are not at a maturity level adequate to gather the wide range of data to perform proper problem triage.

Many organizations do not have the right tooling to identify causal factors to enable them to fix the problem. Not having the correct information to perform problem triage can create problems, protracting problem resolution or leading to a hunt and peck approach to the problem. Adoption of tools can also be hindered by a “not invented here” mentality. There might be a make-or-buy issue to contend with in determining the best tooling to use in problem determination.

In any case, these three issues, expertise, metrics, and monitoring, can be addressed with smarter tooling. Specifically, with tools that can gather the correct data, gather it autonomously, and monitor key performance indicators tuned to support the organization’s SLOs and SLAs.

1.1.2 Management within the larger context

Compounding the issues described above is that it is so difficult to pinpoint and remedy a specific problem that might have occurred in a complex infrastructure that consists of platforms, operating systems, and database management systems (DBMSs) that can be configured together to comprise an organization's enterprise architecture. Regarding problem management, the questions "Why did this happen?" and "How can we prevent this from happening again?" need to be viewed not only within the immediate contextual circumstances but also with an eye on causes from (or impacts to) the larger enterprise infrastructure. It is difficult for smaller businesses to have the required level of communication and expertise across their people and infrastructure and nearly impossible for medium to larger organizations. For this reason, tools for effective problem management need to help IT staff gather and synthesize relevant information across systems.

In addition, the ability to predict and prevent problems is more than just a passing nicety, but has become an integral part of problem management. This ability to predict is tied closely to the ability to see current and historic performance data. In addition, being able to predict usage is key to help organizations plan for and contain costs for the business by maximizing all available capacities across servers, direct access storage devices (DASD), CPU engines, and memory. We need to look at data end-to-end, from the application server to the DBMS server. To facilitate this process and to facilitate communication among IT roles, it is important that key metrics from the various architectural components be automatically correlated and visualized through integrated tooling.

Looking at the bigger picture, it becomes apparent that any problem we have in performance could have been prevented and addressed earlier. It would have been nice if we could have caught a potential bottleneck earlier, even at the design phase. It is possible that the problem might have arisen from a poor initial data model. Even finding out that the problem is relatively simple and can be solved by adding an index is something that causes less impact and cost when found before applications get rolled into production.

The question of application management really requires a fully integrated, yet modular, solution set to address the entire scope of issues that could arise. The solution should include functionality to allow IT professionals to perform the following tasks:

- ▶ Create an initial application design.
- ▶ Create a physical model from a logical model and vice versa.
- ▶ Generate data definition language (DDL) from the physical model.

- ▶ Generate the structured query language (SQL) to access data from the generated database.
- ▶ Add efficiencies of performance to the SQL code by using statically bound packages.
- ▶ Load data to the new database efficiently and quickly.
- ▶ Simplify the administration of the enterprise database environment.
- ▶ Automate the performance by applying service level agreements and objectives to the production environment.
- ▶ Manage growth with solid measurement statistics and metrics.

1.2 Integrated data management

IBM is leading the charge for integrated data management, because of the conviction that moving away from silo's capabilities to integration and collaboration can help enterprises achieve the following goals:

- ▶ Grow the business, by increasing revenues at less cost. Integrated data management enables organizations to accommodate new initiatives such as customer retention programs and entering new markets, without expanding infrastructure.
- ▶ Simplify their infrastructures, including the ability to centralize shared services, thereby reducing the cost of doing business and reducing overhead. Application upgrades, consolidation, and retirement activities are simplified.
- ▶ Adhere to data governance regulations by facilitating alignment, consistency, and governance by defining business policies up front and then sharing, extending, and applying them throughout the data life cycle, thereby reducing risk and avoiding possibly costly litigation.

Integrated data management is about creating an integrated, yet modular, data management environment to design, develop, deploy, operate, optimize, and govern data, databases, and data-driven applications throughout the entire data management life cycle, from requirements to retirement. It is inclusive of problem management, but is much broader. At the heart of integrated data management is the conviction to reap benefits across the life cycle by providing greater integration in the tooling infrastructure through shared policies, artifacts, and metadata. The benefits include enabling more task automation, better alignment and collaboration across teams, and better systems and organizational performance. Figure 1-1 on page 6 illustrates this concept.

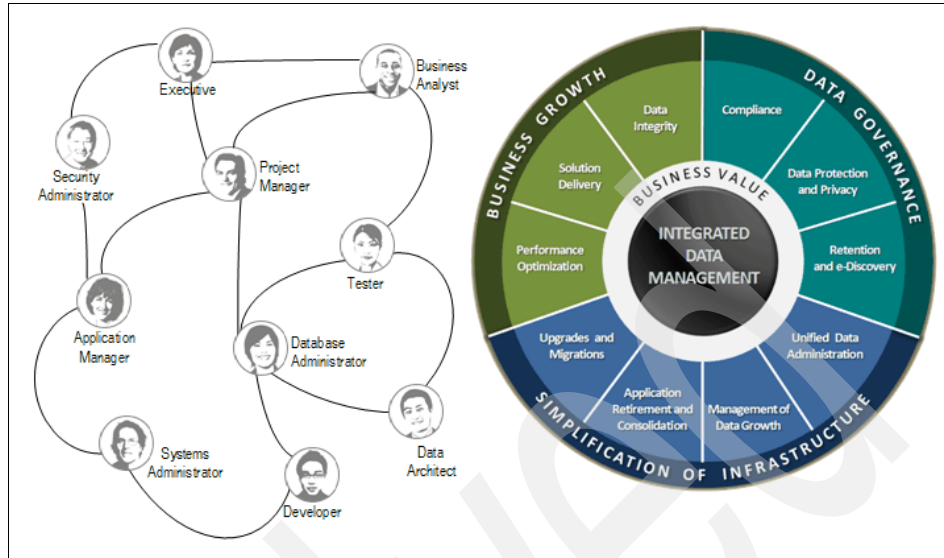


Figure 1-1 Integrated data management helps address business problems

The IBM Integrated Data Management approach helps business and IT work together towards the achievement of business objectives so that there is common understanding, transparency, and consistent execution across the business.

Maintaining alignment is about communication, collaboration, and clarity across organizational roles. Users and business analysts need to capture requirements. Architects are responsible for designing the process, application, and data models.

Developers must produce effective and efficient code using those models. Administrators must understand security and retention policies established by compliance officers and work with their network and systems administration colleagues to achieve compliance and service agreements. It is critical to the effectiveness and productivity of the organization as a whole that the tools each person uses effectively share data assets across roles and tasks.

1.2.1 Integrated data management portfolio

IBM is leading the industry in providing integrated data management environments. Individual tools provide powerful capabilities that target specific data management roles and tasks. More importantly, however, the offerings interoperate, enabling cross-role collaboration, productivity, and effectiveness.

Integration does not end with the Optim offerings, but extends into the InfoSphere™, Rational®, WebSphere®, and Tivoli portfolios as well.

For more details on the portfolio and the solutions offered by IBM, see the following Web page:

<http://www.ibm.com/software/data/optim/>

The following products are some of the IBM Optim solutions tools for integrated data management that are described in this book:

- ▶ Optim Development Studio (formerly Data Studio Developer)
This tool provides a complete development and testing environment for building database objects, queries, database logic, and Java pureQuery applications that can increase development productivity 25-50%.
- ▶ Optim pureQuery Runtime (formerly Data Studio pureQuery Runtime)
This tool delivers a high-performance data access platform that provides the capability to visualize, optimize, and lock down SQL without modifying the application. Complements Optim Development Studio by enabling pureQuery support in production systems, including support for static execution of .NET and Java applications.
- ▶ DB2 Performance Expert and DB2 Performance Expert Extended Insight Feature
This tool provides a comprehensive and proactive performance-management solution for database applications that access DB2 databases on Linux, UNIX, and Windows. The Extended Insight Feature complements Performance Expert by delivering end-to-end database transaction monitoring for Java database applications.
- ▶ Optim query tuning solutions
This tool gives DBA expert advice for tuning single queries (for both DB2 for z/OS and DB2 for Linux, UNIX, and Windows) or query workloads (currently for DB2 for z/OS only) to optimize performance and reduce cost. The Optim Query Workload Tuner for DB2 for z/OS was previously known as DB2 Optimization Expert for z/OS).

Figure 1-2 on page 8 is an illustration of how IBM views the complete life cycle as it pertains to data, and what issues our integrated data management suite of products are addressing. Today, most organizations have a myriad of tools in-house from many vendors supporting different roles and tasks. Each focuses on providing rich task-specific value, but puts little emphasis on linkages with the preceding or next phase in the life cycle.

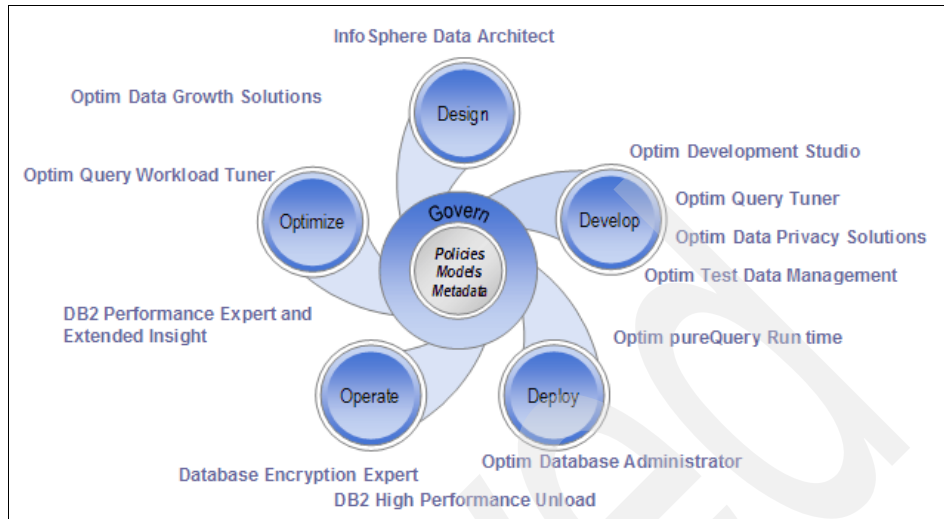


Figure 1-2 Integrated data management capabilities by Optim solutions

With Optim data management solution software, each phase of the life cycle is supported with robust offerings for data-centric tasks and roles. Optim provides support for designing and implementing key cross-phase linkages. The following list defines the key phases in data-centric software development:

- ▶ **Design**
Discover, harvest, model, and relate information to drive a common semantic understanding of the business.
- ▶ **Develop**
Code, generate, test, tune, and package data access layers, database routines, and data services.
- ▶ **Deploy**
Install, configure, change, and promote applications, services, and databases into production.
- ▶ **Operate**
Administer databases to meet service level agreements and security requirements while providing responsive service to emergent issues.
- ▶ **Optimize**
Provide pro-active planning and optimization for applications and workloads including trend analysis, capacity and growth planning, and application retirement including executing strategies to meet future requirements.

- Govern

Establish, communicate, execute, and audit policies and practices to protect and retain data in compliance with government, industry, or organizational requirements and regulations. Not limited to a single phase, governance is a practice that must infuse the entire life cycle.

1.2.2 Portfolio integration

The portfolio encompasses various offerings including InfoSphere, Optim, and Data Management tools. There are broad and deep capabilities for every phase in the life cycle, but what differentiates IBM offerings is the value-added integration across the portfolio (either in current product or in roadmaps) with common user interfaces, common components and services, and shared artifacts.

- Common user interfaces

Whether Eclipse-based or Web-based, Data Studio is adopting a standard and integrated approach to user interfaces that makes moving between roles easy and intuitive. The portfolio includes an Eclipse-based user interface for design, development, object, and command-centric tasks. Here, the Data Studio tools complement and extend the IBM Rational Software Delivery Platform. The fully integrated nature of IBM Data Studio and Rational software simplifies collaboration among business analysts, architects, developers, and administrators. Users can combine tools within the same Eclipse instance, providing seamless movement between tasks. Or, users can share objects across geographically distributed teams to make it easier to maintain alignment and work more efficiently.

By contrast, operations support requires the ability to monitor and respond from anywhere at any time. The Web-based user interface supports operations-oriented administration. Adopting a common approach with Tivoli software for Web-delivered dashboards and portlets provides the greatest flexibility for monitoring, management, and coherent information across the operational stack to improve an organization's ability to meet SLAs. Sharing these capabilities across data servers reduces overall skills requirements and costs. For our z/OS base, existing 3270 interfaces continue to be supported and extended so long as customer investments demonstrate that they are desirable and sustainable.

- Common components and services

Sharing components and services across offerings help organizations achieve cost, productivity and consistency objectives. For example, by using components installed in a common shell instance, your organization can minimize the footprint on the client machine and reduce tool deployment costs. When the tools share components, such as the Data Source Explorer,

it is easier to transfer skills to new products. Shared services, such as data privacy services, means personal identification numbers will be handled consistently whether creating test data or sharing research data.

- ▶ Shared policies, models, and metadata

This is the glue that truly holds everything together. The ability to express policies for machine interpretation, to associate policies with data models or data workloads, and communicate both through shared metadata is the crux of the challenge as well as the critical linchpin for greatest value. Sharing metadata, development, and design artifacts can improve software development alignment, control, and efficiency. Shared configuration information between database administrators and application server administrators can reduce deployment costs while improving quality of service. Shared policies together with the services that implement them can improve security and compliance.

1.2.3 Heterogeneous flexibility

Recognizing the heterogeneity of most organizations, the vision spans IBM and non-IBM databases. While we deliver first on DB2 and Informix® Dynamic Server databases, we are building out the portfolio across Oracle®, Microsoft, Sybase, and leading open source databases.

Whether a data architect, developer, tester, administrator, or steward, the integrated data management portfolio by IBM has capabilities that can help you be more effective and efficient. More importantly, the portfolio and roadmap are delivering a collaborative environment that enables increased productivity and efficiency to make organizations more responsive to opportunities, improve quality of service, mitigate risk and reduce costs for diverse data, databases, and data-driven applications.

For more detailed discussion about IBM Integrated Data Management, refer to the developerWorks article *Integrated Data Management: Managing data across its lifecycle* at the following Web page:

<http://www.ibm.com/developerworks/data/library/techarticle/dm-0807hayes/>

1.3 DBA challenges in modern application environments

There has been an explosion of growth in data access applications that rely on Java-based technologies. This growth has occurred for many reasons, including the growth of vendor-packaged applications that use Java for portability and the rise of Web applications for critical business applications. Mainframe data is a full player in this environment, because critical business data often resides on the mainframe. In addition, favorable pricing available with specialty processors (zAAP and zIIP) on System z encourages applications to run on or against the mainframe.

Along with growth and business importance comes challenges to the DBA who has to manage the data and performance requirements for these applications. In the following sections, we look at some of these challenges in detail.

1.3.1 Inability to identify poorly performing applications

This problem arises because most Java (and .NET) applications run under the same generic identifier used for all dynamic calls. Unless special identifying information is coded into the application (which is a best practice not commonly followed), everything using dynamic SQL will appear under the same identifier in your online monitoring tool. It will be difficult to pinpoint which application is causing a problem. We discuss more about this in Chapter 2, “Optimizing existing Java applications” on page 19.

1.3.2 Lack of insight into source code for database impact analysis and SQL tuning

Let us assume that even though you cannot pinpoint the exact application that is causing the problem, you do manage to extract the bad SQL from your monitoring tool. You walk down the halls (or send politely worded e-mails) in an attempt to find someone, anyone, who recognizes that statement and can fix the problem in the source code.

It is entirely possible that no developer will recognize the SQL because they have never even seen it. The historical background for this is that to address the exploding demand for Java applications, a wide plethora of tools and frameworks rose up to make developers more productive in developing these new applications. Whereas the data access layer was traditionally done through JDBC, which provided developers with a lot of control over the database calls, it

also required a significant coding effort to execute the SQL, retrieve the result sets, and process them such that it could be consumed by the application's business layer.

Several technologies have been developed over time that abstract the developer (to various degrees) from the low level database operations. The general motivation behind these developments was that developers should be able to focus their development efforts on the business logic and let a sophisticated persistence technology (also referred to as Object Relation Mapping [ORM]) take care of the details of how the business objects are represented, accessed, or manipulated in the database. To further abstract the developer from the underlying SQL operations, some technologies have defined their own proprietary query languages that are internally mapped to SQL. Some of the most popular persistence frameworks are Hibernate™ and various implementations of the Java Persistence Architecture (JPA) specification.

The adoption of these technologies has a fundamental impact on the database and the DBA. First, if SQL that accesses and manipulates data is generated by the framework based on some generic rules, performance is likely to be sub-optimal (for example, unnecessary joins). Second, if the framework generates SQL, how would the developer know it came from his or her code, or how would you (or the developer) know which operation in the application has caused this SQL to be executed, if any problems arise? Third, assuming that you need to make a schema change, how will developers be able to correlate the changed table or column with the relevant queries in the Java source to determine if those queries need to change?

1.3.3 Security risks inherent with dynamic SQL

Although SQLJ exists to provide static access for Java applications, most Java (and .NET) applications use dynamic SQL execution. With dynamic SQL, the DB2 DBA has to give appropriate table or view access privileges directly or indirectly to the program's users. This might not seem like a problem, but it means that you have given permission to a user, who, if they find another way to access that table, could execute any SQL statement that their table-level authority allows. A malicious user could use this permission to retrieve or modify table data in unexpected ways.

Another security vulnerability that can occur with dynamic SQL is known as *SQL injection*, which means that an application that is creating the SQL string does not properly sanitize the input (such as cleaning up escape characters), which can lead to all kinds of interesting vulnerabilities. With static SQL, you do not have that problem because the application is not building up a SQL statement for execution at run time. The statement is known at bind time, so the data server can perform type checking and validation prior to execution, preventing

unexpected problems at run time. For more information about SQL injection, see Chapter 5, “Optim Development Studio and Optim pureQuery Runtime: A closer look at version 2.2” on page 153.

1.3.4 Inadequate metrics for capacity planning and prioritization

DB2 for z/OS has a long history of providing extensive metrics for performance tuning and for capacity planning. But the use of dynamic SQL by modern Web applications means that there is a lack of an identifier to correlate accounting information to particular workloads.

A similar problem exists for the use of Workload Manager to classify and prioritize workloads. Without a way to correlate a particular workload with an identifier, all the threads associated with dynamic SQL are treated the same and cannot be classified using WLM goals.

1.4 How pureQuery addresses those challenges

pureQuery is a fundamental component of the IBM Integrated Data Management strategy. It provides an innovative solution to the challenges raised in the section above by helping enterprises to perform the following tasks:

- ▶ Reduce costs by improving system throughput as well as DBA and developer productivity.
- ▶ Improve quality of service for new and existing Java and .NET applications by stabilizing or improving performance and reducing problem resolution cycles.
- ▶ Reduce development time for new Java applications by bridging common Java and SQL development tasks, and balancing productivity and control.
- ▶ Enhance security by limiting user access, minimizing SQL injection risks, and improving audit readiness.

1.4.1 What is pureQuery and how does it help?

pureQuery is a high-performance, data access platform to simplify developing, managing, securing, and optimizing data access for new and existing applications. IBM calls pureQuery a platform because it seems to be the best word to express the vision of pureQuery as a technology that spans the data management life cycle. In other words, there are aspects of pureQuery that are embedded in design, development, test, deployment, monitoring, and optimization.

pureQuery consists of the facets shown in Figure 1-3, which can be used as needed for any particular project or task and that are delivered in specific products related to those tasks.

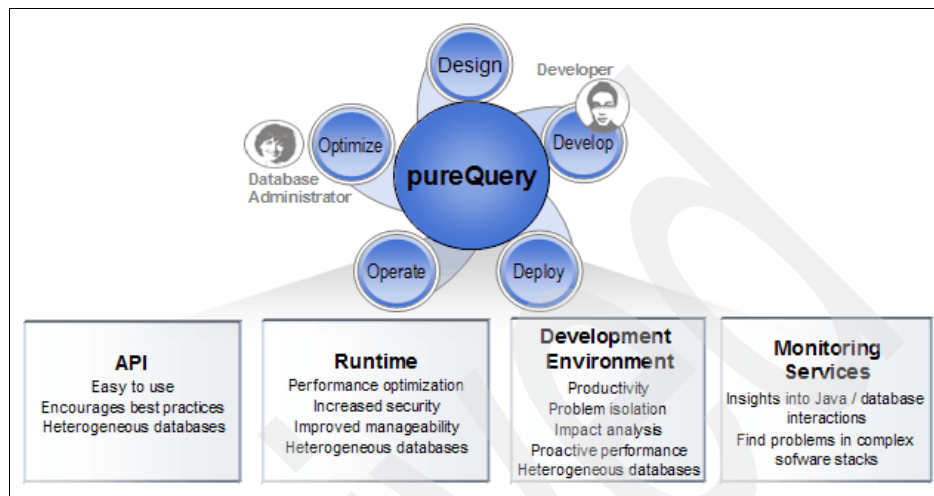


Figure 1-3 pureQuery data access platform

- ▶ Application programming interfaces (APIs), built for ease of use and for simplifying the use of best practices for enhanced database performance when using Java. A key capability of this API is the ability to develop using dynamic SQL and then deploy using static SQL, with no change to the application. In addition, the API includes capabilities that are not even included in JDBC for performance optimization using a capability called *heterogeneous batching of updates*, which means updates of multiple tables can be sent in a batch across the network. We discuss using the API in Chapter 3, “Managing pureQuery application deployment” on page 89.
- ▶ A Runtime, which provides optimized and secure database access, delivered in Optim pureQuery Runtime. A key capability of the Runtime that relates to DBA challenges discussed above is its ability to run Java (or .NET) applications statically, with no change to the application code. This holds true not just for applications that use the pureQuery API, but also for any existing Java applications, even packaged applications or those using a framework such as Hibernate or OpenJPA. The ability to run statically existing Java or .NET applications built for dynamic SQL is called *client optimization*. We discuss client optimization in Chapter 2, “Optimizing existing Java applications” on page 19.

- ▶ An Eclipse-based integrated database development environment for enhancing development productivity, improving data access performance during development, and improving collaboration between developers and DBAs, delivered in Optim Development Studio. For example, with Optim Development Studio, all SQL statements in an application can be visualized as a list (even if the originating query was written in an object query language other than SQL). From that list, you can jump to the originating line of source code, making it much easier to find and fix problematic SQL statements. There is also a way to view the tables and columns referenced in the application so you can determine if a schema change will impact a query in the application.
- ▶ Monitoring services, to provide developers and DBAs with previously unknown insights about performance of database applications. This capability is primarily delivered in DB2 Performance Expert Extended Insight Feature for Linux, UNIX, and Windows, but monitoring services also support development. Optim Development Studio enables developers and DBAs to visualize metrics to identify SQL hot spots quickly and easily.

As you can see, some of the value of pureQuery is in the ease with which it is possible to make use of static SQL execution. Not only does static execution provide performance benefits, it helps with manageability, by providing specific identifying information for accounting traces, for workload prioritization using Workload Manager, and for online monitoring, especially for those using DB2 for z/OS. The use of these products and capabilities to help DBAs with their data management responsibilities are described in more detail throughout this book.

The rest of this book includes detail on how pureQuery can help with data management challenges with Java application environments.

1.5 The Great Outdoor Company

In this book, we demonstrate the functionality of the IBM Integrated Data Management portfolio by using the various tools in the suite to assist The Great Outdoor Company, a fictional retailer of outdoor products. This fictitious company uses Optim solutions to meet specific business objectives. The Great Outdoor Company's continued success has resulted in higher transaction rates and growing data volumes, prompting the chief information officer (CIO) to look for ways to ensure they can continue to meet service level agreements. As a result, the data management team at The Great Outdoor Company uses Optim solutions to monitor and investigate current systems to identify potential efficiency improvements while taking into account cost, benefit, and risk. They implement several projects using a combination of database and application optimization techniques including tuning, archiving, and use of Static SQL.

1.5.1 The products used in this book

At the time this book was being written, many of the products were under the name Data Studio. Many of these have been renamed to Optim, but we use the old names in the actual scenarios and descriptions to match the product releases that were used in the writing of this book. The IBM white paper *Understanding the packaging of Optim Solutions for database development, administration, and performance management* provides a brief overview of the functionality and packaging of the development, administration, and performance management offerings in the IBM Optim products. This article is available at the following Web site:

<http://www.ibm.com/developerworks/data/library/techarticle/dm-0909optimpackaging/>

The products and tools used for all examples in this book are as follows:

- ▶ DBMS
 - DB2 for Linux, Unix, and Windows v9.5
 - DB2 for z/OS
- ▶ Performance monitoring solutions
 - DB2 Performance Expert for Linux, UNIX, and Windows Version 3.2
 - Performance Expert with Extended Insight feature Version 3.2
 - Tivoli Omegamon XE for DB2 Performance Expert on z/OS Version 4.1
 - DB2 Query Monitor Version 2.3
- ▶ Performance tuning solutions

DB2 Optimization Expert for z/OS Version 2.1 (Now renamed Optim Query Workload Tuner for DB2 for z/OS)
- ▶ Development
 - Data Studio Developer 2.1 and Data Studio pureQuery Runtime 2.1 (now renamed Optim Development Studio and Optim pureQuery Runtime)
 - Tivoli Composite Application Manager for WebSphere Version 6.1

Figure 1-4 on page 17 shows the environmental topology used in this book.

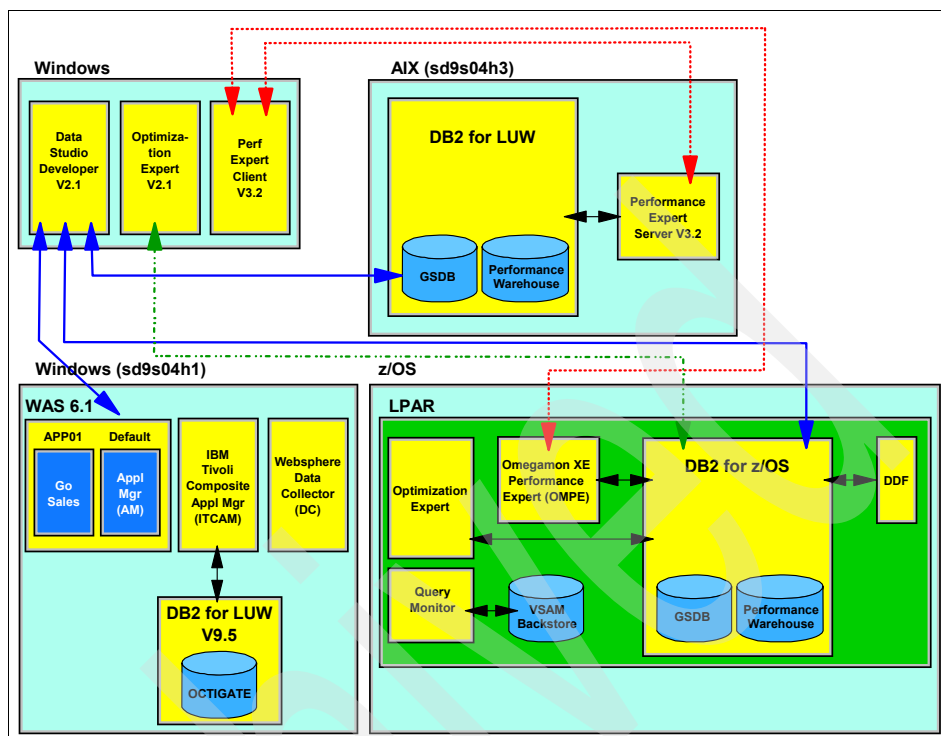


Figure 1-4 Environmental topology

1.5.2 Integrated data management in action

As business grows, the Great Outdoor Company has been experiencing some performance degradation and unexpected fluctuations of the heritage Web applications. The dynamic nature of these applications is a major reason for the performance fluctuations.

Rather than tasking developers to re-architect or engineer a replacement application, the developers are able to improve the applications' performance by making the existing applications execute statically using the client optimization capabilities of pureQuery, as delivered in Data Studio Developer and pureQuery Runtime (renamed to Optim Development Studio and Optim pureQuery Runtime). These tools enable developers to capture the dynamic SQL statements while running their existing applications. After the SQL statements are captured, developers or DBAs can look at and evaluate generated SQL and opt to replace with more efficient queries. The DBA can bind the intact or updated SQL statically into a database package. The statically bound

applications enable the DBA to use Tivoli OMEGAMON® for DB2 (OMPE), DB2 Query Monitor (QM), and DB2 Optimization Expert (OP, renamed to Optim Query Workload Tuner for DB2 for z/OS)) more effectively.

Furthermore, the success of the company also triggers requirements for new applications that will upgrade the functionality of existing applications, as well as requiring changes to the underlying data sources. With IBM InfoSphere Data Architect, beginning at the base model level, the data architect of Great Outdoor Company, can take the existing logical data model (LDM) for the database and add entities to the model based on the new requirements. Referential integrity and relationships are also added to these new tables at the same time. The tool provides the capability for the architect to include the use of domain models to ensure certain rules are applied to the particular schema of interest, or glossary models to enforce certain naming standards for the new entities consistent with the rest of the architecture. From here, the architect has the ability to transform the LDM to a UML model, and share with the development team for use in the expansion of the application. The architect can also transform the LDM to a physical data model (PDM) to share with the DBA to implement the database changes.

The application developers are able to see these changes to the data source from within their development environment using Data Studio Developer. Within the integrated development environment, developers can jump to the appropriate place in the Java source code where the proposed schema changes will have an impact. They are able to generate new Java code to address the new functionality of the application, but can also generate efficient SQL statements to access the data of interest.

The discussed scenarios show a process to addressing common issues using an IDM approach. All of these situations can be addressed and resolved graphically through the IBM suite of tools, eliminating the need for manual impact analysis (which is a huge time saver) and facilitating the use of similar look-and-feel tools across different roles in the organization.

Optimizing existing Java applications

Performance measurement and management is a crucial component of a service level objective (SLO). An SLO violation from a performance degradation can often result in a severe financial penalty. Therefore, performance management should not be viewed as a task that lasts for discrete periods of time such as one day or one week. Rather, it is a continuous, closed-loop process of monitoring, understanding, and tuning.

In most z/OS DB2 shops, performance management is ingrained in the culture and is well-managed based on experience of many years of managing and tuning traditional application workloads. However, for many reasons (introduced in 1.3, “DBA challenges in modern application environments” on page 11) many IT shops face new challenges when managing performance for the relatively new breed of Java distributed workloads compared to the experience of thirty years or more for the traditional workloads.

In this chapter, we describe how use of the pureQuery client optimization capability can help address some of those challenges by optimizing the performance of an existing Java application while at the same time improving its manageability and security so that Java applications can be as manageable and dependable as traditional CICS and COBOL applications.

We use the following products to demonstrate this scenario:

- ▶ Data Studio Developer and Data Studio pureQuery Runtime (which have been renamed to Optim Development Studio and Optim pureQuery Runtime)
- ▶ Tivoli OMEGAMON XE for DB2 Performance Expert
- ▶ DB2 Administration Tool
- ▶ DB2 Query Monitor
- ▶ DB2 Optimization Expert (which has been renamed to Optim Query Workload Tuner)

2.1 Scenario description

The production DBAs of the The Great Outdoor Company, the fictitious company of this book, would like to improve their ability to manage, monitor, tune, and secure one of their existing Java Web applications. By doing so, they are able to perform the following tasks:

- ▶ Identify this application from their online monitoring tool, Tivoli OMEGAMON XE for Performance Expert for DB2 for z/OS
- ▶ Trace the captured SQL back to the originating application source module, which is especially challenging if the application exploits a persistence technology that generates SQL on the fly.
- ▶ Eliminate security exposures of using dynamic SQL
- ▶ Identify and retrieve quickly the dynamic SQL calls that can be analyzed by the explain tool.
- ▶ Collect detailed accounting data to understand the distributed workloads.

2.1.1 Background on DB2 bind process

To facilitate our discussion, we provide a brief introduction of DB2 plan, Database Request Module (DBRM), package, and the bind process of z/OS DB2 programming.

Because the compilers of programming languages, such as COBOL, PL/I, PASCAL and so on, do not recognize SQL statements, a process called *precompile* is used to remove SQL statements before the language compiler can actually process the program code. During precompilation, the SQL statement is replaced with a CALL statement to the appropriate language interface in a modified source file. The actual SQL statements, in an internal format, are added to a separate object, the DBRM.

The modified program source module can be compiled by the programming language compiler to produce an object module, which is then linked into an executable load module (usually, the developer submits the pre-compile, compile, and linkedit in one job/JCL), followed by binding the associated DBRM.

The bind creates the executable form of the SQL, similar to what the linkage editor does for the program. DBRMs can be bound into *packages*. During the bind process, tasks such as access path selection (optimization), authorization, and database object validation are performed. The output of the bind process is either a plan or package that is stored in the DB2 catalog.

Figure 2-1 illustrates this process.

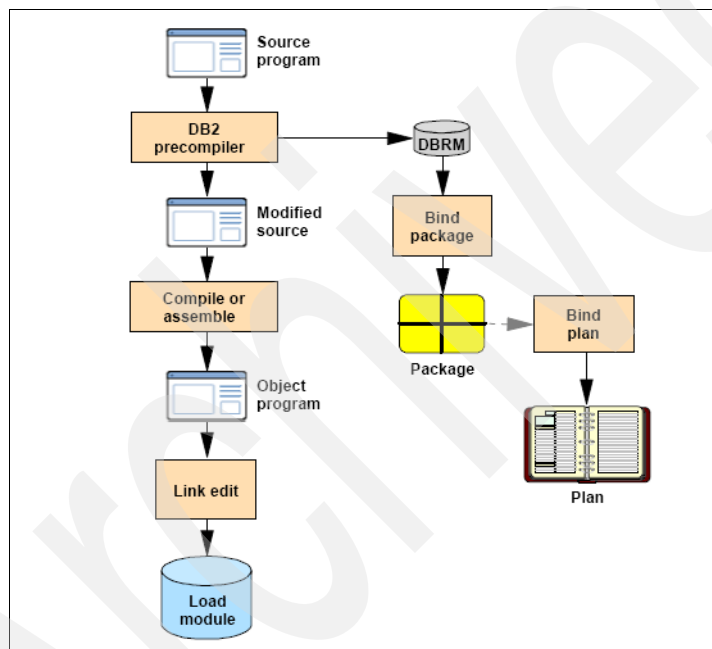


Figure 2-1 Precompile and compile process

A package is an executable representation of a single DBRM. Before the introduction of packages, the DBRMs representing the programs that are linked into a single load module were all bound into a single plan. Using the package bind improves availability and flexibility. If you change an SQL statement in a program, you only have to rebind one package.

For more details of statically bound packages and plans, refer to *DB2 9 for z/OS: Packages Revisited*, SG24-7688.

2.2 Application optimization considerations

The performance of DB2 applications depends on many factors, including architecture, database design, environmental factors, and SQL coding. However, most agree that tuning SQL is the most time-consuming of those factors (Figure 2-2).

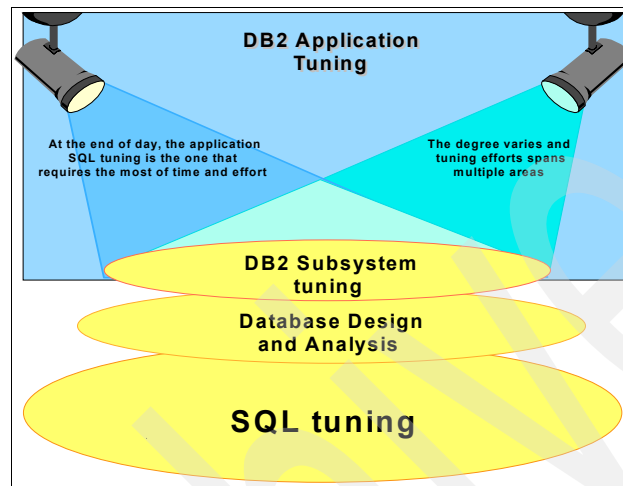


Figure 2-2 DB2 application tuning: time and effort

As mentioned in 1.3, “DBA challenges in modern application environments” on page 11, rapid Java development tools became popular for providing easy development frameworks, but the generated code might not deliver optimal performance. Is there any way to improve the performance of Java applications without enormous amounts of time and energy of the developer and DBA, while also allowing developers to continue to take advantage of the Java development framework that churns out dynamic SQL calls? The solution can be found in using pureQuery client optimization capability.

2.2.1 Improving data access performance

DB2 supports two SQL execution modes:

- Dynamic
- Static

Using the pureQuery technology, Java applications can make use of both without the need to implement the data access using a JDBC-based API for dynamic execution and the SQLJ Java language extension for static execution.

Static versus dynamic SQL

Generally speaking, dynamic SQL calls facilitate a rapid Java application development while the static SQL calls provide the higher security and predictable performance.

The primary difference between static and dynamic SQL is that static SQL requires compiling and binding before an application runs, while dynamic SQL is compiled during the run time. Thus dynamic SQL potentially incurs the extra processing each time it executes, although the dynamic statement cache can help avoid the extra costs. DB2 works smart to minimize the overhead of preparing the access path at run time by using prepared statement caches to make the dynamic SQL execute as fast as possible. However, achieving high cache hit ratios typically requires system tuning of cache sizes and also requires that your SQL statement in JDBC is coded correctly to increase the hit ratio without snapping up the cache space.

In typical environments, ideal optimization is hard to achieve. Therefore, in many cases, static SQL can provide faster performance, more consistent response times, and other benefits as well. The statements are all bound into the package, so you do not get cache misses for an individual query.

Figure 2-3 compares static and dynamic SQL execution processes.

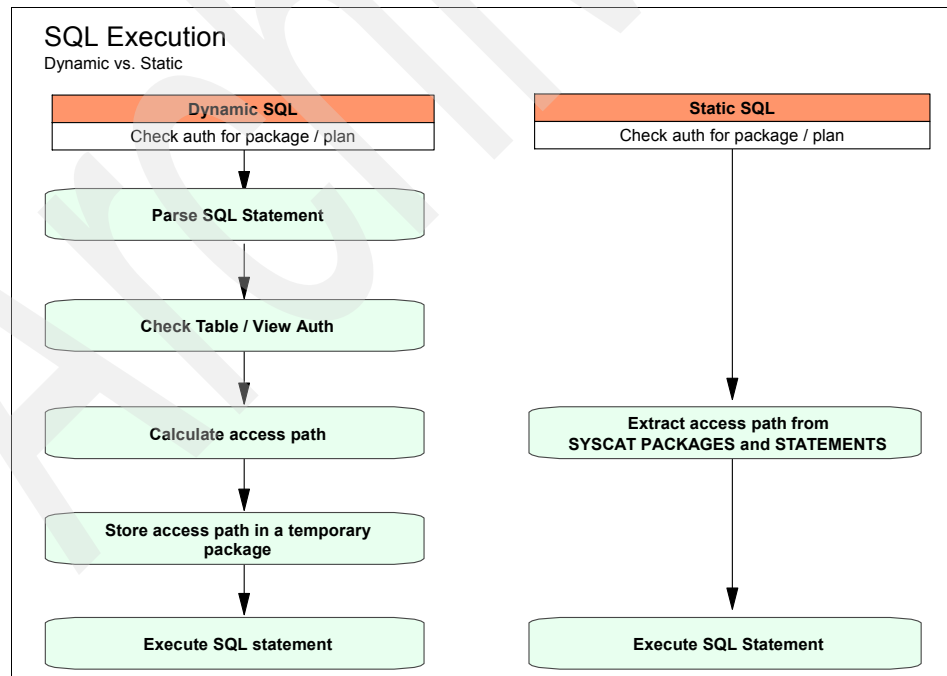


Figure 2-3 Static processing is faster than dynamic

There are additional benefits derived from the statically bound packages. Because a static SQL statement is precompiled and bound into a package before the application is executed, the access plan is locked in before the program runs. These paths remain constant until the program package is rebound, in contrast to dynamic SQL, in which the access path can change from execution to execution.

With the locked-in access path of static SQL, the application gets more predictable performance and a good access path, because a DBA would normally make sure to secure a good access path before it goes into the production system using tools such as DB2 Optimization Expert (now known as Optim Query Workload Tuner). In addition, the DBA can further verify ahead of time which access path will be used with tools such as DB2 Path Checker. All of these provide a certain level of predictable performance as required by SLO.

Another possible performance gain from the static SQL is that it reduces the number of bytes sent through the network and eliminates the full statement preparation during application execution. However, this benefit depends heavily on how much network optimization the JDBC driver can do with the dynamic SQL, and how optimized your existing JDBC code already is.

With dynamic SQL, the package name is usually the same across applications, making it hard to identify the specific poor performing application from monitoring tools. Static packages enable DBAs to provide a distinct name for each program and thus eases the task of identifying programs involved in poorly performing SQL calls, deadlocks, or time-outs. The application owner can be identified through the package naming convention. You can identify the SQL statements issued by a given application program, monitor SQL activity by program, and measure program level CPU time, I/O operations, number of getpages, and other performance characteristics that are associated with each package.

Table 2-1 compares dynamic and static SQL characteristics.

Table 2-1 Dynamic SQL compared with static SQL

Feature	Dynamic SQL	Static SQL
Performance	Can approach static SQL performance with help from dynamic SQL statement cache. Cache misses are costly.	All SQL parsing, catalog access, done at BIND time. Fully optimized during execution.
Access path reliability	Unpredictable. Any prepare can get a new access path as statistics or host variables change.	Consistent. Locked in at BIND time ALL SQL available ahead of time for analysis by EXPLAIN.
Authorization	Privileges handled at object level. All users or groups must have direct table privileges. Security exposure, and administrative burden exists.	Privileges are package-based. Only administrator needs table access. Users/Groups have execute authority. Prevents non-authorized SQL execution.

Feature	Dynamic SQL	Static SQL
Monitoring, Problem determination	Database view is of the JDBC package. No easy distinction of where any SQL statement came from.	Package view of applications makes it simple to track back to the SQL statement location in the application.
Capacity planning, Forecasting	Difficult to summarize performance data at program level.	Package Level Accounting gives program view of workload to aid accurate forecasting.
Tracking dependent objects	No record of which objects are referenced by a compiled SQL statement.	Object dependencies registered in database catalog.

Optimal access plan

Traditionally, Java application developers used low-level APIs to implement data access layers that map in-memory application objects to relational objects, requiring in-depth knowledge and SQL skills. With the introduction of object/relational persistence frameworks, such as Hibernate or JPA, common data access, mapping, and manipulation tasks can be easily integrated into applications. One reason why these frameworks are popular among developers is that they somewhat hide the complexities associated with data access and object/relational mapping. This eliminates the need to implement low-level functionality. While this abstraction can provide big productivity benefits during development, it can impact performance in the following ways:

- ▶ Performance can be less-than-optimal if sub-optimal SQL (which is generated by those frameworks) is processed.
- ▶ Performance can be unreliable because SQL is always executed dynamically.

Using pureQuery client optimization, these issues can be addressed for most types of application that processes SQL dynamically, whether or not they use a persistence framework. If necessary, generated SQL can be replaced with better performing SQL and processing can be performed in static execution mode. By enabling static SQL execution (through the creation of database packages), information about the SQL that an application processes and their access plans are available in the DB2 catalog tables. Because access plans can be locked down, performance is more predictable.

2.2.2 Improving application manageability

Static packages can provide significant benefits to DBAs with respect to application manageability. By enabling the creation of database packages for existing and new Java applications, the pureQuery platform establishes a correlation between an application and its processed SQL. Using their existing

skills and tools, DBAs can apply their experience in managing traditional static workloads such as CICS and COBOL directly to distributed workloads:

- ▶ SQL can be correlated with its originating application, simplifying and shortening the problem determination process.
- ▶ Unique package identifiers can be used to manage workloads more efficiently using Workload Manager (WLM) and its goal-oriented policies.
- ▶ Additional statistics are available for packages that aid in capacity planning.

The following sections illustrate these benefits for a distributed Java workload, which is processed by DB2 for z/OS.

Correlating SQL with its originating application

As stated in 1.3, “DBA challenges in modern application environments” on page 11, performance data of a distributed workload with dynamic calls are all bundled under one name, which makes it difficult to monitor and manage applications and diagnose issues.

Example 2-1 shows a snippet of the thread activity from the Tivoli OMEGAMON XE for DB2 Performance Expert (OMPE) VTAM® classic interface. The displayed thread activity lists multiple Java applications that are all associated with generic packages from the common DISTSERV plan, like any other distributed (dynamic) workload. Without further information and lots of applications running concurrently, it can be difficult to isolate a single application.

Example 2-1 Dynamic SQL thread activity as shown by OMPE

	ZALLT	VTM	02	V410./C DB9B S	05/19/09 17:22	2
> Help PF1	Back PF3	Up PF7	Down PF8	Sort PF10	Zoom PF11	
> T.A						
>	Thread Activity: Enter a selection letter on the top line.					
> *-All-Idle	B-TSO	C-CICS	D-IMS	E-Background	F-Dist Allied	
> G-Dist DBAC	H-Util	I-Inact	J-Filter	K-Functions	L-Stored Proc	
> M-Triggers	N-Sysplex	O-Enclaves	P-Worksta	Q-All+Idle		
=====						
>	Threads Summary Excluding Idle Threads					
THDA						
+ *						
+ Elapsed	Planname	CPU	Status	GetPg	Update	Commit CORRID/JOBN
+ -----	-----	-----	-----	-----	-----	-----
+ 15-06:03	?RRSAF	00.0%	NOT-IN-DB2	131822	0	21965 DB9BADMT
+ 15-06:03	?RRSAF	00.0%	NOT-IN-DB2	0	0	1 DB9BADMT
+ 15-06:03	K02PLAN	00.0%	NOT-IN-DB2	1733	8	17 CA9B02
+ 15-06:03	DB2PM	00.0%	NOT-IN-DB2	3733K	878193	1536K CA9B02
+ 15-06:03	K02PLAN	00.0%	NOT-IN-DB2	0	0	0 CA9B02
+ 00:32:50.4	K02PLAN	00.0%	IN-DB2	500	200	100 CA9B02


```

+ 00:00:00.8  DISTSERV  00.3%  WAIT-REMREQ    34    10    0 db2jcc_appli
+ 00:00:00.8  DISTSERV  02.5%  WAIT-REMREQ    41    12    0 db2jcc_appli
+ 00:00:00.6  DISTSERV  03.1%  WAIT-REMREQ    34    10    0 db2jcc_appli
....
+ 00:00:00.8  DISTSERV  00.5%  WAIT-REMREQ    41    12    0 db2jcc_appli
+ 00:00:00.6  DISTSERV  03.1%  WAIT-REMREQ    34    10    0 db2jcc_appli
=====

```

When a Java application is enabled for static SQL execution, the corresponding threads are associated with a unique package name.

Example 2-2 displays thread activity that can be correlated with the originating Java applications using their associated package names. This chapter's example application uses package DSRED2 (bound at isolation level 1) to process its SQL.

Example 2-2 Thread activity, as shown by OMPE, can be associated with applications using unique package identifiers, such as DSRED21 used by the example application

```

_____ ZALLU   VTM   02       V410./C DB9B S 05/20/09 17:43   2
> Help PF1      Back PF3      Up PF7      Down PF8      Sort PF10      Zoom PF11
> U.A
>
      Thread Activity:  Enter a selection letter on the top line.

> *-All-Idle   B-TSO      C-CICS      D-IMS      E-Background   F-Dist Allied
> G-Dist DBAC   H-Uti1     I-Inact     J-Filter    K-Functions     L-Stored Proc
> M-Triggers    N-Sysplex   O-Enclaves  P-Worksta   Q-All+Idle
=====
>
      Threads Summary Excluding Idle Threads
PTHDA
+ *
+ Elapsed      Package   CPU    Status      GetPg  Update  Commit  CORRID/JOBN
+ -----
+ 00:00:05.6    DSRED21   00.0%  WAIT-REMREQ    41    12    0 db2jcc_appli
+ 00:00:01.0    DEMOSQL   00.0%  WAIT-REMREQ    34    10    0 db2jcc_appli
+ 00:00:00.9    DSRED21  00.0% WAIT-REMREQ   38   10   0 db2jcc_appli
+ 00:00:00.7    KLDFS43   00.0%  WAIT-REMREQ    34    10    0 db2jcc_appli
=====

```

Having unique package names available simplifies the process of isolating problematic applications. Package specific thread statistics provide detailed information that can be queried using monitoring or reporting tools. Example 2-3 on page 28 displays output from DB2 Query Monitor (QM), summarizing SQL activity associated with package DSRED21 during a specific time interval.

Example 2-3 SQL metrics for example application

```

2009/05/20 19:10:45 ----- Operational Summaries ----- Row 1 of 9
Option ==> Scroll ==> PAGE
DB2 QM Subsystem: DB9B          Interval Start Date: 05/20/2009 Time: 19:00:00
Filters Enabled: N              Interval End Date: CURRENT Time: CURRENT
DB2:      Plan:      Pgm:      Authid:
          Section:   Call:      Type:
          WSUser:    WSName:
          WSTran:    CorrID:
C: 1-Plan,2-DB2(Op),4-Authid,5-DB2(St),6-DBase,7-Buff,8-Pageset,9-Objs,
   10-Corr,11-Sect,12-Call,13-WSUser,14-WSName,15-WSTran,16-SQL,19-Delay,
   20-Locks,21-Misc,22-Buffstat,23-Excp,24-CurrAct
----->
CMD  Program          Calls          Elapsed    %Elap    Avg Elapsed
--  -
___  DGO@PC2            222          0.079214   0.94     0.000356
...
...
16  DSRED21          6668          8.110692  97.26    0.001216
...
...
___  CQM@STX8           180          0.019130   0.22     0.000106
***** Bottom of Data *****

```

Taking advantage of WLM

With the introduction of *enclaves*, which are independent dispatchable units of work, Distributed Data Facility (DDF) transactions can be managed separately from the DDF address space itself. The workload coming into DDF is classified as an assigned service class by WLM. Depending on the classification rules specified, WLM can assign service classes to DDF work, and associate different goals with these service classes according to the work type of DDF.

The distributed threads with dynamic SQL calls are sent to DB2 for z/OS as undifferentiated pieces of work, unless the Java program explicitly provides identifying information when it connects to the database.

Note: SAP applications, for example, provide identifying information, such as the correlation ID, workstation user ID, and workstation name, which can be used to manage the workload.

Unfortunately, most Java programs do not provide such identifying information. Their workload is therefore assigned to a single service class with possibly low priority and low level of service. Every distributed thread is treated as equal with no classification of WLM goal and thus cannot be prioritized individually to meet the service level agreement (SLA).

Using pureQuery's ability to associate SQL with unique database packages, you can make use of WLM more efficiently, by assigning service levels to individual applications according to business requirements.

For details about DDF and WLM, refer to *DB2 9 for z/OS: Distributed Functions*, SG24-6952 and *System Programmer's Guide to: Workload Manager*, SG24-6472.

Example 2-4 depicts a default WLM service class and resource group definition for DDF requests. The workload for every Java application that is not uniquely identified and associated with its own service class is processed according to this configuration.

Example 2-4 A typical default setup for DDF requests in WLM

Service Class Selection List		Row 1 to 21 of	
Command ==> _____			
Action Codes: 1=Create, 2=Copy, 3=Modify, 4=Browse, 5=Print, 6=Delete, /=Menu Bar			
Action	Class	Description	Workload
—	ASCH	APPC Transaction Programs	STCTASKS
—	BATCHDIS	Batch default discretionary	BATCH
—	BATCHHI	Batch Workload, above BATCH	BATCH
—	BATCHLOW	Batch Workload, below BATCH	BATCH
—	BATCHMED	Batch Workload	BATCH
—	BATCHWDI	Batch default discretionary	BATCH
—	BATCHWHI	Batch Workload, above BATCH	BATCH
—	BATCHWLO	Batch Workload, below BATCH	BATCH
—	BATCHWME	Batch Workload	BATCH
—	CICS	CICS Transactions	ONLINES
—	IMS	IMS Transactions	ONLINES
4_	DDF	DDF Requests	DATABASE
—	DDFBAT	DDF low priority	DATABASE

* Service Class DDF - DDF Requests

Created by user IBMUSER on 2001/08/13 at 15:05:45
Base last updated by user POKIE on 2006/12/06 at 16:19:22

Base goal:
CPU Critical flag: NO

Duration Imp Goal description

```

-----
1 1000      2  Average response time of 00:00:00.400
2           Discretionary
-----

```

```

-----
* Resource Group DDF - DB2 DISTRIBUTED REQUESTS

```

Created by user BART on 2003/02/18 at 17:05:29

Base last updated by user HOMER on 2003/02/18 at 17:05:29

Base attributes:

Capacity is in Service Units with Sysplex Scope.

Minimum capacity is 0.

Maximum capacity is 1000.

By switching to static SQL processing using pureQuery, unique package names can be used as an identifying token for WLM to assign the appropriate service classes, allowing thread transaction level prioritization, much like CICS and IMS does. Example 2-5 depicts a new service class, DDFRED2, that was created for the pureQuery-enabled example Java application, which uses database package DSRED2 to process its workload.

Example 2-5 Custom service class definition allows for appropriate prioritization of a Java workload

```

-----
Service Class Selection List          Row 1 to 21 of
Command ==>
-----

```

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

Action	Class	Description	Workload
—	ASCH	APPC Transaction Programs	STCTASKS
...			
—	BATCHWME	Batch Workload	BATCH
—	CICS	CICS Transactions	ONLINES
—	IMS	IMS Transactions	ONLINES
—	DDF	DDF Requests	DATABASE
—	DDFBAT	DDF low priority	DATABASE
4_	DDFRED2	DSRED2 DDF app1 by package name	DATABASE

* Service Class **DDFRED2 - DSRED2** DDF appl by package name
Class assigned to resource group DDF

Created by user YIM on 2009/05/21 at 15:17:59
Base last updated by user YIM on 2009/05/21 at 17:23:27

Base goal:
CPU Critical flag: NO

#	Duration	Imp	Goal description
-	-----	-	-----
1	500	2	90% complete within 00:00:00.200
2		3	Execution velocity of 40

Example 2-6 shows that packages named DSRED2* are associated with Subsystem Type - DDF and classified in subsystem DB9B using the DDFRED2 service class.

Example 2-6 Package names in classification rules with associated report class

* Subsystem Type DDF - DDF Work Requests

Last updated by user YIM on 2009/05/21 at 15:23:39

Classification:

Default service class is DDFBAT
There is no default report class.

Qualifier # type	Qualifier name	Starting position	Service Class	Report Class
-	-----	-----	-----	-----
1 SI	DB9B		DDFBAT	DSRED1
2 . PK	. DSRED2*		DDFRED2	DSRED2

Report class DSRED2, shown in Example 2-7 on page 32, is assigned to those packages to identify performance matrixes in SMF or RMF reporting. The classification rules in WLM can be applied to a specific job to collect performance matrixes to fine-tune performance and support capacity planning.

Example 2-7 Report class definitions

Report Class Selection List			Row 28 to 49 of	
Command ==> _____				
Action Codes: 1=Create, 2=Copy, 3=Modify, 4=Browse, 5=Print, 6=Delete, /=Menu Bar				
			----Last Change----	
Action	Name	Description	User	Date
—	DSRED1	Data Studio Redbook Application	YIM	2009/05/21
—	DSRED2	Data Studio Redbook Application	YIM	2009/05/21

Capacity planning for distributed Java applications

If a Java application uses unique packages to process its SQL, relevant accounting and performance statistics can be collected from various sources, such as the DB2 trace, RMF, and SMF. This information can aid in capacity planning.

The OMPE/DB2PM report can provide performance statistics by package (class 7) as well as required statistic, package by package, and application by application for capacity planning of the distributed Java application.

RMF reporting classes monitoring z/OS hardware resources for package names and DB2 accounting record report summarize package activity are also good source for capacity planning.

2.2.3 Enhancing security

Applications that process SQL dynamically require that the privilege to access or manipulate the relevant database objects are explicitly or implicitly granted to users. This can create a security exposure for enterprise information because a malicious user could attempt to use the granted permission to manipulate data by accessing the database using another program.

This weakness is commonly exploited during SQL injection attacks, in which a user (such as an internal or external person accessing a search interface on the Web) can take advantage of the poor application programming practices that do not prevent alteration (or injection) of SQL that database processes on behalf of the application. The results can be devastating as shown by recent incidents where credit card information was illegally obtained.

Note: The pureQuery platform provides a mechanism that can be used to restrict the SQL an application might process dynamically. Refer to 5.3.1, “Preventing SQL injection” on page 179 for further information.

Static SQL processing is not vulnerable to these security risks because database packages are used, which provide a mechanism to restrict data access to a well-defined set of SQL (just as stored procedures do). Users are only granted the privilege to execute SQL that is stored in a package, and are therefore prevented from accessing or manipulating data other than in the way intended by the application programmer. Enterprise data is therefore better protected.

Figure 2-4 visualizes that access to the database can be further restricted through the use of packages to process SQL.

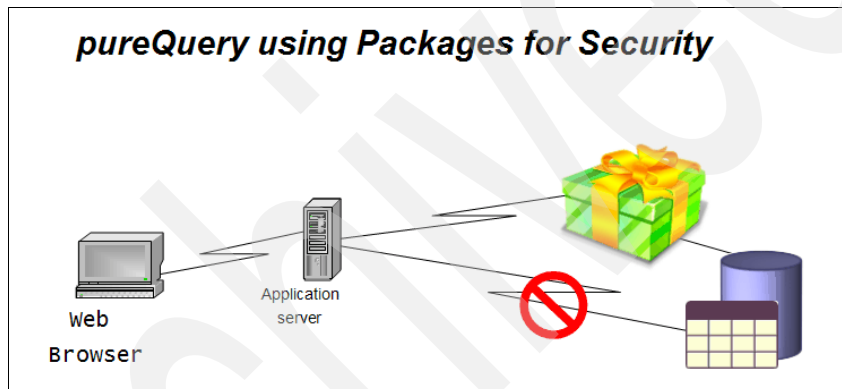


Figure 2-4 Switching from dynamic to static security model

Switching from a user-centric security model to an application-centric security model can also reduce the overhead involved in administering the database, because privileges have to be granted for a few packages instead of a large number of database objects, such as tables and views.

2.3 Monitoring with Tivoli OMEGAMON XE for DB2, DB2 Query Monitor, and DB2 Optimization Expert

The production DBAs are equipped with necessary tools to monitor and manage the workload performance and operation to meet or exceed the SLO. In this section we demonstrate how static package can help the DBA monitor the

application, collect information, and improve performance with Tivoli OMEGAMON XE for DB2 (OMPE), DB2 Query Monitor (QM), and DB2 Optimization Expert (OE).

2.3.1 Correlating SQL with its originating application

For common database administration, tuning, and troubleshooting tasks, it is crucial for the DBA to correlate SQL that is executed with its originating application. This correlation typically cannot be established easily for Java applications that process SQL dynamically because generic packages are used.

Example 2-1 on page 26 depicts a typical thread activity view in OMPE. Using the correlation ID db2jcc_application we are able to determine that various Java applications are accessing the database. However, because they are all using the generic DISTSERV plan, we are unable to distinguish the applications. Is it payroll? A business report? It is unknown.

To work around the issue, some DBAs enforce a strict one-to-one mapping between an application and the user ID that this application is using to connect to the database. For example, a certain business reporting application might be the only application that connects as USER13. While this approach might work for certain types of applications, it cannot be used universally.

Applications that process SQL statically on the other hand can be identified easily, because they are associated with unique package names.

The pureQuery client optimization technology can be used to enable static SQL processing for applications that were originally implemented using an API or persistence framework that only supports dynamic SQL processing. Example 2-8 on page 35 depicts database packages that were created for an existing Java application using pureQuery.

Example 2-8 Listing of DB2 packages created using the pureQuery technology to support static SQL execution for an existing Java application

```
DB2 Admin ----- DB9B Packages ----- Row 1 of 4
Command ==> Scroll ==> PAGE

Commands: BIND REBIND FREE VERSIONS GRANT
Line commands:
DP - Depend A - Auth T - Tables V - Views X - Indexes S - Table spaces
Y - Synonyms RB - Rebind F - Free B - Bind BC - Bind copy GR - Grant
EN -Enab/disab con PL - Package lists P - Local plans LP - List PLAN_TABLE
I - Interpret SQ - SQL in package VE - Versions D - Databases RO - Role

V I V O Quali- R E D
D S A P fier L X R
* * * * * * * *
----->-----
NULLID DSRED22 YIM 2009-05-20-17.24 B S N Y GOSALESC C N B
NULLID DSRED24 YIM 2009-05-20-16.50 B R N Y GOSALESC C N B
NULLID DSRED23 YIM 2009-05-20-17.25 B T N Y GOSALESC C N B
NULLID DSRED21 YIM 2009-05-20-17.26 B U N Y GOSALESC C N B
***** END OF DB2 DATA *****
```

Monitoring application activity again after static SQL processing has been enabled, we are able to distinguish the Java workloads using their unique package names, as shown in Example 2-9.

Example 2-9 DB2 thread activity after static SQL processing has been enabled using the pureQuery client optimization technology

```
----- ZALLU VTM 02 V410./C DB9B S 05/20/09 17:42 2
> Help PF1 Back PF3 Up PF7 Down PF8 Sort PF10 Zoom PF11
> U.A
> Thread Activity: Enter a selection letter on the top line.

> *-All-Idle B-TS0 C-CICS D-IMS E-Background F-Dist Allied
> G-Dist DBAC H-Util I-Inact J-Filter K-Functions L-Stored Proc
> M-Triggers N-Sysplex O-Enclaves P-Worksta Q-All+Idle
=====
> Threads Summary Excluding Idle Threads
PTHDA
+ *
+ Elapsed Package CPU Status GetPg Update Commit CORRID/JOBN
+ -----
+ 00:00:04.9 DSRED21 00.1% WAIT-REMREQ 38 10 0 db2jcc_appli
+ 00:00:02.4 DEMOAPP 00.1% WAIT-REMREQ 34 10 0 db2jcc_appli
+ 00:00:02.3 GLWGENA 00.1% WAIT-REMREQ 34 10 0 db2jcc_appli
+ 00:00:02.3 DNET955 00.0% WAIT-REMREQ 38 10 0 db2jcc_appli
+ 00:00:02.3 DBAP103 00.1% WAIT-REMREQ 41 12 0 db2jcc_appli
+ 00:00:01.9 DSRED51 00.0% WAIT-REMREQ 38 10 0 db2jcc_appli
```

+ 00:00:01.8	DNET95X	00.0%	WAIT-REMREQ	34	10	0	db2jcc_appli
+ 00:00:01.6	DBAP631	00.1%	WAIT-REMREQ	34	10	0	db2jcc_appli
+ 00:00:01.1	DEMOSQL	00.1%	WAIT-REMREQ	34	10	0	db2jcc_appli
+ 00:00:00.8	DSRED21	01.2%	WAIT-REMREQ	34	10	0	db2jcc_appli
+ 00:00:00.8	DSRED21	01.2%	WAIT-REMREQ	34	10	0	db2jcc_appli

=====

These database packages are regular packages that can be maintained and processed using tools with which you are already familiar.

Example 2-10 depicts a DB2 catalog entry for one of the generated packages using DB2 Administration Tool in a mainframe TSO session.

Example 2-10 Viewing package DSRED21 created by pureQuery.

```
DB2 Admin ----- DB9B Packages ----- Row 123 of 139
Command ==>                                     Scroll ==> PAGE

Commands: BIND REBIND FREE VERSIONS GRANT
Line commands:
DP - Depend A - Auth T - Tables V - Views X - Indexes S - Table spaces
Y - Synonyms RB - Rebind F - Free B - Bind BC - Bind copy GR - Grant
EN -Enab/disab con PL - Package lists P - Local plans LP - List PLAN_TABLE
I - Interpret SQ - SQL in package VE - Versions D - Databases RO - Role
```

S	Collection	Name	Owner	Bind Timestamp	V I V O Quali-	R E D
*		*	*		D S A P fier	L X R
*	*	*	*	*	* * * * *	* * *
sq	NULLID	DSRED21	YIM	2009-05-20-17.26	B U Y Y GOSALESC	C N B
...						
...	DSNAOCLI	DSNCLIF4	RC63	2009-04-27-17.25	R Y Y RC63	N
...						
...	DB2OSC	DSN5OWCC	RC63	2009-04-30-11.55	R S Y Y RC63	C N R
	NULLID	DSN5OWSP	RC63	2009-04-30-11.55	R S Y Y RC63	C N R
...						
...	NULLID	DSN5OVE	RC63	2009-04-30-11.55	R S Y Y RC63	C N R

***** END OF DB2 DATA *****

To find the SQLs in the package created by pureQuery, you can use the same DB2 Administration Tool to query the DB2 catalog, as shown in Example 2-11 on page 37, as with any other static package created by a host language compilation and bind. There is nothing unique about packages created by pureQuery.

Example 2-11 SQL statements in the package DSRED21

DB2 Admin ----- Extracted SQL ----- Columns 00001 00072
Command ==> Scroll ==> PAGE

```
***** ***** Top of Data *****
=NOTE= -- SQL statements in PACKAGE : NULLID.DSRED21.()
=NOTE= -- SQL in stmt: 1
000001 DECLARE DB_PDQ_SPC1 CURSOR FOR SELECT P.PRODUCT_NUMBER,
000002 P.BASE_PRODUCT_NUMBER, P.INTRODUCTION_DATE, P.DISCONTINUED_DATE,
000003 P.PRODUCT_TYPE_CODE, P.PRODUCT_COLOR_CODE, P.PRODUCT_SIZE_CODE,
000004 P.PRODUCT_BRAND_CODE, P.PRODUCTION_COST, P.GROSS_MARGIN,
000005 P.PRODUCT_IMAGE, N.PRODUCT_NAME, N.PRODUCT_DESCRIPTION FROM
000006 GOSALES.PRODUCT AS P, GOSALES.PRODUCT_NAME_LOOKUP AS N WHERE
000007 P.PRODUCT_NUMBER = N.PRODUCT_NUMBER FETCH FIRST 5 ROWS ONLY FOR READ
000008 ONLY
=NOTE= -- SQL in stmt: 2
...
=NOTE= -- SQL in stmt: 3
...
=NOTE= -- SQL in stmt: 4
...
=NOTE= -- SQL in stmt: 5
...
=NOTE= -- SQL in stmt: 6
...
=NOTE= -- SQL in stmt: 7
...
=NOTE= -- SQL in stmt: 8
000046 DECLARE DB_PDQ_SPC9 CURSOR FOR SELECT CUST_CODE, CUST_FRST_NAME,
000047 CUST_LAST_NAME, CUST_ADDR1, CUST_CITY, CUST_POST_ZONE, CUST_CTRY_CODE,
000048 CUST_PHN_NBR, CUST_INFO, CUST_EMAIL, CUST_GNDR_CODE, CUST_PROV_STATE
000049 FROM GOSALESCT.CUST WHERE LCASE(CUST_FRST_NAME) LIKE LCASE(CAST(
000050 :H:H AS VARCHAR(128))) AND LCASE(CUST_LAST_NAME) LIKE LCASE(CAST(
000051 :H:H AS VARCHAR(128))) FOR READ ONLY
***** ***** Bottom of Data *****
```

If explain information was captured when the package was created or rebound, performance issues can be investigated more easily, because the access plan that was used during previous executions is well known.

The database package contains a link to further information that can be used to correlate the packages SQL with a particular application module or even the line of code where the SQL was processed. This information provides DBAs with application insights that were traditionally not available for common problem determination tasks.

2.3.2 Dynamic and static SQL performance comparison

One main benefit of static SQL is its potential to improve query performance and reduce CPU usage.

In 2008, an article was published in *DB2 Magazine* by the performance team, describing the findings of a formal performance study. This article is available online at the following Web page:

http://www.ibm.com/developerworks/data/library/dmmag/DBMag_2008_Issue2/pureQueryRuntime/index.html

As part of our book activities, we have taken the Great Outdoor Company Web application running on the distributed platform, measured CPU usage associated with dynamic and static SQL processing on the z/OS database node, and compared the results.

Example 2-12 and Example 2-13 on page 39 show the CPU performance matrixes of the dynamic and static execution as measured using DB2 Query Monitor (QM) tool.

Example 2-12 QM: CPU usage during dynamic execution

```
2009/05/26 15:40:10 ----- Operational Summaries ----- Row 1 of 8
Option ==> Scroll ==> PAGE
DB2 QM Subsystem: DB9B      Interval Start Date: 05/26/2009 Time: 15:00:00
Filters Enabled: N          Interval End Date: CURRENT Time: CURRENT
DB2:      Plan:      Pgm:      Authid:
          Section:    Call:      Type:
          WSUser:      WSName:
          WSTran:      CorrID:
C: 1-Plan,2-DB2(Op),4-Authid,5-DB2(St),6-DBase,7-Buff,8-Pageset,9-Objs,
   10-Corr,11-Sect,12-Call,13-WSUser,14-WSName,15-WSTran,16-SQL,19-Delay,
   20-Locks,21-Misc,22-Buffstat,23-Excp,24-CurrAct
----->
CMD Program      Calls      CPU      %CPU      Avg CPU <%CPU>      DELAY
-----
SYSLH100          2196      0.375453  2.31      0.000170  80.68  0.045555
SYSLN100          17064     15.125765 93.35      0.000886  87.54  0.308731
***** Bottom of Data *****
```

While the applications SQL was executed dynamically, two system packages were used, SYSLH100 and SYSLN100. No other applications were accessing the database using these packages while measurements were taken.

```

...
-----
>
CMD Program          Calls      CPU    %CPU    Avg CPU <%CPU>    DELAY
-----
__  DSRED21          17278    10.841525  90.84    0.000627  88.58  0.163088

***** Bottom of Data *****

```

Note: The number of calls are smaller during static execution due to the avoidance of prepares for the same workload.

	Program Name	# of Calls	CPU
Dynamic	SYSLH100	2196	0.375453
	SYSLN100	17064	15.125765
Dynamic total		19260	15.501218
Static	DSRED21	17278	10.841525
Static total		17278	10.841525
% of CPU savings by static		-1982	43%

The observed improvement in this study varies based on the statement cache hit ratio, and the type of JDBC driver being used.

DB2 Optimization Expert for z/OS (OE) visualizes query information, such as access plan graph and query annotations, and its four advisors (Query Advisor, Access Path Advisor, Statistics Advisor, and Index Advisor) guide the user with expert advice on how to boost SQL statement performance.

This section demonstrates how OE can be used to examine the content of package DSRED21 and how its advisors can be used to improve query performance.

First, we connect to our host subsystem DB9B from OE V2.1 client. See Figure 2-6.

Subsystem Properties

Subsystem Location

Specify the following information about the subsystem that you want to configure and click Catalog. After the subsystem is cataloged, click Next.

Steps

- 1. Location
- 2. Connect
- 3. Manage Users

Subsystem alias: OE

Location: DB9B

Host name: wtsc64.itso.ibm.com

Port number: 12350

Comment: port 12350

Catalog

< Back Next > Finish Cancel

Figure 2-6 The OE panel to connect to the ITSO DB9B subsystem

Figure 2-7 shows that we are connected and ready to use.

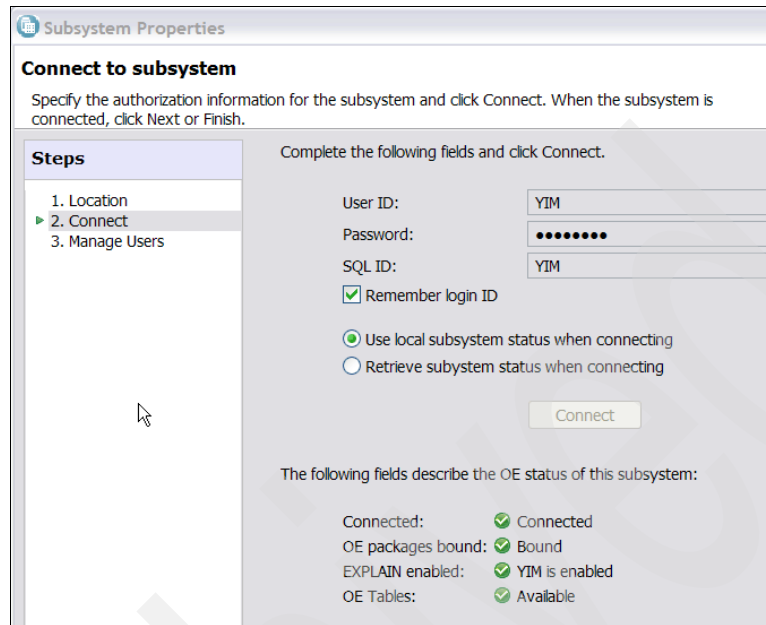


Figure 2-7 Successful connection to DB9B subsystem in ITSO system

The next logical step to see the SQL statements from the package is to retrieve the sample package from the catalog. We build a new view to filter statements by package. See Figure 2-8.

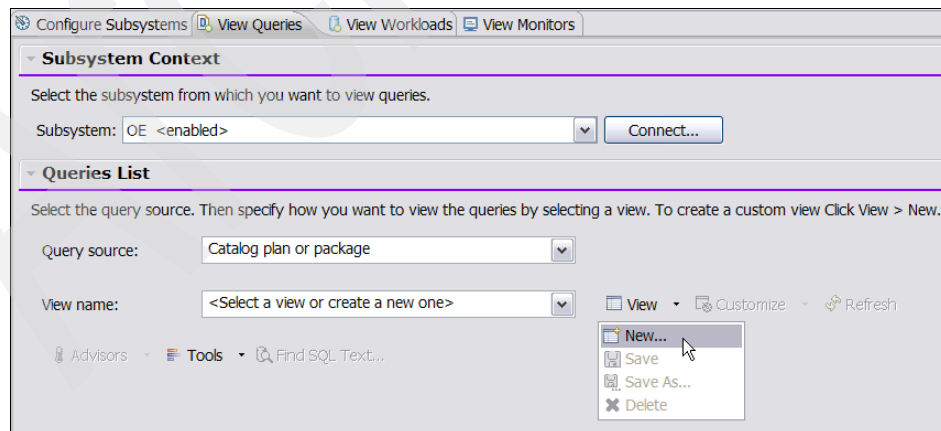


Figure 2-8 Retrieving a package from catalog using OE

We create view DSRED2 and specify that we want to filter the statement by package, as shown in Figure 2-9.

Create View

Filter Rows

Specify criteria to limit the query rows that are returned by typing values in the "Value" column. You must specify at least one package or plan filter. Then click Next or Finish.

Steps

- 1. Filter
- 2. Package Filters
- 3. Cost and Object Filters
- 4. Access Path Filters
- 5. Sort
- 6. Columns

View name: Maximum rows:

Specify whether you want to filter statements by plan, package, or both

- ☐ Filter statements by plan
- ☒ Filter statements by package
- ☐ Filter statement by both plan and package

Figure 2-9 Creating a View name to retrieve a package from the catalog

Figure 2-10 shows that we use DSRED21% with the “like” operator instead of the ‘=’ (equal sign) operator to filter package names to be retrieved.

Create View

Filter Rows
Specify package attributes to control which statements are returned by the view.

Steps

1. Filter
2. Package Filters
3. Cost and Object Filters
4. Access Path Filters
5. Sort
6. Columns

Column Name	Operator	Value	Comment
COLLID	=		Name of the package collection. For a trigger, the value is the name of the package.
NAME	LIKE	DSRED21%	Name of the package.
OWNER	=		Authorization ID of the package owner. For a trigger, the value is the name of the package owner.
TIMESTAMP	=		Timestamp format: yyyy-MM-dd HH:mm:ss
BINDTIME	=		Timestamp format: yyyy-MM-dd HH:mm:ss
QUALIFIER	=		Implicit qualifier for the unqualified table, view, or synonym.
EXPLAIN	=		EXPLAIN option specified for the package; t
VERSION	=		Version identifier for the package. The value
QUERYNO	=		The query number of the SQL statement in

Specify package dependency object filter

Object Qualifier	Object Name	Object Type

Add Object Filter Remove Object Filter

☐ Meet any package filter condition
☒ Meet all package filter conditions

< Back Next > Finish Cancel

Figure 2-10 Creating a view to retrieve packages like 'DSRED21%'.

In our test case, only the package name filter is required to retrieve the sample package from the catalog. After a few clicks of the **Next** button, OE has retrieved eight packages, as shown in Figure 2-11.

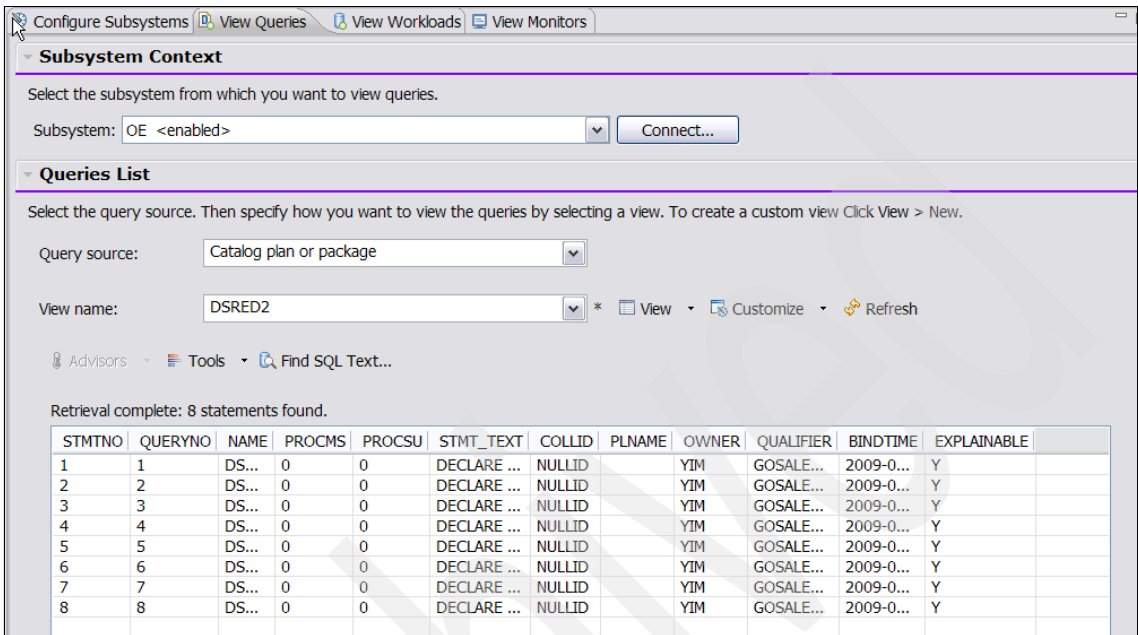


Figure 2-11 The SQL statements of package DSRED21 are listed in OE

Double-click the STMT_TEXT field in a row to bring up a panel showing the SQL statement. Figure 2-12 shows our query number 2 in the package. These are the same eight SQL statements that were retrieved by the DB2 Administration Tool in Example 2-11 on page 37. Both tools retrieve the SQL statement from the DB2 catalog.

The screenshot shows the DB2 Administration Tool interface. At the top, the 'Query source' is set to 'Catalog plan or package' and the 'View name' is 'DSRED2'. Below this, there are buttons for 'Advisors', 'Tools', and 'Find SQL Text...'. A message states 'Retrieval complete: 8 statements found.' Below this is a table with the following columns: STMTNO, QUERYNO, NAME, PROCMS, PROCSU, STMT_TEXT, COLLID, PLNAME, OWNER, QUALIFIER, BINDTIME, and EXPLAINABLE. The table contains 8 rows of data. A dialog box titled 'View SQL Statement' is open, showing the SQL statement for query number 2. The SQL statement is as follows:

```
DECLARE DB_PDQ_SPC2 CURSOR FOR SELECT
P.PRODUCT_NUMBER, P.BASE_PRODUCT_NUMBER,
P.INTRODUCTION_DATE, P.DISCONTINUED_DATE,
P.PRODUCT_TYPE_CODE, P.PRODUCT_COLOR_CODE,
P.PRODUCT_SIZE_CODE, P.PRODUCT_BRAND_CODE,
P.PRODUCTION_COST, P.GROSS_MARGIN, P.PRODUCT_IMAGE,
N.PRODUCT_NAME, N.PRODUCT_DESCRIPTION FROM
GOSALES.PRODUCT AS P, GOSALES.PRODUCT_NAME_LOOKUP AS N
WHERE P.PRODUCT_NUMBER = N.PRODUCT_NUMBER FOR READ
ONLY
```

The dialog box has an 'OK' button at the bottom right.

Figure 2-12 The SQL statement of query number 2 in the DESRED21 package

Assume this was a poorly performing SQL statement in the package identified by OMPE or QM. We can further analyze the SQL statement using the Access Plan Graph from the Tools drop-down menu. See Figure 2-13.

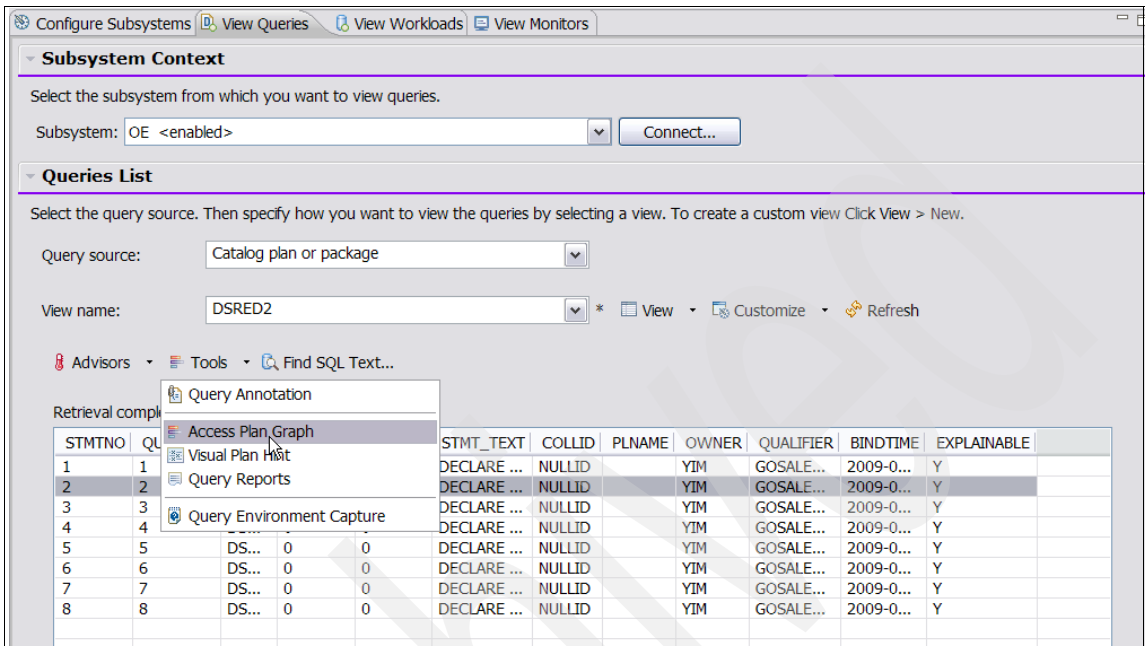


Figure 2-13 Select **Access Plan Graph** from the **Tools** drop-down menu

Select **Access Plan Graph** to visualize the access plan, as shown in Figure 2-14.

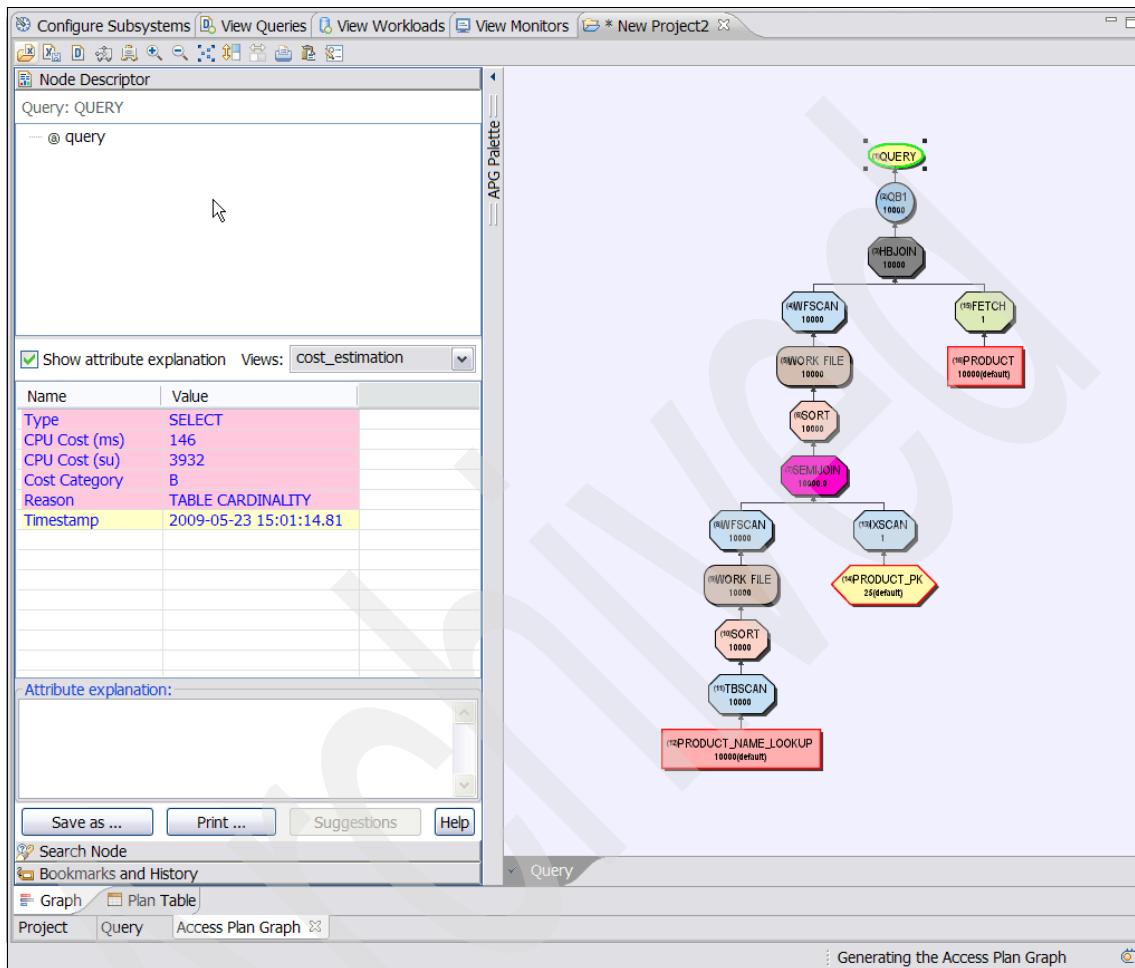


Figure 2-14 Access plan graph by OE

If you hover your cursor around the graph boxes of the access plan, OE provides summary information where the cursor is. If you click one of the boxes, OE provides you detailed information. While we hover over the graph, OE reminds us that Runstats was not run. If we run the Runstats Advisor, it, too, will generate the Runstats advice and include Runstats utility control statements. The Runstats advisor panel will show the Runstats control statement in the left and help descriptions on the right.

We check the table (GOSALES.PRODUCT_NAME_LOOKUP) information from the DB2 catalog directly using the DB2 Administration Tool and find that there is no statistical data available. See Figure 2-15.

```
[32 x 80]
File View Communication Actions Window Help

DB2 Admin ----- DB9B Interpretation of an Object in SYSTABLES ----- 19:03
Option ==>
Top of data
Details for table (label) : GOSALES.PRODUCT_NAME_LOOKUP

Table schema      : GOSALES      Table name       : PRODUCT_NAME_LOOKU >
Created by       : SULLIVA      Created          : 2009-05-07-16.44.36.523523
Table space name : GOSALE      Database name    : GSDB
Object ID for table: 12        DB ID for database : 274
Maximum row length : 327      Primary key OBID  : 64
Number of columns  : 4        Primary key columns:
Validate procedure : N/A      EDIT procedure name: N/A
Parent relations   : 0        Child relations   : 0
Auditing          : AUDIT NONE Status             : X - Primary key
Data capture       : No       Altered           : 2009-05-07-17.20.42.2
Restrict on DROP   : No       Check constraints  : None
Encoding scheme    : E - EBCDIC Col. in part. key : 0
Check pending flag : No       VOLATILE table    : No
Created in DB2 Ver : M - DB2 V9 Dependent MQTS     : 0
Data version       : 0
Table owner        : SULLIVA   Append specified   : No
Owner type         : Auth ID   Clone table name   :
Clone table schema :

Statistical data   : No valid data available
Row count          :
Occupied pages     :
Pct TS pages w/rows:
```

Figure 2-15 Accessing detailed information about a table using the DB2 Administration Tool

Although we can safely assume that we would need to run Runstats on this table, for demonstration purpose, let us use the Runstats Advisor from OE to illustrate the process and functions. Click the Advisors drop-down menu to bring up the Run Advisors drop-down menu. See Figure 2-16.

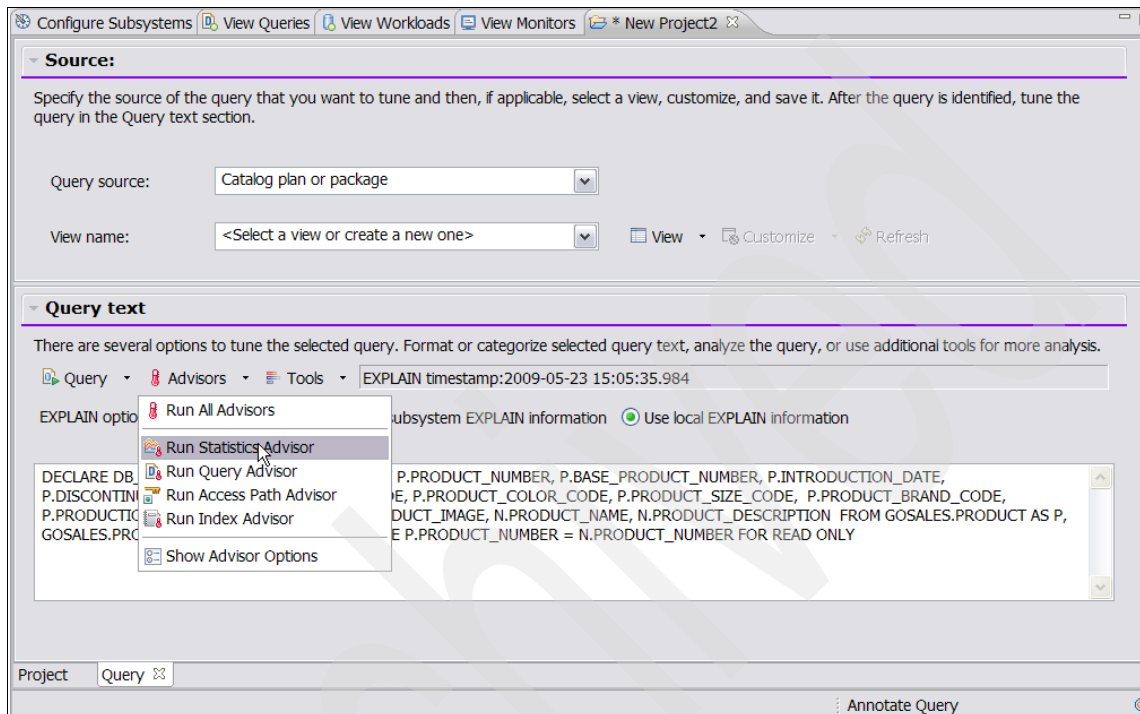


Figure 2-16 Selecting the Statistic Advisor from the drop-down menu in the Advisors tab.

Select **Runstats Advisor**. OE highlights the high priority alert using a red circle with a white horizontal bar inside on top of the panel. When we click the high priority, number 1, OE presents the Runstats control statement with a description on the right panel, as shown in Figure 2-17. To run Runstats, click **Run**.

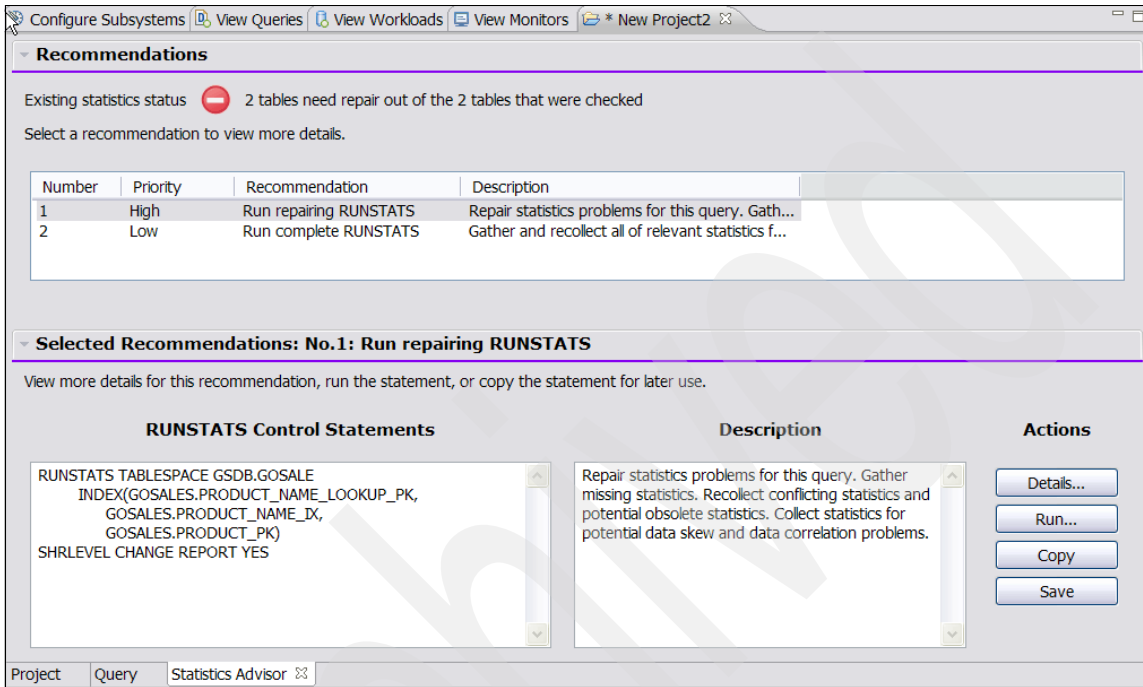


Figure 2-17 Runstats Advisor recommends to gather statistics

For more details about Optimization Expert, refer to *IBM DB2 9 for z/OS: New Tools for Query Optimization*, SG24-7421.

Note: In July 2009, Optimization Expert for z/OS was replaced by IBM Optim Query Tuner for DB2 for z/OS, V2.2 and Optim Query Workload Tuner for DB2 for z/OS, V2.2. For further details refer to the product Web page:
<http://www-01.ibm.com/software/data/optim/query-tuner-z/>

2.4 Optimizing an existing Java application using pureQuery

In the previous sections of this chapter, we demonstrated some of the key advantages that static SQL can provide. This section discusses the pureQuery client optimization feature that provides these benefits, along with others, to existing Java applications that use JDBC to dynamically process SQL.

As part of our discussion, we will cover the following topics:

- ▶ Provide an overview of the pureQuery client optimization process.
- ▶ Cover how to install, configure and validate the prerequisite software.
- ▶ Describe how to use Data Studio Developer to complete the process interactively.
- ▶ Identify alternatives to automate part of the process using the command line tooling.

Wherever appropriate, significant enhancements that are delivered to the pureQuery platform in version 2.2 will be highlighted.

2.4.1 PureQuery client optimization process overview

The process of enabling an existing Java application for pureQuery consists of four phases:

- ▶ **Capture**

The goal of the capture phase is to record the applications SQL. This captured SQL metadata provides the pureQuery platform with the ability to process SQL statically and enables it to provide additional features that can further improve performance and security and assist with common problem determination tasks.

- ▶ **Configure**

During the configuration phase, the captured SQL is mapped to one or more database packages (as needed) and the package characteristics, such as name and version are defined.

- ▶ **Bind**

The configured SQL to package mapping information is used in the bind phase to create the required database packages, that enable the Data Studio (now Optim) pureQuery Runtime to process the applications SQL statically.

► Execute

After the capture, configure and bind phases have been completed, the pureQuery Runtime can be configured to process SQL in the desired execution mode.

This process can be performed in an iterative manner, if necessary.

In the following sections we discuss the four phases in more detail and complete the following tasks:

1. Set up the pureQuery components. (2.4.2, “Setting up pureQuery” on page 52)
2. Configure the applications run time environment to capture its SQL. (2.4.3, “Capturing SQL from a Java application” on page 66)
3. Map the captured SQL to one or more database packages. (2.4.4, “Configuring the captured SQL metadata” on page 73)
4. Bind the database packages. (2.4.5, “Binding the captured and configured SQL” on page 79)
5. Configure the applications run time environment to execute SQL statically(2.4.6, “Configuring the Runtime to execute SQL statically” on page 82)

2.4.2 Setting up pureQuery

To enable static SQL processing for the target application using the pureQuery client optimization feature, you have to install the pureQuery Runtime on the application server node and IBM Data Studio Developer on any client node, as illustrated in Figure 2-18. Note that there is no requirement to install software on the database node.

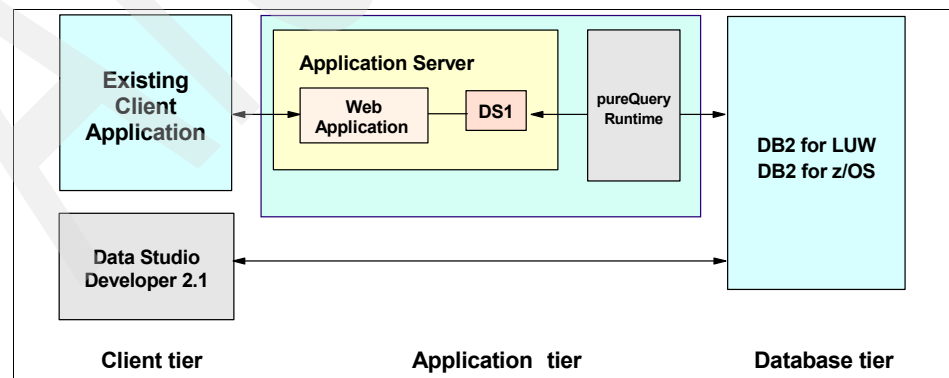


Figure 2-18 Three-tier topology example pureQuery deployment

Note: The client optimization process is unaware to the general topology being used.

Figure 2-19 depicts the lab environment that was used in this book. The Java application that has been selected as a candidate for the client optimization process is running in WebSphere Application Server version 6.1 on a Windows 2003 server. This application accesses DB2 for z/OS version 9.

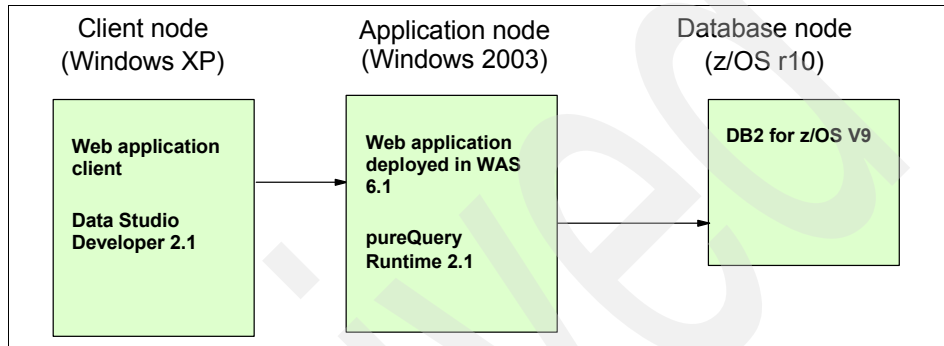


Figure 2-19 Lab environment topology

The Web application GO_Sales was deployed on the application server using the WebSphere Administration Console and was configured to access a DB2 for z/OS database named DB9B using a DataSource that uses a DB2 type-4 JDBC driver.

The Web application is driven by a client application that can simulate typical usage scenarios for the Great Outdoor Company. The simulation was carried out using the Apache Jmeter test suite, available at the following Web Page:

<http://jakarta.apache.org/jmeter/>

The utility simulated 200 users (with a two minute ramp up time), with a maximum number of 1000.

Installation considerations

In preparation for the installation process, create a checklist to identify which pureQuery components you need to install, what the prerequisites for those components are, and whether your environment meets those prerequisites. The release-specific system requirements for each component are listed on the product Web page.

You should also have a basic understanding how the target application's database access is configured, and how the application environment can be modified to accommodate the pureQuery Runtime. General aspects will be covered in the following sections.

Note: The following sections contain links to the latest version of the Integrated Data Management Information Center.

Installing Data Studio Developer

Data Studio Developer provides sophisticated tooling that visualizes captured SQL metadata, allows for its configuration, package creation and basic package management.

To install Data Studio Developer on a client tier node:

1. Download Data Studio Developer for the client platform of your choice. A trial version can be downloaded from the following Web page:

<http://www.ibm.com/developerworks/spaces/optim>

A list of version specific prerequisites can be found at the following Web page:

<http://www-01.ibm.com/support/docview.wss?rs=3368&uid=swg27014476>

2. Follow the installation instructions described in the installing and configuring section of the integrated Data Management Information Center and install Data Studio Developer on the client tier node:

http://publib.boulder.ibm.com/infocenter/idm/v2r1/index.jsp?topic=/com.ibm.datatools.dwb.nav.doc/topics/helpindex_dwb_sdf.html

3. Download and install any applicable Data Studio Developer Fix packs from the support Web site at the following Web page:

<http://www-01.ibm.com/software/data/studio/download.html>

Installing the pureQuery Runtime

The pureQuery Runtime provides the core functionality that enables the pureQuery technology to process SQL statements statically without the need to implement a Java application using the SQLJ language extension, which was used traditionally to provide static SQL support.

Follow these steps to install the pureQuery Runtime on the application tier node:

1. Download the pureQuery Runtime for the platform of your choice for which you have acquired a license. (No trial version of the pureQuery Runtime is available.) A complete list of supported operating systems can be found at the following Web page:

<http://www-01.ibm.com/support/docview.wss?rs=3369&uid=swg27014394>

2. Follow the installation instructions described in the Integrated Data Management Information Center to install the pureQuery Runtime on the application tier node. Two topics discuss platform specific installation aspects:

- *Installing Optim pureQuery Runtime for Linux, UNIX, and Windows*

http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.runtime.overview.doc/topics/helpindex_pq_sdf.html

- *Installation Guide for Optim pureQuery Runtime for z/OS Version 2.2*

http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.runtime.zos.ig.doc/install_ds_pqruntime_zos_22.html

3. Download and install any applicable pureQuery Runtime fix packs from the support Web site:

<http://www-01.ibm.com/software/data/studio/download.html>

Installing the IBM Data Server Driver for JDBC and SQLJ

The pureQuery Runtime requires a compatible level of the IBM Data Server Driver for JDBC and SQLJ, which is also referred to as Java Common Client (JCC) driver. To determine which driver is compatible with the Runtime you are planning to use, refer to the following Web page:

<http://www.ibm.com/support/docview.wss?rs=3369&uid=swg27014394>

Note: The pureQuery Runtime installation package does not include a JCC driver.

To install a compatible driver:

1. Download the appropriate IBM Data Server Driver for JDBC and SQLJ, for example, from the DB2 Fix Pack Web site

<http://www-01.ibm.com/support/docview.wss?rs=71&uid=swg21288110>

2. Install the driver on the application tier node.

Note: The pureQuery Runtime will not work properly if you are using an incompatible version of the IBM Data Server Driver for JDBC and SQLJ.

Configuring the pureQuery Runtime

Applications can only take advantage of the pureQuery Runtime functionality if the pureQuery Java libraries can be loaded by the classloader of the Java virtual machine in which the application executes and if a pureQuery configuration has been defined.

A pureQuery configuration consists of a set of properties that enable, disable, or define the behavior of features. Configurations can be defined at different scopes:

- ▶ At the database connection level (for example, as part of the connection string)
- ▶ At the application level, at the data source level (as a custom property)
- ▶ For the entire Java Virtual machine (as JVM parameter or in a properties file)

For more information refer to *How to set properties for client optimization* in the Information center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.static.doc/topics/rpdqprfhowsetprp.html>

Note: To decide at which scope to define the pureQuery configuration, consider the following: Shall only part of an application, an entire application or multiple applications be governed by a single configuration?

It is often desirable to define one configuration per application. If multiple applications share the same configuration, SQL metadata for these applications is treated by the Runtime as though it belongs to a single application. Packages that are created based on this metadata therefore no longer allow for an easy 1-to-1 correlation between SQL and the originating application (because multiple applications would share the same packages). This can have an impact on problem determination efficiency and workload management.

A property (or generally speaking configuration setting) is identified by a well-defined keyword and is assigned a value. Some properties are mandatory, and some are optional. Optional properties might have a default value assigned that can be overridden.

Note: Property names are case sensitive and are ignored if mistyped. Property values are generally not case sensitive, unless they refer to file system objects, for example on UNIX or Linux.

Some pureQuery properties are specific to the client optimization technology, some are used to configure the behavior of applications that were implemented using the pureQuery API and some are used for both.

Note: Java applications that implement data access using a particular pureQuery API (called method-style) can take advantage of the pureQuery benefits without the need to go through the client optimization process. Chapter 3, “Managing pureQuery application deployment” on page 89 discusses this in more detail.

The most commonly used client optimization related properties are as follows:

- ▶ **executionMode**
This property defines the SQL execution mode. Valid values are DYNAMIC and STATIC. The default is DYNAMIC.
- ▶ **captureMode**
This property enables capturing of SQL that executed successfully. Valid values are ON and OFF, with OFF being the default.
- ▶ **pureQueryXml**
This property identifies the name and location of the SQL metadata file where captured SQL is to be stored if the captureMode property is set to ON. This property must also be specified if executionMode is set to STATIC, because it provides the Runtime with the necessary information that allows for special processing. No default value is defined.
- ▶ **allowDynamicSQL**
This property allows for dynamic SQL execution if SQL cannot be processed statically. Valid values are TRUE and FALSE, with TRUE being the default).

Other properties define additional aspects of the SQL execution and SQL capture process. For a complete list, refer to *Properties that determine the actions of client optimization* in the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.static.doc/topics/cpdqprfprp.html>

Enabling the Great Outdoor Company Web application for pureQuery

The pureQuery Runtime technology was designed to be application server unaware. It can optimize stand-alone applications and applications running in any J2EE container as long as a supported version of Java (1.5 at the time of writing) is used. Generally speaking, as long as an application uses a JDBC connection, it can be enabled for pureQuery.

The Great Outdoor Company Web application, which is deployed in WebSphere Application Server version 6.1, takes advantage of the data source concept, which provides an abstraction layer that enables applications to access databases from different vendors. In this chapter, we enable a data source for pureQuery and configure the application to access the database using this customized data source. Other applications that might be running on the same application server are not impacted unless they access the same data source.

To enable the pureQuery Runtime that we have installed on the application server node, we have to make the pureQuery Runtime Java libraries available to the application's data source and configure the Runtime behavior so the client optimization process can be completed.

A data source in WebSphere Application Server is configured in two steps:

1. Configure a provider that makes a database-specific client infrastructure available.
2. Configure a data source that provides access to a particular database using the infrastructure that above provider makes available.

In the following three sections we guide you through the configuration process and outline basic validation tasks that can be used to identify common setup issues.

Creating a pureQuery-enabled JDBC provider in WebSphere Application Server

Because the Great Outdoor Company Web application accesses a DB2 database, we must create a DB2 JDBC provider on WebSphere Application Server. Then we create a data source named GoSalesApp, which is mapped to the application.

Use these steps to create a WebSphere Application Server JDBC provider that makes use of the pureQuery Runtime libraries:

1. Start the application server.
2. Launch the WebSphere Application Server Administration Console and log in as Administrator.
3. Go to **Resources** → **JDBC** → **JDBC Providers**.
4. Create a DB2 JDBC Provider DB2 Universal JDBC Driver Provider (DB2, DB2 Universal JDBC Driver Provider, connection pool data source). Accept all default settings to create a standard data source definition.

Note: To simplify administration, we have copied the pureQuery Runtime libraries along with the IBM Data Server Driver for JDBC and SQLJ into a dedicated directory, and have defined a WebSphere environment variable named DB2UNIVERSAL_JDBC_DRIVER_PATH, which points to this directory.

Figure 2-20 depicts a basic DB2 JDBC provider definition.

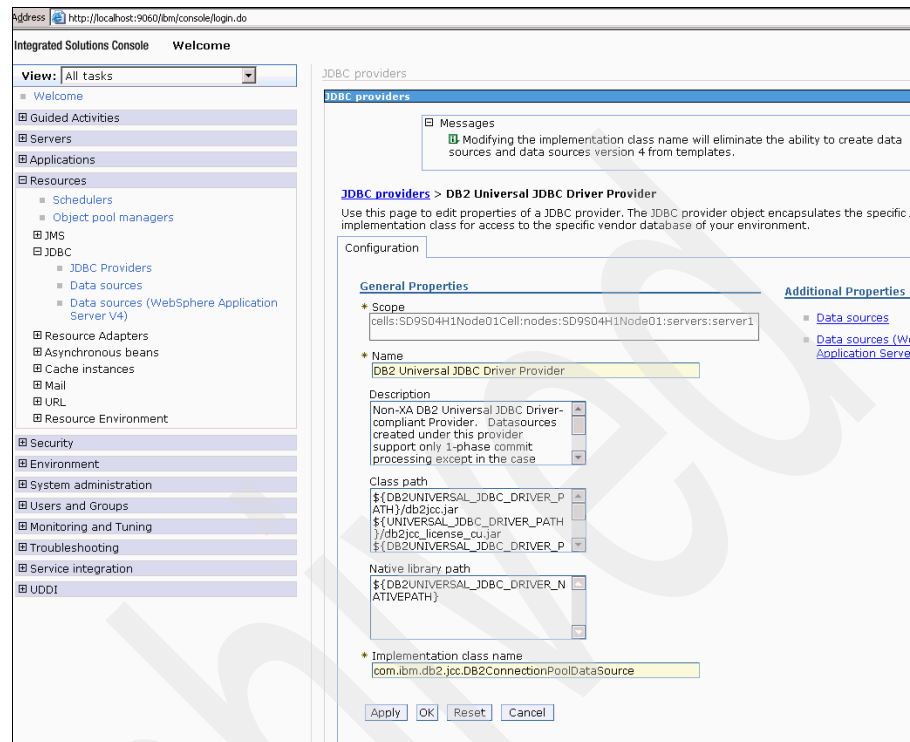


Figure 2-20 DB2 JDBC Provider setup panel in WAS Administration Console

5. Customize the JDBC provider configuration properties to use the pureQuery Runtime libraries and the compatible JDBC drivers. Add the following four JAR files (from the pureQuery Runtime installation and the IBM Data Server Driver for JDBC and SQLJ installation) to the CLASSPATH definition of this provider:

- \${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar
- \${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar
- \${DB2UNIVERSAL_JDBC_DRIVER_PATH}/pdq.jar
- \${DB2UNIVERSAL_JDBC_DRIVER_PATH}/pdqmgmt.jar

6. Save the changes.

Upon successful configuration, this JDBC provider is enabled to take advantage of the pureQuery Runtime Java libraries. Next, you must create a new data source that uses the JDBC provider you have just created.

Creating a WebSphere Application Server data source configuration

A data source configuration identifies the database that this data source makes available and defines various connection and infrastructure properties.

1. Go to **Resources** → **JDBC** → **Data sources**.
2. Create a new data source on the JDBC provider that you created in “Creating a pureQuery-enabled JDBC provider in WebSphere Application Server” on page 59.
3. Configure the required database connection properties.
4. Save the new data source definition.

Figure 2-21 shows GoSalesApp data source as configured in WebSphere Application Server.

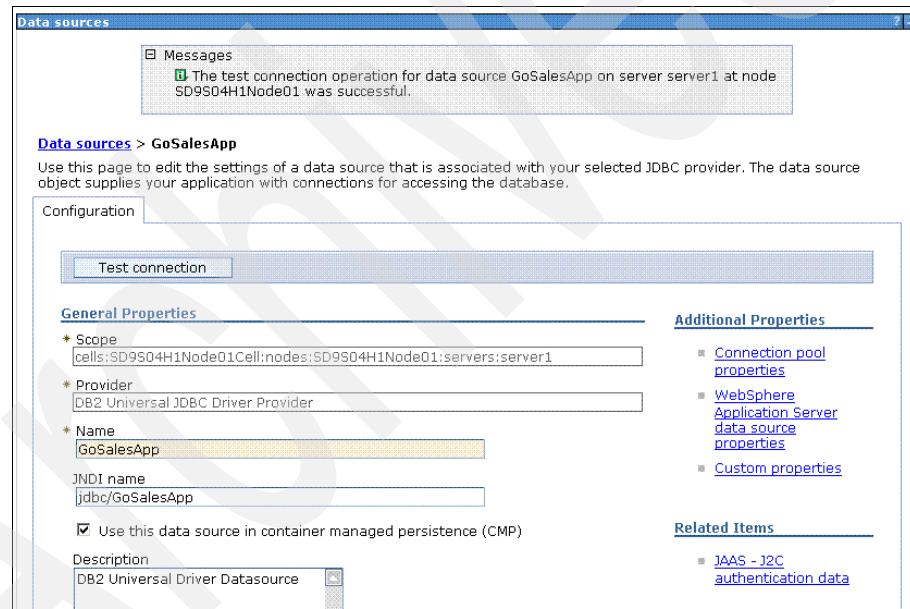


Figure 2-21 Test connection is successful for GoSalesApp data source

5. Map the applications data source reference to this data source.

Note: Changes to a data source configuration in WebSphere Application Server take effect after the application server has been restarted. The instructions in the next section (“Validating the WebSphere Application Server data source configuration” on page 62) ask you to modify the default configuration for this data source to enable some pureQuery functionality. Therefore, there is no need to restart the application server at this point.

Thus far, this data source is not yet configured to make use of any pureQuery client optimization functionality.

Validating the WebSphere Application Server data source configuration

The following steps validate that the data source has been properly configured for use by the pureQuery Runtime. We suggest that you perform this validation to prevent common issues caused by mistyped file names, duplicate or incompatible JDBC driver libraries, missing file permissions, and so on.

The first validation ensures that a connection to the newly-created data source can be established, that the JDBC driver is aware of pureQuery, and that the pureQuery Runtime libraries can be loaded.

1. Launch the WebSphere Application Server Administration Console and log in as Administrator.
2. Open the newly created data source definition by navigating to **Resources** → **JDBC** → **Data sources** → **GoSalesApp**.
3. Click **Custom Properties**.
4. Add a new custom property that configures the pureQuery Runtime behavior:
 - a. Name the property `pdqProperties`.

Note: Custom properties names are case sensitive, like pureQuery properties.

- b. Assign the following value to this property:
`executionMode(STATIC),pureQueryXml(/path/to/nonexistent/capture.pdxml)`

Replace `/path/to/nonexistent/` with the name of an existing directory. No file named `capture.pdxml` should exist in this directory.

Note: The configuration above is commonly used to perform a negative test, which expects a certain operation to fail with a specific error message. The configuration instructs the pureQuery Runtime to statically process SQL statements that are located in a non-existing pureQuery metadata file named `/path/to/nonexistent/capture.pdqml`. Because the file does not exist, the Runtime should reject execution of any SQL statement because it is unable to determine which SQL can be processed statically.

If the test, which you perform next, fails as expected, the following has been verified:

- ▶ The database connection information is correct.
- ▶ The pureQuery Runtime libraries (`pdq.jar` and `pdqmgmt.jar`) can be loaded.
- ▶ The appropriate pureQuery license has been applied.
- ▶ The data sources' JDBC driver is compatible with the pureQuery Runtime libraries.

- c. Save the data source configuration.
5. Restart the application server.
6. Launch the WebSphere Application Server Administration Console and log in as Administrator.
 - a. Open the previously edited data source definition by navigating to **Resources** → **JDBC** → **Data sources** → **GoSalesApp**.
 - b. Click **Test connection**.

Note: This operation connects to the database and submits a dummy SQL statement for processing.

The connectivity test should fail, indicating that file `/path/to/nonexistent/capture.pdqml` was not found. If this test succeeds or a different error is displayed, check your JDBC provider and data source settings. One or more configuration or installation issues need to be resolved.

Note: If the test does not complete as expected, check the application server log files (SystemOut.log and SystemErr.log) for additional error specific information. Validate the following information:

► JDBC provider definition

- The pureQuery Runtime library names (pdq.jar and pdqmgmt.jar) are spelled correctly, located in the specified directory, and accessible to the ID that is used to run the application server.
- The JDBC driver library names (db2jcc.jar and db2jcc_license_cisuz.jar) are spelled correctly, located in the specified directory, and accessible to the ID that is used to run the application server.
- The JDBC driver version is compatible with the pureQuery Runtime version. The application server log file typically displays a data source's JDBC driver version.
- The appropriate pureQuery Runtime license has been applied. To verify the license information, launch program pqactivate.bat (pqactivate.sh on UNIX) (located in the activate directory of your pureQuery Runtime installation), and select **List the active priced features**. An entry named **runtime** should be listed.

► Data source configuration

Custom property pdqProperties is present and its name and values not mistyped.

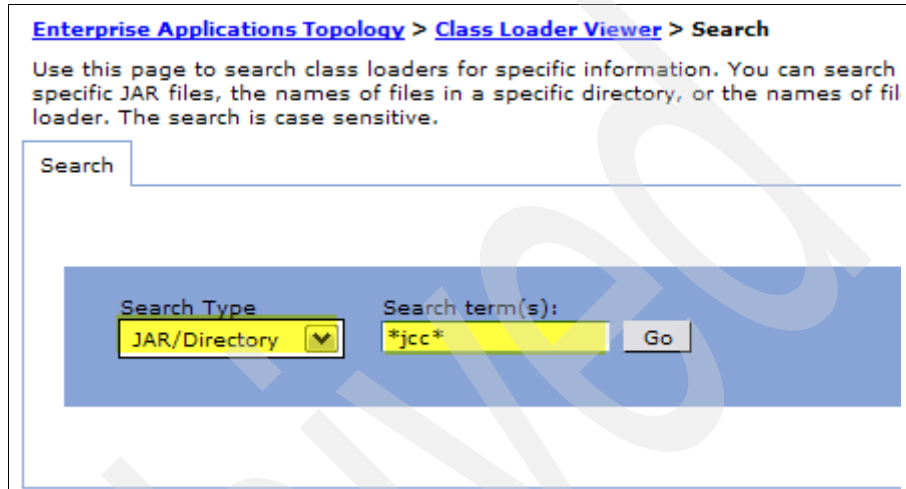
► CLASSPATH definition

The application server environment might be configured to load older versions of the JDBC driver (or the pureQuery Runtime) before the libraries that you have specified in the JDBC provider definition. Perform the steps that follow to determine whether the expected Java libraries are being loaded by the application server.

The second test can be used to validate that the expected IBM Data Server driver for JDBC and SQLJ are used by this data source:

1. Open the Class Loader Viewer by navigating in the WebSphere Administration Console to **Troubleshooting** → **Class Loader Viewer**.
2. Locate the target application in the **Enterprise Applications Topology** tree.
3. Click the application Web module link. In our case it is **Go_Sales.war**.

4. Click **Search** in the Class Loader Viewer to determine from which location the Web application is loading the JDBC drivers and the pureQuery Runtime libraries.
5. Select **JAR/Directory** as the search type and enter `*jcc*` as the search term, as shown in Figure 2-22.



Enterprise Applications Topology > Class Loader Viewer > Search

Use this page to search class loaders for specific information. You can search specific JAR files, the names of files in a specific directory, or the names of file loader. The search is case sensitive.

Search

Search Type: JAR/Directory ▼ Search term(s): Go

Figure 2-22 Using the Class Loader Viewer to validate that the JDBC and pureQuery libraries are loaded from the expected directory

Make sure that the search result indicates that the DB2 JDBC driver (`db2jcc.jar`) is loaded from a directory that contains a compatible JDBC driver. An unexpected directory location could be, for example, a subdirectory of a DB2 V8 installation.

6. Repeat steps 4 and 5 for the pureQuery libraries, by entering `*pdq*` as the search term.

The expected location for `pdq.jar` and `pdqmgmt.jar` is the directory that you have specified in the classpath of the JDBC provider configuration.

Successful completion of both tests is an indication that the environment is configured properly.

2.4.3 Capturing SQL from a Java application

To enable static SQL execution for an existing Java application, you have to capture its SQL statements. Figure 2-23 depicts at a high level the capture phase of the pureQuery client optimization process:

- ▶ The pureQuery Runtime environment is configured to capture successfully executed SQL into a file.
- ▶ The most common use cases for an application are run, causing SQL to be submitted for execution to the JDBC driver.
- ▶ The SQL is forwarded to the database for dynamic execution and the result retrieved.
- ▶ The pureQuery Runtime stores information about successfully executed SQL in an external proprietary metadata file.
- ▶ The result is returned to the application.

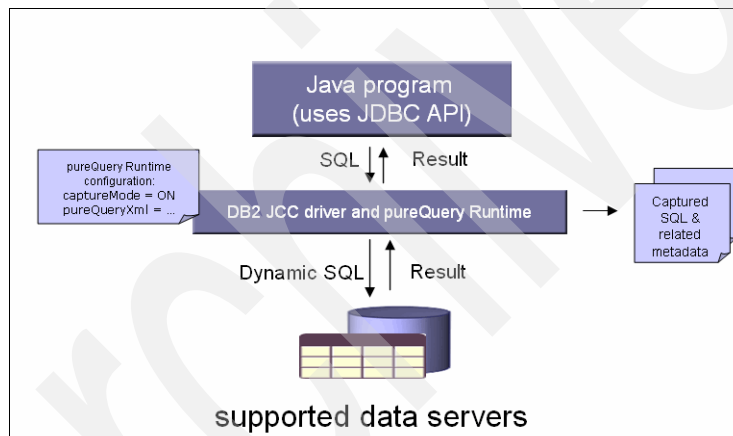


Figure 2-23 pureQuery client optimization - capture process

The capture process is transparent to the application and is typically performed using an automated process in a development or test environment. It is suggested that you not capture SQL in a production environment to avoid the overhead associated with this activity.

Two configuration settings are required to enable the Runtime's capture mode: captureMode and pureQueryXml. If set to ON (OFF is the default), the captureMode property instructs the Runtime to capture successfully executed SQL. The pureQueryXml property identifies the name and location of a file where the captured information is saved for later processing. The file is created if it does not exist or is appended to if it exists.

Generally speaking, three aspects of the capture process can be customized, impacting the amount of information that is gathered, and therefore the operations that can be performed based on the collected information:

► Which SQL is captured?

Any SQL that executes successfully is captured. Certain limits can be imposed by configuration settings such as `captureStatementBatchSQL` that, if enabled (default), causes batched SQL DML to be captured and `maxNonParmSQL`, which restricts the number of non-parametrized SQL that is recorded (unlimited by default).

Note: The ability of the pureQuery Runtime to perform special SQL processing (such as execute statically) is impacted by these settings.

► How much information related to the originating application is gathered?

The amount of information being captured can be customized according to your needs. For example, a limit can be imposed on the number of source code layers to which SQL can be traced back (`stackTraceDepth`), the software layers that are to be ignored (`packagePrefixExclusions`) or the number of distinct locations in the source code where the same SQL is executed (`maxStackTracesCaptured`).

To determine how much information you need to capture, you have to decide how much information you (or developers) would like to have available for trouble-shooting and what-if analysis tasks:

- Do you need to know every location in the source code where the same SQL is processed, or is it sufficient to know that a particular SQL is executed?
- How much of the Java stack trace information is relevant? If, for example, an application runs in an application server, the stack trace includes information about modules that belong to the infrastructure that might be relevant to you.

The more information the Runtime captures, the larger the metadata file grows.

Note: Special Runtime SQL processing (such as static execution) is not impacted by any of these settings.

► Is SQL special processing required?

By default, SQL is captured as is. In version 2.2 of pureQuery, additional functionality is made available that allows for the consolidation of SQL by removing literal values. Refer to 5.2.1, “SQL literal substitution” on page 155 for more details.

A complete list of the Runtime configuration settings that impact the SQL capture process, along with motivations when to enable certain features, can be found in *Descriptions of properties for capturing and running restricted sets of SQL statements dynamically* in the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.static.doc/topics/cpdqpqxmlnondb2dcr.html>

Recommendation: The IBM Data Server Driver for JDBC and SQLJ can be configured to defer the SQL prepare requests until the SQL is executed by the application. Doing so, by setting the JDBC driver property `deferPrepares` to `TRUE`, can reduce network delays. However, we suggest disabling deferred prepares, by setting the `deferPrepares` property to `FALSE`, while the pureQuery Runtime is configured to capture SQL.

Capturing SQL for the Great Outdoor application in WAS

The following steps guide you through the process of configuring a pureQuery-enabled WebSphere data source to capture SQL using the default settings:

1. Launch the WebSphere Application Server Administration Console and log in as Administrator.
2. Open the target applications data source definition by navigating to **Resources** → **JDBC** → **Data Source** → **GoSalesApp**.
3. Click **Custom Properties**.

The red arrow in Figure 2-24 on page 69 shows the location of the **Custom Properties** link.

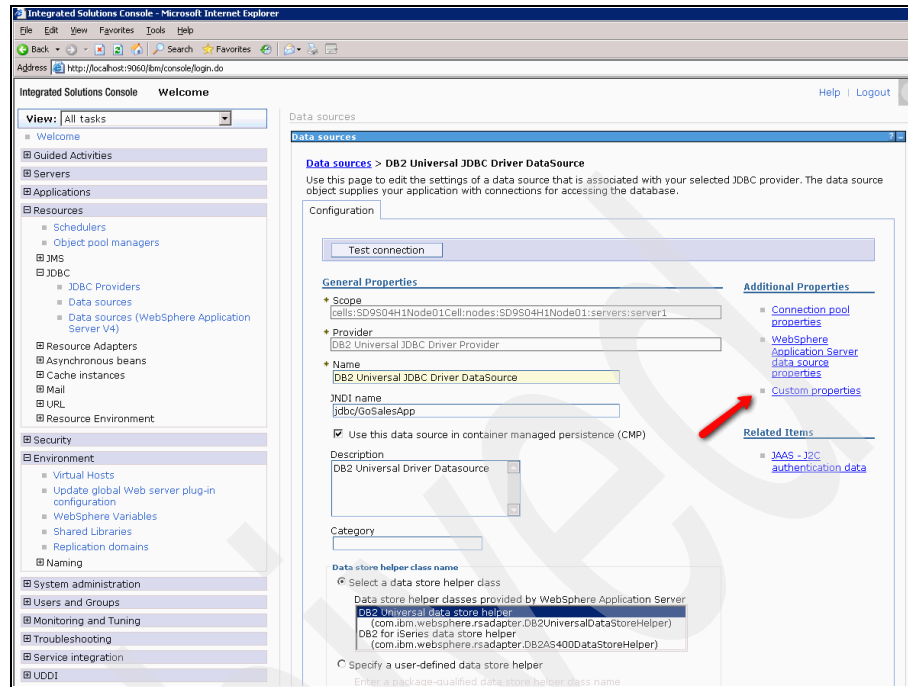


Figure 2-24 Custom Properties is found when you click data source

4. Add a new custom property that defines the pureQuery Runtime environment configuration (or modify the property value if the property is already defined).

- a. Name the property pdqProperties.

As indicated earlier, two mandatory properties have to be configured to enable the capture process: captureMode and pureQueryXml. For illustrative purposes we are also defining the SQL execution mode as dynamic.

- b. Assign the following value to pdqProperties:

executionMode(**DYNAMIC**), pureQueryXml (/path/to/capture.pdqxml), captureMode(**ON**)

This configuration instructs the Runtime to execute all SQL dynamically and capture successfully executed SQL statements in /path/to/capture.pdqxml. Note that the directory /path/to/ must exist or the capture operation will fail. The file will be created if it does not exist, or appended to if it exists.

Note: Special consideration has to be given if an application runs in a clustered environment, because each cluster might process a subset of the SQL, raising the need to consolidate the captured SQL to make it available to all clusters. Version 2.2 of pureQuery provides the necessary support through a new configuration property named `outputPureQueryXml` and a utility that allows for merge of multiple capture files. Refer to *Capturing SQL statements from Web applications that are vertically clustered, horizontally clustered, or both* in the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.static.doc/topics/tpdqpxmlnondb2capsqlclswebappverthori.html>

Similarly noteworthy is a version 2.2 feature which allows for capturing SQL statements from Web applications that share a data source as described in the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.static.doc/topics/tpdqprfcapsqlclswebappshrdatsrc.html>

- c. Save the data source configuration and restart the application server.
- d. Run the client application that drives the Web applications workload.

Note: In a typical scenario you will likely capture SQL incrementally, meaning only SQL that was previously not captured is recorded in subsequent iterations. The pureQuery Runtime appends SQL that was previously not captured to the end of the capture file.

- e. Stop the application server after the workload has been completely processed.

Stopping the application server forces the pureQuery Runtime to write all in-memory SQL metadata information to disk.

The captured SQL metadata can now be processed using Data Studio Developer.

Note: If your application server node is different from the client node, transfer the SQL metadata file to the client node where Data Studio Developer is installed. Make sure to transfer the file in binary mode to prevent data corruption.

Importing captured SQL metadata into DSD

Data Studio Developer provides an easy-to-use set of tools that can be used to browse, configure, and bind the captured SQL. The following steps walk you through the process of importing the captured SQL file into Data Studio Developer.

1. Launch Data Studio Developer.
2. Select a new workspace, for example, C:\dsredbook\.
3. Switch to the Java perspective.
 - a. Select from the tool bar **Window** → **Perspective** → **Java**.
 - b. Open the Data Source Explorer view (it provides access to local and remote databases) **Window** → **Show View** → **Other ...** → **Data Management** → **Data Source Explorer**.
4. Create a new database connection for the database that was accessed by the application. (In our example, the z/OS database is DB9B.)
 - a. In the Data Source Explorer right-click the Database Connections node.
 - b. Select **New** to create a new database connection.
 - c. Select the appropriate database manager and enter database, host, port number, user name, and password information for the DB9B database.
 - d. Click **Test Connection** to validate database connectivity.
 - e. Click **OK** to save the new database configuration.

The DB9B database should now be displayed in the Data Source Explorer.
5. Create a new Java project (into which you will import the SQL metadata file).
 - a. Select from the tool bar **File** → **New** → **Java project**.
 - b. Enter a project name of your choice, for example, DSREDBOOK.
 - c. Click **Finish** to create the project.

Note: You do not have to know how to program Java to use the relevant pureQuery tooling, even though the perspective name and project type are both named Java.

6. Enable pureQuery support for the newly created project.
 - a. In the Package Explorer right-click the DSREDBOOK Java project and select **pureQuery** → **Add pureQuery support** from the context menu.
 - b. Select the DB9B database connection and click **Next..** This association between the project and the database provides the tooling with access to the database, which, for example is required when we bind the captured SQL to a package.

- c. Select the **Enable SQL capturing and binding for JDBC applications** check box.
 - d. Override the default schema with the qualifier that you would like to assign to unqualified database objects that are referred to in the captured SQL. (You might ask the DBA who created the DB9B tables which schema was used when the tables were created.)
 - e. Click **Finish**.
7. Import the SQL metadata file into the DSREDBOOK project.
- a. Select from the tool bar **File** → **Import ...** to launch the import wizard.
 - b. Expand the General folder and select **File System** to import the SQL metadata file, which you have stored on your local hard disk.
 - c. Click **Next**.
 - d. In the From directory input box, browse to the directory where you stored the SQL metadata file.
 - e. Select the box next to the file name to select it. Select the Into folder input box and browse to directory DSREDBOOK/pureQueryFolder.
 - f. Select the **Create selected folders only** option.
 - g. Click **Finish** to import the selected file into the pureQuery folder in the DSREDBOOK project.
 - h. Locate the SQL metadata file in the Package Explorer. If it is not displayed in the pureQueryFolder directory, select the pureQueryFolder folder and press F5 to refresh the view.
 - i. Locate the pureQuery Outline view. It should display the captured SQL statements.
 - j. If you are unable to find this view, select the DSREDBOOK project in the Package Explorer, right-click and select **pureQuery** → **Show pureQuery Outline** from the context menu.

If the pureQuery Outline view does not display any data, click the outline refresh button.

You are ready to work with the imported SQL metadata.

2.4.4 Configuring the captured SQL metadata

The captured SQL metadata is by default not mapped to a database package and can therefore not be executed by the Runtime in static execution mode.

The goal of the configuration phase is to define the logical mapping between the captured SQL and one or more packages. As part of this process the general package characteristics such as name, collection id and version are defined, as depicted in Figure 2-25. Note however, that the actual package (or packages) are not yet created. You can use the tooling in Data Studio Developer to interactively define the mapping and package characteristics or the command line tooling to partially automate the process.

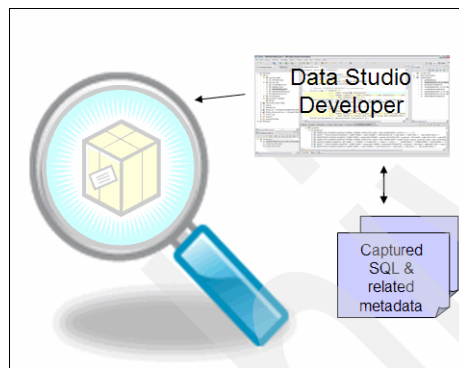


Figure 2-25 PureQuery client optimization configure process overview

Note: The configuration operation enriches the SQL metadata file with mapping and package information. Without this information the pureQuery Runtime would not be able to process SQL statically.

Detailed information about the configure utility can be found in Configure utility of the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.utils.doc/topics/rpdqprfsyncnf.html>

The following steps walk you through the configuration using Data Studio Developer.

Configuring the SQL package mapping and package characteristics using DSD

SQL package mapping and package characteristics can be defined in Data Studio Developer using a configuration file named `Default.genProps`. This file is automatically created in the `pureQueryFolder` of each project that is `pureQuery`-enabled.

To configure the SQL package mapping, perform the following steps:

1. In the Package Explorer expand the **DSREDBOOK** project, expand folder `pureQueryFolder` and double-click the **Default.genProps** file to open it in an editor. Each entry in this file uses the syntax shown in the example below:

```
<pureQuery artifact name> = <configuration settings for this artifact
```

A `pureQuery` artifact name is, for example, the name of an existing SQL metadata file. The associated configuration settings define the package mapping and the package characteristics. A special kind of entry named `defaultOptions` can be specified, defining default configuration options that apply to all artifacts. Configuration options that are specified for a particular artifact name override the default configuration settings.

2. The configuration file contains, by default, an entry that defines the mapping between an SQL metadata file (`capture.pdqml`) and a database package (DSRED). The default package name is derived from the first few characters of the project name in which the metadata file is located. The following path name shows the default capture file SQL package mapping entry in `Default.genProps`:

```
C:\dsredbook\DSREDBOOK\pureQueryFolder\capture.pdqml = -rootPkgName  
DSRED
```

Note: If no entry for the imported metadata file exists, right-click and select **pureQuery** → **Add or Remove Entries** from the context menu to create one.

3. You can change the default package name to a more descriptive name. By choosing a descriptive package name one can easily correlate the content of a database package with the corresponding metadata file. In our example, we use `DSRED2` to indicate that this package is associated with the application that we use. The following path name shows the customized SQL to package mapping in `Default.genProps`:

```
C:\dsredbook\DSREDBOOK\pureQueryFolder\capture.pdqml = -rootPkgName  
DSRED2
```


Note: The `-rootPkgName <base package name>` option is used to provide a base name that the tool will use to generate the name of one (or more if necessary) packages that will hold the captured SQL.

4. Additional package characteristics can be defined by appending options to the end of the line.

Note: Most editors in Data Studio Developer support content assistance. Pressing `<Ctrl>+<space>` in an editor opens up a list of supported options or values that can be specified. This concept is commonly used in integrated development environments to improve productivity and reduce common errors (such as typos).

For example, enter a space after the package name, press `<Ctrl> + <Space>`, choose **-collection** from the context menu and enter a collection ID under which the package will be created. If this option is not specified, NULLID will be used.

5. Save the updated `Default.genProps` file by pressing `<Ctrl> + <s>`. The tooling detects that the configuration file has changed and displays a warning indicating that the DSREDBOOK project has to be updated.
6. Select **Yes** to rebuild the project. The changes that you have made to the configuration file will, upon completion of this task, be reflected in the SQL metadata (`capture.pdqml`) file.

Now that you have configured the database package characteristics you can preview the content and characteristics of the database package using the pureQuery Outline view.

Note: In release 2.1, all SQL statements that are stored in a single SQL metadata file are mapped to the same package. With the introduction of the incremental configure feature in release 2.2, SQL statements from a single capture file can be mapped to individual packages, providing greater control over the package content. For example, SQL that accesses a particular set of tables can be manually grouped so that it resides in a single package, whereas other SQL resides in other packages.

7. If you do not have the pureQuery Outline view open, open it by selecting your Java project in the Package Explorer and right-clicking the Java project and going to **pureQuery → Show pureQuery Outline View**.

8. In the pureQuery Outline view, select the SQL tab. If nothing is shown, click **Refresh**. Figure 2-26 shows a snapshot of the captured SQL and its mapping to a package named DSRED2. Only statements that are shown in this tab will be bound to a package during the bind phase.

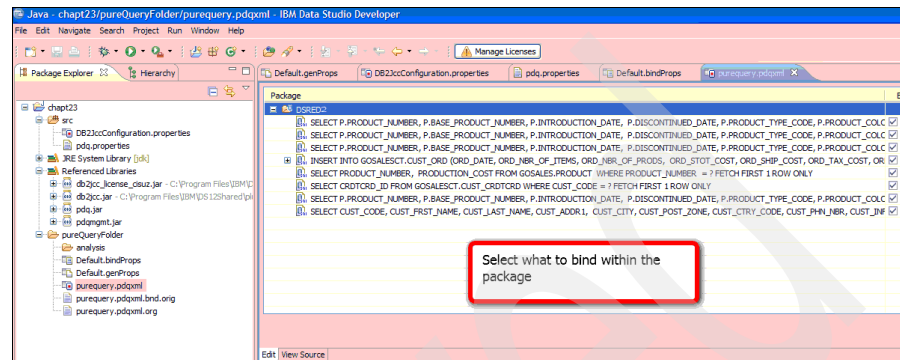


Figure 2-26 View SQL statements that will be bound to a package

9. You should see the package name that you have associated with the capture file when you customized the package mapping using the `-rootPkgName` option in configuration file `Default.genProps`.

Note: The Database and Java tabs display the captured SQL metadata using two different perspectives:

- ▶ In the Database tab, information is displayed in a hierarchical structure, identifying the database objects that were accessed or manipulated by the application, along with trace-back information to the source code where in the application the SQL was processed. This feature is useful if the application uses a persistence framework, such as JPA or Hibernate that generates SQL on the fly, hiding the exact SQL that is processed from the developer.
- ▶ In the Java tab, information is shown in a hierarchical structure as well, but from a source code perspective, allowing developers to identify which application modules process which SQL.

Not every captured SQL has to be mapped to a package and will therefore be bound. DDL statements, for example, are by default not mapped to a package.

Data Studio Developer provides a capture file editor that allows for the customization of individual SQL statements. It can be used to define the following situations:

- ▶ Captured SQL is to be removed from the file
- ▶ Captured SQL is not mapped to a package, will be ignored during the bind process and can therefore only be executed dynamically (if dynamic execution is allowed)
- ▶ Captured SQL is not mapped to a package, will be ignored during the bind process and a user designated replacement SQL is executed instead during dynamic SQL execution (that is, to improve performance)
- ▶ Captured SQL is mapped to a package and will be bound as is and can be processed statically
- ▶ Captured SQL is mapped to a package, but a user-designated replacement SQL is bound in the package instead (that is, to improve performance).

Section 5.2, “Improving database access performance for Java applications” on page 155 illustrates how to use the editor to remove SQL from the capture file and how to designate replacement SQL to improve performance.

Various articles and tutorials on developerWorks® discuss the features and capabilities that the pureQuery Outline view provides to support common problem determination and what-if analysis tasks. Some information can be also found in *Examining relationships between SQL statements and Java code* in the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.ui.doc/topics/cpdqre1sqljvatop.html>

Another good resource is the popular “What’s new and cool in ... “ series on developerWorks at the following Web page:

<http://www.ibm.com/developerworks/data/products/devstudio/>

After you have defined how the captured SQL maps to a database package and the package characteristics, you can configure the behavior of the pureQuery StaticBinder utility.

Configuring binder and bind options using DSD

The StaticBinder utility takes as input a configured SQL metadata file, binder options and bind options. Binder options define the behavior of the utility, that is, the artifact type that is created (DBRM files, packages) whereas bind options define certain package characteristics (default qualifier, and so forth).

Both types of options can be defined in Data Studio Developer using a configuration file named *Default.bindProps*. This file is automatically created in the pureQueryFolder of each project that is pureQuery enabled.

To customize the StaticBinder utility options for the SQL metadata file, complete the following steps:

1. In the Package Explorer expand project DSREDBOOK, folder pureQueryFolder and double-click the **Default.bindProps** configuration file to open it in an editor. Each entry in this file uses the syntax shown in the following example:

```
<pureQuery artifact name> = <bind settings for this artifact>
```

A pureQuery artifact name is, for example, the name of an existing SQL metadata file. The associated configuration settings define the binder or bind options. A special kind of entry named defaultOptions can be specified, defining default bind and binder options that apply to all artifacts. Configuration options that are specified for a particular artifact name override the default settings.

Note: If no binder or bind options are defined, internal defaults are used. The binder utility binds the captured and configured SQL in a package at four isolation levels and does not use any bind options.

To override the default behavior, select the desired option. For example, to create DRBM files instead of packages, specify the -generatedDBRM true option.

2. In the Default.bindProps file editor, right-click anywhere. From the context menu, select **pureQuery** → **Add or Remove Entries**. A dialog box displays, listing the SQL metadata files (capture.pdqxml in our example) for which no entry exists yet in this configuration file.
3. In the “Add or Remove Entries” dialog box, select the **capture.pdqxml** entry from the Possible Entries column and click **ADD**. Click **OK** to create a configuration entry for this file.
4. Use content assist to get a list of available options. Press <Ctrl> + <Space> to open the content assist menu.
5. Select **-bindOptions** from the content assist menu to customize the bind options that will be applied during the bind activity.
6. The bind option entry should now look like the following example:
`C:\dsredbook\DSREDBOOK\pureQueryFolder\capture.pdqxml = -bindOptions`
7. Append a space to the end of the line and enter an opening double quote (").

8. Press <Ctrl> + <Space> to invoke content assist again.
9. Let us assume that the example application processes SQL that contains unqualified database object names. For the bind operation (and static SQL execution) to succeed, we have to define a default qualifier that uniquely identifies those objects in the database. Scroll down to the QUALIFIER bind option entry and select it.
10. Type a space and a default schema name followed by a closing double quote. The final line in your file should look like the following example:

```
C:\dsredbook\DSREDBOOK\pureQueryFolder\capture.pdqxml = -bindOptions  
"QUALIFIER GOSALESCT"
```

Note: Multiple bind options are specified by separating entries with a space. For example:

```
-bindOptions "QUALIFIER GOSALESCT EXPLAIN YES"
```

11. Save the changes to the binder configuration file by pressing <Ctrl> + <s>.

The bind options and binder options that are defined in the configuration file are not saved in the SQL metadata file because they are not needed at run time.

2.4.5 Binding the captured and configured SQL

The StaticBinder utility creates one or more database packages (or DBRM files as configured by the binder options) based on mapping information that was added to the SQL metadata file during the configuration phase. The binder and bind options, which we have defined in Default.bindProps, configure its behavior. Figure 2-27 depicts the bind process.

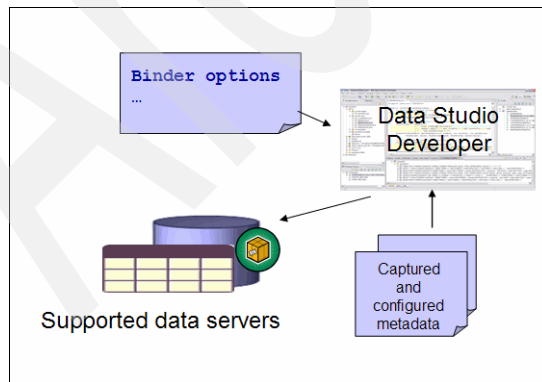


Figure 2-27 PureQuery client optimization bind process overview

You can use the tooling in Data Studio Developer to bind interactively the necessary packages (or create the DBRM files) or the command line tooling to automate the process.

Note: The bind operation does not modify the SQL metadata file.

Detailed information about the StaticBinder utility can be found in *StaticBinder utility, as used for client optimization* in the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.utils.doc/topics/rpdqprfsynstcbnd.html>

Binding SQL metadata using DSD

Data Studio Developer provides a bind wizard that can be used to bind one or more SQL metadata files.

Note: In Data Studio Developer version 2.1 a bind operation always binds all packages that are defined within a single SQL metadata file. In version 2.2 individual packages that are defined within a single SQL metadata can be bound as well, avoiding unnecessary or lengthy bind operations.

To bind a single SQL metadata file, perform the following steps:

1. Expand the DSREDBOOK project and locate the SQL metadata file `capture.pdqxml` in the `pureQueryFolder` folder. Right-click the **capture.pdqxml** file and select **pureQuery** → **Bind** from the context menu. The bind wizard is displayed, prompting for a database connection name, which identifies the database in which the artifacts are to be created.
2. Select an existing connection to the target database, DB9B in our example, where the packages will be created.
3. Click **Finish** to bind the captured and configured SQL to a database package in the database. The binder utility outputs status messages in the Console view, indicating whether any problems occurred.

Note: By default, only the user who performed the bind operation holds the EXECUTE privilege for the package. To grant other users the same privilege, specify binder option `-grant` in the bind configuration file `Default.genProps`.

4. To confirm that you have successfully created the packages, select the DSREDBOOK project in the Package Explorer.

5. Right-click the project and select **pureQuery** → **Show pureQuery Outline** from the context menu.
6. In the pureQuery Outline view, select the SQL tab to display the packages that are mapped to the SQL metadata file. If nothing is shown, click **Refresh**.
7. In the SQL tab double-click the package name (or right-click and select **Show in Data Source Explorer** from the context menu). In our example the package is DSRED2.

The Data Source Explorer expands the Packages node in the DB9B database and highlights the DSRED2 packages. The number of packages shown depends on the isolation level that you have optionally specified when you configured the binder options. By default four packages should be present.

Now that you have configured and bound the captured SQL statements, you have to transfer the modified SQL capture file to the application server node where the pureQuery Runtime can access it.

Note: If you forget to transfer the configured SQL metadata file to a location on the application node where the pureQuery Runtime can access it, an error will be raised indicating that this file has not been configured and bound yet.

Exporting the configured SQL metadata from DSD

To export the updated SQL metadata file, perform the following steps:

1. Select **File** → **Export** from the tool bar to launch the export wizard.
2. Expand the General folder and select **File System** to export the captured SQL metadata file. Click **Next**.
3. Clear the “DSREDBOOK” project check box.
4. Expand the DSREDBOOK and the pureQueryFolder nodes.
5. Select **pureQueryFolder** and the SQL metadata file that you would like to export from you workspace. (capture.pdqxml in our example)
6. Enter a destination directory in the **To directory** text box.
7. Click **Finish** to export the selected file into the specified directory.
8. Close Data Studio Developer.
9. Transfer the exported SQL metadata file in binary mode to the application server node.

Note: Transfer the file in binary mode from one system to another to prevent data corruption.

Now that the SQL metadata file has been configured, the associated packages have been created, and the updated SQL metadata file been transferred to the application node, the pureQuery Runtime can be configured to process SQL statically.

2.4.6 Configuring the Runtime to execute SQL statically

The first three phases (capture, configure, and bind) of the pureQuery client optimization process provide the prerequisite artifacts that the pureQuery Runtime requires to process SQL statically: the configured SQL metadata file and one or more database packages. After these artifacts are in place, the Runtime behavior can be configured as desired.

Figure 2-28 depicts the SQL execution process at a high level if an application is enabled for pureQuery.

- ▶ A Java application submits SQL for processing using a JDBC database connection.
- ▶ The pureQuery Runtime determines how to process this SQL (execute dynamically, execute statically or raise an error) based on its configuration and the content of the SQL metadata file.
- ▶ The SQL is processed dynamically or statically (unless the Runtime has determined that it cannot process this SQL) and the result is returned to the application.

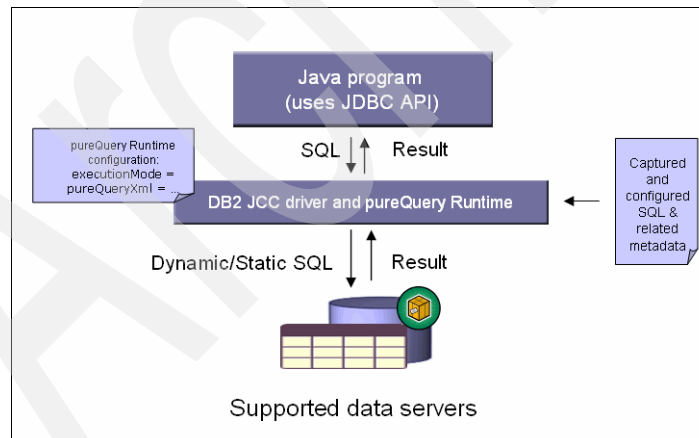


Figure 2-28 PureQuery client optimization execution phase overview

The execution mode is configured using two pureQuery properties:

- ▶ executionMode
- ▶ allowDynamicSQL

Table 2-2 describes the effective execution mode, if none, one, or both of the properties have been defined.

Table 2-2 Effective pureQuery execution modes

pureQuery property		executionMode	
	value	DYNAMIC (or not set)	STATIC
allowDynamicSQL	TRUE or not set	All SQL is executed dynamically	<ul style="list-style-type: none">▶ If SQL package mapping information can be located, SQL is executed statically▶ SQL is executed dynamically otherwise
	FALSE	All SQL is executed dynamically	<ul style="list-style-type: none">▶ If SQL package mapping information can be located, SQL is executed statically▶ Error is raised otherwise

If SQL is to be processed statically, the pureQueryXml property must be defined, identifying the name and location of a previously configured SQL metadata file.

Additional Runtime configuration settings can be defined to enable special SQL processing, such as the previously mentioned ability to replace literals with parameter markers, which we explain in 5.2.1, “SQL literal substitution” on page 155.

For a complete list of pureQuery SQL execution scenarios, refer to *Required settings for running non-pureQuery API applications in STATIC mode* in the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.static.doc/topics/rpdqprfsynpdqrunstc.html>

Enabling static SQL execution for the Great Outdoor Company application in WebSphere Application Server

To enable static SQL processing for the example Web application, we have to edit the data source configuration and modify the pureQuery Runtime properties that we have used to capture the applications SQL.

1. Launch the WebSphere Application Server Administration Console and log in as Administrator.
2. Open the data source definition for the Web application, for example, by navigating to **Resources** → **JDBC** → **Data Source** → **GoSalesApp**.
3. Click **Custom Properties**.
4. Edit the existing pdqProperties custom property that defines the pureQuery Runtime configuration.
5. Assign the following value to this property:

```
executionMode(STATIC),pureQueryXml (/path/to/capture.pdqxml)
```

This configuration instructs the Runtime to execute previously captured and configured SQL statically. The pureQueryXml identifies the SQL metadata file that you have transferred from the client node to the application node in the previous phase of the client optimization process. SQL that was not captured is executed dynamically.

6. Save the data source configuration and restart the application server.
7. Run the client application that drives the Web application workload. The SQL is executed statically if it was previously captured and configured. If the Runtime is unable to locate a package mapping for a given SQL statement, it executes SQL dynamically.

This concludes the basic introduction to the pureQuery client optimization process that enables existing Java applications to benefit from static SQL.

2.4.7 Enabling client optimization using command line utilities

Previously, we have described how one would go about enabling an existing Java application for pureQuery using Data Studio Developer. While there are major advantages to using the GUI to browse and configure the captured SQL metadata, there are certain scenarios where the command line utilities might be a viable alternative for the configuration and bind steps.

Assume, for example, that your organization already has an automated process in place that deploys applications to multiple machines. To support static SQL execution on these machines, you must create the supporting database packages in the databases that are being accessed. The command line utilities

that are included in the pureQuery Runtime installation can be integrated into your existing automated processes, streamlining artifact creation and deployment.

Configure utility

The configure utility is a Java program that takes as input the SQL metadata file that is to be configured as well as the configuration options that are to be applied. The simple example batch file shown in Example 2-14 can be used to configure individual SQL metadata files. It expects the name of the SQL metadata file that is to be configured (parameter #1), along with a package name (parameter #2) and a collection ID (parameter #3) as input parameters.

Example 2-14 Simple batch script that invokes the pureQuery configure utility

```
@echo off
@setlocal
REM ADD pureQuery Runtime libraries to classpath
set CLASSPATH=.\lib\pdq.jar;.\lib\pdqmgmt.jar;%CLASSPATH%
REM ADD JDBC libraries to classpath
set CLASSPATH=.\lib\db2jcc.jar;.\lib\db2jcc_license_cisuz.jar;%CLASSPATH%

REM run java com.ibm.pdq.tools.Configure -help to display command line options
REM
REM invoke configure utility using the following parameters
REM parameter 1: SQL metadata file
REM parameter 2: base package name that will be assigned to this file
REM parameter 3: collection id
java com.ibm.pdq.tools.Configure -pureQueryXml %1 -rootPkgName %2 -collection %3
```

Using a similar approach you can also create UNIX shell scripts or ANT scripts to automate the SQL metadata file configuration.

Note: As an alternative, you can also specify an options file as a parameter that contains the desired configuration options. To do so, modify the last line to read as follows:

```
java com.ibm.pdq.tools.Configure -optionsFile %1
```

Doing so provides more flexibility, because the script does not need to change if you want to specify additional configuration options. Using the default.genProps editor in Data Studio Developer you can easily assemble an options file and customize it according to your needs.

In the following section, when we discuss the binder utility, we take advantage of this approach.

To process a capture file, simply invoke the batch script and pass the desired parameters, as shown in Example 2-15.

Example 2-15 Example configure batch script invocation

```
.\run_configure.bat "pureQueryData\salesanalysis.pdxml" DSRED2 NULLID
```

Note: The command line configure utility does not provide the ability to customize the individual SQL package mapping or the content of the SQL metadata files (removal of SQL, designation of replacement SQL, and so forth). You must use Data Studio Developer to complete these advanced tasks.

Bind utility

Automating the bind process for an SQL metadata file is simple. The binder utility is a Java program that is configured through command line options or an options file.

You can use the simple example batch file in Example 2-16 to bind one or more SQL metadata files. It expects as input parameter the name of a bind configuration file that identifies the SQL metadata files that are to be bound, along with applicable binder and bind options.

Example 2-16 Simple batch script that invokes the pureQuery StaticBinder utility

```
@echo off
@setlocal

REM ADD pureQuery Runtime libraries to classpath
set CLASSPATH=.\lib\pdq.jar;.\lib\pdqmgmt.jar;%CLASSPATH%
REM ADD JDBC libraries to classpath
set CLASSPATH=.\lib\db2jcc.jar;.\lib\db2jcc_license_cisuz.jar;%CLASSPATH%

REM run java com.ibm.pdq.tools.StaticBinder -help to display command line options

REM invoke bind utility using the following parameter
REM parameter 1: bind configuration file

java com.ibm.pdq.tools.StaticBinder -optionsFile %1
```

Note: The command line bind utility does not support binding of individual packages that are defined in a single SQL metadata file. The utility always makes an attempt to bind all packages that are defined in a metadata file.

To bind one or more configured SQL metadata files, invoke the script and pass an options file name as a parameter that configures the binder and bind options for these files. See Example 2-17.

Example 2-17 Example bind batch script invocation

```
.\run_bind.bat "pureQueryFiles\bind_salesanalysis.properties"
```

Option files, which provides the content assist capabilities, can be created using the Default.bindProps editor in Data Studio Developer. Example 2-18 lists the content of an options file, that defines the binder and bind options for a single SQL metadata file.

Example 2-18 Example options file

```
defaultOptions=-isolationLevel CS -url  
jdbc:db2://lost.in.some.domain:50000/DB9B -username db2admin -password  
fun2poke  
pureQueryData\salesanalysis.pdqxml = -bindOptions "EXPLAIN YES"
```

Note: In our example, we embed the user credentials as well as the database URL in the options file. If this raises a security concern, you can also provide this information as a command line parameter, by altering the example script to expect two additional parameters that are passed through to the StaticBinder utility.

```
.\run_bind.bat "pureQueryFiles\bind_salesanalysis.properties"  
db2admin hard2guess
```

The sample scripts that were shown are meant to be used as a starting point. Error checking and further customization is necessary to use them in an enterprise environment.

2.4.8 Application maintenance

Over time, applications, database objects, or software prerequisites change, raising the need to maintain the SQL metadata files as well as the associated database packages. In 3.7, “pureQuery application maintenance” on page 111 we discuss various aspects for applications that take advantage of client optimization and pureQuery applications, which are implemented using the pureQuery API.

2.5 Evaluation

In this chapter, we exposed common challenges that DBAs face when it comes to Java applications that dynamically process workloads, and identified a solution for existing Java applications that provides the following advantages:

- ▶ Improves performance by eliminating the overhead involved with dynamic SQL processing
- ▶ Allows for custom WLM configuration to prioritize individual Java workloads according to the needs of the business
- ▶ Allows for the correlation of SQL with its originating application, application and source code module to streamline common problem determination and application maintenance tasks
- ▶ Tightens security by preventing SQL injection and limiting the number of SQL that application users might execute

Chapter 3, “Managing pureQuery application deployment” on page 89 provides insight into some of the deployment and application maintenance aspects. Even though the scenario in the next chapter focuses on applications that use the native pureQuery API to exploit the benefits of static SQL, the aspects discussed are just as relevant to existing Java applications that are enabled for pureQuery using the client optimization technology.

At the time of writing, a new version of the pureQuery technology and Data Studio Developer was released. In Chapter 5, “Optim Development Studio and Optim pureQuery Runtime: A closer look at version 2.2” on page 153, we discuss some of the key features that have been introduced.

For the up-to-date in-depth information, articles, tutorials about the pureQuery client optimization technology as well as trial downloads go to the following Web page:

<http://www.ibm.com/developerworks/data/products/optim>

Managing pureQuery application deployment

The pureQuery API and the tooling in Data Studio Developer provide rich and powerful functionality to facilitate efficient development of Java applications as well as effective use of system resources upon execution and deployment.

In Chapter 2, “Optimizing existing Java applications” on page 19 we reviewed the main benefits that the pureQuery client optimization technology can provide to existing Java data access applications. In this chapter, we focus on new Java applications that are implemented using the pureQuery API, which takes advantage of these same benefits (and more, such as heterogeneous batching) without the need to complete the client optimization process. We focus our discussion on the deployment aspect, with consideration given to the usual enterprise application deployment concerns: application change, software dependency, and database object change. Throughout this chapter the term *pureQuery application* refers to a Java application in which the data access layer was implemented using the pureQuery API.

3.1 Scenario description

To meet new business requirements, The Great Outdoor Company has implemented a new application that uses the pureQuery API as its underlying data access layer. They have completed the development phase and are ready to move the application to a test system where they plan to run the application in static SQL execution mode. The new application will run on a WebSphere application server with the SQL packages statically bound in a z/OS DB2 database.

First we discuss the requirements for a successful deployment of pureQuery applications from a DBA perspective.

3.2 Deployment with pureQuery

Changing business requirements and new business opportunities often lead to requirements in IT to rewrite or create new applications and then to deploy those applications. In this section we show the deployment steps required when working with pureQuery applications for both dynamic and static execution.

3.2.1 Common deployment process

Before jumping into the specifics of deployment of pureQuery and what it entails, we first examine common application deployment processes from a high level.

Usually when deploying an application, several things are taken into consideration, including the following components:

- ▶ The application itself
- ▶ The environment on which it will be executing
- ▶ The operating system (OS) level
- ▶ The hardware resources
- ▶ The application server configurations

This level of discussion is beyond the scope of this publication, so we focus on the details that are important to a System DBA.

As a point of clarification, many organizations have application DBAs and DBA modelers in the same reporting chain-of-command as the application programming groups. These application DBAs in larger organizations usually do

not have the task of configuring, tuning, and maintaining the execution of the production environment. For this task, the role of environmental issues and considerations falls to the System DBA.

It is the System DBA who is responsible for the subsystems or instances and for the various communication connectivity issues such as drivers and end-to-end application connectivity. In some organizations, these roles can and will overlap. While we are cognizant of this political reality, for the sake of this chapter we will refer to both an application DBA and a System DBA simply as “DBA”.

When a new application is deployed, there are several points of concern from a DBA's perspective, among which, are the following issues:

- ▶ Ensuring correct drivers are installed so that the application will be able to communicate efficiently with the database
- ▶ Ensuring that the required data structures exist
- ▶ The ability to reasonably gauge the workload from the database management system (DBMS) perspective and being confident that the new application workload will not impact the service level agreements (SLA) of other data-centric applications reliant on the same DBMS instance, subsystem, or server
- ▶ Maintaining data security

3.2.2 pureQuery dynamic SQL application deployment

There are no significant differences to deploying a dynamic pureQuery application from deploying any other Java application. From a DBA's perspective, the concerns are exactly the same. So the question arises, what if anything needs to be done to deploy a pureQuery application? The answer is simple. For successful deployment of a pureQuery application, you must first install and configure the Data Studio (now Optim) pureQuery Runtime product.

As mentioned in Chapter 1, the pureQuery API along with the development environment provided by Data Studio Developer (Optim Development Studio) make it much easier to develop more manageable, high-performance Java data access applications. The pureQuery Runtime functions as the underlying foundation to enable the advantages to be gained by making use of the pureQuery API. The pureQuery Runtime is the key component in achieving improved performance, manageability, security, and troubleshooting through optimization of SQL without modification of the original application. System requirements are covered later in this chapter.

3.2.3 pureQuery static SQL application deployment

To take full advantage of everything that pureQuery has to offer when the target DBMS is DB2, the deployment of a pureQuery application for static execution is ideal. Again from a DBA perspective, concerns still do not change. However, given the functionality that this type of deployment enables, there is a bit more involvement in the deployment realm for the DBA.

As part of the deployment process there are at least three tasks that have to be completed to enable static SQL execution for a pureQuery application:

- ▶ Create one or more database packages that provide static execution support for the application's SQL
- ▶ Grant execution privileges for those packages
- ▶ Modify the pureQuery Runtime configuration to execute SQL statically

In the following sections we review considerations and prerequisites, discuss in detail parts of the deployment process and take a closer look at the maintenance aspect of pureQuery applications with respect to its impact on the database.

3.3 Deployment planning

Consider the following questions before deploying a pureQuery application:

- ▶ Which artifacts need to be deployed?
- ▶ Which additional software needs to be installed and configured?
- ▶ Which prerequisites does the target system have to meet where the application is to be deployed?
- ▶ Which tasks need to be performed post deployment?

In this section, we briefly review these planning activities. As always, additional steps might be required depending on the exact deployment scenario.

3.3.1 Identifying deployment artifacts

To repeat, a pureQuery application is not much different from any other Java application. If we compare the artifacts of a regular Java application that only processes SQL dynamically with the artifacts of a pureQuery application (which processes SQL dynamically or statically), we would find that both consist of (compiled) source code, configuration files, or other application-specific supporting artifacts.

There are essentially just two additional deployment artifacts that have to be taken into account for a pureQuery application deployment:

- ▶ pureQuery Runtime

The pureQuery Runtime is comprised of a set of Java libraries that provide the pureQuery Runtime environment, along with its software prerequisites. Typically the pureQuery Runtime is installed only once per system. However, it is also possible to package it as part of the application, eliminating the need for a separate installation.

- ▶ Database packages

The database packages support static SQL execution for the SQL that this application processes.

Before delving into the specifics, we want to clarify a few things about the pureQuery API and its ability to process SQL statically. The pureQuery platform provides two API types:

- ▶ Method-style
- ▶ Inline style

For our discussion it is not relevant what these APIs look like or how a developer would use them. What is important to know is that the method-style API supports static SQL processing out of the box, whereas data access that is implemented using the inline style API does not yet support static SQL processing directly.

What does that mean? If the method-style API was used to implement the data access layer, Data Studio Developer can extract from the source code the information (for example, the SQL) it needs to create database packages that will enable the Runtime to process this SQL statically. If, however, the inline style API was used to implement the data access layer, the required information cannot be extracted from the source code. Thus, if inline style is used, it is necessary to complete the client optimization process described in Chapter 2, “Optimizing existing Java applications” on page 19 to capture the information in an external file, which can subsequently be used to create the database packages.

Note: You might be wondering why one would want to use the inline style API if it does not natively support static SQL execution. There are certain use cases in which it can be desirable to use it, such as when queries are dynamically generated. You can find a good summary in *Write high performance Java data access applications, Part 3: Data Studio pureQuery API best practices*, available from the following Web page:

<http://www.ibm.com/developerworks/data/library/techarticle/dm-0808rodrigues/index.html>

As you can see, as part of the deployment planning process it is important to find out which API they are using to determine whether the application needs to go through the client optimization process to enable static SQL execution.

3.3.2 Identifying software prerequisites

At run time, a pureQuery application requires three additional pieces of software:

- ▶ A supported Java run time environment (Version 1.5 at the time of writing)
- ▶ An installation of the pureQuery Runtime product
- ▶ A compatible IBM Data Server Driver for JDBC and SQLJ (“JDBC driver”)

During the deployment process, access to an installation of Data Studio Developer (for GUI- driven database package creation) or the pureQuery Runtime (for command line-driven package creation) is required.

3.3.3 Identifying target system requirements

The pureQuery Runtime product is available for distributed platforms (pureQuery Runtime for Linux, UNIX, and Windows) and z/OS (pureQuery Runtime for z/OS). You can find a comprehensive, version specific up-to-date list of supported operating systems, database servers, and prerequisites at the following Web page:

<http://www.ibm.com/support/docview.wss?rs=3369&uid=swg27014394>

The pureQuery Runtime uses the IBM Data Server Driver for JDBC and SQLJ to perform its database operations, thus requiring a compatible version to be pre-installed. (A list of compatible JDBC drivers can be found at the Web site above.) Issues can occur if an incompatible set of JDBC drivers is used:

- ▶ If the JDBC driver is too old (for example, if you are using one that was shipped with DB2 Version 8), it will not recognize the pureQuery Runtime configuration settings, and pureQuery cannot be used.
- ▶ If the JDBC driver level does not meet the required minimum, pureQuery Runtime might not be adequately supported. This might lead to problems at execution time.
- ▶ If a pureQuery application was developed using a particular version of the pureQuery API, it cannot be executed on a system where an older version of the pureQuery Runtime is installed. This is because it might exploit features that are not supported by the older installed version of the pureQuery Runtime on the production system.

Note: The pureQuery Runtime installation does not include the required JDBC drivers. You must download them from the DB2 support Web site if the installed version is not current:

http://www.ibm.com/software/data/db2/support/db2_9/download.html

3.3.4 Identifying post deployment tasks

If SQL is to be executed statically, additional thought has to be given to package management:

- ▶ Which additional users or groups require special privilege on those packages to allow for common maintenance tasks?
- ▶ Can existing database object privileges be revoked from users or groups that access database objects only through this application to increase security?
- ▶ Should a service class other than the default class be assigned to the packages, allowing for custom workload management prioritization?

Because all of these considerations are generally applicable to applications that use database packages, we will not delve into specifics in this book.

3.4 Installing the pureQuery Runtime

In a three-tier application architecture, as shown in Figure 3-1, the pureQuery Runtime is installed on the application node to make it accessible to pureQuery applications. No additional software is required to be installed on the database node.

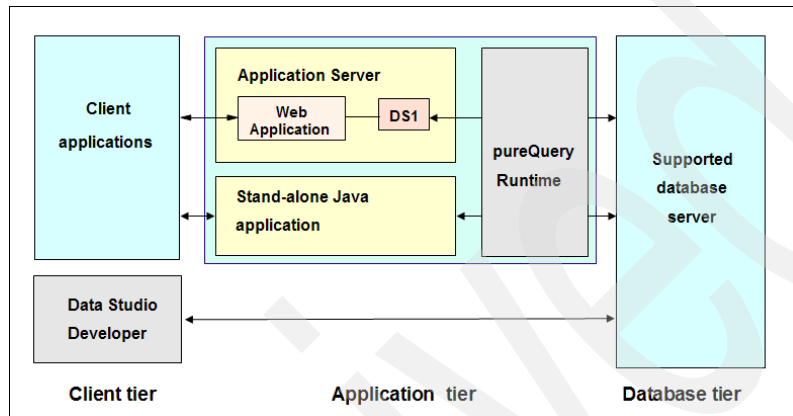


Figure 3-1 Example pureQuery 3-tier deployment topology

The simple installation instructions are well documented in the *Installing, upgrading, and configuring* article of the Integrated Data Management Information Center at the following Web page:

http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.dstudio.nav.doc/topics/ic_install.html

Note: Because version 2.2 of the product will be available by the time this book is published, all Information Center references in this chapter refer to the latest release.

The pureQuery Runtime installation program does not alter the application environment. The program copies the Java archives `pdq.jar` and `pdqmgmt.jar` to a user-designated location on the machine. You must manually change the `CLASSPATH` to pick up these libraries. Remember that the compatible JDBC drivers must also be made accessible to the application's class loader to complete the setup.

In Chapter 2, "Optimizing existing Java applications" on page 19, when we discuss how to enable an existing application for pureQuery, we provide one example how a WebSphere Application Server can be configured such that its

applications can make use of the pureQuery libraries. Depending on the type of application that you are deploying and your topology, other approaches might be viable alternatives.

Note: Because you can install different versions of the pureQuery Runtime, we recommend that you install each version in separate directory. Different versions of the Runtime can coexist on a single machine. However, an application should use only a single version.

3.5 Package deployment using pureQuery bind utility

The general process used to create the database packages for a pureQuery application is similar to the process that described in Chapter 2, “Optimizing existing Java applications” on page 19, when we discussed how to enable static SQL execution for an existing Java application:

1. Define the mapping between SQL statements and one or more database packages (which includes a description of the package characteristics).
2. Configure the binder options.
3. Create the packages according to the mapping definitions and bind configurations.

In the case of a pureQuery application, the mapping information between the SQL statements and database packages is already embedded in the compiled Java data access code, eliminating the need to define the mapping at the time of deployment.

Note: The approach to embed the mapping information into the source code is different from the approach that the pureQuery client optimization process takes. The mapping information is stored in the external SQL metadata file to provide a non-invasive solution.

The bind process can be performed interactively using Data Studio Developer or in batch mode using executable utility classes, which are packaged as part of the pureQuery Runtime installation.

In the following sections we focus on the bind process in general and provide examples that illustrate the batch processing approach because the interactive approach using Data Studio Developer is similar to the one that was already covered in 2.4, “Optimizing an existing Java application using pureQuery” on page 51 when the captured SQL metadata file was bound.

3.5.1 Process using pureQuery StaticBinder

With the StaticBinder utility, you can create and bind DB2 packages based on SQL metadata files that were created and configured as part of the pureQuery client optimization process or based on a pureQuery application that was implemented using the method-style data access API. You can also choose to create DBRM files that you can later transfer to a z/OS data set and use to create DB2 packages. Figure 3-2 illustrates the inputs and outputs, establishing the link between an application and the database.

Note: The StaticBinder utility does not alter the source that contains the application's SQL, which is either pureQuery Java code or an SQL metadata file.

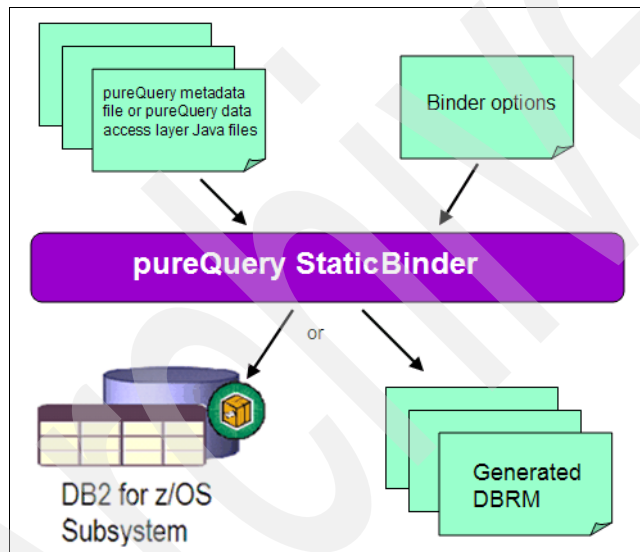


Figure 3-2 pureQuery StaticBinder utility

To clarify the terminology used for the pureQuery StaticBinder utility, the binder options are configuration settings that define the behavior of the pureQuery StaticBinder utility as well as bind options (in the traditional sense) that will be applied. Some of these bind configuration settings are applicable only to one of the two potential outputs (package or DBRM), some apply to both, and some are mutually exclusive. If a DBRM file is created, it will not be bound.

Whether a package binds directly into DB2 for z/OS or is generated as a DBRM is controlled by binder option `-generatedDBRM`. By default, or if the option is set to `FALSE`, the StaticBinder utility creates database packages.

Having the ability to control whether to bind static packages directly into a DB2 for z/OS subsystem or to generate a DBRM is mostly dictated by the change control process installed in your organization. Many organizations do not allow direct changes in the production environment due to auditing and regulatory compliance such as internal regulation, industrial standard generally accepted principles, or governmental regulation mandate.

Conversely, it is more efficient to bind static packages directly into the test environments, because the overhead of additional steps are not required by any regulation. The steps involved with using pureQuery StaticBinder with DBRM generation are as follows:

1. Generate DBRM file.
2. Upload DBRM to z/OS.
3. Bind DBRM using DB2 Interactive (DB2I) or batch job.
4. Grant package execution privileges.

Determining whether to use pureQuery StaticBinder to generate a DBRM or a package is an organizational issue. It usually depends on how your organization handles regulatory compliance and how your organization moves application modules into production.

Authority

A user who invokes the StaticBinder utility must hold one of the following authorities, in addition to the privileges required to compile the SQL:

- ▶ SYSADM authority
- ▶ DBADM authority
- ▶ If the package does not exist, the BINDADD privilege, and one of the following privileges:
 - CREATEIN privilege
 - DB2 for z/OS®: PACKADM authority on the collection or on all collections
 - DB2 Database for Linux®, UNIX®, and Windows®: IMPLICIT_SCHEMA authority on the database if the schema name of the package does not exist
- ▶ If the package exists:
 - DB2 for z/OS: The BIND privilege on the package
 - DB2 Database for Linux, UNIX, and Windows: ALTERIN privilege on the schema and BIND privilege on the package

3.5.2 The scenario usage of pureQuery StaticBinder

To illustrate the use of pureQuery StaticBinder, we describe the environment used in this book. Figure 3-3 shows the environment at The Great Outdoor Company.

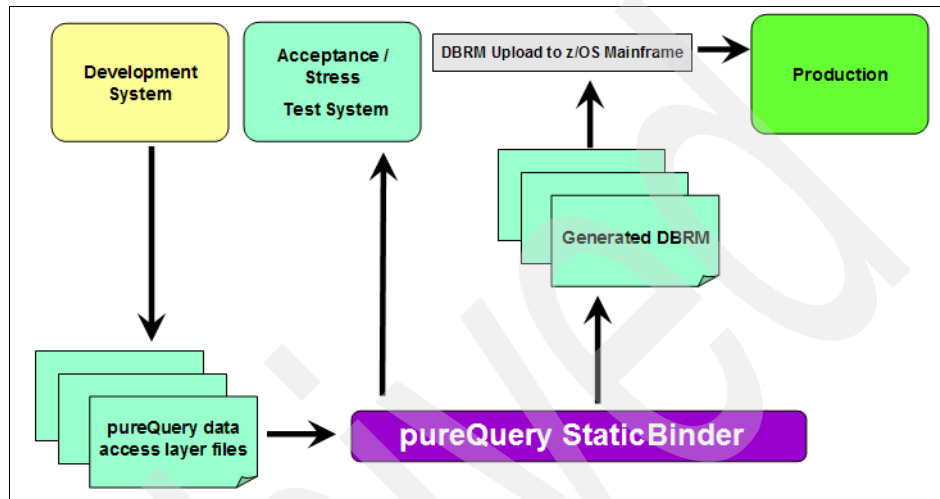


Figure 3-3 The Great Outdoor Company environment

The company has three environments consisting of one production system and two test systems. Further, the two test systems consist of one development system and one acceptance/stress test system. The application staff performs the majority of development work in the development system. After development work is completed (including any unit test work), the application is packaged into an enterprise archive file (.EAR) and turned over to the stress test staff for system testing in the acceptance/stress test system.

This archive file contains, among other things, the Java data access layer that was implemented using the pureQuery method-style API. As part of the pureQuery build (code generation) process, these Java classes have been enriched with information identifying the package name, collection ID, package version, and consistency token, enabling the StaticBinder utility and pureQuery Runtime to correlate source code with its associated packages. The archive file or .EAR file is used as input to the StaticBinder utility along with binder options as shown in Figure 3-2 on page 98.

Note: The StaticBinder utility can be used on any .ear, .jar, .war, or .zip archive that contains the pureQuery data access code for a given application.

To save time, the stress test staff informally builds statically-bound packages using the static binding method of the pureQuery StaticBinder utility. After all of the stress testing is completed and the application is ready for production, the stress test staff creates the necessary documentation in the change control system to move the application into production. The change control system is used to meet auditing compliance for having an audit trail for production changes.

The change control system is automated for all changes performed in production at The Great Outdoor Company. There is also a corporate security mandate that only the change control system can make these production changes. When an application is ready for production, the stress test staff along with the applications staff and the production staff work out the date and timing on when to move the application into production. Prior to that production date, the stress test staff must get ready for the move by performing tasks such as creating all DBRMs required for the change control system. Among the many things the change control system performs, one of the tasks is to bind and grant all of the packages being either added or changed.

For the binding into the acceptance/stress test system and for the DBRM generation process, the stress test staff uses the simple batch script shown in Example 3-1. Having the binding, granting, and DBRM generation process in a single script significantly simplifies the process. The script connects direct to DB2 with the following format:

```
jdbc:db2://<mainframe domain name>:<DB2 for z/OS port>/<location>
```

This information is found at DB2 startup on the <ssid>MSTR address space on z/OS.

Example 3-1 Script to bind/grant direct into DB2 and DBRM generation

```
@echo off
@setlocal

REM JDBC type-4 URL identifying the target database
set TARGET_URL=jdbc:db2://wtsc64.itso.ibm.com:12350/DB9B
set USER_NAME=<mainframe userid here>
set PASSWORD=<mainframe password here>

:CHECK_PARM_ONE
IF "%1" == "" GOTO SYNTAX

:CHECK_PARM_TWO
IF "%2" == "" GOTO SYNTAX

GOTO DO_BIND

:DO_BIND
set APPLICATION_ARCHIVE=%1
set BIND_OPTIONS_FILE=%2
```

```

:Found from pQ Runtime Installation
set CLASSPATH=.\lib\db2jcc.jar;.\lib\db2jcc_license_cisuz.jar;%CLASSPATH%
set CLASSPATH=%CLASSPATH%;.\lib\pdq.jar;.\lib\pdqgmt.jar

REM invoke binder utility
java com.ibm.pdq.tools.StaticBinder -url %TARGET_URL% -username %USER_NAME% -password
%PASSWORD% -archive %APPLICATION_ARCHIVE% -optionsFile %BIND_OPTIONS_FILE%

GOTO END

:SYNTAX

ECHO Command syntax: %0 archive_name bind_options_file
echo where archive_name      identifies the archive file that contains the pureQuery
data access classes
echo      bind_options_file identifies the file that contains the bind and binder
options

:END

```

Acceptance and stress test turnover: Direct binding packages into DB2 for z/OS

As mentioned before, the stress test staff informally builds statically-bound packages using the static binding method of the pureQuery StaticBinder utility. The application team provides all required application artifacts to the stress test staff. This is done to insure that no changes happen to the application during system testing and stress testing. Obviously, if problems are found the application is given back to applications team for review and problem resolution.

To save time, and because the acceptance/stress test system is not formally managed by change control, The Great Outdoor Company performs static binding direct into DB2 for z/OS without creating a DBRM. The stress test staff uses the batch file in Example 3-1 on page 101 for both binding direct into DB2 for z/OS as well as to generate the DBRM when it comes time to build the production objects. Example 3-2 shows an example of a binder options file for binding directly into DB2 for z/OS.

Example 3-2 Bind input for package binds directly into DB2 for z/OS

```

defaultOptions=-bindOptions "QUALIFIER GOSALEST" -isolationLevel CS
-differenceOnly true -grant grantees(db2admin)
com.jkenterprise.dao.JKData
com.jkenterprise.dao.ProductData=-bindOptions "QUALIFIER GOSALES"
com.jkenterprise.dao.CustData
com.jkenterprise.dao.Cust_ordData
com.jkenterprise.dao.Cust_ord_detlData

```

An options file can contain a defaultOptions entry, which defines options that are applied to all Java interfaces that are listed in this file. For each listed Java interface, specific options can be defined that take precedence over default options.

Note: Think of a Java interface as a data access interface. Such an interface defines data access methods that the application's business logic uses to retrieve or modify (insert, update, delete) data that is stored in the database. For example, Interface com.jkenterprise.dao.ProductData listed in Example 3-2 defines methods that can be used to retrieve product information based on product number, product category, and so on.

One unique aspect to these interfaces is that they contain pureQuery annotations that the pureQuery tooling (such as StaticBinder) and the pureQuery Runtime can process, eliminating the need to store the information in an external file.

Let us take a closer look at the example. The application uses five different interfaces to access or manipulate data. Generally speaking, the StaticBinder utility identifies the SQL in each listed interface and determines which options need to be used to generate the requested output (DBRM or package). For the JKData interface, no explicit options are defined, thus the specified default options are used:

- A package will be created containing all SQL that is defined in the interface if this package does not exist yet (-differenceOnly true) at the cursor stability isolation level (-isolationLevelCS).

By default, the pureQuery StaticBinder utility creates four packages or DBRM files, one for each of the four DB2 isolation levels. The pureQuery StaticBinder utility identifies the isolation level by appending the following numbers to the root names of the packages or DBRM files:

- 1: For isolation level Uncommitted Read (UR)
- 2: For isolation level Cursor Stability (CS)
- 3: For isolation level Read Stability (RS)
- 4: For isolation level Repeatable Read (RR)

If you use the -isolationLevel option when you perform a bind, or specify the isolation level in the bind options string, only the package or DBRM file for the isolation level that you specify is created. The name follows the convention that the StaticBinder uses when creating packages or DBRM files for all four isolation levels.

- ▶ All unqualified database object references in the SQL statements will be resolved using the GOSALEST qualifier (-bindOptions "QUALIFIER GOSALEST"). Qualified database object references are not impacted by this setting.
- ▶ After the package has been created successfully, the EXECUTE privilege is granted to the package owner and user db2admin (-grant grantees(db2admin)).

The exact same options will be used for all other interfaces, except ProductData. Because a specific bind option (-bindOptions "QUALIFIER GOSALES") is defined for this interface, unqualified database objects will be qualified using the GOSALES qualifier instead of GOSALEST. All other binder options remain in effect for this interface.

A formal syntax definition for the options file can be found in *Syntax of entries in the Default.bindProps file for configuring support of static SQL in pureQuery code* in the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.static.doc/topics/rpdquistsqlsynbnd.html>

Note: Invoke the StaticBinder utility in Data Studio Developer by right-clicking the project name in the Project Explorer and selecting **pureQuery** → **Bind pureQuery Application** from the context menu.

If Data Studio Developer is used to invoke the StaticBinder utility, binder options are expected to be defined in configuration file Default.bindProps, which is located in the pureQueryFolder of your project. To create a custom binder configuration file, take advantage of the built-in configuration file editor. It provides content assistance (press <ctrl>+<space>) along with help information, describing available options and their usage.

Example 3-3 shows the output of the StaticBinder utility, if the above configuration file is used. Five packages are created at isolation level CS, each containing the SQL of a single interface. The package owner and user db2admin hold the EXECUTE privilege for each package.

Example 3-3 Execution of pureQuery StaticBinder utility direct bind to DB2 for z/OS

```
C:\Projects\Residency_May2009\chapter3_bind_artifacts\chapter3_bind_artifacts>bind SalesAnalysis.ear SalesAnalysis.bindProps
```

```
IBM Data Studio pureQuery Runtime 2.1 build 2.1.206
Licensed Materials - Property of IBM
5724-U16
(c) Copyright IBM Corp. 2006, 2008 All Rights Reserved.
```

```

=====
The StaticBinder utility successfully bound the package 'SAPD2' for the isolation level
CS.
Executed successfully : GRANT EXECUTE ON PACKAGE "NULLID"."SAPD2" TO db2admin

The StaticBinder utility successfully bound 'com.jkenterprise.dao.ProductDataImpl'.

=====
The StaticBinder utility successfully bound the package 'OSSPKG2' for the isolation
level CS.
Executed successfully : GRANT EXECUTE ON PACKAGE "NULLID"."OSSPKG2" TO db2admin

The StaticBinder utility successfully bound 'com.jkenterprise.dao.JKDataImpl'.

=====
The StaticBinder utility successfully bound the package 'SAC02' for the isolation level
CS.
Executed successfully : GRANT EXECUTE ON PACKAGE "NULLID"."SAC02" TO db2admin

The StaticBinder utility successfully bound 'com.jkenterprise.dao.Cust_ordDataImpl'.

=====
The StaticBinder utility successfully bound the package 'SACOD2' for the isolation level
CS.
Executed successfully : GRANT EXECUTE ON PACKAGE "NULLID"."SACOD2" TO db2admin

The StaticBinder utility successfully bound
'com.jkenterprise.dao.Cust_ord_detlDataImpl'.

=====
The StaticBinder utility successfully bound the package 'SACD2' for the isolation level
CS.
Executed successfully : GRANT EXECUTE ON PACKAGE "NULLID"."SACD2" TO db2admin

The StaticBinder utility successfully bound 'com.jkenterprise.dao.CustDataImpl'.

=====
Results of the StaticBinder utility's activity:

    Number of implementation classes and pureQueryXml files for which the bind operation
SUCCEEDED: 5

Bind action executed successfully.

```

An in-depth overview of how to use the StaticBinder utility, the options file, and pureQuery application specific binder options can be found in *StaticBinder utility, as used with annotated methods* of the Integrated Data Management Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.utils.doc/topics/rpdqsqlbndutl.html>

Production turnover: Creating a DBRM

After the stress test staff completes the testing of the application in the acceptance and stress test system, the application is readied for production. Tasks such as determining a turnover date, moving the required modules to the turnover libraries, and creating the change control record are all performed. Included among these tasks is creating the DBRMs for the statically-bound packages. Because the change control process is automated for auditing purposes, the stress test staff creates the DBRMs as required for the change control system.

One of the tasks the change control system performs is to bind and grant all of the packages being added or changed. This is beyond the scope of tasks necessary for The Great Outdoor Company production turnover because it is automated.

The stress test staff must perform the following tasks:

1. Generate the DBRM.
2. Upload the DBRM to the z/OS system into a library referenced by the change control automated process for binding and granting later.

As before, the stress test staff uses the common script in Example 3-1 on page 101 for both binding direct into DB2 for z/OS and to generate the DBRM when it comes time to build the production objects. In this case, they use the DBRM generation process.

Example 3-4 shows an example of binder options for creating a DBRM for a single isolation level. Note the usage of an output path associated with DBRM generation. The StaticBinder utility creates a DBRM file for each Java interface at the cursor stability isolation level and stores them in the location identified by the outputDBRMPath parameter.

Example 3-4 Bind input for creating DBRM

```
defaultOptions=-isolationLevel CS -generateDBRM true -outputDBRMPath
c:\DBRMrepository\SalesAnalysis
com.jkenterprise.dao.JKData
com.jkenterprise.dao.ProductData
```



```
com.jkenterprise.dao.CustData  
com.jkenterprise.dao.Cust_ordData  
com.jkenterprise.dao.Cust_ord_detlData
```

Note: If the target output of the bind activity is a DBRM file, the -bindOptions option must not be specified. Bind options can be specified when the DBRM is bound to a package or collection.

Example 3-5 shows the execution output from running with the binder options for pureQuery StaticBinder utility creating a DBRM.

Example 3-5 Execution of pureQuery StaticBinder utility for DBRM Creation

```
C:\Projects\Residency_May2009\chapter3_bind_artifacts\chapter3_bind_artifacts>bi  
nd SalesAnalysis.ear SalesAnalysis.bindDBRMProps
```

```
IBM Data Studio pureQuery Runtime 2.1 build 2.1.206  
Licensed Materials - Property of IBM  
5724-U16  
(c) Copyright IBM Corp. 2006, 2008 All Rights Reserved.
```

```
=====
DBRM generation started.
The StaticBinder utility generated the DBRM file 'SAPD2' for isolation level CS.
DBRM generation completed successfully for 'com.jkenterprise.dao.ProductData
Impl'.

The StaticBinder utility successfully bound 'com.jkenterprise.dao.ProductDataImpl'.

=====
DBRM generation started.
The StaticBinder utility generated the DBRM file 'OSSPKG2' for isolation level CS.
DBRM generation completed successfully for 'com.jkenterprise.dao.JKDataImpl'.

The StaticBinder utility successfully bound 'com.jkenterprise.dao.JKDataImpl'.

=====
DBRM generation started.
The StaticBinder utility generated the DBRM file 'SAC02' for isolation level CS.
DBRM generation completed successfully for 'com.jkenterprise.dao.Cust_ordDataImpl'.

The StaticBinder utility successfully bound 'com.jkenterprise.dao.Cust_ordDataImpl'.

=====
DBRM generation started.
The StaticBinder utility generated the DBRM file 'SAC0D2' for isolation level CS.
DBRM generation completed successfully for 'com.jkenterprise.dao.Cust_ord_detlDataImpl'.

The StaticBinder utility successfully bound
'com.jkenterprise.dao.Cust_ord_detlDataImpl'.
```

```

=====
DBRM generation started.
The StaticBinder utility generated the DBRM file 'SACD2' for isolation level CS.
DBRM generation completed successfully for 'com.jkenterprise.dao.CustDataImpl'.

The StaticBinder utility successfully bound 'com.jkenterprise.dao.CustDataImpl'.

=====
Results of the StaticBinder utility's activity:

    Number of implementation classes and pureQueryXml files for which the bind operation
SUCCEEDED: 5

Bind action executed successfully.

```

Figure 3-4 shows the DBRM file upload to the z/OS system. Note the name generated ends with a “2”, indicating the isolation level was Cursor Stability.

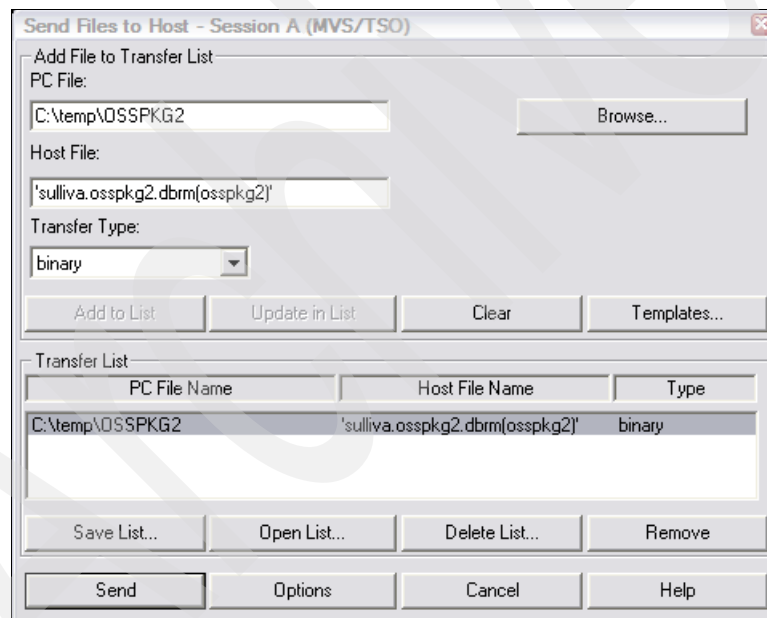


Figure 3-4 Upload of DBRM to z/OS system

After uploading, the DBRM can then be bound manually, as shown in Figure 3-5 on page 109, or processed using a change control system.

```

                                BIND PACKAGE                                SSID: DB9B
COMMAND ==>

Specify output location and collection names:
1  LOCATION NAME ..... ==>                                     (Defaults to local)
2  COLLECTION-ID ..... ==> SULLIVA                               (Required)
Specify package source (DBRM or COPY):
3  DBRM:          COPY:      ==> DBRM                           (Specify DBRM or COPY)
4  MEMBER      or  COLLECTION-ID ==> OSSPKG2
5  PASSWORD    or  PACKAGE-ID .. ==>
6  LIBRARY     or  VERSION ..... ==> 'SULLIVA.OSSPKG2.DBRM'
                                           (Blank, or COPY version-id)
7  ..... -- OPTIONS ..... ==>                                (COMPOSITE or COMMAND)
Enter options as desired:
8  CHANGE CURRENT DEFAULTS? .. ==> NO                           (NO or YES)
9  ENABLE/DISABLE CONNECTIONS? ==> NO                           (NO or YES)
10 OWNER OF PACKAGE (AUTHID) .. ==> SULLIVA                     (Leave blank for primary ID)
11 QUALIFIER ..... ==>                                         (Leave blank for OWNER)
12 ACTION ON PACKAGE ..... ==> ADD                             (ADD or REPLACE)
13 INCLUDE PATH?..... ==> NO                                   (NO or YES)
14 REPLACE VERSION ..... ==>
                                           (Replacement version-id)
PRESS:  ENTER to process      END to save and exit      HELP for more information

```

Figure 3-5 Manual bind of uploaded DBRM to z/OS system

Note: The StaticBinder utility cannot be used to bind an existing DBRM file to a package or collection.

As a final step, the production DBA must make sure adequate security is established for the statically-bound package. While this is an obvious step in most organizations, it is a highly critical step that must not be overlooked.

Whether the package is granted access to secondary authority IDs or to individuals is based on use, criticality, and sensitivity of the data. You should protect your data and adopt the philosophy that minimal access is the best policy.

3.6 Configuring the pureQuery Runtime to process SQL statically

After a pureQuery application has been deployed onto its target system and the necessary database packages have been created, the pureQuery Runtime environment can be configured to process SQL statically.

Do this by setting the executionMode property to STATIC.

Note: Unlike for existing Java applications that use are enabled for static SQL execution using client optimization there is no need to set the pureQueryXml property for pureQuery applications. The necessary package mapping information is already embedded in the pureQuery data access code.

Conceptually, the approach that you take to configure the pureQuery Runtime is identical to the approach that discussed in Chapter 2, “Optimizing existing Java applications” on page 19:

- ▶ Determine the scope at which the pureQuery Runtime is to execute SQL statically:
 - For a single application
 - For multiple applications
 - For all applications
- ▶ Configure the SQL execution mode and other Runtime-specific settings at the desired scope:
 - Locally (for example, at the connection or application level)
 - Custom (for example, at the data source level in an application server)
 - Globally (for example, at the Java Virtual Machine level)
- ▶ Restart the application (or application server) if necessary for the changes to take effect.

The following two examples illustrate how this property can be set globally.

Example 3-6 shows how to enable static SQL execution for a pureQuery application at the JVM level.

Example 3-6 Enabling static SQL execution at the JVM level

```
java -Dpdq.executionMode="STATIC" myjavapkg.myPQApplication
```

Example 3-7 shows how to enable static SQL execution for a pureQuery application at the JVM level using a custom configuration file containing an entry `pdq.executionMode=STATIC`.

Example 3-7 Enabling static SQL execution at the JVM level using a custom configuration file

```
java -Dpdq.configFile=myPDConfigFile myjavapkg.myPQApplication
```

For other ways to enable static execution refer to *Configuring pureQuery API applications to run SQL statements statically* in the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.static.doc/topics/tpdqsqlsttsqlrun.html>

Note: Refer to Chapter 2, “Optimizing existing Java applications” on page 19 for a detailed description on how to set the `executionMode` property at a data source level in WebSphere Application Server.

The basic deployment for the application is complete and you can proceed with the post deployment tasks.

3.7 pureQuery application maintenance

Eventually, after the pureQuery application and its dependent artifacts have been deployed to its target environment, it becomes necessary to perform common package maintenance tasks. In this section of this chapter we explore how changes to the application, the database, or the pureQuery Runtime can impact an existing deployment. The general aspects being discussed apply to pureQuery applications as well as Java applications that were enabled for pureQuery using the client optimization capability.

Bind and rebind general overview

After the initial binding to create the static package, subsequent pureQuery application maintenance tasks are primarily in the bind or rebind of the packages resulting from the changes in the application database. The choice of whether to use BIND or REBIND to rebind a package depends on the circumstances. BIND must be used in the following situations:

- ▶ When the package does not currently exist in the database.
- ▶ When there are modifications to the program. Anytime a modification happens to a program, including changes to any SQL statements, a rebind is always necessary.
- ▶ When you wish to modify any of the bind options. REBIND does not support any bind options. For example, if you want to have privileges on the package granted as part of the bind process, BIND must be used, because it has an `SQL_GRANT_OPT` option.
- ▶ When detection of all bind errors is desired. REBIND only returns the first error it detects, whereas the BIND command returns the first 100 errors that occur during binding.

Because REBIND can take significantly less time than that of BIND, we recommended using REBIND whenever the situation does not specifically require the use of BIND. An example of using REBIND is to recreate the packages after the DB2 Runstats has been executed, thereby taking advantage of the new statistics.

While rebind is faster and does not require bind options, it has restrictions on when it can be applied. For example, REBIND does not automatically commit the transaction following a successful rebind. The user must explicitly commit the transaction. This enables the “what if” analysis, in which the user updates certain statistics, and then tries to rebind the package to see what changes. This behavior also permits multiple rebinds within a unit of work.

If REBIND is executed on a package that is in use by another user, the rebind will not occur until the other user's logical unit of work ends, because an exclusive lock is held on the package's record in the SYSCAT.PACKAGES system catalog table during the rebind.

When REBIND is executed, the database manager recreates the package from the SQL statements stored in the SYSCAT.STATEMENTS system catalog table. If many versions with the same package number and creator exist, only one version can be bound at once. If not specified using the SQL_VERSION_OPT rebind option, the VERSION defaults to be “”. Even if there is only one package with a name, and creator that matches the name, and creator specified in the rebind request, it will not rebind unless its VERSION matches the VERSION specified explicitly or implicitly. If REBIND encounters an error, processing stops, and an error message is returned.

A package can become inoperative due to several reasons such as change in SQL, schema, package authority. The inoperative packages must be explicitly rebound by invoking either the bind or rebind utilities. Though the invalid packages will be automatically (or implicitly) rebound by the database manager when they are executed. This might result in a noticeable delay in the execution of the first SQL request for the invalid package. It might be desirable to rebind invalid packages explicitly, rather than let the system automatically rebind them, to eliminate the initial delay and to prevent unexpected SQL error messages that might be returned if the implicit rebind fails. For example, following database migration, all packages stored in the database are invalidated by the MIGRATE DATABASE command. Given that this might involve a large number of packages, it might be desirable to rebind all of the invalid packages explicitly at one time. This explicit rebinding can be accomplished using BIND (or DSN subcommand BIND PACKAGE on z/OS), REBIND (or DSN subcommand REBIND PACKAGE on z/OS)

3.7.1 Application changes

If changes are made to a pureQuery application's data access layer, the associated database packages will be impacted. Before we categorize some application change scenarios, we informally define two terms:

- ▶ Data access method

A *data access method* executes SQL to create, read, update, or delete data that resides in a database. These types of methods are typically invoked by the business logic of an application. For example, a data access method might return all customers that joined a loyalty club within the past 30 days.

- ▶ Data access interface

A *data access interface* groups related data access methods. A data access interface is typically associated with a single database package to support static SQL execution. For example, a customer data access interface might provide several data access methods that register new customers, update customer related information, or delete customers that have not made a purchase within a certain amount of time.

Note: Earlier in this chapter we referred to data access interfaces in a more generalized manner as Java interfaces.

Application changes might include the following scenarios:

- ▶ Addition of a new data access interface (the application will execute one or more new SQL statements).

If a new interface was added to a data access layer, a new package has to be bound to support static SQL execution.

- ▶ Addition of one or more data access method to an existing data access interface (the application will execute one or more new SQL statements).

If a method that executes new SQL is added to an existing data access interface, the associated database packages have to be bound again because the embedded consistency token was changed. For example, the addition of a new data access method that retrieves from a database all customers that joined a loyalty club during the past 14 days requires a bind of the associated package.

Note: In the case of client optimization, the new SQL has to be incrementally captured and is associated with an existing package. Starting with Version 2.2, it can be associated with a new or existing package.

- Modification of SQL in an existing data access method (the application will execute one new SQL statement instead of an existing one).

If an existing method is modified to execute different SQL, the associated database packages have to be bound again because the embedded consistency token was changed. For example, if an existing SQL statement was modified to return an additional column, a bind of the associated package is required.

Note: In the case of client optimization, the modified SQL has to be incrementally captured and is associated with an existing package. Starting with Version 2.2, it can be associated with a new or existing package. The modified SQL is not automatically removed from the metadata file. Over time, this might lead to an unnecessarily large SQL metadata file (and therefore increase package size or the number of packages), because the original SQL might never be processed again. To reduce the resource consumption, consider capturing all SQL from scratch after major data access layer changes.

- Removal of one or more methods from an existing data access interface (the application will no longer execute one or more SQL statements).

If an existing method is removed, the associated database packages have to be bound again, because the embedded consistency token was changed.

Note: In the case of client optimization, consider removing the SQL manually from the SQL metadata file (using the SQL metadata file editor in Data Studio Developer) to reduce the file size. If SQL is manually removed, the associated package has to be bound again because the consistency token has changed. After major application changes it is considered good practice to recapture SQL from scratch to reduce resource consumption.

- Removal of a data access interface (the application will no longer execute one or more SQL statements).

If a data access interface is no longer used by the business layer the associated packages can be dropped.

Note: In the case of client optimization, consider removing the SQL manually from the SQL metadata file (using the SQL metadata file editor in Data Studio Developer) to reduce the file size. If SQL is manually removed, the associated package has to be bound again because the consistency token has changed. After major application changes it is considered good practice to recapture SQL from scratch to reduce resource consumption.

Common to all discussed application changes is that they result in the creation of a new consistency token in the Java source code (for pureQuery applications) or the SQL metadata file (for client optimization). Taking advantage of this fact, you can manually use the StaticBinder utility in Data Studio Developer or in batch mode (through command line invocation) to identify the packages that need to be bound again or to bind them for the changed application. Example 3-8 and Example 3-9 depict two basic default option settings that can be used to detect which packages need to be bound or to bind them.

Example 3-8 Configuring the StaticBinder utility to determine which packages need to be created for a particular pureQuery application or SQL metadata file

```
defaultOptions=--verifyPackages SUMMARY
# add pureQuery application interface names below
# e.g. com.example.myApp.MyInterface1=
# e.g. c:\pqcmetadata\salesapp\capture.pdqxml=
```

Example 3-9 Configuring the StaticBinder utility to bind only packages for applications that have changed for a particular pureQuery application or SQL metadata file

```
defaultOptions=--differenceOnly true
# add pureQuery application interface names below
# e.g. com.example.myApp.MyInterface1=
# e.g. c:\pqcmetadata\salesapp\capture.pdqxml=
```

3.7.2 Software dependencies

The pureQuery platform is continuously improved and extended. As releases are made available, you must decide whether there is a need to upgrade or not. If you are upgrading from an older release to a newer release, take the following points into consideration:

- Can a newer version of Data Studio Developer, which is installed on a developer machine, be used to develop a data access layer that will be used by an application that uses an older version of the pureQuery Runtime in the test or production environment?

We suggest that you use the same version of Data Studio Developer and pureQuery Runtime to avoid compatibility issues.

- What has to be considered if Data Studio Developer is upgraded to a newer release?

An upgrade of Data Studio Developer (which includes a compatible version of the pureQuery Runtime libraries) can have an impact on existing applications. The Runtime version used during development must not be newer than the Runtime version used in test and production environments. This is to avoid any incompatibility issues that might arise in light of the possibility that newer releases might introduce features that are not supported by an already installed older release.

The same is true for existing Java applications that have been enabled for pureQuery using the client optimization technology. If an SQL metadata file that was captured using an older release of the Runtime is processed using a newer release of Data Studio Developer, its format will be automatically upgraded to the latest version. The updated metadata file version might not be compatible with older versions of the Runtime, raising the need to upgrade both, Data Studio Developer and existing Runtime installations to the same release.

- Can multiple release installations coexist?

Multiple releases of the pureQuery Runtime can coexist on the same machine. However, each version of the pureQuery Runtime requires a compatible level of JDBC drivers. It is advantageous to copy the corresponding JDBC libraries into the corresponding pureQuery installation directory. By using variables to identify the exact location in the file system, upgrades from older releases to a newer release can be simplified.

- Does a pureQuery upgrade impact database packages that were created using tooling that was provided in older releases?

An upgrade from one release of pureQuery to a newer release does not have an impact on database packages that were created using the older release.

- I upgraded to a newer version of the pureQuery Runtime. Can I continue using the current version of the IBM Data Server Driver for JDBC and SQL?

Each version of the pureQuery Runtime requires a compatible version of the IBM Data Server Driver for JDBC and SQL. Refer to the system requirements information for the Runtime version that you are planning to use, to determine whether a JDBC driver upgrade is required.

3.7.3 Schema changes

PureQuery applications depend on one or more packages when SQL is executed statically. These packages can become invalid when any object that the package depends on is altered or dropped. Because packages that were created using the pureQuery tooling are regular packages, common approaches can be used to maintain those packages. For further details, refer to *DB2 9 for z/OS: Packages Revisited*, SG24-7688.

3.7.4 Access path changes

The static packages must be rebound after executing RUNSTATS to make use of the updated statistics. A rebind is required after performing REORG only if the runstats is run in conjunction with the reorg. However, it is a common practice to run RUNSTATS followed by REORG in a production environment. In this case, it is imperative to rebind after the REORG.

For further details, refer to *DB2 9 for z/OS: Packages Revisited*, SG24-7688.

3.7.5 Maintaining packages using Data Studio Developer

Data Studio Developer provides support for basic package management tasks.

Package creation

If the applications pureQuery data access code is available as Java source code or in compiled form (for example, packaged in an archive), the associated packages can be created using the bind wizard. This wizard is the same wizard that was used in Chapter 2, “Optimizing existing Java applications” on page 19 to bind packages from SQL metadata files that were created using the pureQuery client optimization process.

Package management

The Data Source Explorer, shown in Figure 3-6, provides a live database view with access to the following common package maintenance tasks

- ▶ Inspecting the properties, content and validity of a package
- ▶ Reviewing granted authorization privileges
- ▶ Rebinding existing packages

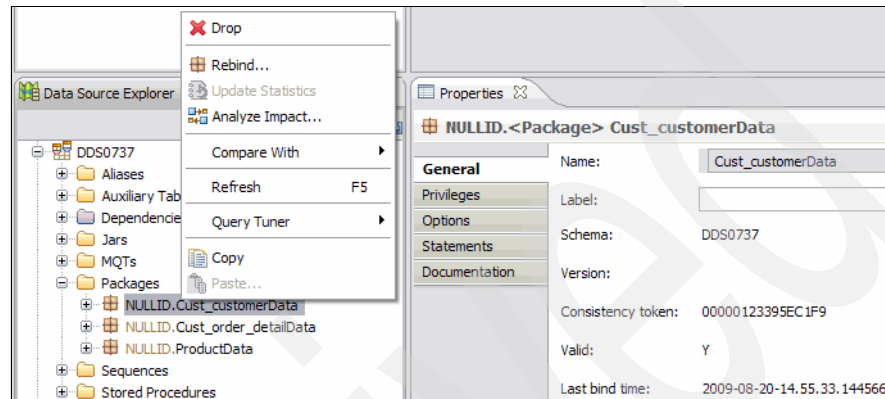


Figure 3-6 Inspecting package properties using the Data Source Explorer in Data Studio Developer

The package properties view can also be used to associate a package with its originating data access interface (or SQL metadata file), because correlation information was added by the StaticBinder utility to the package description.

3.8 Evaluation

In this chapter, we described the deployment process for pureQuery applications and discussed some of the common application maintenance aspects from a DBA perspective.

To learn more about the pureQuery API and how developers can use the sophisticated tooling in Data Studio Developer to build data access layers more efficiently, go to *Develop - Runtime & API* of the Integrated Data Management community page at the following Web page:

<http://www.ibm.com/developerworks/spaces/optim>

Managing enterprise systems with PE/EI and ITCAM

DB2 Performance Expert and DB2 Performance Expert Extended Insight Feature (PE/EI) provide important detailed monitoring and analysis information to help you with end-to-end problem solving related to database applications accessing DB2 for Linux, UNIX, and Windows. IBM Tivoli Composite Application Manager (ITCAM) monitors system-wide performance, especially for application monitoring. In this chapter we discuss how to manage enterprise systems with PE/EI, Tivoli OMEGAMON XE for DB2 Performance Expert for z/OS (OMPE), and ITCAM.

4.1 The need for enterprise management

To achieve scalability and performance, most Internet application deployments have evolved into multi-tier infrastructures where the Web server tier serves as the Web front-end, the business logic is executed on the middleware application servers, and the backend storage and access are provided through the database servers. While the multi-tier infrastructures offer a variety of scalability and extensibility benefits, they are also more difficult to operate and manage. When a problem occurs (for example, a slowdown), an administrator often has difficulty in figuring out which applications in the multi-tier infrastructure could be the cause of the problem. Is it the network? The database? The Web server? Comprehensive routine monitoring of every component in the infrastructure is essential to maintain proactively the system at its optimal performance.

The application server middleware that hosts and supports the business logic components is often the most complex of the multi-tier infrastructure. To offer peak performance, an application server provides a host of complex functions and features including database connection pooling, thread pooling, database result caching, session management, bean caching and management, and so on. To ensure that the application server is functioning effectively at all times, all of these functions have to be monitored and tracked proactively and constantly.

When monitoring a multi-tier infrastructure, problems are best solved by a top-down approach. Before any diagnosis can be made, it is essential to understand the inter-relationships between all of the components in the application.

The various components include the database tier, the application tier, and the graphical and persistent session components of WebSphere Application Server. It might also include gateways and connection tiers on enterprise-wide applications. When a user of an application reports that the application is slow, it is not always easy to tell where the problem or problems might be happening. It could be in the database where it is manifested as a lower buffer pool hit ratio or as a low read-to-pages-read ratio. It might be in the application tier and be manifested as a connection issue. Perhaps it is a non-issue and is working as designed. If this is true, was the design flawed from the beginning? Could the application been underestimated on usage, connection, and SQL access? The desired goal is to have a well-tuned application that spans many tiers and database management systems but is transparent in operation from a user perspective.

4.2 Monitoring with PE/EI

Finding where the problem is occurring can be as daunting as knowing what to look for. It is rare to find a single person or entity who knows the whole application from the user perspective to the database perspective. This is largely due to the fact that in most large shops it is far more efficient to have groups organized around a structural discipline (such as a database management system, system administration, or application tier) than it is to be organized on an end-to-end basis of a single application. And because many companies have hundreds of applications, it makes little sense to have a single person chartered with the responsibility of managing many applications end-to-end.

But being organized around structural discipline has a disadvantage when trying to find where a problem could be happening. Finding an issue somewhere across connections and tiers presents a problem and untold wasted time in getting around the politics of problem determination.

This is where IBM DB2 Performance Expert and DB2 Extended Insight Feature (DB2 PE/EI) can help with end-to-end problem solving by helping you resolve emergent problems before they can affect the business.

Java-based composite application environments have created huge benefits from their flexibility in application development and deployment. However, with this added benefit, they have created huge challenges for DBAs in understanding and managing overall database performance within those environments. DB2 Performance Expert Extended Insight Feature enhances DB2 Performance Expert to solve these problems by providing the DBA with an end-to-end database monitoring solution for a complete view of database performance across the database client, the application server, the database server, and the network between them.

Figure 4-1 on page 122 shows a typical Java application on a two-tier construct. While DB2 Performance Expert alone can easily allow for monitoring in the database server tier, it cannot cover monitoring end-to-end. For this type of monitoring, Performance Expert Extended Insight can provide the total picture by showing the performance and connection activity on the database server just outside of the DBMS context and can also provide the performance, connection, and application functional activity on the application server tier.

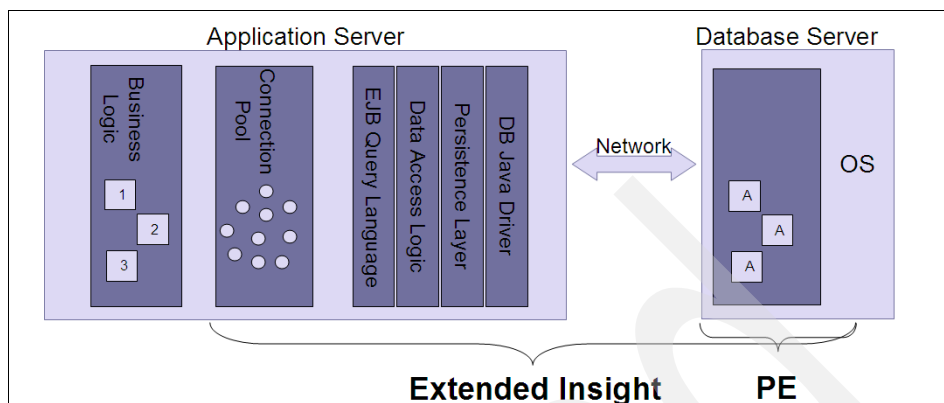


Figure 4-1 Typical Java application in a two-tier construct

With the Extended Insight Feature, database administrators obtain a complete view of the overall database system performance from the same perspective as the database application, which means they can now perform the following tasks:

- ▶ Monitor performance indicators that better reflect user experience and proactively detect negative trends in response times for database APIs, network congestion between the application and the database, or other factors key to sustaining service level agreements.
- ▶ See where workloads, transactions, and SQL requests spend their time across the database client, the application server, the database server, and the network between them.
- ▶ Isolate database problems to the correct layer in the stack, including visibility into database-related performance indicators in WebSphere application servers.

The end-to-end monitoring is available today for dynamic Java workloads with DB2 Performance Expert and Performance Expert Extended Insight Feature.

4.2.1 Performance Expert Extended Insight Feature

By extending DB2 Performance Expert with the Performance Expert Extended Insight Feature, you get graphical reporting and analytical drilldown capabilities related to activity outside of the database server itself. In other words, DBAs can use the same product used for DB2 for Linux, UNIX, and Windows performance measurement to gather important metrics from the application tier or server, and any of the connectivity servers and gateways end-to-end (for Java applications).

Performance Expert Extended Insight Feature allows the DBA to see the big picture:

- ▶ The end to end response time across the database client
- ▶ The application server
- ▶ The network

This provides the database administrator immediate insight into where Java database workloads, transactions, and SQL requests are spending their time. Performance Expert Extended Insight Feature reduces the time needed to isolate performance issues and makes it easier to meet service level agreements for Java applications

Figure 4-2 shows the end-to-end tab of Performance Expert Extended Insight on the Performance Expert Overview window.

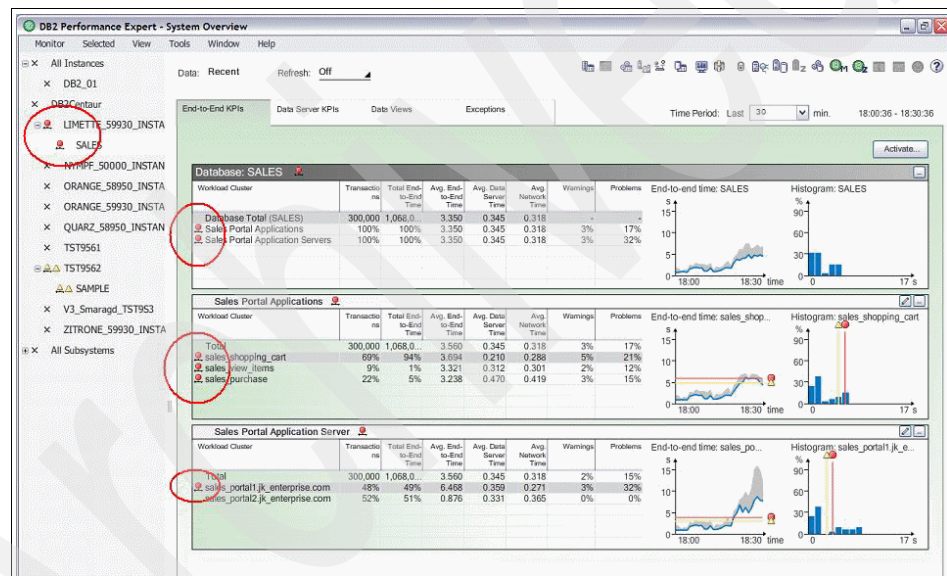


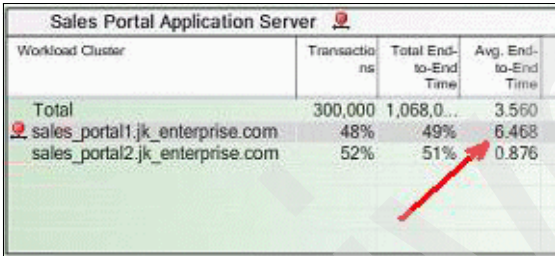
Figure 4-2 Performance Expert Extended Insight overview

Performance Expert can notify the user of alerts through e-mail, UserExit, or by flagging the event in the overview window with alert icons. In this example, red indicators show that some key objectives are being missed.

The key objectives are established as event exceptions and fully modifiable. The event exception handling is covered in 4.2.2, “Performance Expert” on page 128.

When a threshold reaches a warning level setting, a yellow triangle appears next to all textual and graphic displays on the system overview. When a threshold reaches a problem level setting, a red dot appears in place of a warning level yellow triangle. By using this method, a DBA is quickly able to see and perform further drill-down on the problem.

Figure 4-3 shows a view of an issue seen in the end-to-end tab of Performance Expert Extended Insight overview window. Note the average end-to-end time shows a fairly high average time for application server sales_portal1.jk_enterprise.com. Further analysis can be obtained by double-clicking this field.



Workload Cluster	Transactions	Total End-to-End Time	Avg. End-to-End Time
Total	300,000	1,068,0...	3.560
sales_portal1.jk_enterprise.com	48%	49%	6.468
sales_portal2.jk_enterprise.com	52%	51%	0.876

Figure 4-3 End-to-end Drill Down on Application Server

Figure 4-4 on page 125 shows the details of the activity on the server shown in Figure 4-3. At issue is the spike in connection wait time within the WebSphere application server, shown in detail in Figure 4-5 on page 125.

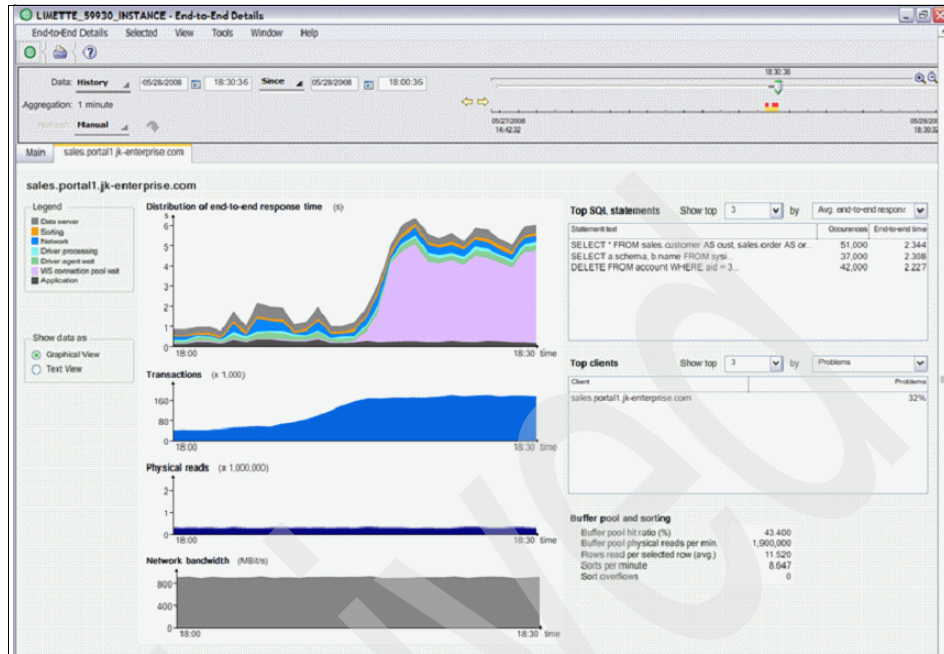


Figure 4-4 End-to-end server details

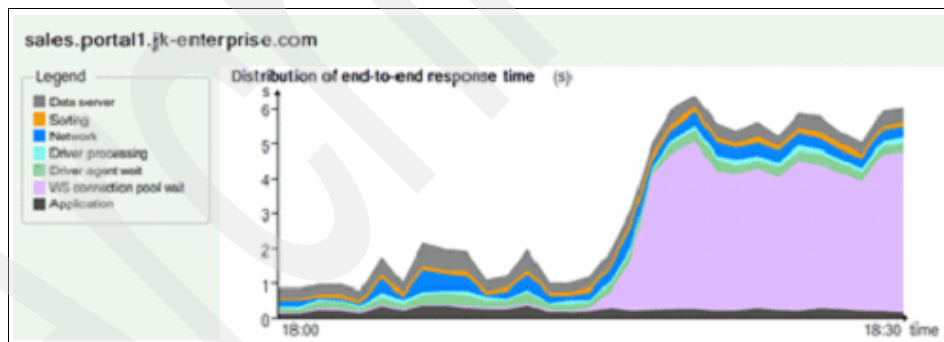


Figure 4-5 End-to-end response time

At this point, we can answer the question of, “What happened?” but to answer “Why did it happen?”, further drill down is necessary on the connection pool information. By double-clicking the top problem window, shown in Figure 4-6, we can drill in on the issue to determine resolution.

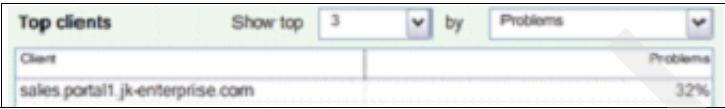


Figure 4-6 Problem drill-down

Figure 4-7 shows the details of the WebSphere connection pool problem.

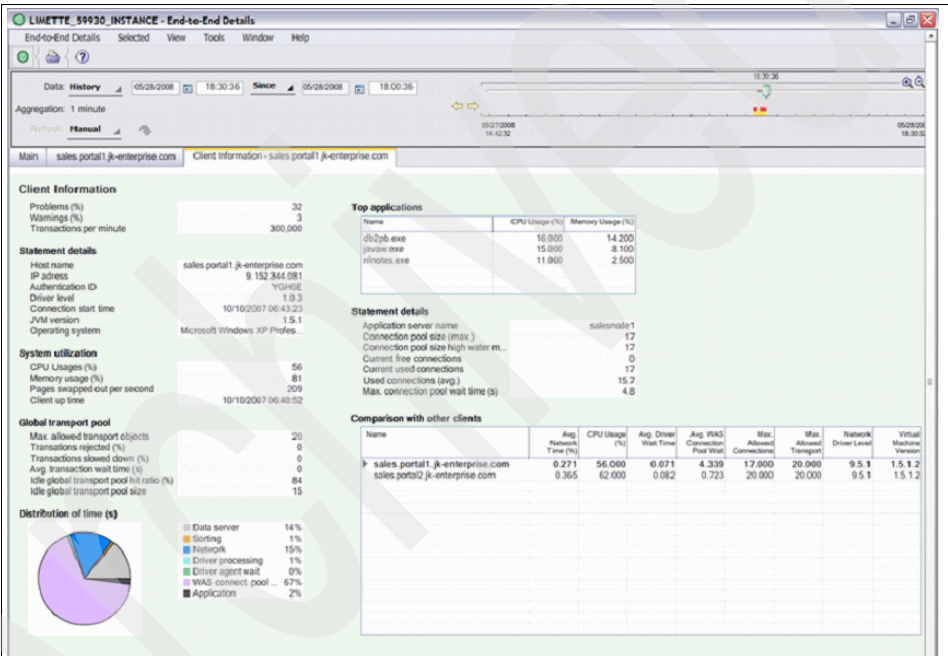


Figure 4-7 WAS connection details

Looking at the close-up of the problem shown in Figure 4-8 on page 127, we can see that the 4.8 second wait time indicates that the maximum number of allowed connections is not sufficient. The high-water mark is 17 connections and both maximum allowable connections and current connects are all at 17.

With this data the DBA has two possible courses of action:

- ▶ Raise the allowable connections.
- ▶ Ask the application why this value jumped in size.

Either way, the DBA is armed with more than a guess on why things are slow. There is actual data to explain the condition across the application enterprise.

Statement details	
Application server name	salesnode1
Connection pool size (max.)	17
Connection pool size high water m...	17
Current free connections	0
Current used connections	17
Used connections (avg.)	15.7
Max. connection pool wait time (s)	4.8

Figure 4-8 WAS connection details close-up

Performance Expert Extended Insight works with any Java application at JDK 1.5 or above. Additionally, the methods shown in Table 4-1 provide additional information for Performance Expert Extended Insight.

Table 4-1 Performance Expert Extended Insight Java methods

Method	Information provided
setDB2ClientAccountingInformation	Accounting information.
setDB2ClientApplicationInformation	Name of the application that is working with a connection.
setDB2ClientDebugInfo	The CLIENT DEBUGINFO connection attribute for the Unified debugger.
setDB2ClientProgramID	A caller-specified string that helps the caller identify which program is associated with a particular SQL statement. Also provides user name for a connection.
setDB2ClientWorkstation	Client workstation name for a connection.

To summarize Performance Expert Extended Insight feature:

- ▶ PE/EI improves the availability of mission-critical Java database applications by detecting negative trends sooner. This includes the following monitored trends:
 - Eroding response times for database APIs
 - Network congestion between the application and the database
 - Other factors key to sustaining service level agreements
- ▶ It can manage applications to meet service level agreements more easily and effectively. DBAs can see the response time for single SQL requests or complete database transactions from the applications point of view, enabling creation of performance indicators that more directly relate to the user's experience.

- ▶ The time needed to isolate performance issues is reduced from days to minutes. It uses graphical displays to see where workloads, transactions, and SQL requests originate in the application, and how the time spent is spread across the database client, the application server, the database server, and the network between them.
- ▶ It improves collaboration and resolution of performance problems across the software stack by providing additional performance indicators (such as connection pool wait times and other vital database-related performance statistics) when used with WebSphere Application Server.

4.2.2 Performance Expert

DB2 Performance Expert is a tool to help resolve emerging problems before they can affect the business. While Performance Expert Extended Insight helps with end-to-end monitoring, Performance Expert provides a deep drill-down capability for DB2 for LUW by collecting statistics and application information and storing it in its DB2 Performance Expert Server. In addition, Performance Expert has a repository that takes current historical information and aggregates this information into its performance warehouse database.

Figure 4-9 shows a sample configuration topology.

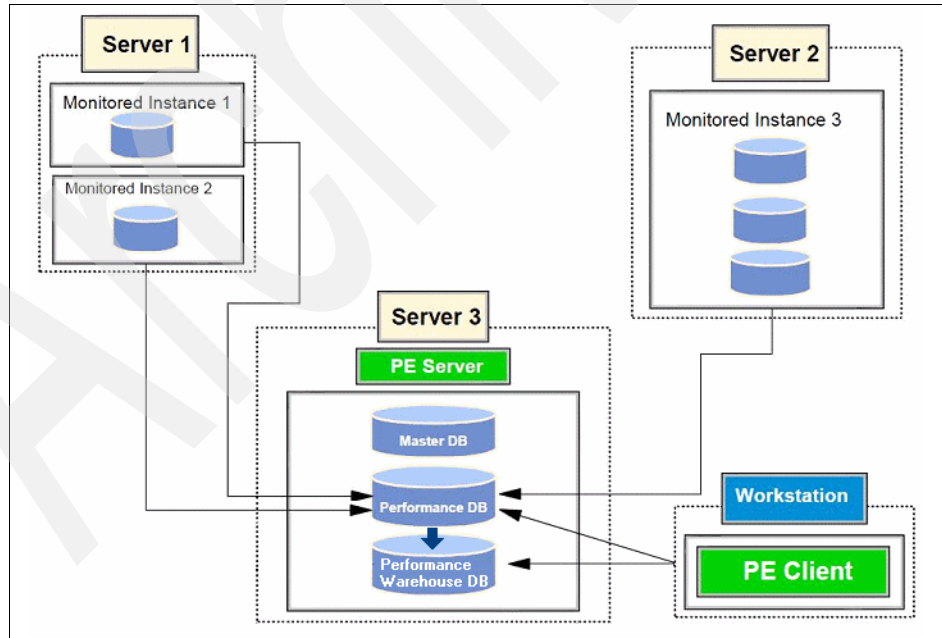


Figure 4-9 Performance Expert example configuration

Server 1 has 2 DB2 for LUW instances with only one database in each instance being monitored. Server 2 has a single instance of DB2 for LUW with three separate databases each being monitored in Performance Expert. Server 3 is the host server for DB2 Performance Expert Server. Performance Expert Server also has an instance of DB2 for LUW running with the tables required for DB2 Performance Expert repository.

Note the arrow pointing from the performance database to the performance warehouse database. While this is a simplistic representation of the database structure within DB2 Performance Expert, the object of this representation is that the data from the performance database is periodically aggregated and moved to the performance warehouse, based on installation configuration of the performance warehouse.

Among the many features in DB2 Performance Expert, its stand-out functions include the following features:

- ▶ Event Exceptions to service level agreement (SLA)
Performance Expert provides a rich, deep set of metric measurement with the ability to establish warning and problem settings. When any threshold is reached, PE has the capability to issue notification through SMTP or by a UserExit through SNMP.
- ▶ Overview Alert
A set of key performance indicators (KPI) will show warning (yellow triangle) or problem (red dot) level issues. These issues can then be investigated to find details.
- ▶ Application thread drill down and statistics drill down
Performance Expert can take the data provided and present it in any dimension required:
 - By thread details and drill-down
 - By statistics details and drill-down
- ▶ Database and Instance settings
Configuration settings can be viewed easily with no more than two clicks of the mouse.
- ▶ Data warehouse of performance history
Performance Expert keeps recent performance data, based on configurable settings, and periodically saves this data to a performance data warehouse.

Additionally, no agent is required on the monitored server. Having a performance monitoring tool that is agent-free makes Performance Expert easy to manage and maintain.

Deadlocks

To illustrate how DB2 Performance Expert with drill-down capability can be used to find bottlenecks and resource contention in the enterprise, let us look at an all-too-common problem: *deadlocks*.

What is a deadlock? A deadlock is a type of contention where an initiating SQL locks either a row or a table in update intent and where there is a another SQL (victim SQL) that needs access to that row or table.

After a lock is acquired, it is held until the owning or initiating transaction is terminated. At that point, the lock is released and the data resource is made available to other transactions. A lock is released when one of three courses of action occurs:

- ▶ The initiating thread terminates abnormally, which forces a rollback on any updates made to rows.
- ▶ The initiating thread issues a COMMIT, which will release the lock.
- ▶ The application completes or an implied COMMIT.

There are strategies to prevent deadlock timeouts, namely in setting the isolation level of the access. DB2 for LUW provides different levels of protection to isolate the data from each of the database applications while it is being accessed. By choosing an appropriate isolation level, we can maintain data integrity and avoid unnecessary locking. The following isolation levels are supported by DB2 in order of concurrency, starting with the maximum concurrency:

- ▶ Uncommitted read

The uncommitted read (UR) isolation level, also known as “dirty read,” is the lowest level of isolation supported by DB2. It can be used to access uncommitted data changes of other applications. For example, an application using the uncommitted read isolation level will return all of the matching rows for the query, even if that data is in the process of being modified and might not be committed to the database. You need to be aware that if you are using an uncommitted read, two identical queries might get different results, even if they are issued within a unit of work, because other concurrent applications can change or modify those rows that the first query retrieves. Uncommitted read transactions will hold few locks. Thus, they are not likely to wait for other transactions to release locks. If you are accessing read-only tables or it is acceptable for the application to retrieve uncommitted data updated by another application, use this isolation level, because it provides better performance.

► Cursor stability

The cursor stability (CS) isolation level is the default isolation level and locks any row on which the cursor is positioned during a unit of work. The lock on the row is held until the next row is fetched or the unit of work is terminated. If a row has been updated, the lock is held until the unit of work is terminated. A unit of work is terminated when either a COMMIT or ROLLBACK statement is executed. An application using cursor stability cannot read uncommitted data. In addition, the application locks the row that has been currently fetched, and no other application can modify the contents of the current row. As the application locks only the row on which the cursor is positioned, two identical queries might still get different results even if they are issued within a unit of work. When you want the maximum concurrency while seeing only committed data from concurrent applications, choose this isolation level.

► Read stability

The read stability (RS) isolation level locks those rows that are part of a result set. If you have a table containing 100,000 rows and the query returns 10 rows, then only 10 rows are locked until the end of the unit of work. An application using read stability cannot read uncommitted data. Instead of locking a single row, it locks all rows that are part of the result set. No other application can change or modify these rows. This means that if you issue a query twice within a unit of work, the second run can retrieve the same answer set as the first. However, you might get additional rows, as another concurrent application can insert rows that match to the query.

► Repeatable read

The repeatable read (RR) isolation level is the highest isolation level available in DB2. It locks all rows that an application references within a unit of work, no matter how large the result set. In some cases, the optimizer decides during plan generation that it might get a table level lock instead of locking individual rows, because an application using repeatable read might acquire and hold a considerable number of locks. The values of the LOCKLIST and MAXLOCKS database configuration parameters will affect this decision. An application using repeatable read cannot read uncommitted data of a concurrent application. As the name implies, this isolation level ensures the repeatable read to applications, meaning that a repeated query will get the same record set as long as it is executed in the same unit of work. Because an application using this isolation level holds more locks on rows of a table, or even locks the entire table, the application might decrease concurrency. You should use this isolation level only when changes to your result set within a unit of work are unacceptable.

The two most widely used strategies to prevent deadlocks within an application is the use of COMMIT and the use of isolation level.

For strategies outside of the application in dealing with deadlocks, review the database configuration parameters which control locking:

► LOCKLIST

This parameter indicates the amount of storage that is allocated to the lock list, in 4K pages.

► MAXLOCKS

This parameter defines the minimum percentage of the lock list held by an application that must be filled before the database manager performs lock escalation.

► LOCKTIMEOUT

This parameter specifies the number of seconds that an application will wait to obtain a lock. This parameter helps avoid global deadlocks for applications.

Make sure LOCKLIST and MAXLOCKS DB configuration parameters are large enough by monitoring occurrences of lock escalation. Insufficient locklist size results in DB2 attempting to escalate many row locks to a single table level lock, causing deadlocks if the escalation fails, and greatly impacting concurrency if the escalation succeeds.

In summary, deadlocks usually do not occur or happen on static, stable, and non-changing databases and applications. They usually happen in the following circumstances:

► On new application where they are deadlocking within the application itself

This is mostly as a result of inadequate system testing or “shake-out” integration testing.

► On new applications which access an existing database

This is also due to inadequate system testing but can also be caused of inadequate estimation of the usage of the existing system.

► On changes to an existing application such as batch work timing, change to SQL, or change in ordering

You can reduce lock waits and deadlocks with the following approaches:

► Issuing COMMIT statements at the right frequency

► Choosing the appropriate isolation level

► Specifying the FOR FETCH ONLY clause in the SELECT statement

► Releasing read locks using the WITH RELEASE option of the CLOSE CURSOR statement, if acceptable

► Tuning the LOCKLIST and MAXLOCKS database configuration parameters to avoid lock escalations that impact concurrency

Performance Expert usage with deadlocks

The next area to cover is setting up Performance Expert to monitor for deadlocks. Deadlocks, along with the many available matrixes for measurement in Performance Expert are set up in the event exceptions. The next several figures show how this process is accomplished.

To begin with, from the DB2 Performance Expert Client, the event exception icon is selected (as shown in Figure 4-10), which brings up the exception processing window.

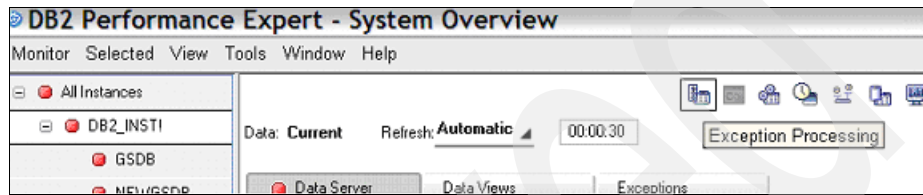


Figure 4-10 Event exception icon

In the event exception window, a new threshold for deadlocks can be added to an existing threshold set or a whole new threshold can be built. Figure 4-11 shows how to select a new threshold set.

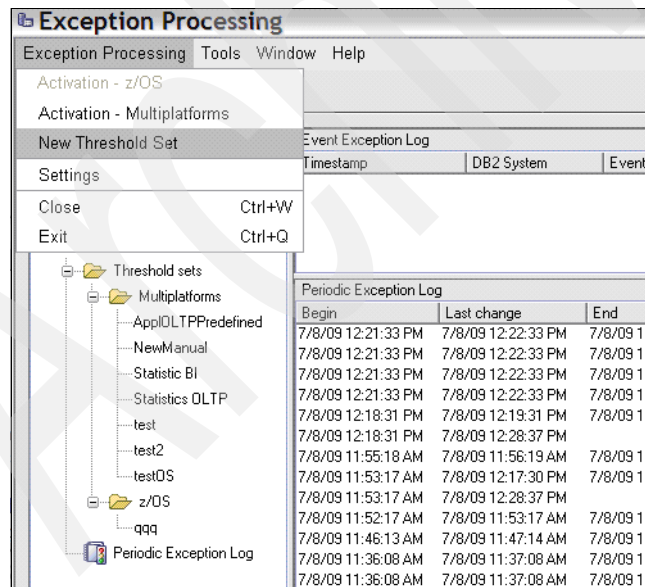


Figure 4-11 New threshold set for event exceptions

DB2 Performance Expert threshold settings can be built manually by adding and configuring each threshold setting or a set can be added using four possible templates with default settings. We suggest using a template for setting up event monitoring whenever you begin monitoring a new set of servers, instances, and databases. Choose from the following template sets:

- ▶ Statistical OLTP
- ▶ Application OLTP
- ▶ Statistics Business Intelligence (BI)
- ▶ IBM Content Manager

Figure 4-12 shows how to set up new event exceptions from a template. In this example, the application OLTP template is used.

New threshold set

Operating system: ☒ Multiplatforms ☐ z/OS

Name: Application

Author: tqb

Creation date: Jul 8, 2009 12:31:07 PM

Modification date: Jul 8, 2009 12:31:07 PM

Description:

Create a new threshold set or select a predefined one.

☐ New

☒ Predefined

Name	Description
Statistics OLTP	Thresholds for an OLTP environment
Application OLTP	Thresholds for an OLTP application
Statistics BI	Thresholds for an BI/DW environment
IBM Content Manager	Thresholds for IBM Content Manager...

OK Cancel Help

Figure 4-12 Establishing new threshold event exceptions from template

Because of the need to set some additional warning and problem event exceptions for deadlocks, Figure 4-13 shows the modifications performed to the event exception periodic event for deadlocks.

Edit threshold - Exclusive lock escalations

Exception field

Select the exception category
Applications

Select the exception subcategory
Agents
Buffer Pool
Cache
Locks
Miscellaneous
SQL Activity
Sort
Workspace

Select the exception field
Agents waiting on locks
Deadlocks detected
Exclusive lock escalations
Lock escalations
Lock timeout (sec)
Lock timeouts since connected
Lock waits since connect
Locks held by application

Warning and problem threshold

Value > WARNING threshold 0 per minute
PROBLEM threshold 1

Qualifier

Name
Application information - ID
Application information - Name
Application status
Authorization ID
Client - Communication protocol
Client - Configuration NNAME
Client - Operating platform
Client - Product / version ID
DB alias
DB name
DB path

Operator
=

Value

Conditions

Qualifier	Operator	Value/Pattern
-----------	----------	---------------

Remove Remove all

OK Cancel Help

Figure 4-13 Adjusting event for lock escalations

After the event exceptions are established, they must be activated. Figure 4-14 shows how this is done.

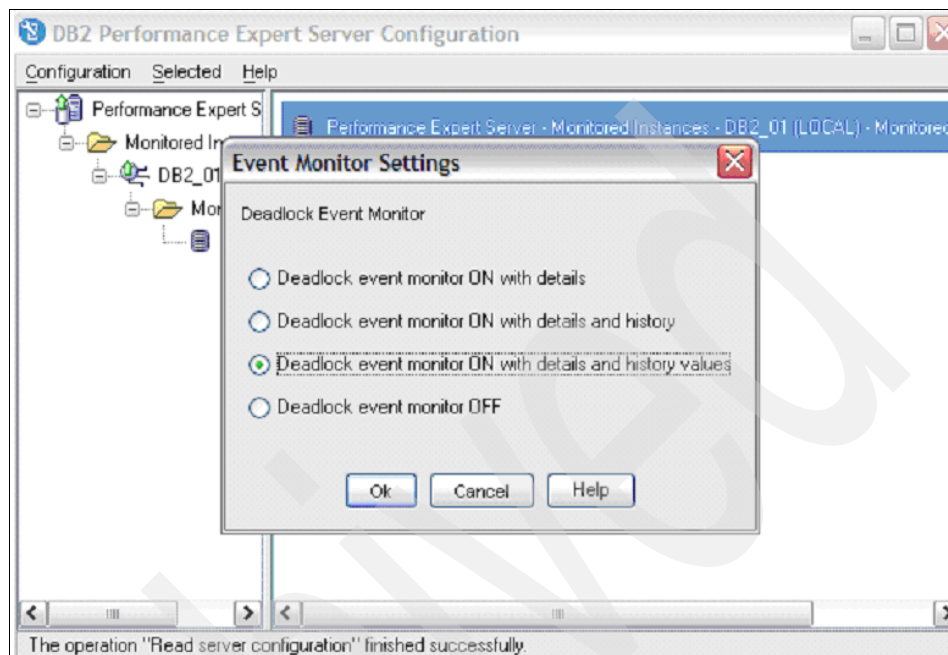


Figure 4-14 Activation of event exceptions

When a threshold is reached on a monitored event, Performance Expert can notify by E-mail or by SNMP UserExit. Figure 4-15 demonstrates how to activate these two notification processes.

The SMTP server is listed under Host and Port while the SNMP UserExit, which resides on the Performance Expert Server specifies to location of the UserExit file.

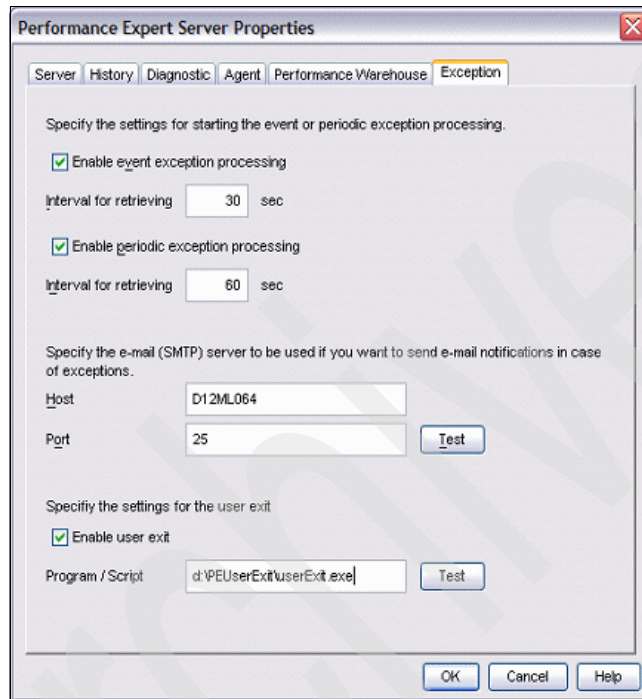


Figure 4-15 SMTP and SNMP notification setup

Figure 4-16 shows an example of an e-mail notification of a deadlock condition.

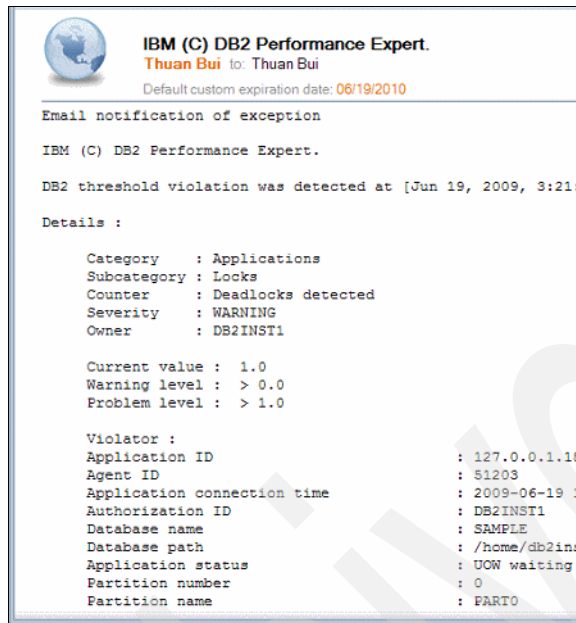


Figure 4-16 E-mail notification of a deadlock

After event exceptions are established and activated, DB2 Performance Expert starts collecting data through snapshots and using DB2 for LUW event monitoring, which is activated under the auspices of DB2 Performance Expert.

Over time, data is collected and matrixes are compared against the established matrixes defined in the event exceptions. Depending on the metric and its associated threshold setting, Performance Expert issues warnings or event errors for the periodic exception.

Figure 4-17 shows the DB2 Performance Expert's Overview window. In the window you can see both periodic exceptions and the event exceptions. In this example, we look at the deadlock event exception and try to find not only what happened, but why it happened.

Expert - System Overview

tools Window Help

Data: Current Refresh: Automatic 00:00:30

Data Server KPIs Data Views Exceptions 6/18/09 5:00:52 PM

Periodic Exception Log

Name	Object Name	Partition Name	Frequency	Timestamp	
Data (page) hit ratio (%)	SAMPLE	PART0	53	6/18/09 5:00:52 PM	Delete
Index hit ratio (%)	SAMPLE	PART0	30	6/18/09 5:00:52 PM	Delete
Hit ratio (%)	SAMPLE	PART0	32	6/18/09 5:00:52 PM	Delete
Deadlocks detected	SAMPLE	PART0	6	6/16/09 6:15:07 PM	Delete
Free Space (%)	/dev/disk/by-id/ata-HITA...	N/P	1	6/15/09 6:09:39 PM	Delete
Used pages in table spac...	SAMPLE	PART0	3	6/15/09 6:09:39 PM	Delete
Statements - Failed operat...	SAMPLE	PART0	47	6/15/09 3:34:16 PM	Delete
Average number of rows t...	SAMPLE	PART0	4	6/14/09 7:35:46 PM	Delete

Event Exception Log

Name	Object Name	Specific Information	Frequency	Timestamp	
Deadlock	SAMPLE	Database: SAMPLE	8	6/16/09 6:12:17 PM	Delete

Figure 4-17 Exception shown on overview with deadlock condition

Also note that the Lock Timeout value is -1. This means there is no timeout on this database. Will this create problems downstream? Possibly, depending on other access to the database.



Figure 4-19 shows another method to obtaining this information by selecting applications in lock conflict.

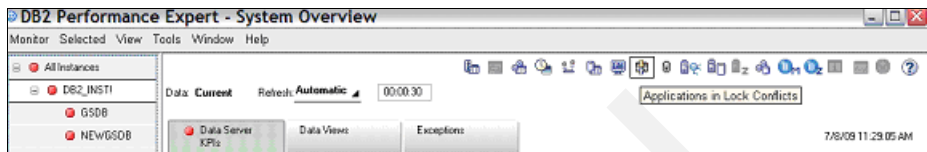


Figure 4-19 Applications in lock conflict

By selecting the icon for application in lock conflict, we can see the database and mode for the lock in Figure 4-20.



Figure 4-20 Database and mode

Right-clicking displays a drop-down menu to Application Details shows the details of the lock (Figure 4-21).

We see the details of the two participants of the deadlock and have the information about why it happened.

What this example has shown is the ease of use and drill-down capabilities of Performance Expert. Performance tuning to event exceptions based on service level agreements and objectives is the best way to manage the DB2 for LUW enterprise.

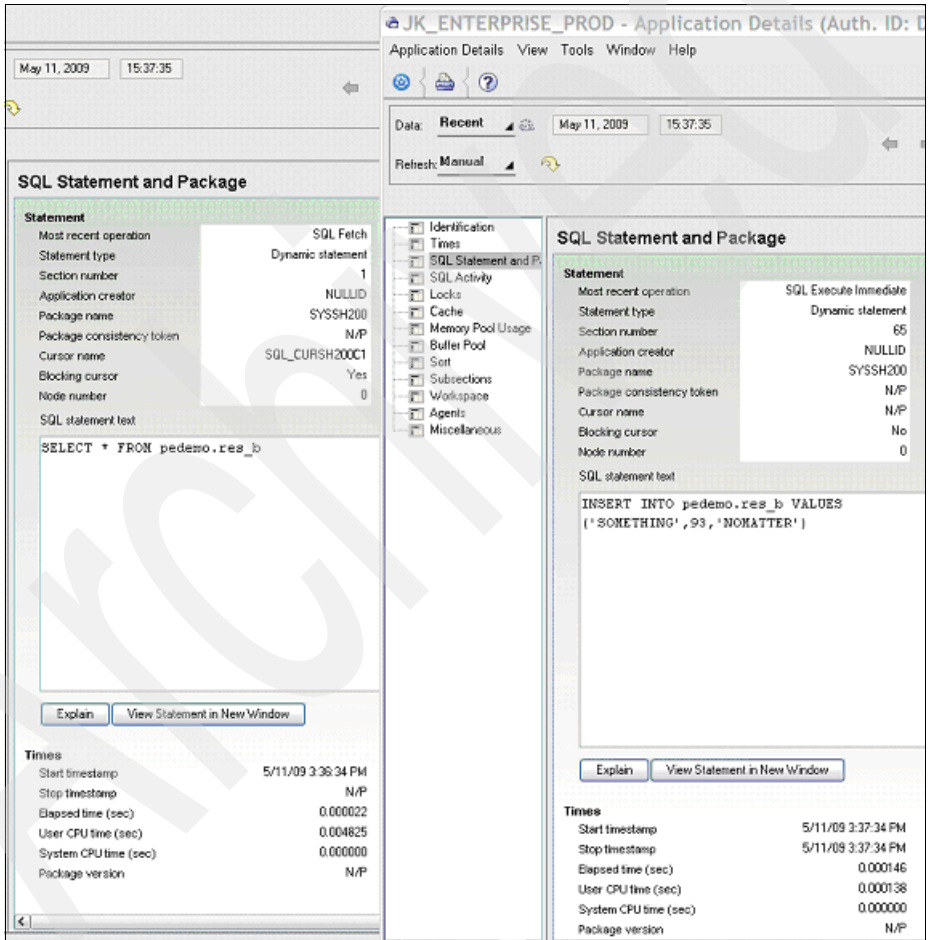


Figure 4-21 Deadlock participant comparison

4.3 Monitoring with ITCAM

IBM Tivoli Composite Application Manager (ITCAM) can be used to monitor system-wide performance. Given a service level agreement, ITCAM can be geared to generate triggers and alerts on problem areas, especially the application server. You can use ITCAM and database server performance monitoring tools such as OMPE or DB2 Performance Expert to monitor and diagnose root causes of performance problem. In this section, we discuss ITCAM features that can be applied for monitoring the application.

4.3.1 ITCAM introduction

ITCAM monitors application transactions end-to-end, alerts management teams for the potential application problems, and provides intensive diagnostic information for repair. ITCAM for WebSphere and J2EE offers a comprehensive deep-dive diagnosis capability into WebSphere and J2EE application server environment. You can define the proactive problem monitoring alerts and events to automate the diagnostic data collection and the system support notification. It also provides request correlation capability for diagnosing requests and transactions that span multiple J2EE application servers, CICS, and IMS.

Table 4-2 shows a set of ITCAM alerts that we suggest for monitoring the general health of a Java virtual machine (JVM). These alerts are independent of each other so a user can mix and match them to suit their environment.

Table 4-2 ITCAM alerts

Alert name	Time (min)	Condition
DB Connection Pool Thread Timeout	5	Thread Timed Out > 0
WebSphere Error	5	Monitor SystemOut.log for error severity
High CPU Percent Used	0.5	CPU Used > 80%
High GC Time Percent	5	Real Time % > 80%
High App Response Time	5	Avg Response Time > 2000 ms
Out of Heap Space	0.5	Allocation_Failure.Heap_Status = 1
Servlets JSP's Error	0.5	Error Count > 0
Thread Pool Percent Maxed	5	% of Time Thread Pool at Max > 80%
Web Application Error	5	Error Count > 0

Alert name	Time (min)	Condition
JCA Connection Pool Waiting Threads	5	Concurrent Waiting Threads > 0
WebContainer ThreadPool Percent Maxed	5	% of time Web Container Pool at Max > 80%

You also can use pureQuery with ITCAM to diagnose and fix database-related problems. While ITCAM can be used to diagnose and isolate the problem effecting the service-level agreement, pureQuery can be used to fix it, regardless of whether it is database performance-, security-, or administration-related.

4.3.2 Configuring ITCAM for WebSphere for application monitoring

In this section, we demonstrate application monitoring with ITCAM by The Great Outdoor Company pureQuery application. The topology of our lab environment is shown in Figure 1-4 on page 17 with the following ITCAM related details:

- ▶ ITCAM 6.1 for WebSphere and J2EE is installed on a Windows server `sdns04h1.itso.ibm.com`.
The ITCAM installation and configuration is described in detail in *IBM Tivoli Composite Application Manager Family Installation, Configuration, and Basic Usage*, SG24-7151.
- ▶ ITCAM admin module runs on the default profile.
- ▶ Managing server is installed on the same machine.
- ▶ Go_Sales enterprise application used by the Great Outdoor Company is installed on a profile called AppServ01. We monitor this application server.
- ▶ Data Collector for WebSphere is installed. Go_Sales pureQuery application is instrumented for data collection. ITCAM collects performance data and displays as performance matrixes on the admin module.
- ▶ `dsredbookgroup` is the group defined for instrumenting the server for monitoring the Go_Sales enterprise application.

You must verify that all the required modules of ITCAM are up and running. Do this by navigating from the ITCAM administrative module: **Administration** → **Managing Server** → **System Diagnostics**.

The publisher, data collector, and other selected ITCAM modules should be up and running. Make sure that for every module you configured to start with the ITCAM kernel, there is a correspondent diagnostic component shown in the ITCAMWAS administrative diagnostic menu. For example, if you start the data collector with kernel, there should be an entry in the kernel startup log indicating

that the data collector module is up and running and a component shown in the ITCAM administrative module system diagnostic menu.

Figure 4-22 shows the correlation between ITCAM kernel startup message and the ITCAMWAS administrative module system diagnostic menu.

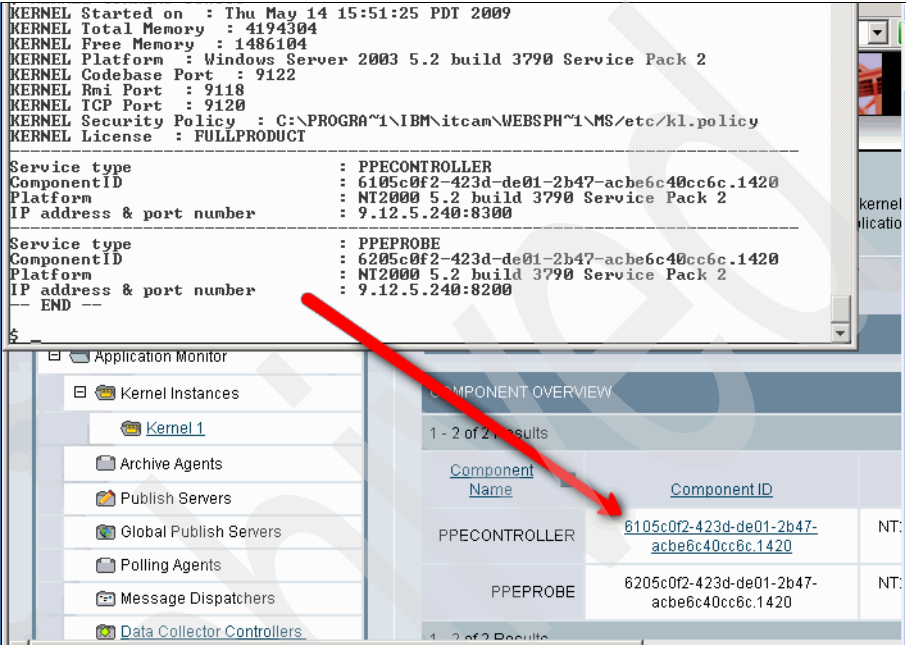


Figure 4-22 ITCAM Kernel and diagnostics messages match

Figure 4-23 shows how Go_Sales datasource is configured inside ITCAM.

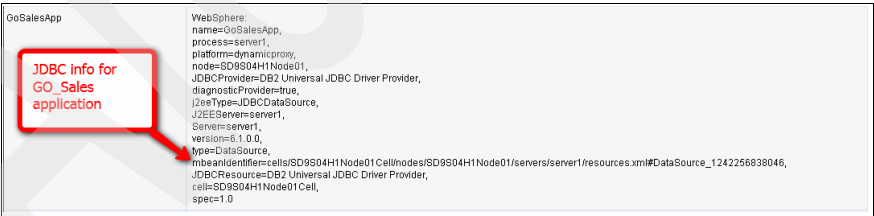


Figure 4-23 Go_Sales datasource configuration for ITCAM

Figure 4-24 shows a snapshot of the key indicators as the online transactions proceed.

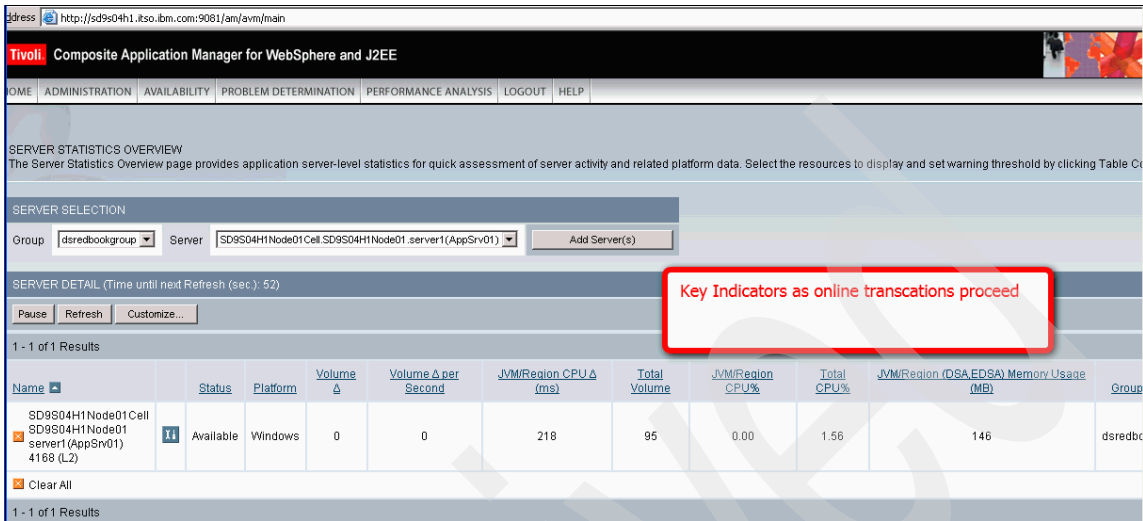


Figure 4-24 Key indicators inside ITCAM admin module

4.3.3 Application monitoring using ITCAM

In this section, we illustrate how performance diagnosis is achieved with the help of ITCAM and OMPE. We run 100 and 1000 users simulations using Apache Jmeter that generates client requests against our DB2 for z/OS database. Jmeter calls the dynamic SQL application deployed on WAS server AppSrv01. We monitor transaction response times, JVM settings, and CPU usage.

First, we started observing high activity on ITCAM. Figure 4-25 shows a snapshot of our capture.

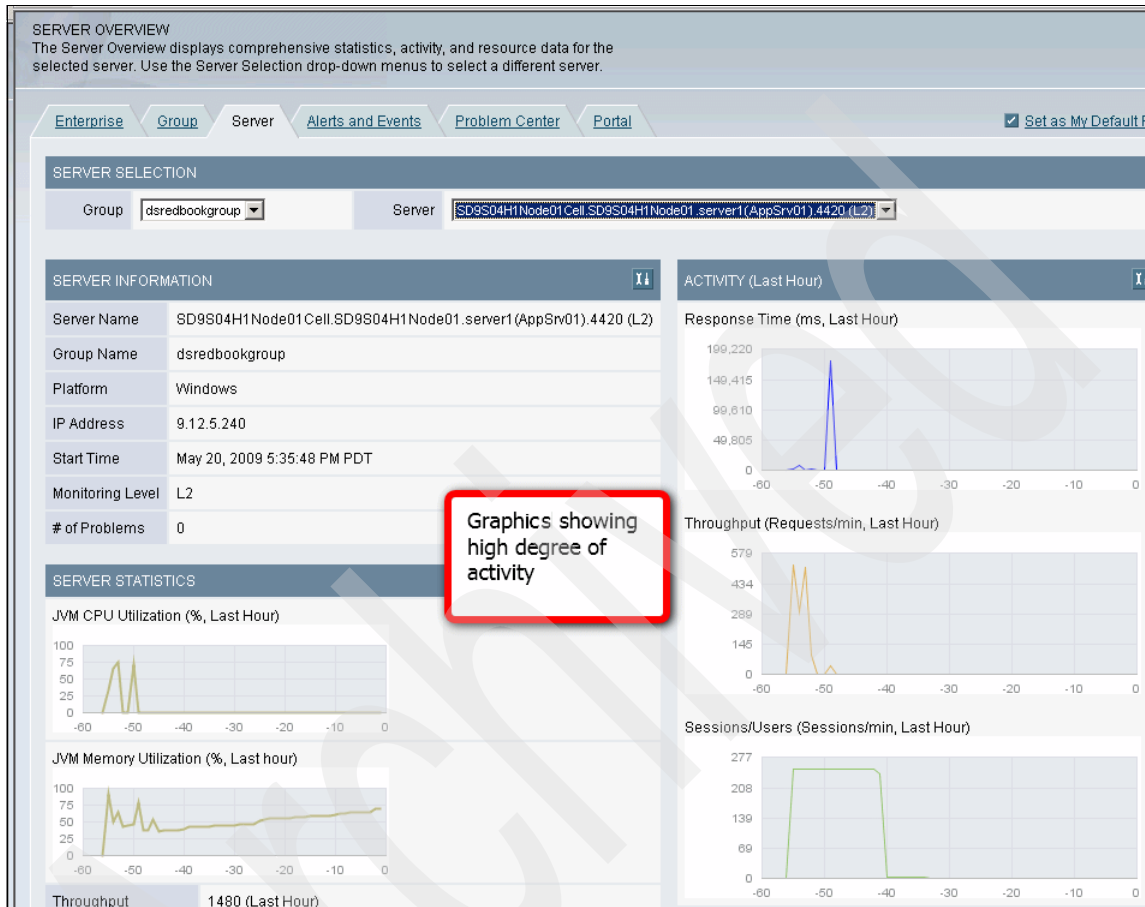


Figure 4-25 High activity as shown on ITCAM

To test our scenario, we ramped up users and their think time. Figure 4-26 shows the activities with 100 users at two seconds ramp time.

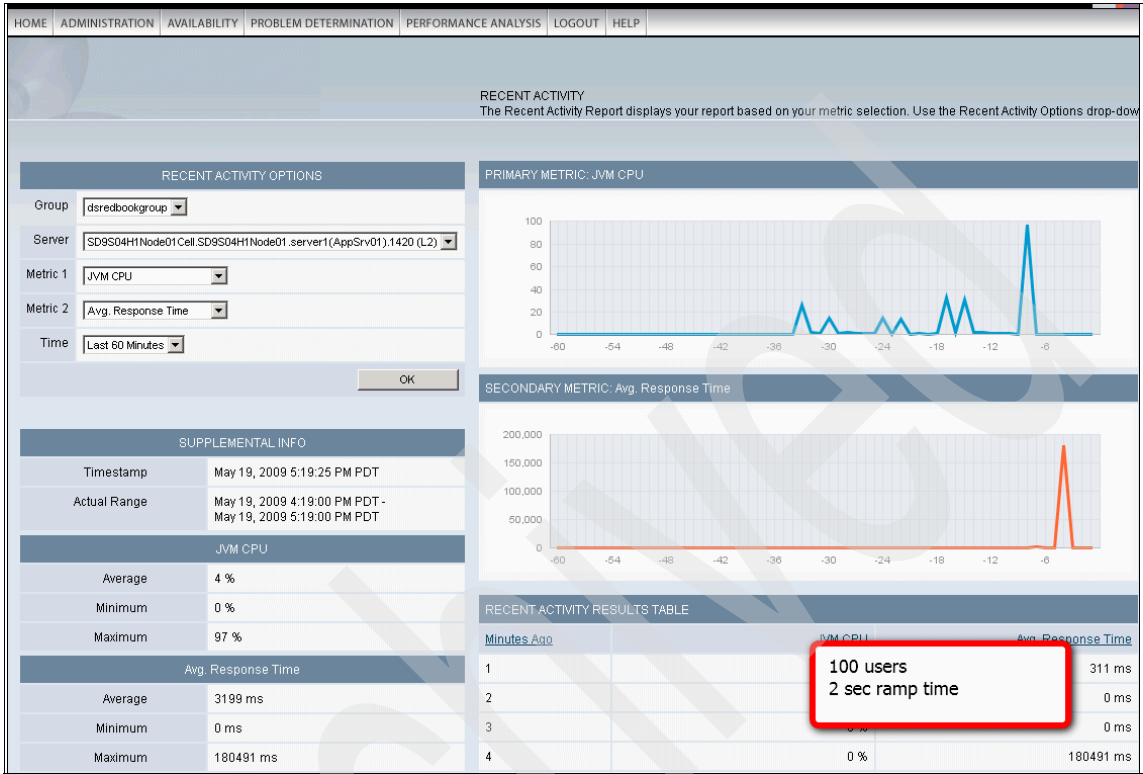


Figure 4-26 ITCAM status with 100 users at 2 seconds ramp time

While the test loads are running, we observe that we have relatively high CPU usage, as shown in Figure 4-27.

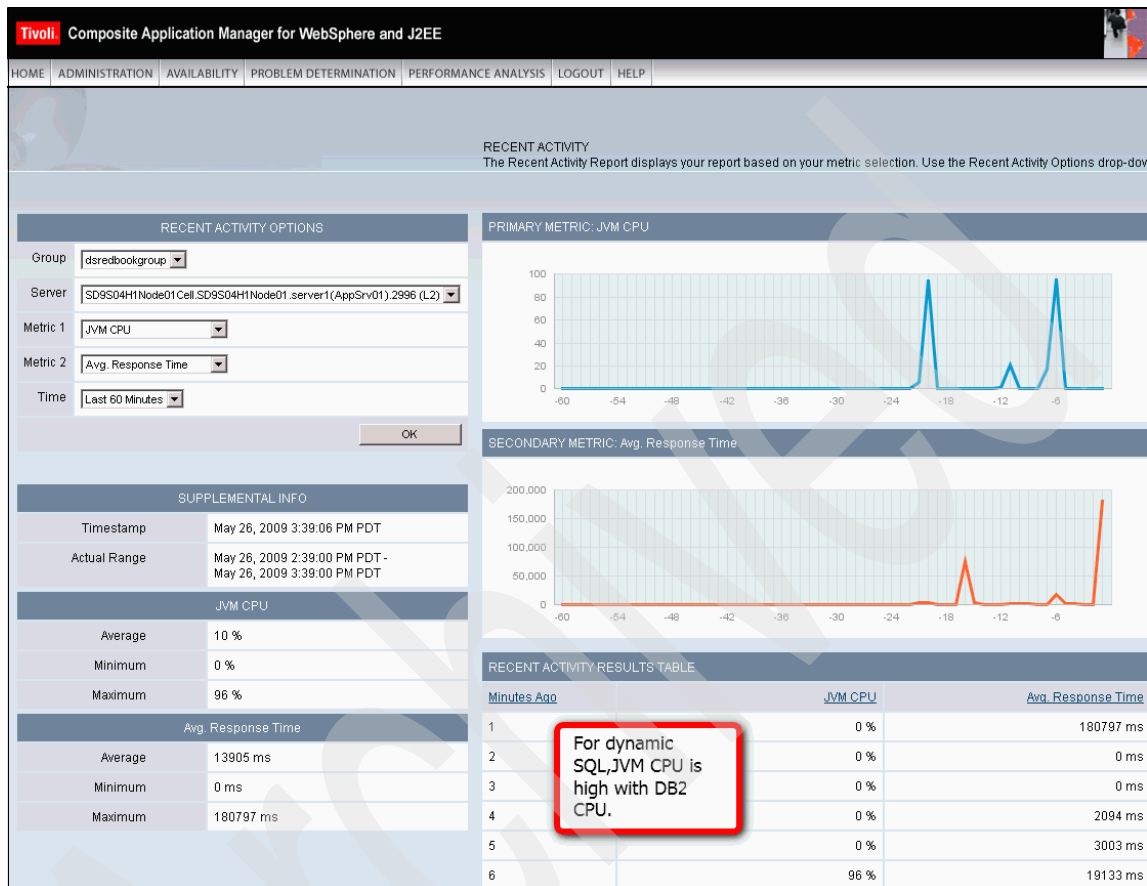


Figure 4-27 High JVM CPU usage

Other unusual values we see are high use time and the Free Pool Size zero. (Figure 4-28). This gives us a clue to investigate the dynamic threads hitting the DB2 on z/OS server.

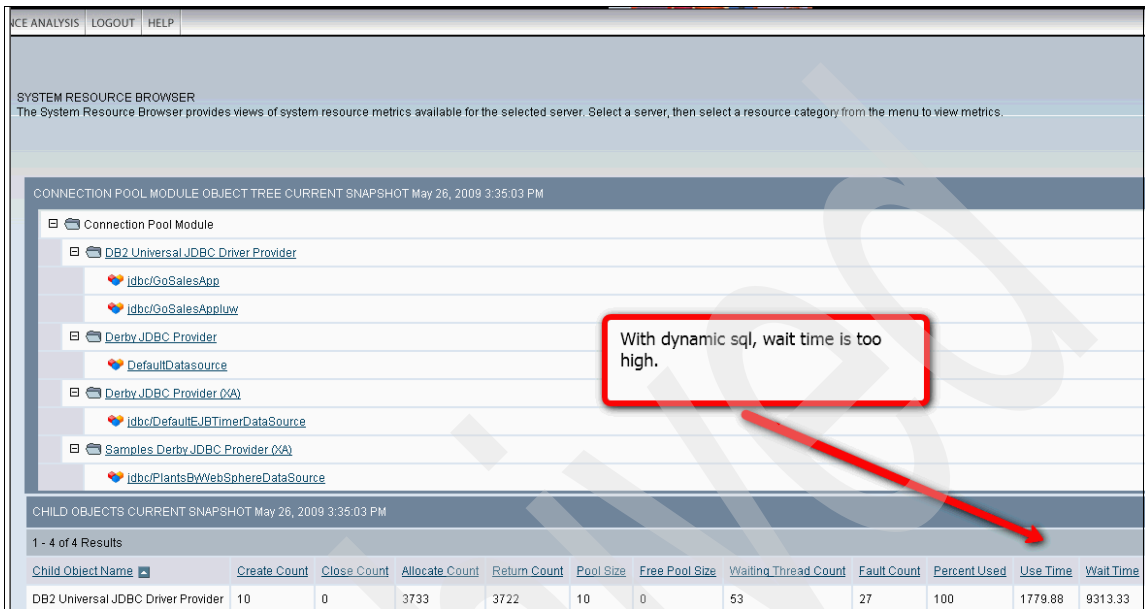


Figure 4-28 Connection pool status

We monitor the OMPE thread output. We further synchronized high CPU usage on WebSphere Application Server and threads of DB2 for z/OS. We timed high JVM/CPU usage with OMPE thread CPU usage. This correlation helps us conclude that the dynamic SQL is the problem area that can be improved.

As our Web application is a pureQuery application, we captured the dynamic SQL and redeployed as the static package to the database as described in Chapter 2, “Optimizing existing Java applications” on page 19.

When the static packages were deployed, we observed that the CPU usage subsides, as shown in Figure 4-29.

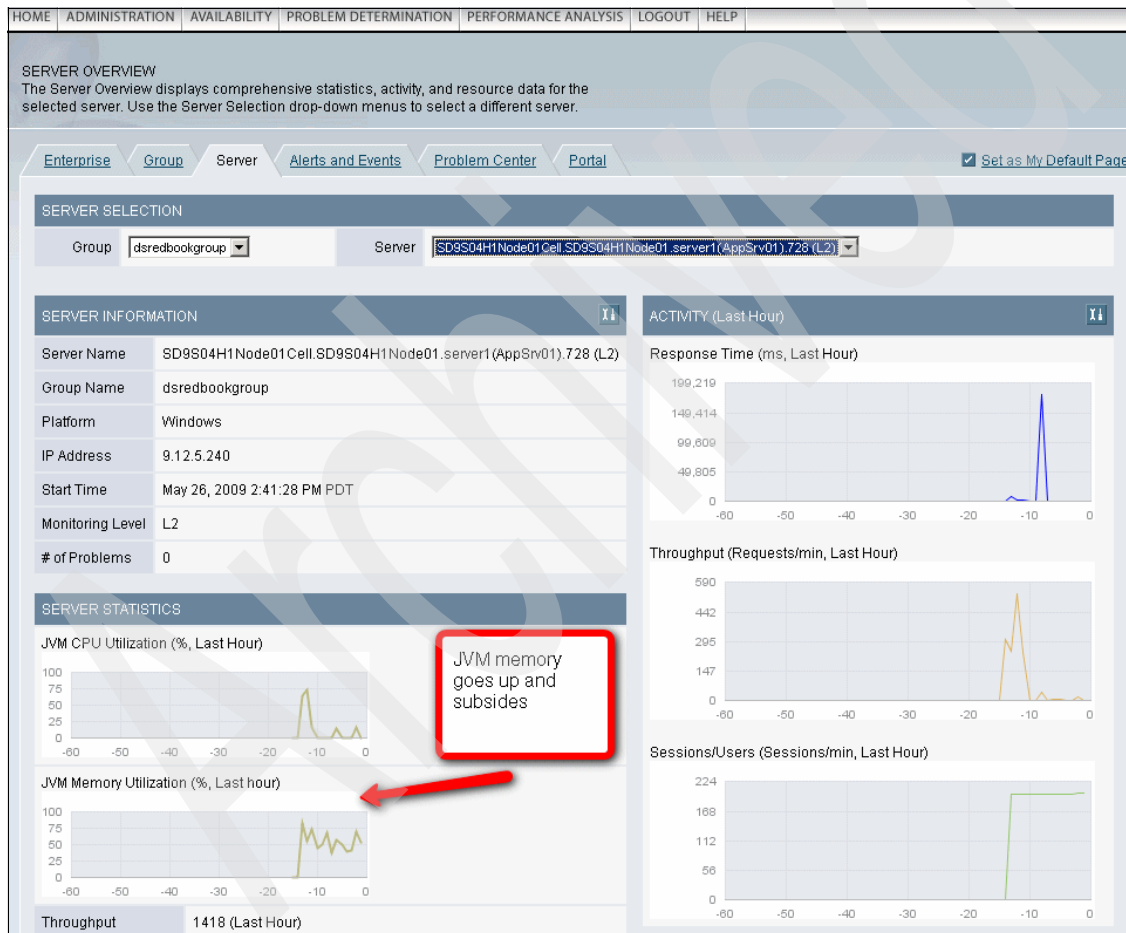


Figure 4-29 Observe how JVM subsides with deployment of static packages

Using the static SQL, the connection pool performance also improved, as there are more connections free. This is shown in Figure 4-30.

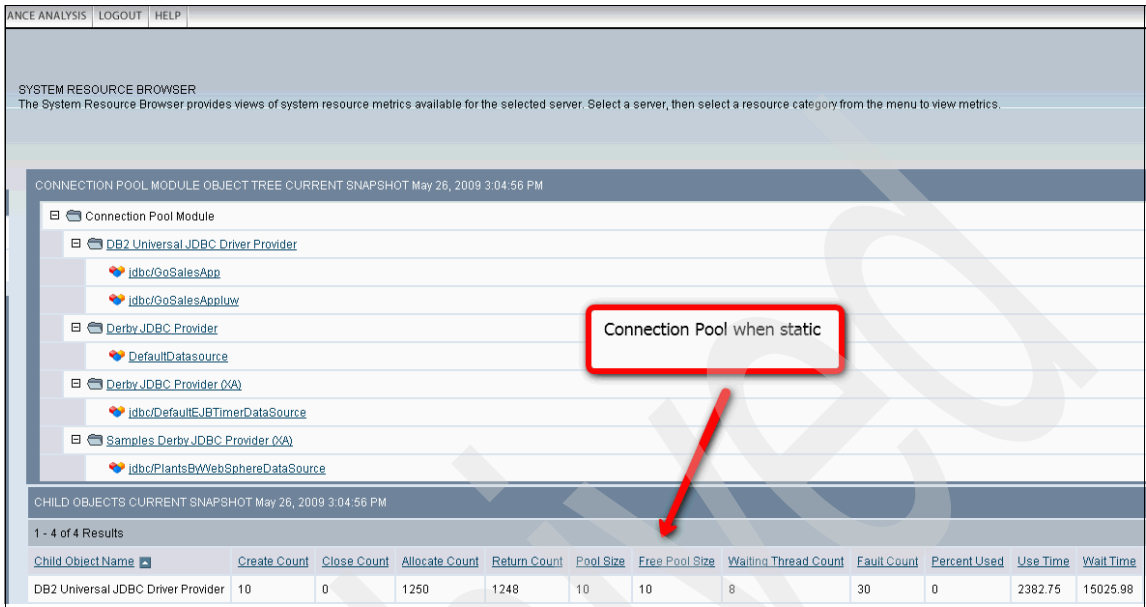


Figure 4-30 Connection pool status with the static SQL

By using ITCAM's response time feature and correlating with online/live sessions, you can diagnose the problems and optimize the query performance that often seems to be SLA detractor. ITCAM can identify hot spots. In our scenario, response time improves because we are taking advantage of pureQuery and static SQL. This enriches your SLA with the combination of ITCAM and pureQuery.

Optim Development Studio and Optim pureQuery Runtime: A closer look at version 2.2

As an enterprise your company likely relies on database technology from IBM (DB2 for Linux, Windows, and UNIX, DB2 for z/OS, DB2 for i, and Informix Dynamic Server) and other vendors (such as Oracle). Release 2.1 of Data Studio Developer and the pureQuery Runtime support the major IBM Data Servers, DB2, and Informix. With the availability of Version 2.2, Java applications that access Oracle databases can benefit from the pureQuery platform as well.

In June of 2009, new versions of Data Studio Developer and the pureQuery Runtime were released under the Optim name:

<http://www.ibm.com/developerworks/data/products/devstudio/>

Data Studio Developer is now Optim Development Studio (ODS) and Data Studio pureQuery Runtime is now Optim pureQuery Runtime. In this chapter, we refer to both products by their new names.

Although the Data Studio name lives on as the basic administration and development tool that you can use at no additional charge with your IBM Data Server license, it is with Optim Development Studio that you get the pureQuery capabilities that we describe in this IBM Redbooks publication.

5.1 Scenario description

In previous chapters we describe how the fictitious company Great Outdoor Company makes use of parts of the pureQuery platform (along with other Integrated Data Management capabilities) to meet their service level agreements for their distributed and z/OS DB2 systems. Without a doubt, providing existing and new Java applications with the ability to process SQL statically provides significant benefits. But the value of the pureQuery platform extends beyond this capability, which makes it not only relevant to applications that access DB2 data servers, but to any application that accesses a supported database.

In this chapter, we review how some of the new features in combination with existing features can be used to gain the following benefits for Java applications that access DB2, IDS and Oracle databases:

- ▶ Reduce resource consumption
- ▶ Improve Java data access performance
- ▶ Streamline the process for new data access development
- ▶ Improve security

Notations and terminology

Throughout this chapter we are using the notation depicted in Example 5-1 to define a pureQuery Runtime configuration. Note that line feeds between configuration properties are only shown to improve readability. They should therefore be omitted.

Example 5-1 Formatted pureQuery Runtime configuration

```
pdqProperties=propertyname1(value1),  
propertyname2(value2),  
propertyname3(value3)
```

Note: There are different means to configure the pureQuery Runtime, depending on the scope (local or global) to which the configuration settings apply. Refer to the Information Center at the following Web page for alternative notations:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.static.doc/topics/rpdqprfhowsetprp.html>

Note that even if you use another notation, the property names and values listed in the examples remain the same.

Path references are shown as /path/to/ and should be replaced with valid and existing path names, such as c:\pureQueryConfigurations\captureFiles\ on Windows.

Some features of the pureQuery Runtime use an external file to store and process SQL statement information. Throughout this chapter we refer to this file as the SQL metadata file or capture file. The official documentation refers to it as the pureQueryXml file.

5.2 Improving database access performance for Java applications

In this section we discuss how the 2.2 releases of Optim Development Studio and Optim pureQuery Runtime can improve database access performance for Java applications.

5.2.1 SQL literal substitution

Database access performance of applications that process SQL dynamically can depend significantly on the database systems SQL statement cache hit ratio. Before an SQL statement can be executed by the database system, it has to be compiled by the database engine. In other words, an access plan has to be created.

Database systems, such as DB2 and Oracle, provide caching mechanisms that temporarily store SQL statements and their access plans in memory, avoiding the need to determine the access plan each time an application submits identical SQL statements. A well-tuned cache can significantly reduce the number of times an access plan has to be determined for identical SQL, reducing CPU usage and improving query response times. (Static SQL processing, on the other

hand, typically does not require this compilation step because the SQL access plan is predetermined and stored in the database package at bind time.)

Since the SQL statement cache size is limited and more SQL is typically processed than can remain cached, SQL statements are sooner or later removed from the cache according to the caching strategy. Some suboptimal application coding practices may therefore result in poor cache performance, resulting in decreased application performance.

One such coding practice is the (sometimes) unnecessary use of string literals of otherwise identical SQL statements. Let us take a look at example queries that retrieve customer information to illustrate a key benefit of the pureQuery Runtime. Example 5-2 shows a set of SQL that retrieves information about a particular customer using literal values to identify the customer.

Example 5-2 SQL SELECT statements

```
SELECT * FROM GOSALESC.T.CUST WHERE CUST_CODE = 22307
SELECT * FROM GOSALESC.T.CUST WHERE CUST_CODE = 95123
SELECT * FROM GOSALESC.T.CUST WHERE CUST_CODE = 18107
```

Assuming that these queries have been issued by an application that processes customer data, each SQL is treated by the database as distinct. Unless the exact query has been processed before and is therefore present in the cache, a new access plan has to be determined for each query. Each instance of these compiled SQL statements might displace another statement from the cache, potentially affecting not only this application but others as well.

On the other hand, if the application had issued SQL containing a parameter marker to retrieve customer information, only one SQL statement would require compilation and would likely remain in the cache if it is used frequently. Example 5-3 shows a SQL statement that retrieves information about a particular customer using a parameter marker.

Example 5-3 SQL statement with a parameter maker

```
SELECT * FROM GOSALESC.T.CUST WHERE CUST_CODE = ?
```

With the pureQuery platform, it is possible, without changing any application code, to consolidate all the SQL statements shown in Example 5-2 to the single statement shown in Example 5-3, which can significantly improve the ability to find the statement in the cache in applications that make liberal use of literals.

The literal substitution feature can also provide significant benefits to applications that process SQL statically using the client optimization feature. This is because by consolidating many different statements into one, fewer SQL statements have

to be captured. This has a direct impact on resource consumption and may further improve performance, because a smaller capture file can offer the following benefits:

- ▶ Reduce the number of database packages that have to be created and maintained.
- ▶ Reduce the pureQuery-related memory and CPU consumption on the application node.
- ▶ Reduce the Optim Development Studio resource usage during SQL metadata file-related processing.

In the following sections we discuss how the literal substitution feature, which was introduced in Version 2.2 of the pureQuery Runtime, can be enabled for basic and enhanced SQL processing. First, let us clarify what these terms mean:

- ▶ Basic (dynamic) SQL processing mode

In basic (dynamic) SQL processing mode, the Runtime passes the SQL through to the database and does not perform any (relevant) pre-processing and postprocessing tasks.

- ▶ Enhanced SQL processing mode

In enhanced SQL processing mode, the pureQuery Runtime can perform the following tasks:

- Execute SQL statically.
- Prevent dynamic SQL execution.
- Limit which SQL can be executed dynamically (whitelist processing).
- Execute replacement SQL instead of the original SQL (SQL replacement).

To take advantage of enhanced processing, parts (or all) of the client optimization process have to be completed, because it relies on the presence of the SQL metadata file.

The general literal substitution enablement process is similar for both modes and differs only in a few details.

Enabling literal substitution for dynamic SQL processing

In Version 2.2 the Runtime performs literal substitution only if enhanced processing is enabled. To take advantage of literal substitution even if no enhanced SQL processing is desired, you must enable one of the enhanced processing features. For the use case in this section, the SQL replacement feature is the ideal candidate, because it can be configured to perform its processing without any side effects on the SQL that is to be executed.

There are two steps that need to be completed to perform literal substitution:

1. Create a dummy SQL metadata file containing at least one SQL statement. The presence of this file is required by the SQL replacement feature.
2. Configure the pureQuery Runtime environment to replace literal values and enable SQL replacement.

The details of how the SQL replacement feature works is discussed in section 5.2.2, “Improve query performance using SQL replacement” on page 163.

The dummy SQL metadata file can be created by configuring the Runtime to capture the applications SQL, as shown in Example 5-5. Note that no literal substitution needs to be performed.

Example 5-4 Enabling SQL capturing to create a dummy SQL metadata file

```
pdqProperties=executionMode(DYNAMIC),  
captureMode(ON),  
pureQueryXml (/path/to/dummy.pdqxml)
```

You can capture any number of the applications SQL statements, but a single one is sufficient. It is irrelevant which SQL is captured, because no actual enhanced processing will be performed. If you have captured more than one SQL, you can use SQL metadata file editor in Optim Development Studio to remove the others.

Since only dynamic SQL execution is desired, no database package mapping has to be defined for this metadata file.

Once the dummy metadata file is created, change the pureQuery Runtime environment configuration as shown in Example 5-5. The literal substitution feature, which is disabled by default, is enabled by setting property `sqlLiteralSubstitution` to `ENABLE`. Property `enableDynamicSQLReplacement` is set to `TRUE`, enabling enhanced SQL processing.

Example 5-5 Enabling literal substitution for dynamic SQL execution

```
pdqProperties=executionMode(DYNAMIC),  
pureQueryXml (/path/to/dummy.pdqxml),  
sqlLiteralSubstitution(ENABLE),  
enableDynamicSQLReplacement(true)
```

If configured as shown in the example, the pureQuery Runtime validates whether literals in an SQL statement can be substituted by preparing a derived (consolidated) SQL statement that contains parameter markers instead of the

original literals. If the derived SQL can be successfully prepared, the literals in the original SQL can be substituted and the SQL is executed as though it contained parameter markers.

We suggest enabling the JDBC driver's internal cache to improve performance. Without this caching, the JDBC driver would have to send an avoidable prepare request to the database each time the same derived SQL statement is processed by the Runtime. Figure 5-1 illustrates a common scenario for an application that processes three variations of the same SQL containing different literals. With JDBC driver caching enabled, only one prepare request is sent to the database. Subsequent requests are unnecessary if the statement is still present in the cache. Without caching, three identical prepare requests would have to be processed.

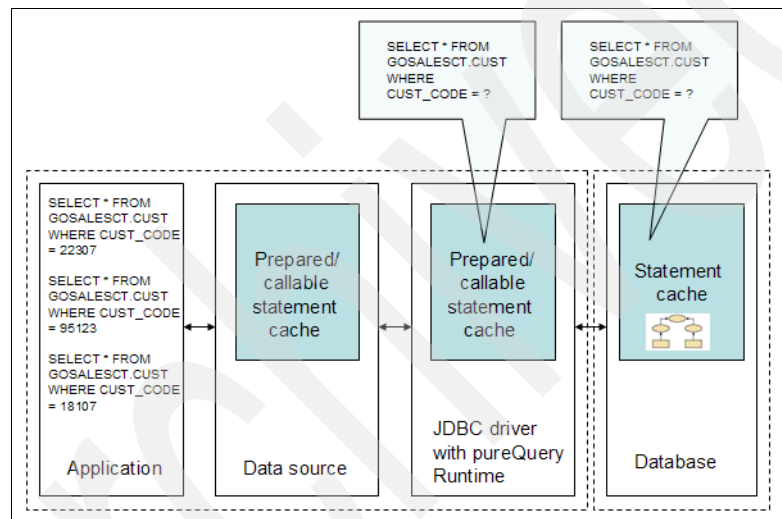


Figure 5-1 Improving literal replacement performance through caching

JDBC driver caching can be enabled for applications that access the database using a connection pool data source object, by defining property `maxStatements` and assigning a numeric value to it, which configures the cache size.

Note: If an application runs in an application server, such as WebSphere Application Server, JDBC statement caching is configured by defining custom property `maxStatements` for data sources that use the IBM Data Server Driver for JDBC and SQLJ. Refer to the product documentation for details.

The main difference between how literal substitution is enabled for basic and enhanced processing is the pureQuery Runtime environment configuration that is used. As you have seen in this section, while the dummy metadata was captured, literal replacement does not have to be enabled. Let us take a look at the more common enhanced SQL processing scenario.

Enabling literal substitution during enhanced SQL processing

In a typical scenario, the pureQuery Runtime environment is configured to take advantage of enhanced SQL processing, because it provides the main pureQuery benefits including:

- ▶ Execute SQL statically.
- ▶ Prevent dynamic SQL execution.
- ▶ Limit which SQL can be executed dynamically (“whitelist processing”).
- ▶ Execute replacement SQL instead of the original SQL (“SQL replacement”).

If literal substitution is to be performed in conjunction with enhanced processing, the `sqlLiteralSubstitution` property has to be defined during the capture and the execution phase of the client optimization process, which is described in Chapter 2, “Optimizing existing Java applications” on page 19.

To turn on literal substitution (it is turned off by default) during capture, set the `sqlLiteralSubstitution` property in the pureQuery Runtime configuration to `ENABLE`, as shown in Example 5-6.

Example 5-6 Enabling literal substitution during SQL capture

```
pdqProperties=executionMode(DYNAMIC),
captureMode(ON),
pureQueryXml(/path/to/anna.pdqxml),
sqlLiteralSubstitution(ENABLE)
```

The literal replaced SQL is stored in the SQL metadata file instead of the original SQL.

Note: As described earlier, it is sufficient to capture a single instance of an SQL statement, because the literal values are replaced by parameter markers. Without literal substitution enabled, all instances of otherwise identical SQL would have to be captured, increasing resource usage.

Figure 5-2 on page 161 depicts a snapshot of an example capture file, which is visualized using the SQL Outline view (referred to as pureQuery Outline in Version 2.1) in Optim Development Studio. The literal value in the first SELECT statement was replaced with a parameter marker.

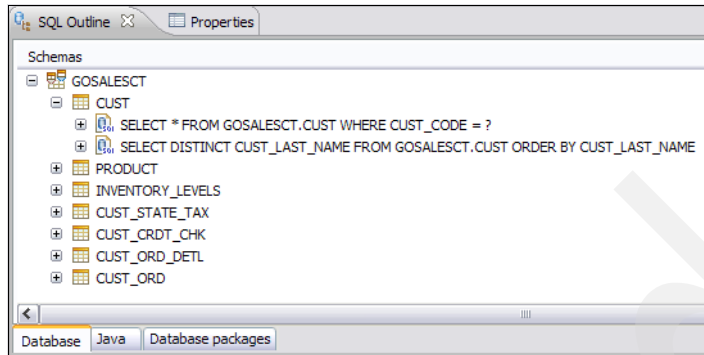


Figure 5-2 Visualizing consolidated SQL statements using the SQL Outline view

During enhanced SQL processing (or in other words, during the execution phase of the client optimization process), the literal replaced SQL is processed instead of the original one under the following conditions:

- ▶ Property `sqlLiteralSubstitution` is set to `ENABLE`
- ▶ Property `sqlLiteralSubstitution` is set to `NOT_SET` or not specified but was set during the last capture activity which created the SQL metadata file that is being used

Example 5-7 depicts a pureQuery Runtime configuration in which literal substitution is enabled during static SQL processing.

Example 5-7 Explicitly enabling literal substitution during static SQL processing

```
pdqProperties=captureMode(OFF),
executionMode(STATIC),
pureQueryXml(/path/to/anna.pdqxml),
sqlLiteralSubstitution(ENABLE)
```

If the feature is enabled, it replaces in the incoming SQL literal values with parameter markers and matches this statement with the information that was previously stored in the SQL metadata file. If a match is found or, if no match was found but the incoming (original) SQL is present in the SQL metadata file, processing is performed according to the current configuration settings.

The following scenarios illustrate two common use cases:

- ▶ If the Runtime is configured to run SQL statically and replace literals, the incoming SQL is only executed statically if its literal replaced version or the incoming SQL (containing literals) is present in the SQL metadata file. If no match is found, the literal replaced SQL is executed dynamically if the Runtime property `allowDynamicSQL` is set to `TRUE`. If neither of the conditions are met, the SQL is not executed.
- ▶ If whitelist processing for dynamic SQL execution is enabled (that is, the Runtime configuration property `capturedOnly` is set to `true`) in combination with literal substitution, the incoming SQL is executed only if its literal replaced equivalent (or, if no equivalent is found the incoming SQL) is present in the SQL metadata file.

Note: To improve performance, the JDBC drivers internal cache should be enabled as outlined in the previous section.

Literal substitution considerations

pureQuery string literal processing is performed exclusively on the application node and is transparent to the database. Even though some overhead may be observed on the application node, the database node is not impacted, making this a good solution in environments where resource usage on the database node is costlier than resource usage on the application node.

Not every literal that is present in an SQL statement can be replaced with a parameter marker. Generally speaking, there are several scenarios in which literals are not replaced even if the feature is enabled:

- ▶ If an SQL statement contains literals and parameter markers, no substitution is performed. If a developer made a conscious decision to use both literals and parameter markers, it is not desirable to alter the SQL.
- ▶ Literals that are used in a certain context may not be replaced, depending on the pureQuery Runtime version and the database being accessed. For example a certain literal might be replaceable if the SQL is executed against a DB2 for LUW system but not replaceable if it is executed against DB2 for z/OS.

For more details, refer to the Integrated Data Management Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.javatool.static.doc/topics/rpdqprfsynpdqcapdcrsqllitsub.html>

The pureQuery literal substitution feature does not require any database vendor-specific functionality, making it a good alternative for applications that access a data server that does not provide natively similar functionality.

Some database servers provide similar substitution capabilities or may provide them in a future release, raising the question whether client-based technology provides a viable alternative. It is beyond the scope of this book to perform an in-depth analysis, so we provide you with a core set of questions that will help you make a decision that is appropriate for your situation:

- ▶ What is the primary motivation that prompts you to consider literal substitution? Would you use it to improve performance of dynamic SQL or to enable you to leverage static SQL? Native support by the database server does not enable a broader class of applications for static SQL processing, unlike the pureQuery Runtime feature.
- ▶ Is the application accessing a data server that natively supports this capability? At the time of writing, only DB2 for LUW version 9.7 provides serverside literal substitution functionality. Applications that access older versions of DB2 for z/OS cannot take advantage of this capability.
- ▶ Is it desirable to limit the scope of SQL substitutions to individual applications or sets of statements? SQL performance may differ significantly for SQL that contains specific literal values as opposed to parameter markers. In some scenarios it may therefore be desirable to retain the original statements and not perform literal consolidation. Using the pureQuery feature, literal substitution can be configured at a more granular level.
- ▶ The savings potential that is achieved by performing SQL literal substitution could be partially offset by the costs associated with the substitution activity. Is it acceptable to incur these costs on the database node or is it preferable to perform this operation on the application node? Depending on where processing is more costly, one approach might be better than the other.

Thus far we have focused on the pureQuery ability to replace literals within SQL statements. While this feature may address some problems, let us look at another capability that enables you to replace entire SQL statements that an application processes.

5.2.2 Improve query performance using SQL replacement

Data access performance issues can surface throughout an application's active life cycle: during development, in test, or in production. If these issues are caused by poorly performing SQL that was either hand-coded or automatically generated (for example, by an object-relational mapping (ORM) persistence framework, such as Hibernate), it may not always be possible to tune the database well enough to remedy the situation. Changing the application's SQL is typically not

an option once a project has moved past the development phase and the application has been rolled out, or if the SQL is generated on the fly by a framework.

The pureQuery Runtime introduced in version 2.1 a feature that can be used to (temporarily, or if necessary permanently) override SQL statements that an application processes without the need to change the applications source code or even having access to the source code.

To take advantage of this feature, you must complete the pureQuery client optimization process:

- ▶ Capture the SQL should be replaced.
- ▶ Replace the SQL using the SQL metadata file editor in Optim Development Studio.
- ▶ Optionally, if static SQL execution is desired and supported by the database, bind the modified SQL to a package.
- ▶ Enable the SQL substitution feature in the pureQuery Runtime configuration.

Since we have already discussed the general process at length in Chapter 2, “Optimizing existing Java applications” on page 19, we focus now on illustrating the key tasks.

Let us assume we have identified a particular SQL statement that performs sub-optimally and should therefore be replaced with a better performing one or augmented with optimizer hints. For illustrative purposes we are using an Oracle application and will add an SQL hint to achieve better performance. The general tasks being demonstrated are the same for all supported databases.

With the help of the application owner (who could be a developer, quality assurance professional, or systems administrator) we have installed the pureQuery Runtime, configured it to capture the application's SQL and executed the relevant use cases that cause the application to process the desired SQL.

Example 5-8 depicts a typical pureQuery Runtime configuration that could be used for this purpose.

Example 5-8 Capturing SQL that requires optimization

```
pdqProperties=executionMode(DYNAMIC),
pureQueryXml (/path/to/capturedSQL.pdqxml),
captureMode(ON)
```

Having captured the SQL, you can import it into Optim Development Studio where you designate the replacement SQL:

- ▶ Transfer the capture file in binary mode to a machine where Optim Development Studio is installed.
- ▶ Create a new Java project and enable pureQuery support (right-click the project name and select **pureQuery** → **Add pureQuery support**). See Figure 5-3.

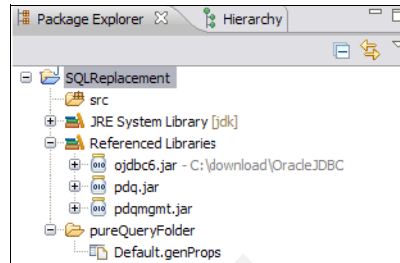


Figure 5-3 Example of a pureQuery enabled project in Optim Development Studio

- ▶ Use the import wizard (**File** → **Import** → **General** → **File System**) to import the capture file into the pureQueryFolder subdirectory of the Java project. See Figure 5-4.

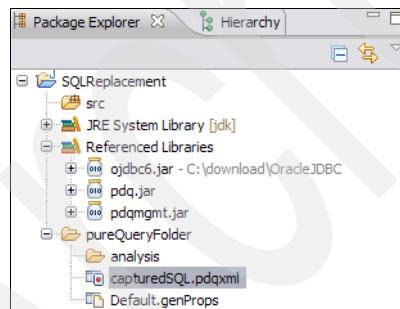


Figure 5-4 Example of an imported SQL capture file

Note: If you are working with a capture file that was created using an application that accesses a DB2 data server, you will have to assign a package identifier in configuration file Default.genProps to this capture file in order to support static SQL execution. For Oracle applications or applications that execute SQL only dynamically no association has to be established.

Upon completion of these steps you are now ready to edit the capture file using the capture file editor (Figure 5-5). The type of capture file (DB2, Oracle, IDS) that you are working with determines which editor features are enabled. For example, the DB2 capture file editor provides features specific to database package management.

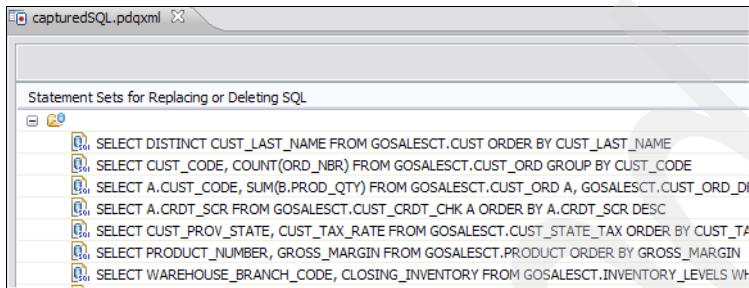


Figure 5-5 Browsing an SQL capture file using the capture file editor

The capture file likely contains SQL statements that are irrelevant to the task at hand, which is to replace problematic SQL. Most editor tasks can be invoked by right-clicking anywhere in the editor. To invoke a task that is specific to a single SQL statement, select the statement first and right-click to open a context-sensitive menu. Figure 5-6 depicts the SQL context menu that provides access to the Delete Statement task, which removes SQL from the capture file.

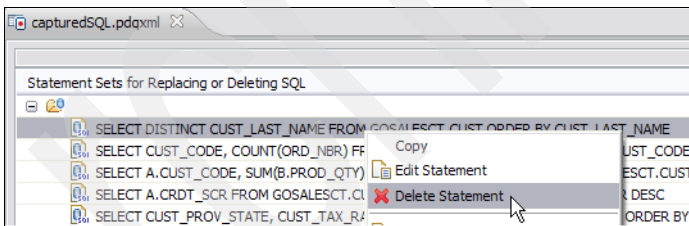


Figure 5-6 Deleting SQL from a capture file

Figure 5-7, Figure 5-8, and Figure 5-9 illustrate the steps that you must complete to replace an SQL statement.

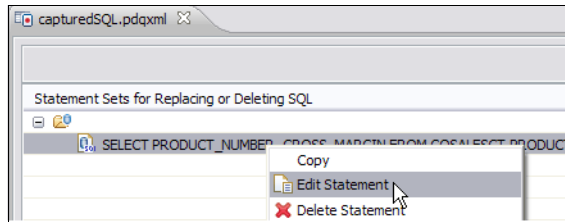


Figure 5-7 Invoking the SQL editor

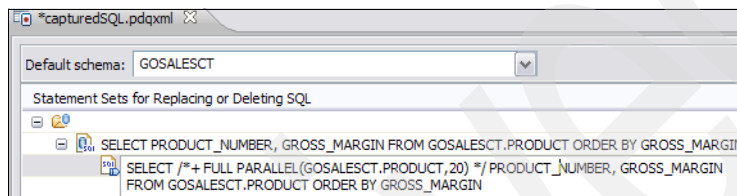


Figure 5-8 Providing Oracle hints using the statement editor

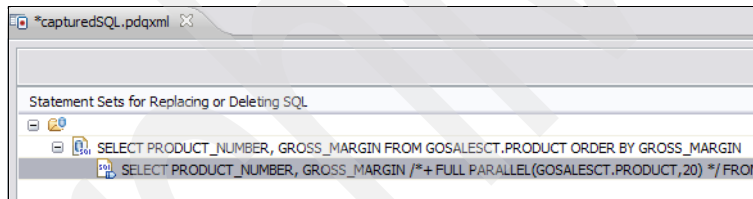


Figure 5-9 Reviewing the modified SQL capture file

The designated replacement SQL has to be equivalent to the original SQL, meaning it has to be of the same type (for example, an INSERT statement cannot be replaced with a SELECT statement) and it has to contain the same number and types of input parameters and result columns.

The following examples illustrate invalid replacement SQL designations:

► Example 1: Adding result columns

If an application issues SQL, it expects the result set to contain certain information, such as a fixed number of columns. Without a change to the application, additional columns would likely not be processed or might be processed incorrectly. Assume that the original SQL reads as follows:

```
SELECT PRODUCT_NUMBER, GROSS_MARGIN  
FROM GOSALESC.T.PRODUCT
```

The replacement SQL cannot be rewritten to return additional columns

```
SELECT PRODUCT_NUMBER, GROSS_MARGIN, PRODUCT_ORIGIN  
FROM GOSALESC.T.PRODUCT
```

► Example 2: Adding input parameters

This change is not permitted because an application would not provide any input values for additional parameters that were not present in the original SQL. Assuming the original SQL reads as follows:

```
SELECT PRODUCT_NUMBER, GROSS_MARGIN  
FROM GOSALESC.T.PRODUCT
```

The replacement SQL cannot be rewritten to restrict the result set using a parameter because the Runtime would not know which value to assign to the parameter

```
SELECT PRODUCT_NUMBER, GROSS_MARGIN  
FROM GOSALESC.T.PRODUCT  
WHERE GROSS_MARGIN > ?
```

Note: No semantic checking is performed. It is therefore possible to designate substitution SQL that, for example, returns a subset or superset of the data that would be returned by the original query. For example:

```
SELECT PRODUCT_NUMBER, GROSS_MARGIN  
FROM GOSALESC.T.PRODUCT
```

can be replaced with

```
SELECT PRODUCT_NUMBER, GROSS_MARGIN  
FROM GOSALESC.T.PRODUCT  
WHERE GROSS_MARGIN > 20
```

Permitted changes include, but are not limited to, modifications that affect the following elements:

- ▶ The access path that the optimizer would choose (for example, ORDER BY clause, OPTIMIZE FOR clause, optimizer hints for Oracle like shown in an earlier example).
- ▶ The queries fetch size (for example, adding FETCH FIRST clause, additional predicates that limit the result set).
- ▶ The locking behavior (for example, adding WITH ISOLATION clause, SKIP LOCKED DATA clause, FOR UPDATE clause).
- ▶ Which database objects are used (for example, changing the schema qualifier for a view or table).
- ▶ Use of the PLAN table (for example, adding QUERYNO clause).

Having assigned replacement SQL using the editor, save the changes (**File** → **Save**) and export the modified capture file using the export wizard (**File** → **Export** → **General** → **File System**). To make the updated capture file available to the pureQuery Runtime, you must transfer it in binary mode to the application node where the target application is installed and replace the original capture file with the modified version.

Note: If static SQL processing is desired, you will have to bind the content of the updated file to a package, as described in Chapter 2, “Optimizing existing Java applications” on page 19.

To take advantage of the replaced SQL, configure the Runtime to enable the SQL replacement feature, by defining property `enableDynamicSQLReplacement`, as shown in Example 5-9.

Example 5-9 Configuring the pureQuery Runtime to execute substitution SQL

```
pdqProperties=executionMode(DYNAMIC),  
pureQueryXml (/path/to/capturedSQL.pdqml),  
captureMode(OFF),  
enableDynamicSQLReplacement(TRUE)
```

Note: If static SQL processing is enabled, the substitution SQL is always executed instead of the original SQL, because the original SQL is not stored in the associated database package.

We have demonstrated how the pureQuery Runtime and Optim Development Studio provide you with the ability to use optimized SQL. Optim Development Studio includes Visual Explain and can therefore be used to analyze how DB2 or Oracle process a particular query. It is important though to understand that no automatic query rewriting is performed and no tuning advice is given. You can take advantage of other products, such as Optim Query Tuner, which was previously available as Optimization Expert, to obtain advice on how to optimize queries or the database to achieve better performance.

Optimizing DB2 queries

Optim Development Studio provides seamless integration with Optim Query Tuner for DB2 for Linux, UNIX, and Windows, Optim Query Tuner for DB2 for z/OS, and Optim Query Workload Tuner for DB2 for z/OS. If either product is co-installed (commonly referred to as shell-shared) with Optim Development Studio, you can take advantage of their powerful DB2 query tuning capabilities by invoking them directly from the capture file editor's context menu, as shown in Figure 5-10.

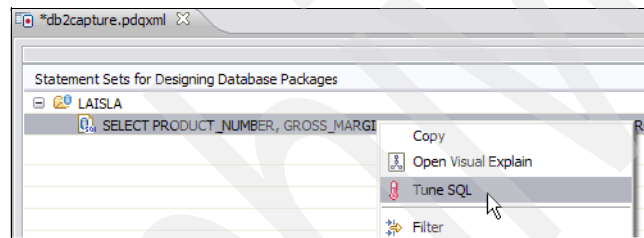


Figure 5-10 Invoking query tuning from the capture file editor in ODS

The Query Tuner products provide an Index Advisor, a Query Advisor, a Statistics Advisor, and an Access Path advisor along with advanced query tools. For introductory information about tuning and visualization capabilities along with product integration aspects, refer to the developerWorks article *SQL Tuning: Not just for hardcore DBAs anymore* at the following Web page:

http://www.ibm.com/developerworks/data/library/dmmag/DMMag_2009_Issue2/DataStudio/

For in-depth information, refer to *Tuning queries with Optim Query Tuner* in the Information Center at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=/com.ibm.datatools.qrytune.nav.doc/topics/tuningqueries.html>

We have focused on how the pureQuery platform can be used by the DBA to further optimize data access for existing Java applications. In the following section we take a look at how some of the tuning activities can be performed during the development phase of the application life cycle.

Query optimization considerations

While it is always desirable to make the SQL changes in the application itself to improve performance, there may be scenarios where this is not possible. The ability to replace SQL provides you with the ability to put a temporary or (if necessary) permanent fix in place that can improve data access performance.

One can argue that this ability poses a security risk because an unauthorized person might be able to alter a capture file. While this is true, there are safeguards in place that mitigate this risk. First of all, write access to the capture file should be limited at the operating system level, just like for any other protected resource. If one manipulates a capture file (for example by adding replacement SQL), the Runtime will not execute this SQL in dynamic execution mode unless it is explicitly configured to do so (by setting property `enableDynamicSQLReplacement` to `TRUE`). In static execution mode the Runtime cannot process the unauthorized replacement SQL because it is not present in the original database package. One would have to bind the updated capture file first, which requires special database privileges.

5.2.3 Apply SQL development best practices and prevent rogue queries

Application developers are commonly much more experienced in writing efficient source code than SQL that is used to access or manipulate database objects that embody business processes. This causes some challenges because without proper tuning, application performance may be suboptimal. Given that many development environments have little support for SQL development and tuning effort, problematic SQL is usually identified only late in the development cycle when performance tests uncover previously hidden issues.

Optim Development Studio and the pureQuery Runtime provide capabilities that foster a more efficient application development process cycle, which we describe in more detail.

Using the highly optimized pureQuery API, developers can focus on developing business logic instead of having to implement the complex code associated with common SQL processing tasks, such as statement preparation, execution, and result set processing. Source code for common data access and manipulation

tasks can be automatically generated and easily customized. Several built-in SQL development features in Optim Development Studio provide seamless SQL development integration into the development environment.

SQL content assist capabilities, which are similar to Java IDE content assist features that developers are familiar with, lead to more rapid SQL creation and prevent common errors, such as misspelled column or table names or syntax violations. By using custom SQL templates for common query types, developers can benefit from DBA expertise, providing more consistency across application modules and applications.

Figure 5-11 depicts the content assist capabilities of the Java editor in Optim Development Studio, which allows for the creation of SQL statements or the customization of template SQL. Note that the content assist drop-down list contains a list of columns that are present in the customer table.

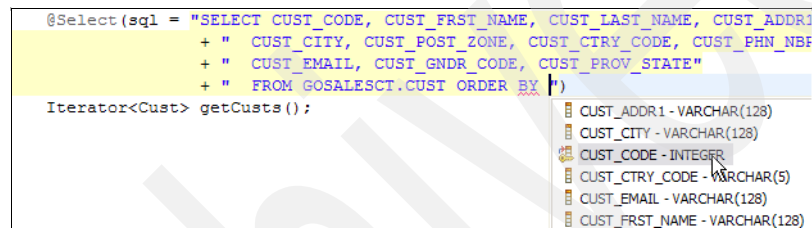


Figure 5-11 Using content assist to develop an SQL statement

Real-time error checking while SQL statements are edited provides instantaneous feedback, eliminating the need to execute the source code to verify that the constructed SQL can be executed successfully.

Figure 5-12 illustrates how a mistyped column name is brought to the developer's attention while the code is being modified.

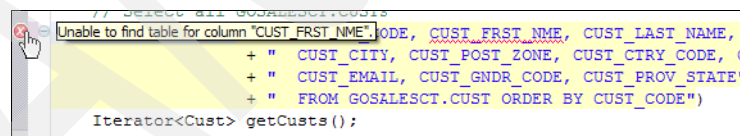


Figure 5-12 Real-time semantic and syntactic SQL error checking

Figure 5-13 displays a context menu of the Java editor that provides access to common SQL development tasks, such as query formatting, execution, metadata lookup and access plan visualization. This function eliminates the need to learn how to use multiple tools.

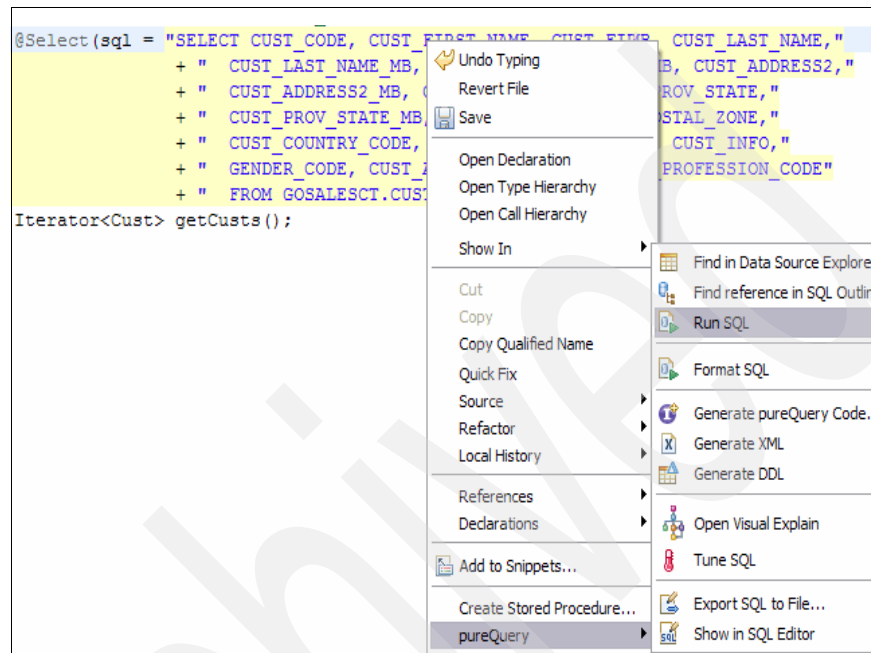


Figure 5-13 Access to common SQL development tasks from the Java editor

Built-in integration with the Optim Query Tuner encourages novice and experienced developers alike to tune queries during development, instead of postponing the task until performance tests are conducted.

Figure 5-14 depicts the output of a query tuning activity in Optim Development Studio. The query access plan along with the advisor recommendations are shown in the query tuning view on the right side. A list of SQL that are embedded in the application is displayed in the lower right. On the lower left, it shows the database objects that are presented in the database that the application is accessing.

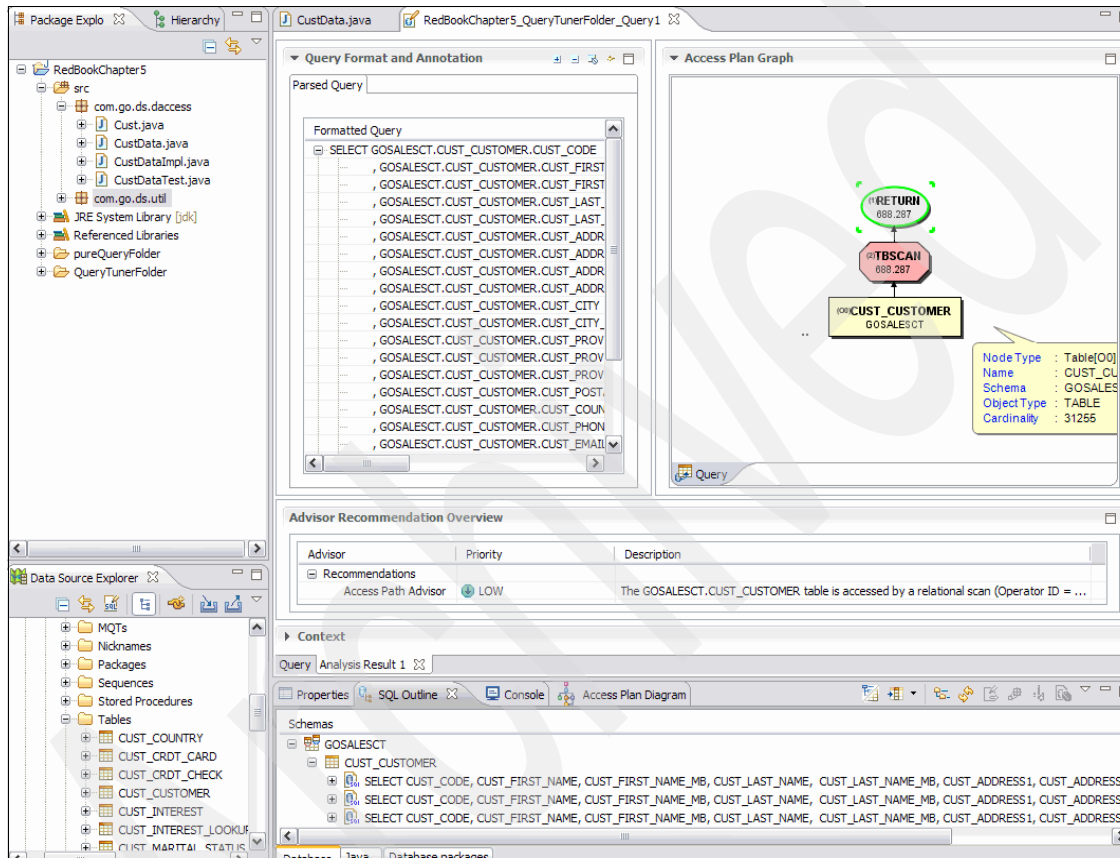


Figure 5-14 A typical query tuning view in ODS

One common performance tuning challenge is to determine which SQL tuning efforts will have the biggest impact on application performance. Optim Development Studio can be used to capture, visualize and compare SQL execution metrics, making it easier to identify high-impact queries that take a long time to process or are executed frequently.

In version 2.2 of the pureQuery Runtime, the following execution metrics are captured for each SQL statement:

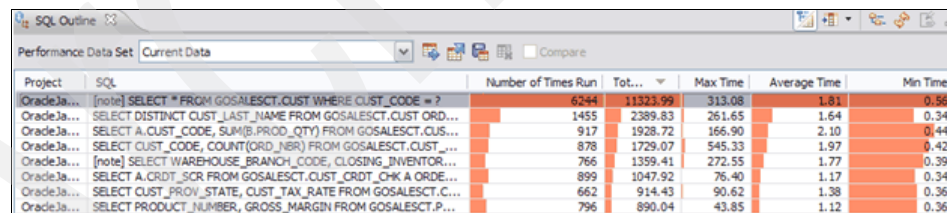
- ▶ Execution count
- ▶ Minimum SQL execution time
- ▶ Maximum SQL execution time
- ▶ Average SQL execution time

Note: These performance metrics are gathered on the client side (the application node) and currently do not include any database internal metrics that traces or other monitoring tools, such as Performance Expert, may capture. Because no database vendor-specific functionality is used to gather these metrics, this feature can be used to identify performance hot-spots in applications that access any supported database.

To enable SQL performance metrics capturing, the application has to be executed using the pureQuery run configuration in Optim Development Studio.

Note: To use this run configuration to capture performance metrics for a stand-alone Java application, select the Java program in the Package Explorer, right-click and, from the context menu, navigate to **Run As** → **Run Configurations** → **PureQuery**.

The SQL outline view visualizes the captured metrics and provides the ability to sort, manage (save, delete, import, export), and compare them, as shown in Figure 5-15.



Project	SQL	Number of Times Run	Tot...	Max Time	Average Time	Min Time
OracleJa...	[note] SELECT * FROM GOSALESC.T.CUST WHERE CUST_CODE = ?	6244	11323.99	313.08	1.81	0.56
OracleJa...	SELECT DISTINCT CUST_LAST_NAME FROM GOSALESC.T.CUST ORD...	1455	2389.83	261.65	1.64	0.34
OracleJa...	SELECT A.CUST_CODE, SUM(B.PROD_QTY) FROM GOSALESC.T.CUS...	917	1928.72	166.90	2.10	0.44
OracleJa...	SELECT CUST_CODE, COUNT(ORD_HER) FROM GOSALESC.T.CUST...	878	1729.07	545.33	1.97	0.42
OracleJa...	[note] SELECT WAREHOUSE_BRANCH_CODE, CLOSING_INVENTOR...	766	1359.41	272.55	1.77	0.39
OracleJa...	SELECT A.CRDT_SCR FROM GOSALESC.T.CUST_CRDT_CHK A ORDE...	899	1047.92	76.40	1.17	0.34
OracleJa...	SELECT CUST_PROV_STATE, CUST_TAX_RATE FROM GOSALESC.T.C...	662	914.43	90.62	1.38	0.36
OracleJa...	SELECT PRODUCT_NUMBER, GROSS_MARGIN FROM GOSALESC.T.P...	796	890.04	43.85	1.12	0.36

Figure 5-15 Identifying application performance hot-spots using SQL performance metrics

Using the grid view, one can identify potential hot-spots by sorting the data by execution count or total elapsed time. By saving and comparing performance metrics, the impact of database tuning efforts are immediately visible.

Note: The DB2 SQL tuning capabilities of the Query Tuner products that were mentioned earlier can be invoked from the SQL Outline view as well, by right-clicking on an SQL statement and selecting **Tune SQL** from the context menu.

We have covered some features in Optim Development Studio that help application developers become more productive in their SQL development tasks, but how can the Runtime support you in your efforts to achieve good application performance with respect to the database? For example, the following two examples are both situations that can cause performance problems or even application breakage:

- ▶ Lack of insights into how an application accesses or modifies database objects delays tuning efforts into late in the project cycle
- ▶ Lack of code reviews and code modifications that are made without appropriate notification of impacted users can break applications and cause delays.

The pureQuery Runtime and Optim Development Studio can be used to foster and enforce some of the SQL development related best practices.

Let us take a look at two example scenarios.

Scenario 1: Preventing execution of unauthorized SQL

Have you heard of the infamous “one last change to the source code” that would not have any impact on the database? Right. It would be helpful if it was possible to lock down the SQL that an application can process and detect any changes that were made, would it not? The pureQuery Runtime provides two features that allow you to do just that without the need to modify the database or the application itself.

The Runtime can be configured to prevent execution of any SQL that has not been captured, and likely not been reviewed and approved. This special processing (also referred to as whitelisting) is enabled by setting the `capturedOnly` property in the Runtime configuration to `TRUE` (the default is `FALSE`), which essentially triggers an error if any SQL-related application changes are made. Example 5-10 depicts a pureQuery Runtime configuration that prevents execution of SQL that has not been approved.

Example 5-10 Enabling whitelist processing for dynamic SQL;

```
pdqProperties=executionMode(DYNAMIC),  
pureQueryXml (/path/to/myapprovedSQL.pdqxml),  
capturedOnly(TRUE)
```

Note: At the time of writing, whitelist processing cannot be enforced for applications that were implemented using the method-style pureQuery API.

If your goal is to determine which SQL has been changed or added (since the application's SQL was reviewed the last time), configure the pureQuery Runtime to capture this SQL in a separate file that can then be reviewed using Optim Development Studio. To enable delta detection, configure the `outputPureQueryXml` property and assign a file name to it, as shown in Example 5-11.

Example 5-11 Capturing delta SQL to identify application changes

```
pdqProperties=executionMode(DYNAMIC),  
pureQueryXml (/path/to/myapprovedSQL.pdqxml),  
outputPureQueryXml (/path/to/deltaSQL.pdqxml)
```

If the runtime is configured as shown in Example 5-11, any SQL that is not present in file `myapprovedSQL.pdqxml` will be captured in file `deltaSQL.pdqxml`.

Note: You can use the merge utility in Optim Development Studio or the command line to merge the two files if the new or updated SQL is acceptable

Scenario 2: Schema change impact analysis

In most environments multiple applications are accessing the same database. What if the database model changes and new columns must be added to tables or removed from tables or data types have to be modified to reflect changing requirements? How does your team currently identify the impact on applications that are accessing or modifying these objects?

The SQL Outline view in Optim Development Studio can be used to identify whether an application is impacted by a database schema change, because it visualizes how an application accesses or modifies database objects and where the operation is performed. Figure 5-16 shows the Database tab in the SQL Outline. It displays the database objects that are being accessed or modified by the application for which the metadata is displayed.

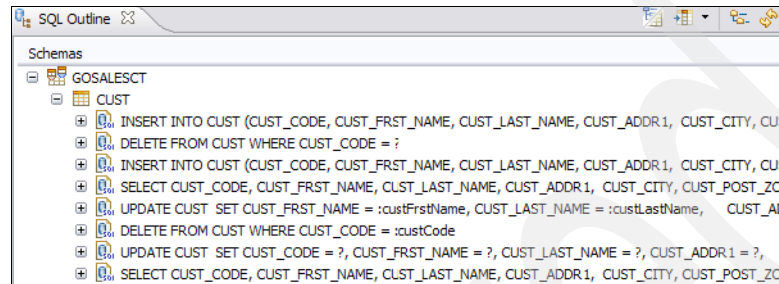


Figure 5-16 The database tab in the SQL Outline

Note: The SQL Outline displays information for applications that have been implemented using the pureQuery API as well as the content of capture files that provide insights for applications that use any other JDBC-based data access technology. Basic impact analysis can be performed for any type of Java application that accesses a database.

Query optimization considerations

Providing insights into how and how often an application accesses or manipulates database objects during development can significantly improve data access performance, because potential hot-spots can be identified during the development cycle. To gain these insights for existing Java applications, you must complete the capture phase of the client optimization process. If an application's data access layer was implemented using the method-style pureQuery API, the metadata is embedded in the source code and available without the need to capture it explicitly.

5.3 Improving database access security for Java applications

“You have been selected for an audit!” is probably one of the things you do not look forward to hearing as a tax paying citizen. Being prepared for a business audit is part of your responsibilities as an enterprise database administrator. What processes are in place to enforce data integrity and privacy? Who can access the data? Can you show your audit reports? And on it goes ...

Preventing unauthorized access to business data is a crucial requirement for every business. Exposure of confidential information can result in costly litigation and may have a significant impact on the business. As a DBA you are responsible for safekeeping information by enforcing strict access control to database objects.

The general database security model that is applied when applications access database objects varies based on the SQL execution mode that an application uses to access the database. Dynamic SQL execution is governed by the implicitly or explicitly granted authority to access or manipulate the database objects that are referenced by SQL statements. Sometimes this level of granularity is not good enough, because using this model makes it possible to access or manipulate data in unintended ways.

For applications that are accessing DB2 data servers you have the option to switch to static SQL execution to enforce much stricter security. Applications are basically only allowed to process only a well-defined set of SQL (which is stored in database packages). Users do not have to have any database object access. If for one reason or another you cannot exploit this security model (for example, because it is not supported by the database server), the pureQuery platform can provide you with an additional layer of security.

In the following section we take a closer look at how the pureQuery platform can prevent some common security issues for dynamic SQL.

5.3.1 Preventing SQL injection

The subject of SQL injection has gained a lot of attention during the past years. Companies small and large have been affected by this code injection technique, which exploits certain database access programming approaches. It can be used to obtain unauthorized data access or, even worse, to manipulate or destroy data.

To understand how pureQuery can help prevent SQL injection, let us take a look at a simple example query that retrieves credit card information for a specific user. Example 5-12 shows a simple SQL statement that uses a literal value.

Example 5-12 Simple SQL statement that uses a literal value

```
SELECT * FROM CRD_CRDT WHERE CUST_CODE = '0381'
```

This query would be vulnerable to SQL injection if the query string was composed and executed by the Java application, as shown in Example 5-13.

Example 5-13 Java source code to generates SQL statement using external input

```
stmt.execute("SELECT * FROM CRD_CRDT WHERE CUST_CODE = ' +  
customer_code + '");
```

If the variable `customer_code` contains a value that was specified by the user (for example, '0381' OR '1' = '1' as shown in Example 5-14), the query could be actually rewritten to return credit card information for more than one user.

Example 5-14 Resulting SQL statement if user input is not pre-processed properly

```
SELECT * FROM CRD_CRDT WHERE CUST_CODE = '0381' OR '1' = '1'
```

The SQL injection threat can be mitigated by using parameter markers in the query because user input is treated as a data token instead of part of the SQL, as shown in Example 5-15.

Example 5-15 Prevents SQL injection by using parameter markers

```
pStmt = conn.prepareStatement("SELECT * FROM CRD_CRDT WHERE CUST_CODE = ?");  
pStmt.setString(1, customer_code);
```

To avoid this issue, if an application's data access layer was not developed according to best practices, you can use the pureQuery Runtime's SQL literal substitution feature in combination with its whitelist processing by defining the pureQuery configuration in Example 5-16.

Example 5-16 PureQuery Runtime configuration that can prevent SQL injection

```
pdqProperties=executionMode(DYNAMIC),  
pureQueryXml(/path/to/whitelist.pdqxml),  
capturedOnly(TRUE),  
sqlLiteralSubstitution(ENABLE)
```

This two-pronged approach mitigates the security threat because it enforces the use of parameter markers and prevents execution of SQL that is not present in the whitelist.

To understand why both features should be used together, we must take a brief detour and review how SQL literal substitution processing works. During the capture phase SQL is consolidated into a single statement if it matches exactly (with exception of the string literal values). This consolidated SQL statement is stored in the SQL capture file and matched against incoming SQL at runtime. If, at runtime, an incoming SQL statement can be reduced to the parameterized statement that is present in the capture file, the parameterized SQL will be executed instead and the SQL literal passed through the database as a typed parameter value.

In the case of a malicious user, however, the incoming SQL statement in the example above cannot be matched because it contains more SQL literals. Since no matching occurred, the query is executed as though whitelist processing is not enabled and an unauthorized data access occurred. The Runtime's whitelist processing feature prevents this unauthorized execution, because this form of SQL (with additional SQL literals) is not present in the approved whitelist.

You might have noticed that this solution is an extension of the approach described in scenario 1 in 5.2.3, "Apply SQL development best practices and prevent rogue queries" on page 171, where we outline how you can restrict which SQL an application may process. This runtime functionality essentially mirrors what happens during static SQL execution without the need to go through a client optimization process.

Security considerations

The pureQuery platform does not replace the database's security framework, but it can be used to augment it. Using the client optimization process, Java data access applications that were implemented using the JDBC API can be enabled to take advantage of the static SQL security model. Java applications whose data access layer was implemented using the pureQuery method style API natively support static SQL execution and can therefore use this security model. If dynamic SQL execution is desired, the pureQuery Runtime can be used to restrict which SQL can be executed by a particular application.

Summary

In this chapter we have reviewed some of the existing and new features of the pureQuery Runtime and Optim Development Studio in the context of common challenges that DBAs and developers are facing. Up-to-date information about the latest features can be found in the popular “*What's new and cool in ...*” series on developerWorks, at the following Web page:

<http://www.ibm.com/developerworks/data/products/devstudio/>

Also, refer to the community page, which provides access to the latest articles, tutorials, demos, trial versions, videos, blogs and podcasts:

<http://www.ibm.com/developerworks/spaces/optim>

Dedicated forums at this Web site provide answers to commonly asked questions.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 186. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *DB2 9 for z/OS: Packages Revisited*, SG24-7688
- ▶ *DB2 9 for z/OS: Distributed Functions*, SG24-6952
- ▶ *System Programmer's Guide to: Workload Manager*, SG24-6472
- ▶ *IBM DB2 9 for z/OS: New Tools for Query Optimization*, SG24-7421
- ▶ *IBM Tivoli Composite Application Manager Family Installation, Configuration, and Basic Usage*, SG24-7151

Other publications

These publications are also relevant as further information sources:

IBM DB2 for Linux, UNIX, and Windows manuals

- ▶ *Administrative API Reference*, SC27-2435
- ▶ *Administrative Routines and Views*, SC27-2436
- ▶ *Call Level Interface Guide and Reference, Volume 1*, SC27-2437
- ▶ *Call Level Interface Guide and Reference, Volume 2*, SC27-2438
- ▶ *Command Reference*, SC27-2439
- ▶ *Data Movement Utilities Guide and Reference*, SC27-2440
- ▶ *Data Recovery and High Availability Guide and Reference*, SC27-2441
- ▶ *Database Administration Concepts and Configuration Reference*, SC27-2442
- ▶ *Database Monitoring Guide and Reference*, SC27-2458
- ▶ *Database Security Guide*, SC27-2443

- ▶ *DB2 Text Search Guide*, SC27-2459
- ▶ *Developing ADO.NET and OLE DB Applications*, SC27-2444
- ▶ *Developing Embedded SQL Applications*, SC27-2445
- ▶ *Developing Java Applications*, SC27-2446
- ▶ *Developing Perl, PHP, Python, and Ruby on Rails Applications*, SC27-2447
- ▶ *Developing User-defined Routines (SQL and External)*, SC27-2448
- ▶ *Getting Started with Database Application Development*, GI11-9410
- ▶ *Getting Started with DB2 Installation and Administration on Linux and Windows*, GI11-9411
- ▶ *Globalization Guide*, SC27-2449
- ▶ *Installing DB2 Servers*, GC27-2455
- ▶ *Installing IBM Data Server Clients*, GC27-2454
- ▶ *Message Reference Volume 1*, SC27-2450
- ▶ *Message Reference Volume 2*, SC27-2451
- ▶ *Net Search Extender Administration and User's Guide*, SC27-2469
- ▶ *SQL Procedural Languages: Application Enablement and Support*, SC23-9838
- ▶ *Partitioning and Clustering Guide*, SC27-2453
- ▶ *pureXML Guide*, SC27-2465
- ▶ *Query Patroller Administration and User's Guide*, SC27-2467
- ▶ *Spatial Extender and Geodetic Data Management Feature User's Guide and Reference*, SC27-2468
- ▶ *SQL Procedural Language Guide*, SC27-2470
- ▶ *SQL Reference, Volume 1*, SC27-2456
- ▶ *SQL Reference, Volume 2*, SC27-2457
- ▶ *Troubleshooting and Tuning Database Performance*, SC27-2461
- ▶ *Upgrading to DB2 Version 9.7*, SC27-2452
- ▶ *Visual Explain Tutorial*, SC27-2462
- ▶ *What's New for DB2 Version 9.7*, SC27-2463
- ▶ *Workload Manager Guide and Reference*, SC27-2464
- ▶ *XQuery Reference*, SC27-2466
- ▶ *Installing and Configuring DB2 Connect Personal Edition*, SC27-2432
- ▶ *Installing and Configuring DB2 Connect Servers*, SC27-2433

- ▶ *DB2 Connect User's Guide*, SC27-2434
- ▶ *Information Integration: Administration Guide for Federated Systems*, SC19-1020-02
- ▶ *Information Integration: ASNCLP Program Reference for Replication and Event Publishing*, SC19-1018-04
- ▶ *Information Integration: Configuration Guide for Federated Data Sources*, SC19-1034-02
- ▶ *Information Integration: SQL Replication Guide and Reference*, SC19-1030-02
- ▶ *Information Integration: Introduction to Replication and Event Publishing*, SC19-1028-02

Online resources

These Web sites are also relevant as further information sources:

DB2

- ▶ IBM Integrated Data Management
<http://www-01.ibm.com/software/data/optim/>
- ▶ DB2 Information Center
<http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/>
- ▶ Database and Information Management home page
<http://www.ibm.com/software/data/>
- ▶ DB2 Technical Support
http://www.ibm.com/software/data/db2/support/db2_9/
- ▶ DB2 Product Family Library
<http://www.ibm.com/software/data/db2/library/>
- ▶ DB2 developerWorks
<http://www.ibm.com/developerworks/db2/>
- ▶ DB2 for Linux
<http://www.ibm.com/software/data/db2/linux/>
<http://www.ibm.com/software/data/db2/linux/validate/>
- ▶ DB2 Universal Database V9 Application Development
<http://www.ibm.com/software/data/db2/ad/>

- Planet DB2

<http://www.planetdb2.com/>

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Numerics

3-tier application 96

A

access path selection 21
access plan 37, 155
access plan graphic 47
address space 28
alert icon 123
application centric security model 33
application environment 96
application manageability 25
application server 122
application server node 52
archive file 100
artifact creation 85
authorization 21

B

batch job 99
bean caching 120
bind command 111
bind operation 79
bind process 21
bind utility 86
binder command utility 86
binder option 102
binder utility 78
-bindOptions option 102
buffer pool 120
business logic 12, 120
business object 12

C

cache 156
caching strategy 156
capture file 86
change control process 99
change control system 101
classification rule 31
classloader 56
CLASSPATH definition 60

client node 52
client optimization technology 34
client tier node 54
COBOL 20
command line utility 84
COMMIT 130
common component 9
common user interface 9
configuration setting 94
configure command utility 85
connection pool 126
connection pooling 120
connection string 56
consistency token 100
content assist 78
context menu 75
CPU consumption 39
CPU usage 38, 155
cursor 131
cursor stability 103, 131
custom property 56, 69

D

data access interface 113
data access layer 25
data access code 110
data access method 113
data definition language 4
data management lifecycle 5
data privacy service 10
data source 56, 58, 60–61
data structure 91
database application 122
database client 122
database connection 56
database node 52
database object 79
database object validation 21
database package 25, 73, 79, 92
database request module 20, 98
database result caching 120
database server 122
database system performance 122

- data-centric application 91
- data-driven application 5
- DB2 Interactive 99
- DB2I 99
- db2jcc.jar 65
- DBRM 79
- DDF 28
- DDL 4
- deadlock 130
- deadlocks 133
- default isolation level 131
- default qualifier 79
- deployment 85, 92
- direct access storage device 4
- dirty read 130
- dispatchable unit 28
- distributed data facility 28
- distributed thread 28
- distributed workload 20, 26
- dynamic call 26
- dynamic SQL 28
- dynamic SQL execution 12
- dynamic SQL processing 34

E

- end-to-end monitoring 122
- enterprise database environment 5
- executable load module 21
- execution mode 83
- execution privilege 92, 99
- extensibility 120

F

- failover 2

G

- generateDBRM option 98
- generic package 34

H

- hardware resource 90
- heterogeneous batching of updates 14
- hierarchical structure 76
- hit ratio 120

I

- industrial standard 99

- infrastructure property 61
- initial application design 4
- initial binding 111
- in-memory application object 25
- integrated data management 5, 7
- internal regulation 99
- Internet application 120
- isolation level 78, 103, 130–131
- isolationLevel option 103

J

- Java language extension 22
- Java persistence architecture 12
- Java runtime environment 94
- JCC driver 60
- JDBC driver 94, 96
- JPA 12
- JVM parameter 56

K

- key component 91
- key performance indicator (KPI) 129

L

- language interface 20
- LDM 18
- level of service 28
- literal replacement feature 156
- literal value 161
- load data 5
- lock escalation 132
- locklist 131
- locktimeout 132
- logical 112
- logical data model 18
- logical model 4

M

- maintenance task 111
- manage growth 5
- manageability 91
- maxlocks 131–132
- maxStackTracesCaptured 67
- measurement statistics 5
- medium-sized business 91
- metadata 10
- method-style API 100

metrix 3, 138
modified source file 20
multi-tier infrastructure 120

N

network congestion 122

O

object module 21
object-relational mapping 12, 163
optimization 21
optimization phases
 bind 51
 capture 51
 configuration 51
 execution 52
option file 85
ORM 12
overview window 123

P

package 98
package characteristics 75
package name 100
package version 100
packagePrefixExclusions 67
parameter marker 158, 160–161
PASCAL 20
performance 91, 120
performance indicator 122
performance metrix 31
performance statistics 32
performance warehouse 129
periodic exception 138
persistence framework 12, 34, 76, 163
persistence technology 12
physical data model 18
physical model 4
PL/I 20
plan 34
portlet 9
precompiler 20
problem determination task 51
problem diagnosis 26
problem resolution cycle 13
provider 58
pureQuery Outline view 72

pureQuery properties
 allowDynamicSQL 57
 captureMode 57, 69
 executionMode 57
 pureQueryXml 57, 69

Q

query response time 155

R

read stability 103, 131
rebind command 111
Redbooks Web site 185
 Contact us xii
regulatory compliance 99
relational object 25
relational persistence framework 25
repeatable read 103, 131
research data 10
resource contention 130
resource group 29
response time 122
runstats advisor 49

S

scalability 120
security 91
security exposure 20
service class 28, 31
service level agreement 2, 129
service level objective 2
session management 120
shared artifact 9
SMB 91
software layer 67
source module 20
SQL 5
SQL injection attack 32
SQL metadata 54
SQL replacement 157, 160
stackTraceDepth 67
statement cache 23
statement preparation 24
static execution 38
static execution mode 25, 73
static package 111
static SQL 38

StaticBinder utility 98
stress test 101
subsystem 99
system catalog table 112
system throughput 13

T

test environment 66
thread activity 26
thread pooling 120
thread statistics 27
threshold 124, 137
troubleshooting 91

U

uncommitted read 103, 130
unit of work 28, 131
user-centric security model 33

W

white list processing 162
whitelist processing 157, 160
workaround 3

Using Integrated Data Management To Meet Service Level Objectives

(0.2" spine)
0.17" <-> 0.473"
90 <-> 249 pages



Using Integrated Data Management To Meet Service Level Objectives



Client optimization with pureQuery

In this IBM Redbooks publication, we learn how The Great Outdoor Company, a fictional retailer of outdoor products, uses Integrated Data Management solutions by IBM to optimize data management for performance, productivity, and manageability, allowing them to get the most from their data assets and to ensure they can meet service level objectives (SLOs).

Deploy pureQuery-enabled application

End-to-end monitoring with IBM tools

We discuss how organizational and infrastructure complexities can negatively affect overall performance and productivity and how an integrated data management approach can help bridge these silos and gaps to provide end-to-end data and data application life cycle support. To illustrate, we focus on how to optimize an enterprise Java application using the Optim Development Studio (formerly Data Studio Developer) and pureQuery Runtime to improve performance, manageability, and security.

We discuss the practical aspects of managing the deployment and maintenance of pureQuery applications. Because an integrated data management approach extends to the problem solving aspect of the data life cycle, we also discuss how DB2 Performance Expert, DB2 Performance Expert Extended Insight Feature (PE/EI), and IBM Tivoli Composite Application Manager (ITCAM) can help you pinpoint problems in complex application environments.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks