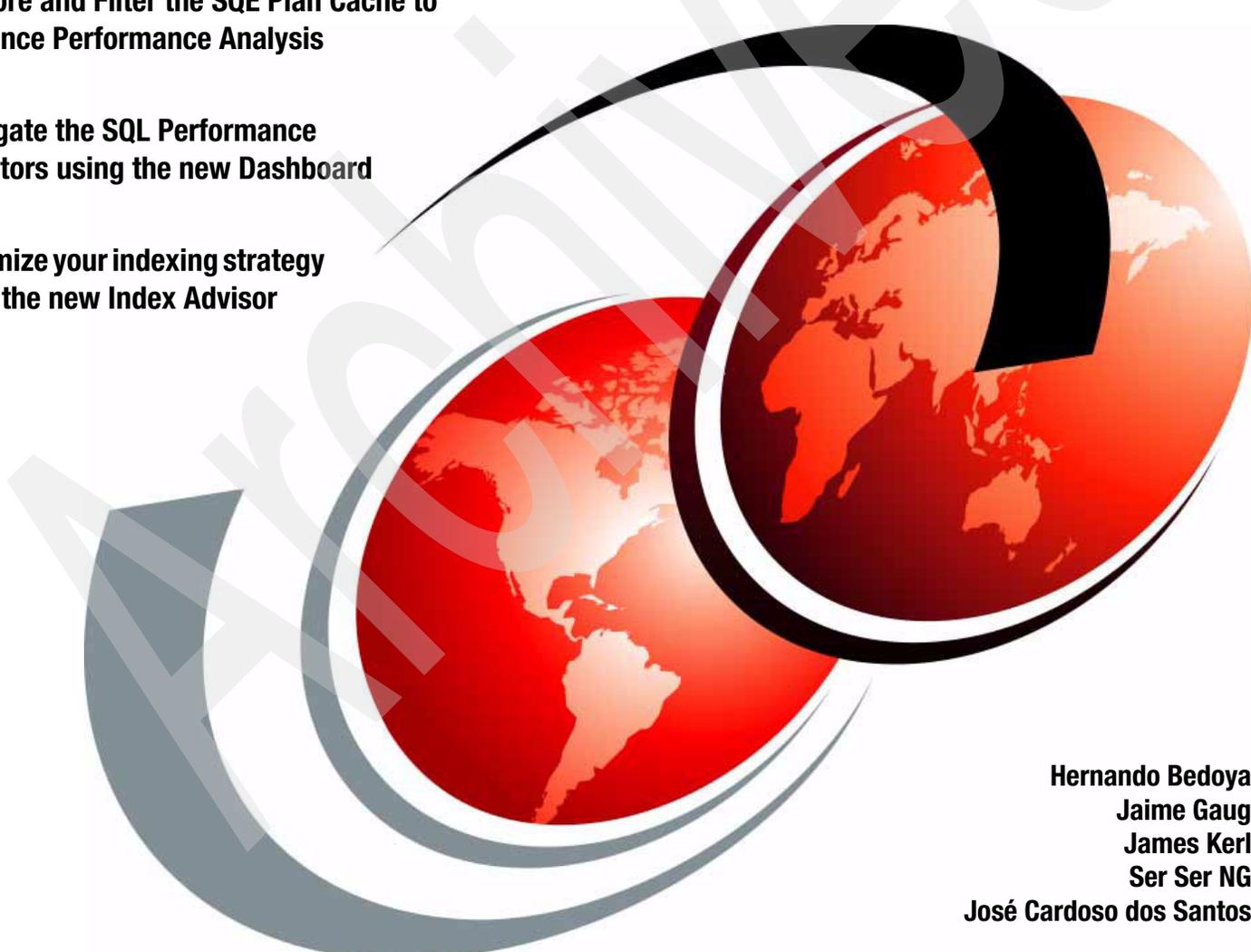


OnDemand SQL Performance Analysis Simplified on DB2 for i5/OS in V5R4

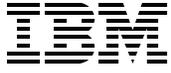
Explore and Filter the SQE Plan Cache to
enhance Performance Analysis

Navigate the SQL Performance
Monitors using the new Dashboard

Optimize your indexing strategy
with the new Index Advisor



Hernando Bedoya
Jaime Gaug
James Kerl
Ser Ser NG
José Cardoso dos Santos



International Technical Support Organization

**OnDemand SQL Performance Analysis Simplified on
DB2 for i5/OS in V5R4**

March 2007

Archived

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

Archived

First Edition (March 2007)

This edition applies to Version 5, Release 4, of i5/OS (5722-SS1).

Contents

Contents	iii
Notices	ix
Trademarks	x
Preface	xi
The team that wrote this redbook.	xi
Become a published author	xiv
Comments welcome.	xiv
Chapter 1. Determining whether you have an SQL performance problem	3
1.1 Questions to ask yourself	4
1.2 How do you know that there is a problem?	4
1.3 Where is the problem occurring?	5
1.4 Did you ever have satisfactory performance?	7
1.5 Do SQL queries appear to have performance problems?	8
Chapter 2. DB2 for i5/OS performance basics.	11
2.1 Basics of indexing	12
2.1.1 Binary radix tree indexes	12
2.1.2 Encoded-vector index	13
2.2 Query engines: an overview	15
2.2.1 Database architecture before V5R2M0.	17
2.2.2 Current database architecture	17
2.2.3 Query Dispatcher	19
2.2.4 Statistics Manager	22
2.2.5 SQE Optimizer	24
2.2.6 Data Access Primitives	25
2.2.7 Access plan.	25
2.2.8 SQL packages	26
2.2.9 SQE Plan Cache	27
2.2.10 Open Data Path	29
Chapter 3. Overview of tools to analyze database performance	33
3.1 Introduction to the tools.	34
3.2 SQE Plan Cache	35
3.3 SQE Plan Cache Snapshots	39
3.4 The Database Performance Monitors	43
3.4.1 Detailed Monitor	44
3.4.2 Summary Monitor or Memory Resident Database Monitor.	51
3.4.3 Importing a Database Monitor to SQL Performance Monitor	57
3.4.4 The Monitor Comparison feature of SQL Performance Monitor	57
3.5 Visual Explain	60
3.6 Index Advisor	63
3.7 Evaluators	67
3.7.1 Index Evaluator	67
3.7.2 MQT (Materialized Query Tables) Evaluator.	70
3.8 Current SQL for a Job function	78
3.9 Debug messages	82

3.10 Print SQL information	86
Chapter 4. Gathering SQL performance data	89
4.1 Types of SQL Performance Monitors	90
4.2 Collecting Detailed SQL performance data	90
4.2.1 Starting a Detailed Database Monitor using the command interface	90
4.2.2 The SQL Performance Monitor Wizard	96
4.2.3 Starting a Detailed SQL Performance Monitor	97
4.2.4 Enabling Database Monitors in ODBC clients	101
4.2.5 Enabling Database Monitors in OLE DB clients	103
4.2.6 Enabling Database Monitors in JDBC clients	105
4.2.7 Enabling Database Monitors in .NET clients	106
4.2.8 Enabling Database Monitors using an exit program	106
4.3 Collecting Summary SQL Performance Monitor data	106
4.4 Importing Database Monitors into iSeries Navigator	109
4.4.1 Importing detailed monitor data	110
4.4.2 Importing detailed monitor data from a previous release	110
4.4.3 Importing summary monitor data	111
4.5 SQL Performance Monitors properties	111
4.5.1 Detailed monitor properties	111
4.5.2 Summary monitor properties	112
4.5.3 Imported monitor properties	113
4.6 Summary or Detailed SQL Performance Monitor	114
Chapter 5. Analyzing SQL performance data using iSeries Navigator	117
5.1 Detailed SQL Performance Monitor Analysis overview	118
5.1.1 Analysis overview	119
5.1.2 Amount of work requested	122
5.1.3 Environmental factors	123
5.1.4 Implementation information	124
5.1.5 Types of statements	126
5.1.6 Miscellaneous information	127
5.1.7 I/O information	128
5.2 In-depth analysis reports	128
5.2.1 Getting detailed reports from Summary and Statement buttons	129
5.2.2 Additional information reports	133
5.2.3 Action menu items	140
5.3 Summary SQL Performance Monitor analysis overview	143
5.3.1 SQL performance report information from summary data	145
5.3.2 Examples and application of Summary SQL Performance Monitor	148
5.3.3 Limitations of the Summary monitor	148
5.4 Show Statements	149
5.4.1 Filtering options	149
5.4.2 Launching Visual Explain	152
5.5 Compare monitors	152
5.6 Case study	156
5.6.1 A poor performing SQL statement	157
5.6.2 Why are table scans being done?	159
5.6.3 Why is CQE being used?	167
5.6.4 Comparison	171
Chapter 6. Custom Database Monitor Analysis	173
6.1 The Database Monitor record types	174
6.1.1 Database Monitor record types	174

6.1.2	The 1000 Record: SQL statement summary	177
6.1.3	The 30XX Records: Query Optimization Row Type	179
6.1.4	The 3000 Record: Arrival sequence (table scan)	179
6.1.5	The 3001 Record: using an existing index	181
6.1.6	The 3002 Record: temporary index created	182
6.1.7	The 3003 record: query sort	183
6.1.8	The 3004 record: temporary file	184
6.1.9	The 3006 record: access plan rebuild	185
6.1.10	The 3007 record: index evaluation	186
6.1.11	The 3010 record: host variables	187
6.1.12	The 3014 record: general query optimization information	188
6.1.13	The 3015 record: SQE statistics advised	189
6.1.14	The 3018 record: STRDBMON/ENDDDBMON	189
6.1.15	The 3019 record: rows retrieved detail	189
6.1.16	The 3020 record: index advised (SQE)	190
6.1.17	The 3030 record: materialized query table	191
6.2	Introduction to query analysis	192
6.3	Tips for analyzing the Database Monitor files	194
6.3.1	Using an SQL ALIAS for the Database Monitor table	194
6.3.2	Using a subset of the Database Monitor table for faster analysis	194
6.3.3	Using SQL views for the Database Monitor table	195
6.3.4	Creating additional indexes over the Database Monitor table	195
6.4	Database Monitor query examples	196
6.4.1	Finding SQL requests that are causing problems	197
6.4.2	Total time spent in SQL	198
6.4.3	Individual SQL elapsed time	199
6.4.4	Analyzing SQL operation types	202
6.4.5	Full open analysis	203
6.4.6	Isolation level used	208
6.4.7	Table scan	209
6.4.8	Temporary index analysis	211
6.4.9	Index advised	214
6.4.10	Access plan rebuilt	218
6.4.11	Query sorting	223
6.4.12	SQE advised statistics analysis	227
6.4.13	Fetches and Retrieved detail rows	231
6.4.14	Materialized query tables	234
	Chapter 7. SQE Plan Cache and SQE Plan Cache Snapshots	237
7.1	SQE Plan Cache and SQE Plan Cache Snapshot	238
7.2	SQE Plan Cache	238
7.2.1	Viewing the properties of the SQE Plan Cache	238
7.2.2	Viewing the content of the SQE Plan Cache	242
7.2.3	Using the filter options	243
7.2.4	Finding and Visual Explaining a query from the SQE Plan Cache	245
7.3	SQE Plan Cache Snapshot	248
7.3.1	Creating an SQE Plan Cache Snapshot using iSeries Navigator	248
7.3.2	Creating an SQE Plan Cache Snapshot using Stored Procedure	252
7.3.3	Creating an SQE Plan Cache Snapshot using an Exit Program	256
7.3.4	Analyzing an SQE Plan Cache Snapshot	258
7.3.5	Working with SQL statements from an SQE Plan Cache Snapshot	264
7.3.6	An example of finding table scans in a SQE Plan Cache Snapshot	266
7.3.7	Comparing SQE Plan Cache Snapshots	269

Chapter 8. Analyzing database performance data with Visual Explain	275
8.1 What is Visual Explain.	276
8.2 Finding Visual Explain.	276
8.2.1 The SQL Script Center	278
8.2.2 Explain Only	279
8.2.3 Run and Explain	280
8.2.4 Explain While Running	280
8.3 Navigating Visual Explain	281
8.3.1 Toolbar	283
8.3.2 Menu options	288
8.3.3 Controlling the diagram level of detail.	291
8.3.4 Displaying the Environment Settings	293
8.3.5 Visual Explain query attributes and values	294
8.4 Using Visual Explain with Database Monitor data.	298
8.5 Using Visual Explain with imported data	301
8.5.1 Show Statements	303
8.6 Using Visual Explain with SQE Plan Cache and Plan Cache Snapshot.	304
8.6.1 Using Visual Explain with SQE Plan Cache	304
8.6.2 Using Visual Explain with SQE Plan Cache Snapshot	307
8.7 Non-SQL interface considerations	310
8.8 The Visual Explain icons	311
Chapter 9. Index Advisor	319
9.1 What is the Index Advisor	320
9.2 System Wide Index Advised Table	320
9.3 Levels of Index Advisor access.	324
9.3.1 Index Advisor access at Database level	325
9.3.2 Index Advisor access at Schema level	326
9.3.3 Index Advisor access at Table level	327
9.4 Index Advisor interface in iSeries Navigator	327
9.5 Interfaces to Index Advised information	330
9.5.1 Access to Index Advised information from Detailed SQL Performance Monitor screen interface	331
9.5.2 Access to Index Advised information from SQE Plan Cache screen interface	334
9.5.3 Access to Index Advised information from SQE Plan Cache Snapshot screen interface	336
9.5.4 Access to Index Advised information from Visual Explain screen interface	339
9.5.5 Access to Index Advised information from the Debug messages	346
9.6 Temporary Indexes	355
9.6.1 CQE - Temporary Indexes	356
9.6.2 SQE - Temporary Indexes	356
Chapter 10. SQL performance analysis: a methodology	363
10.1 Performance methodology	364
10.2 Performance troubleshooting	364
10.2.1 Problem source determination	364
10.2.2 Performance data capture	365
10.2.3 Performance analysis process	366
10.3 Application of the tools to the methodology	366
10.4 Example of using the methodology	370
Chapter 11. Environmental settings that affect SQL performance	379
11.1 Introduction	380
11.2 Optimization goal	381

11.2.1	What is the goal?	382
11.2.2	Setting the Optimization Goal	385
11.3	Sensitive Cursors	387
11.3.1	Performance and query optimization impacts	388
11.3.2	Cursor sensitivity programming interfaces	389
11.4	SMP (Symmetrical multiprocessing) Degree.	393
11.4.1	iSeries i5/OS Architecture	393
11.4.2	What is SMP?	394
11.4.3	SMP parallel-enabled functions	395
11.4.4	Parallel Database Processing	397
11.4.5	Enabling parallel processing	398
11.4.6	Feedback	399
11.4.7	Available and balanced resources	403
11.4.8	SMP considerations	404
Chapter 12. Tips to proactively prevent SQL performance problems		407
12.1	Indexing strategy	408
12.1.1	Access methods	408
12.1.2	Guidelines for perfect indexes	409
12.1.3	Additional indexing tips	410
12.1.4	Index Advisor	411
12.2	Coding of your SQL statements	411
12.2.1	Avoid using logical files in your select statements	411
12.2.2	Avoid using SELECT * in your select statements	413
12.2.3	Avoid using the relative record number to access your data	414
12.2.4	Avoid numeric data type conversion	415
12.2.5	Avoid numeric expressions	416
12.2.6	Using the LIKE predicate	419
12.2.7	Avoid scalar functions in the WHERE clause	420
Chapter 13. Using Collection Services data to identify jobs using system resources		423
13.1	Relationship of Collection Services, Database Monitor data and Performance Trace	424
13.2	Collection Services and Database Monitor data	425
13.2.1	Starting Collection Services	425
13.2.2	From iSeries Navigator	427
13.2.3	Using Performance Management APIs.	428
13.2.4	V5R4 STRPFRCOL command	428
13.3	Using Collection Services data to find jobs using CPU	429
13.3.1	Finding jobs using CPU with the Component Report	429
13.3.2	Finding jobs using CPU with iSeries Navigator Graph History	436
13.3.3	Finding jobs using CPU with Management Central System Monitors.	439
13.4	Using Collection Services data to find jobs with high disk I/O counts	442
13.5	Using Performance Data of the Database Monitor to find the query that needs optimization	446
13.6	Using Performance Trace to find object locks.	452
Appendix A. Tools to check a performance problem		457
	WRKACTJOB command	458
	WRKSYSACT command	459
	WRKSYSSTS command.	461
	WRKOBJLCK command.	462
	WRKJOB command	463
	iDoctor for iSeries Job Watcher	464

Related publications	467
IBM Redbooks	467
Other publications	467
Online resources	467
How to get IBM Redbooks	468
Help from IBM	468
Index	469

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which 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

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

Redbooks (logo) ™
eServer™
iSeries™
i5/OS®
AS/400®
DB2 Universal Database™

DB2®
DRDA®
IBM®
OS/400®
POWER™
Redbooks™

System i™
System/38™
SQL/400®
WebSphere®

The following terms are trademarks of other companies:

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

Snapshot, and the Network Appliance logo are trademarks or registered trademarks of Network Appliance, Inc. in the U.S. and other countries.

Java, JDBC, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

The goal of database performance tuning is to minimize the response time of your queries. It is also to optimize your server's resources by minimizing network traffic, disk I/O, and CPU time.

This IBM® Redbook helps you to understand the basics of identifying and tuning the performance of Structured Query Language (SQL) statements using IBM DB2® for i5/OS®. DB2 for i5/OS provides a comprehensive set of tools that help technical analysts tune SQL queries. The SQL Performance Monitors are part of the set of tools that IBM i5/OS provides for assisting in SQL performance analysis since Version 3 Release 6. These monitors help to analyze database performance problems after SQL requests are run. In V5R4 of i5/OS, iSeries™ Navigator provides a series of new tools to do SQL Performance analysis that we cover in this book. Among the new tools that we will covering are:

- ▶ Capability of visualizing the contents of the SQE Plan Cache
- ▶ SQE Plan Cache Snapshots
- ▶ The new reporting tool - Dashboard
- ▶ OnDemand Index Advisor
- ▶ Evaluators such as Index and Materialized Query Tables

This book also presents tips and techniques based on the SQL Performance Monitors and other tools, such as Visual Explain and all the tools provided in V5R4. You will find this guidance helpful in gaining the most out of both DB2 for i5/OS and query optimizer when using SQL.

Some of the material and foundation of this book was originally published in the IBM Redbook *SQL Performance Diagnosis on IBM DB2 Universal Database for iSeries*, SG24-6654.

Note: In this book, we use the name “SQL Performance Monitor” when using iSeries Navigator. SQL Performance Monitor has two versions: Detailed Database Monitor and Summary Monitor. We refer to the tool as “Database Monitor” when using a green screen and accessing the tool by running the Start Database Monitor (STRDBMON) CL command.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Poughkeepsie Center.



Hernando Bedoya is an IT Specialist at the IBM ITSO, in Rochester, Minnesota. He writes extensively and teaches IBM classes worldwide in all areas of DB2 Universal Database™ for iSeries. Before joining the ITSO more than six years ago, he worked for IBM Colombia as an IBM AS/400® IT Specialist doing pre-sales support for the Andean countries. He has 24 years of experience in the computing field and has taught database classes in Colombian universities. He holds a master degree in computer science from EAFIT, Colombia. His areas of expertise are database technology, application development, and data warehousing.



Jaime Gaug is a Staff Software Engineer in the award-winning iSeries Technical Support Center in Rochester, Minnesota. He provides on demand support of DB2 UDB for i5/OS for clients across the globe. Over the decade he has spent with IBM, he has been involved with identifying and resolving numerous issues involving SQL Performance. He holds a bachelor's degree in Computer Science from Case Western Reserve University, Cleveland, Ohio.



James Kerl is a Senior Certified IT Specialist for System i™. He provides pre-sales technical support for database and Linux® on System i for the IBM Americas Linux Advanced Technical Support Organization. James is located in Dallas, Texas where he has worked in the computing industry for IBM for 38 years. He has expertise in the iSeries database, Business Intelligence, linux, and most application languages. In the past, he has worked with midrange and host communications, network management and attached workstations. He holds a Bachelor of Science degree in Physics from the University of Texas in Arlington and has enhanced his education with many master level courses in

Business Administration. He has presented many technical topics on Business Intelligence and database on the IBM AS/400 and IBM eServer™ iSeries to customer and IBM internal conferences, such as, COMMON, the IBM iSeries Technical Conferences, IBM Professional Leadership Exchange (PLTE), and other IBM conferences. He is active in the IBM sponsored Try Science program where he speaks to local elementary schools in his home town of Fort Worth, Texas.



Ser Ser NG is an accredited Advisory IT Specialist at IBM Malaysia. She has 9 years of working experience in supporting iSeries. She performs post-sales software support, maintenance and services deliveries for iSeries. Her areas of expertise are performance study, Linux, SAP® setup and configuration for iSeries.



José Cardoso dos Santos is an IT Certified Specialist for System i platforms at Grupo Intercompany which is an IBM Premier Business Partner in São Paulo city in state of São Paulo, Brazil. He joined Grupo Intercompany in 2003. In 1986 he started his studies in Information Technology in São Bernardo do Campo city in state of São Paulo, Brazil. He has worked with iSeries platforms since 1995. His areas of expertise are database technology, database performance, application development and security. He provides pre-sales and post-sales support of DB2 for i5/OS, WebSphere® Application Server for i5/OS, Security on iSeries platforms and others. He also works as instructor of i5/OS

courses such as Control Language Programming, Security, Administration, DB2 for i5/OS, Query/400 and others. He holds IBM certifications such as IBM Certified Database Associate – DB2 UDB V8.1 Family Application Development, IBM Certified System Administrator

WebSphere Application Server ND V6.0 and IBM Certified System Administrator WebSphere Portal V5.0. He also holds a Bachelor of Mathematics with emphasis in Computer Science degree from Faculdade de Filosofia Ciências e Letras da Fundação de Santo André-SP, Brazil. He attended a post-graduation course of specialization in J2EE™ Technology at Faculdade de Informática e Administração Paulista (FIAP) – SP, Brazil.

The authors involved in the previous edition of this book were:

Elvis Budimlic
Director of Development at Centerfield Technology

Morten Buur Rasmussen
IT Specialist at the IBM Client Technology Center in La Gaude, France

Peggy Chidester
Staff Software Engineer in IBM Rochester Support Center

Fernando Echeveste
Staff Software Engineer in IBM Rochester

Birgitta Hauser
Software Engineer at Lunzer+Partner GmbH in Germany

Kang Min Lee
IT Specialist at IBM Korea

Dave Squires
iSeries Support Center in United Kingdom

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

Thomas Gray
Marvin Kulas
Joanna Pohl-Miszczyk
Jenifer Servais
ITSO, Poughkeepsie Center

Mark Anderson
Robert Bestgen
Michael Cain
Kevin Chidester
Daniel Cruikshank
Jim Flanagan
Scott Forstie
Kent Milligan
Brian Muras
Denise Voy Tompkin
IBM Rochester

Luis Guirigay
IT Specialist - Ascendant Technology LLC

Nicolas Bueso Quan
Certified IT Specialist IBM Brazil

Peter Bradley
IBM UK

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- ▶ Send your comments in an e-mail to:

redbook@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



Part 1

Introduction to DB2 for i5/OS and database performance tools

In this part, we introduce basic information about DB2 for i5/OS performance. We also introduce the different tools for analyzing database performance.

This part includes the following chapters:

- ▶ Chapter 1, “Determining whether you have an SQL performance problem” on page 3
- ▶ Chapter 2, “DB2 for i5/OS performance basics” on page 11

Note: In this book, we use the name “SQL Performance Monitor” when using iSeries Navigator. SQL Performance Monitor has two versions: Detailed Database Monitor and Summary Monitor. We refer to the tool as “Database Monitor” when using a green screen and accessing the tool by running the Start Database Monitor (STRDBMON) CL command.

Archived



Determining whether you have an SQL performance problem

In this chapter, we explain how to determine if a performance problem is the result of poorly performing Structured Query Language (SQL) queries. We describe the various system tools and a methodology to help you find an SQL performance problem.

This chapter guides you in:

- ▶ Asking the right questions to determine whether you have a performance problem
- ▶ Knowing what to check when you have a performance problem
- ▶ Determining if a performance problem is related to SQL

1.1 Questions to ask yourself

If you are reading this book, you most likely currently have an SQL performance problem, have experienced an SQL performance problem in the past, or are interested in learning how to diagnose SQL performance. An important step in looking into any SQL performance problem is to ask a few questions to determine whether the problem is with your SQL or a system-wide performance problem. To clarify the problem, you must ask yourself the following questions:

- ▶ How do you know that there is a problem?
- ▶ Where is the problem occurring?
- ▶ Did you ever have satisfactory performance?
 - If you have a job or program that used to run well but does not do so now, what changed since it last ran well?
- ▶ Do SQL queries appear to have a performance problem?
 - If the problem is with multiple queries, is there any commonality among the queries?
 - If the problem is with one query, what is the change in run time?
 - Have you ever tuned your queries for performance?

We explain each of these questions in more details in the sections that follow.

1.2 How do you know that there is a problem?

This question is very basic, yet vital. Identifying how you know that there is a problem determines the steps to take to analyze the problem.

For example, you are reading a report that shows that CPU spiked during a time frame. Is this a problem? Is CPU spiking bad? Are you seeing a trend or is this a one-time occurrence? During the time frame, did you receive complaints from users saying that they were unable to get their work done? Such complaints would be a strong indicator that there is an actual problem.

We recommend that you create a spreadsheet to document and keep a history of problems. Table 1-1 shows an example of how problem record keeping can be helpful in problem solving.

Table 1-1 Example of problem record keeping

Date of problem	Time of problem	User reporting	Problem job	Reported problem	How problem discovered	How problem resolved	Data gathered
01/05/05	13:01:00	Sales Report	QZDASOINIT	Report taking longer to generate	User found	^a	SQL Performance Monitor started

a. The queries to generate the sales report usually run in a dedicated memory pool. However, for some reason still unknown, the pool identifier in the QZDASOINIT prestart job was changed so that the queries started to run in *BASE along with some other high demanding jobs on the system. By changing the prestart job entry, the queries went back to run in their dedicated memory pool and the response time reverted to what it was in the past.

In the example shown in Table 1-1, it could look like an SQL problem since an SQL query was taking a long time. However, further analysis showed that other work done in the pool was the

cause of the problem. In Appendix A, “Tools to check a performance problem” on page 457, we explain how to use the system commands and tools to investigate whether the problem is related to the system or to an SQL query. There are a number of solutions to try to resolve the problem shown in Table 1-1:

- ▶ Separate the QZDASOINIT jobs into their own pool.

For more information, see Software Knowledge Base document “Separating Batch Work from *BASE”, document number 349033974 on the Web at:

http://www-912.ibm.com/s_dir/slkbases.NSF/1ac66549a21402188625680b0002037e/9fa68bd7573e48af862565c2007d3d9b?OpenDocument

Another more basic document in the Software Knowledge Base is document “Assigning Prestart Jobs to a Specific Pool”, document number 392581292 on the Web at:

http://www-912.ibm.com/s_dir/slkbases.nsf/1ac66549a21402188625680b0002037e/d34520dad99d66238625709200595ff9?OpenDocument

- ▶ Verify that the QPFRADJ system value is set to automatically adjust memory in the pools.
- ▶ Add more memory to the pool.

In the above example, a user is experiencing a problem when a report is generated. They found it to be a system-wide related problem, but if the above fix did not resolve the problem, then a more detailed analysis of the SQL job would be required.

1.3 Where is the problem occurring?

Another important step in analyzing an SQL performance problem is to identify where the problem is occurring. It is helpful to understand the components of work involved whenever a user executes any SQL request. Figure 1-1 illustrates the different components of work involved in the execution of an SQL request.

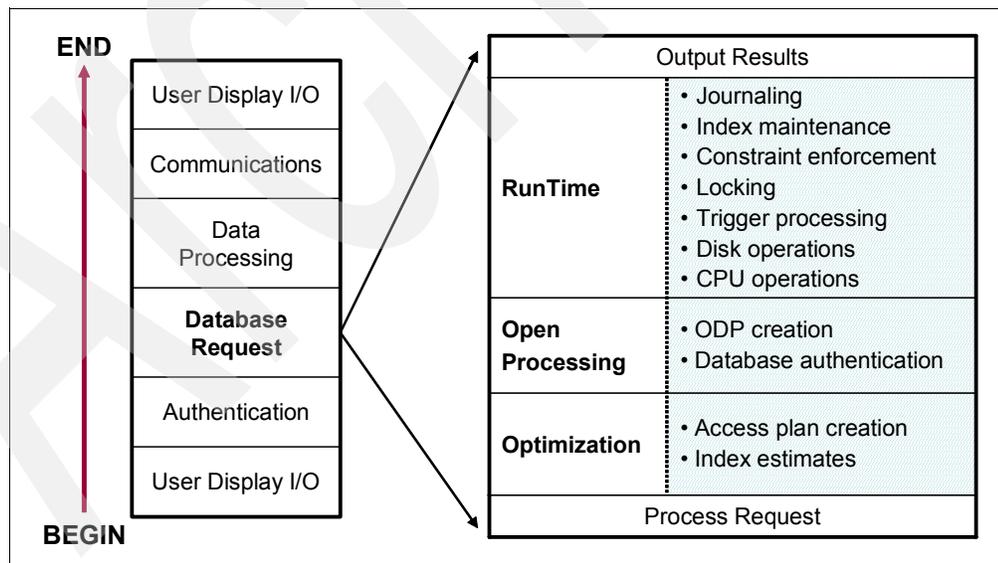


Figure 1-1 Components of work

Isolate the problem to the smallest number of factors as possible. Before we start analyzing the SQL or database request, we must understand that other variables are involved in the total response time such as:

- ▶ User display I/O
- ▶ Authentication
- ▶ Data processing
- ▶ Communications

You might ask some additional questions to find where the problem is occurring, such as:

- ▶ Is the problem occurring only in one pool?
 - What activity is occurring in the pool?
- ▶ Are you having a performance problem running tasks on the iSeries server, even simple commands?
- ▶ Is the problem occurring only within specific jobs, that is batch or QZDASOINIT?
- ▶ Do you hear only from one set of users when the problem occurs?
 - What is the commonality among the users?
- ▶ Does the problem occur only when going through a specific interface, such as WebSphere, SQL, or Query/400?
- ▶ Are the jobs having problems running remotely or locally?
- ▶ Can the problem be isolated to a specific program?
 - Is SQL embedded?
 - What type of program is it?
- ▶ Can the problem be isolated to a specific SQL statement?

You need to examine all of the answers to these questions to see if you can isolate the problem to jobs running SQL. If the jobs that have a problem are QZDASOINIT or QSQSRVR, then it is likely that they are running SQL. QRWTSRVR jobs are quite often used to run SQL, but are also used if you are using distributed data management (DDM) files. When the problem occurs only in certain jobs or for a certain group of users, you must review the environment, such as communications, pools, job priorities, and so on.

After you isolate the problem and are now certain that the problem is on the SQL request, you must understand what it takes to execute an SQL request. Upon the execution of an SQL request, three main operations are done as shown in Figure 1-2:

- ▶ Optimization time
- ▶ Open processing time
- ▶ Run time

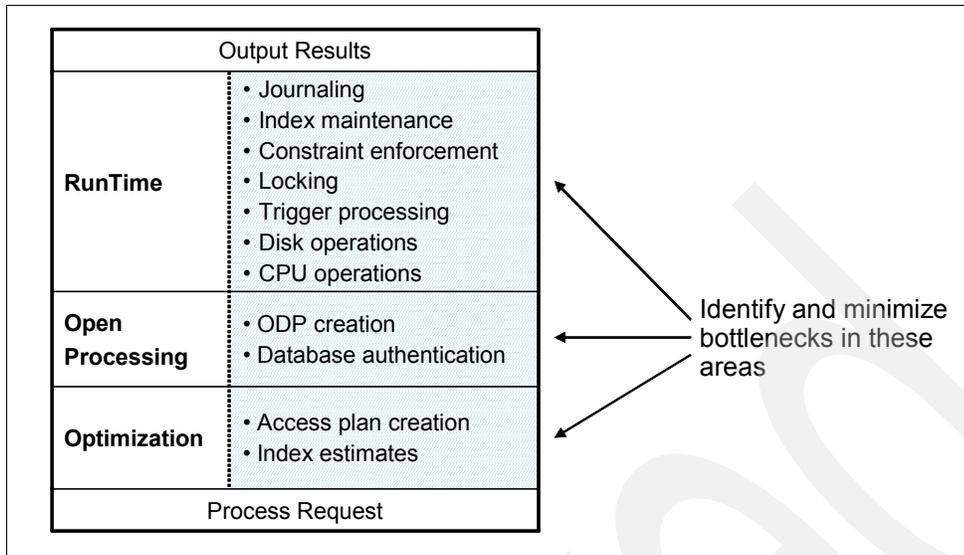


Figure 1-2 Components of work: Database request

In Figure 1-2, you can see which operations affect optimization time, open processing time, and run time. It is key that you identify and minimize the bottlenecks in these three main areas of the processing of an SQL request.

If it is believed to be SQL related because the problem remained after making the above changes then let's look at how you could approach resolving it using the tools available in V5R4 of i5/OS.

In order to make a quick determination if an SQL problem exists, an analyst could use SQE Plan Cache described in Chapter 7., "SQE Plan Cache and SQE Plan Cache Snapshots" on page 237 to see if there are SQL problems indicated for the jobs described above. This can be done while waiting for data to be collected by the performance monitor that was started. This initial analysis can begin to identify how the SQL is being processed by the database engine in DB2 for i5/OS.

With the information in Table 1-1 in hand and an indication during the initial analysis of the SQE Plan Cache whether there needs to be additional detailed analysis, we will need to wait for the detailed performance monitor data being collected.

When the problem occurs again you can then select to quickly analyze the monitor data using the iSeries Navigator tools described in Chapter 5., "Analyzing SQL performance data using iSeries Navigator" on page 117 either using the summary or detailed monitor analysis depending on the type of data that was being collected.

Understanding the tools that we describe in this book will help you to identify the bottlenecks in these areas.

1.4 Did you ever have satisfactory performance?

Satisfactory performance is when the system or an application is running without any performance problems. Knowing if you had satisfactory performance involves understanding the history of an application on the system. Some SQL performance problems are caused by:

- The introduction of a new application

- ▶ The application of new program temporary fixes (PTFs)
- ▶ An upgrade to a newer i5/OS release
- ▶ Changes to system values

In this book, you learn how to use some tools to gather performance information prior to any major change on the system. You learn how to determine SQL performance problems in cases where previously there was a good working situation or where you are unsure whether you previously had a good working situation. In a scenario where you know that you had satisfactory performance, it is vital to document the timeline of what happened since the system or application last ran well. Make sure that you document any changes, such as those that we previously listed.

The key is documenting exactly what the performance was prior to a change in the environment or application. You should always capture performance monitor data to have a historical record of how things were working prior to significant changes. With V5R4, we've made that easier with the ability to capture a snapshot of the current running environment if you haven't saved any performance monitor data lately. Although, it is not the ideal and does not provide a complete detailed view of all jobs, it can help if it becomes necessary to identify a change in the runtime environment.

Review Chapter 7., "SQE Plan Cache and SQE Plan Cache Snapshots" on page 237 and see how the SQE Plan Cache Snapshot™ can help build a partial picture of what the system is currently doing with SQL jobs.

1.5 Do SQL queries appear to have performance problems?

It might be difficult to answer this question if just one SQL query has a problem or if multiple SQL queries have a problem, since the issue is often found at the job level. In the following chapters, we show you how to use a different set of tools to investigate which queries are having problems, if you have not already made that determination. It is important to differentiate between one query having a performance problem and many SQL queries having a problem.

- ▶ One SQL query having a performance problem

When it appears that a single SQL query has a performance problem, you must know the run time of the specific query before the performance problem appeared. Additionally, you must know how you gather the runtime data.

- ▶ Multiple SQL queries having performance problems

In situations where it appears that multiple SQL queries have performance problems, you must ask additional questions to try to find any commonality among the queries:

- Do the queries that have the performance problem all use the same table?
- Does the problem appear to be with a specific type of query, such as left outer joins or updates?
- Do only specific users run the queries?
- Do all the queries run in the same pool?
- Have you ever tuned your queries for performance?

When multiple queries have a problem, it indicates the need for you to review the environment and examine such aspects as communications, pools, job priorities, and so on. Refer to Appendix A, "Tools to check a performance problem" on page 457, for more information.

One question to keep in mind when you are examining SQL performance problems is: "Did you ever tune your queries?" SQL queries require the existence of indexes to obtain statistics

and for implementation (access of data) of the query request. For more information about an adequate indexing strategy, refer to the white paper *Indexing and statistics strategy for DB2 UDB for iSeries* available on the Web at:

http://www.ibm.com/servers/enable/site/education/abstracts/indxng_abs.html

Through the use of the SQE Plan Cache and performance monitor data in the new iSeries Navigator GUI “dashboard” described in Chapter 5., “Analyzing SQL performance data using iSeries Navigator” on page 117, you can quickly and methodically research a possible problem in SQL jobs, statements and related database objects. This will allow you to answer the question: Does my SQL statement or query have a problem, where, and when?

In the following chapters, we explain how to use SQL Performance Monitors and other tools to determine if your queries should be tuned for performance. In Chapter 12, “Tips to proactively prevent SQL performance problems” on page 407, we provide tips to help you avoid SQL performance problems.

Archived



DB2 for i5/OS performance basics

In this chapter, we introduce some of the basic concepts of SQL performance on DB2 for i5/OS. We discuss the indexing technology on DB2 for i5/OS. We also introduce the query engines on DB2 for i5/OS, Classic Query Engine (CQE) and SQL Query Engine (SQE).

2.1 Basics of indexing

DB2 for i5/OS has two kinds of persistent indexes:

- ▶ *Binary radix tree indexes*, which have been available since 1988
- ▶ *Encoded-vector indexes (EVIs)*, which became available in 1998 with V4R3

Both types of indexes are useful in improving performance for certain kinds of queries. In this section, we introduce this indexing technology and how it can help you in SQL performance.

2.1.1 Binary radix tree indexes

A *radix index* is a multilevel, hybrid tree structure that allows a large number of key values to be stored efficiently while minimizing access times. A key compression algorithm assists in this process. The lowest level of the tree contains the leaf nodes, which house the address of the rows in the base table that are associated with the key value. The *key value* is used to quickly navigate to the leaf node with a few simple binary search tests.

Figure 2-1 shows the structure of a binary radix tree index.

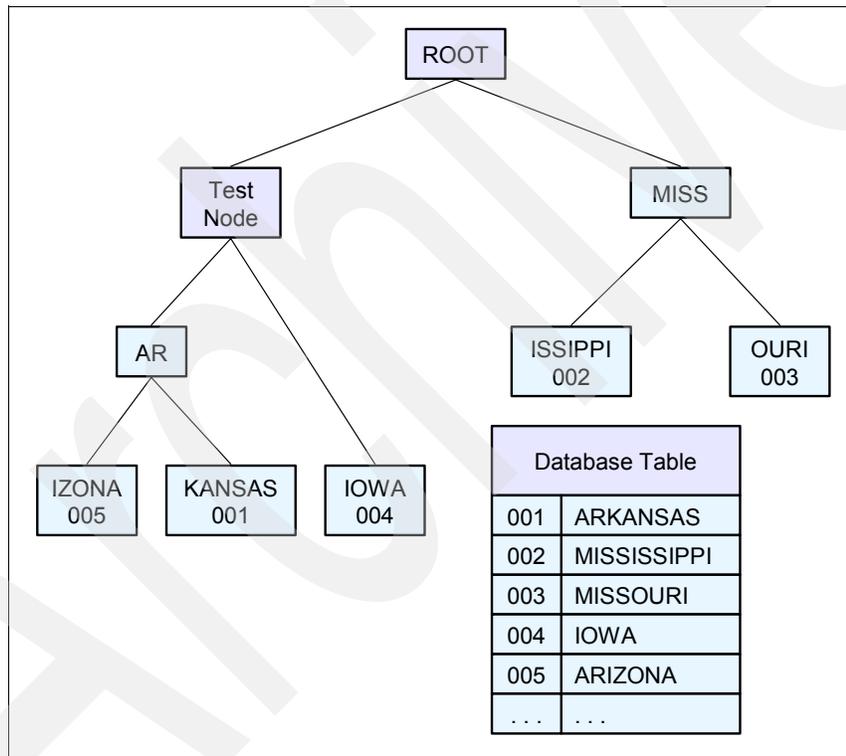


Figure 2-1 Binary radix tree index

Thus, a single key value can be accessed quickly with a small number of tests. This quick access is generally consistent across all key values in the index, since the server keeps the depth of the index shallow and the index pages spread across multiple disk units.

The binary radix tree structure is good for finding a small number of rows because it can find a given row with a minimal amount of processing. For example, using a binary radix index over a customer number column for a typical online transaction processing (OLTP) request, such as “find the outstanding orders for a single customer,” results in fast performance. An index created over the customer number field is considered the perfect index for this type of

query because it allows the database to focus on the rows that it needs and perform a minimal number of I/Os.

2.1.2 Encoded-vector index

To understand EVIs, you should have a basic knowledge of bitmap indexing. DB2 for i5/OS does not create permanent bitmaps. SQL creates dynamic bitmaps temporarily for query optimization.

The need for newer index technologies has spawned the generation of a variety of similar solutions that can be collectively referred to as *bitmap indexes*. A bitmap index is an array of distinct values. For each value, the index stores a bitmap, where each bit represents a row in the table. If the bit is set on, then that row contains the specific key value.

Table 2-1 shows the bitmap representation of an index.

Table 2-1 *Bitmap index*

Key	Bit-Array
Arkansas	10000000110000010110
Arizona	01000100000011010000
...	
Virginia	00000011000000100000
Wyoming	00001000000100000001

With this indexing scheme, bitmaps can be combined dynamically using Boolean arithmetic (ANDing and ORing) to identify only those rows that are required by the query. Unfortunately, this improved access comes with a price. In a very large database (VLDB) environment, bitmap indexes can grow to an ungainly size. For example, in a one billion row table, you might have one billion bits for each distinct value. If the table contains many distinct values, the bitmap index quickly becomes enormous. Usually, relational database management systems (RDBMSs) rely on a type of compression algorithm to help alleviate this growth problem.

An EVI is created using the `CREATE ENCODED VECTOR INDEX` SQL command as shown in Example 2-1.

Example 2-1 *Creating an EVI using the CREATE ENCODED VECTOR INDEX SQL command*

```
CREATE ENCODED VECTOR INDEX MySchema.EVI_Name
  ON MySchema.Table_Name (MyColumn)
  WITH n DISTINCT VALUES
```

An EVI is an index object that is used by the query optimizer and database engine to provide fast data access in decision support and query reporting environments. EVIs are a complementary alternative to existing index objects (binary radix tree structure, logical file, or SQL index) and are a variation of bitmap indexing. Because of their compact size and relative simplicity, EVIs provide faster scans of a table that can also be processed in parallel.

An EVI is a data structure that is stored basically as two components:

- ▶ Symbol table

The symbol table contains a distinct key list, along with statistical and descriptive information about each distinct key value in the index. This table maps each distinct value to a unique code. The mapping of any distinct key value to a 1-, 2-, or 4-byte code

provides a type of key compression. Any key value, of any length, can be represented by a small bytecode. Additional key information, such as first and last row and number of occurrences, helps to get faster access to the data.

► Vector

The vector contains a bytecode value for each row in the table. This bytecode represents the actual key value found in the symbol table and the respective row in the database table. The bytecodes are in the same ordinal position in the vector, as the row it represents in the table. The vector does not contain any pointer or explicit references to the data in the table.

Figure 2-2 shows the components of an EVI.

Symbol Table					Vector	
Key Value	Code	First Row	Last Row	Count	Row Number	Code
Arizona	1	1	80005	5000	1	1
Arkansas	2	5	99760	7300	2	17
...					3	18
Virginia	49	1222	30111	340	4	9
Wyoming	50	7	83000	2760	5	2
					6	7
					7	50
					8	50
					9	1

Figure 2-2 Encoded-vector index

Note: Because the vector represents a relative record number list, an EVI cannot be used to order records. EVIs also have a limited use in joins.

When executing queries that contain joins, grouping, and ordering, a combination of binary radix indexes and EVIs might be used to implement the query. When the selected row set is relatively small, a binary radix index usually performs faster access. When the selected row set is roughly between 20% and 70% of the table being queried, table probe access using a bitmap, created from an EVI or binary radix index, is the best choice.

Also, the optimizer and database engine have the ability to use more than one index to help with selecting the data. This technique might be used when the local selection contains AND or OR conditions, a single index does not contain all the proper key columns, or a single index cannot meet all of the conditions. Single key EVIs can help in this scenario since the bitmaps or relative record number (RRN) lists created from the EVIs can be combined to narrow down the selection process.

Recommendation for EVI use

EVIs are a powerful tool for providing fast data access in decision support and query reporting environments. However, to ensure the effective use of EVIs, you must implement them using the guidelines:

Create EVIs on:

- ▶ Read-only tables or tables with a minimum of INSERT, UPDATE, and DELETE activity
- ▶ Key columns that are used in the WHERE clause: local selection predicates of SQL requests, and fact table join columns when using Star Join Schema support
- ▶ Single-key columns that have a relatively small set of distinct values
- ▶ Multiple-key columns that result in a relatively small set of distinct values
- ▶ Key columns that have a static or relatively static set of distinct values
- ▶ Nonunique key columns, with many duplicates

Create EVIs with the maximum bytecode size expected:

- ▶ Use the WITH n DISTINCT VALUES clause on the CREATE ENCODED VECTOR INDEX statement.
- ▶ If unsure, consider using a number greater than 65535 to create a 4-byte code, avoiding the EVI maintenance overhead of switching bytecode sizes as additional new distinct key values are inserted.

When loading data, keep in mind that:

- ▶ You drop EVIs, load the data, and then create EVIs.
- ▶ EVI bytecode size is assigned automatically based on the number of actual distinct key values found in the table.
- ▶ The symbol table contains all key values, in order; there are no keys in the overflow area.

2.2 Query engines: an overview

Data is the key. Quick and reliable access to business data is critical to making crucial business decisions. A robust database management system (DBMS) has excellent performance capabilities and automated, built-in management and administration functionality. It allows businesses to concentrate on making decisions based on the information contained in their database, rather than managing the database.

Integrated into IBM OS/400® (i5/OS), DB2 for i5/OS has its roots in the integrated relational database of the IBM System/38™, the predecessor of the AS/400 and iSeries servers. Although the database was always relational in nature, native file operations were used to access the data.

With the debut of the AS/400 in 1988 came the introduction of SQL on the platform. SQL is an industry standard (SQL 2003) to define database objects (Data Definition Language (DDL)) and manipulate database data (Data Manipulation Language (DML)). SQL provides an alternative and additional method for accessing data. Both SQL and native methods can coexist. Focusing primarily on OLTP applications, the database has satisfied customer requirements for well over 20 years.

More recently, a new breed of applications started to dominate development efforts. These applications are designed to accommodate rapidly changing business needs and processes. To address the issues and satisfy the demands of the new application world, IBM considered the following options:

- ▶ Continue to enhance the existing product
- ▶ Acquire a new database technology
- ▶ Re-engineer the existing product

The continual enhancement of the product did not seem to be a viable proposition. The increasing development resources required to maintain the existing code resulted in a reduction of resources available to provide new functionality in a timely manner.

Acquiring a new database technology would compromise the basic tenets that distinguish the iSeries from the rest of the industry. These include the integration of the database within OS/400 and the ease-of-use characteristics of the database that minimize administration efforts. Losing these characteristics would significantly reduce the cost of ownership benefits of the iSeries.

Re-engineering the existing product was a more practical solution. However, this could easily become an overwhelming and potentially unsustainable task if an attempt was made to re-engineer the entire product. It could also impact portions of the product that continue to provide solid and efficient support to existing applications and functions.

After considering the options, IBM chose to re-engineer the product. We did so with the added decision to focus only on those aspects of the product for which re-engineering offered the greatest potential. The potential offered the ability to:

- ▶ Support modern application, database, and transactional needs
- ▶ Allow the continued development of database functionality in an efficient and timely manner
- ▶ Maintain and enhance the self-managing value proposition of DB2 for i5/OS
- ▶ Provide a foundation to handle increasingly more complex query environments
- ▶ Improve query performance consistency and predictability
- ▶ Incorporate state-of-the-art techniques

In line with this decision, the query engine was identified as an area that would benefit substantially from such a re-engineering effort. The best current technologies and algorithms, coupled with modern object-oriented design concepts and object-oriented programming implementation, were applied in the redesign of the query engine and its components.

To guarantee existing applications continue to work and to make new or even existing applications profit from the new designed product, IBM decided to implement an additional query engine. The newly redesigned query engine in DB2 for i5/OS is the *SQL Query Engine (SQE)*. The existing query engine is referred to as the *Classic Query Engine (CQE)*. Both query engines coexist in the same system.

The staged implementation of SQE enabled a limited set of queries to be routed to SQE in V5R2 and even more in V5R3. In general, read-only single table queries with a limited set of attributes were routed to SQE but a PTF to V5R2 and V5R3 increased the number. Over time, more queries will use SQE, and increasingly fewer queries will use CQE. At some point, all queries, or at least those that originate from SQL interfaces, will use SQE.

Note: SQE processes queries only from SQL interfaces, such as interactive and embedded SQL, Open Database Connectivity (ODBC) and Java™ Database Connectivity (JDBC™).

2.2.1 Database architecture before V5R2M0

For systems prior to the release of V5R2M0, all database requests are handled by the CQE. Figure 2-3 shows a high-level overview of the architecture of DB2 for i5/OS before OS/400 V5R2. The optimizer and database engine are implemented at different layers of the operating system.

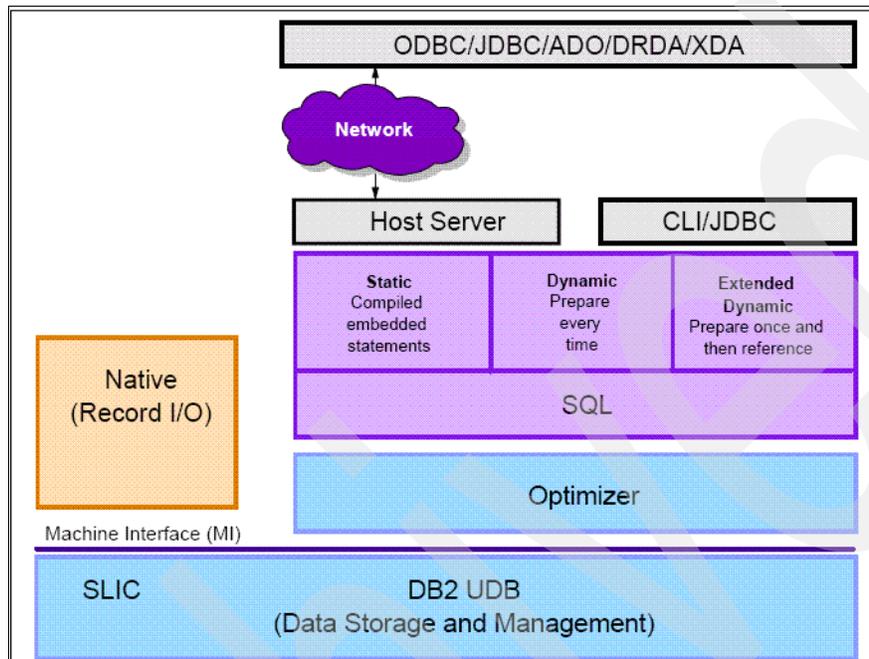


Figure 2-3 Database architecture before the release of V5R2M0: Classic Query Engine

Most CQE query decisions are made above the machine interface (MI) level. In CQE, the interaction between the optimizer and the query execution component occurs across the MI, resulting in interface-related performance overhead.

2.2.2 Current database architecture

With the release of V5R2M0, a new SQE was shipped. SQE and CQE coexist in the same database environment. Depending on the database requests, the Query Dispatcher (refer to 2.2.3, “Query Dispatcher” on page 19) decides to route the query to either the CQE or SQE.

While both the new SQE and the existing CQE can handle queries from start to finish, the redesigned engine simplifies and speeds up queries. In addition to providing the same functionality as CQE, SQE also performs these functions:

- ▶ Moves the optimizer below the MI for more efficient query processing
- ▶ Separates and moves improved statistics to the Statistics Manager dashboard
- ▶ Uses an object-oriented design that accelerates the delivery of new database functionality
- ▶ Uses more flexible, independent data access options to provide autonomous query cruise control
- ▶ Uses enhanced algorithms to provide greater responsiveness and query handling
- ▶ Provides enhanced performance on long-running complex query terrains
- ▶ Retains road maps to provide ease of use in query driving

- ▶ Provides additional and enhanced query feedback and debug information messages through the Database Monitor and Visual Explain interfaces

There are several new and updated components of SQE in OS/400 V5R2 through i5/OS V5R4, including:

- ▶ Query Dispatcher
- ▶ Statistics Manager
- ▶ SQE Optimizer
- ▶ Data Access Primitives
- ▶ SQE Plan Cache

Figure 2-4 shows an overview of the DB2 for i5/OS architecture on i5/OS V5R4 and where each SQE component fits. The functional separation of each SQE component is clearly evident. In line with design objectives, this division of responsibility enables IBM to more easily deliver functional enhancements to the individual components of SQE, as and when required.

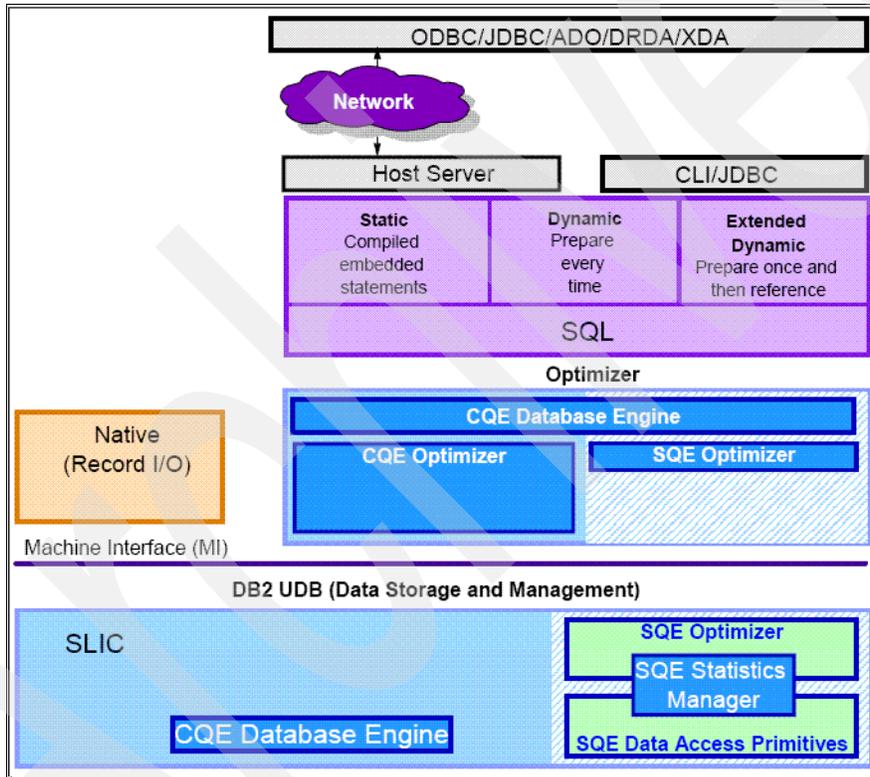


Figure 2-4 Current database architecture: coexisting CQE and SQE

Note: Most of the SQE Optimizer components are implemented below the MI level, which translates into enhanced performance.

Object-oriented design

The SQE query optimizer was implemented using an object-oriented design. It uses a tree-based model of the query, where each node is an independent and reusable component. These components can interact and interface with each other in any given order or combination. Each node can be optimized and executed independently. This design allows greater flexibility when creating new methods for query implementation.

With this new design, the ability to use an index in reverse order can be easily implemented by simply adding a new node. The procedural nature of CQE prevents it from being easily enhanced to read an index in reverse order.

Another example that demonstrates how the object-oriented tree model makes SQE easier to enhance is SQE support for *nonsensical* queries. The term nonsensical describes a query statement that does not return any result rows, for example:

```
Select * from testtable where 1 = 0
```

Surprisingly, many applications use this type of query to force no rows to be returned. Because of the procedural nature of CQE, it is virtually impossible to enhance CQE to recognize the fact that 1 will never equal 0. Therefore, CQE implements this query using a table scan. In contrast, the tree node model of SQE easily allows a node to be added to check for nonsensical predicates before reading any rows in the specified table.

Figure 2-5 shows an example of the node-based implementation used in SQE. In this example, NODE1 represents a typical index probe access method. New nodes to check for nonsensical queries and to index in reverse order are added.

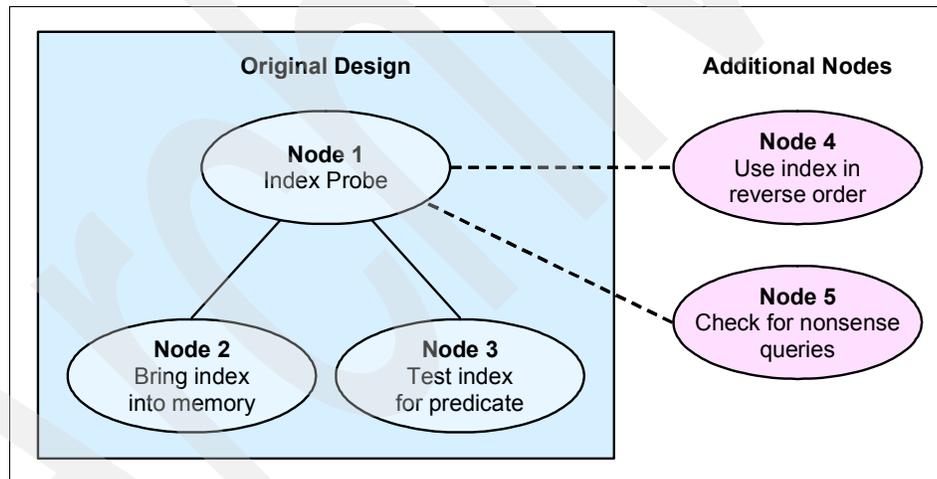


Figure 2-5 Object-oriented design: tree-based model of queries

2.2.3 Query Dispatcher

The function of the Query Dispatcher is to route the query request to either CQE or SQE, depending on the attributes of the query. One of these attributes includes the interface from which the query originates, which is either SQL-based (embedded SQL, ODBC, or JDBC) or non-SQL based (OPNQRYF and Query/400). All queries, irrespective of the interface used, are, therefore, processed by the dispatcher. It is not possible for a user or application program to influence this behavior or to bypass the dispatcher.

Note: Only SQL queries are considered for the SQE. Non-SQL queries, such as, those created by OPNQRYF and Query/400 are *not* SQL-based.

Figure 2-6 illustrates how different database requests are routed to the different query engines.

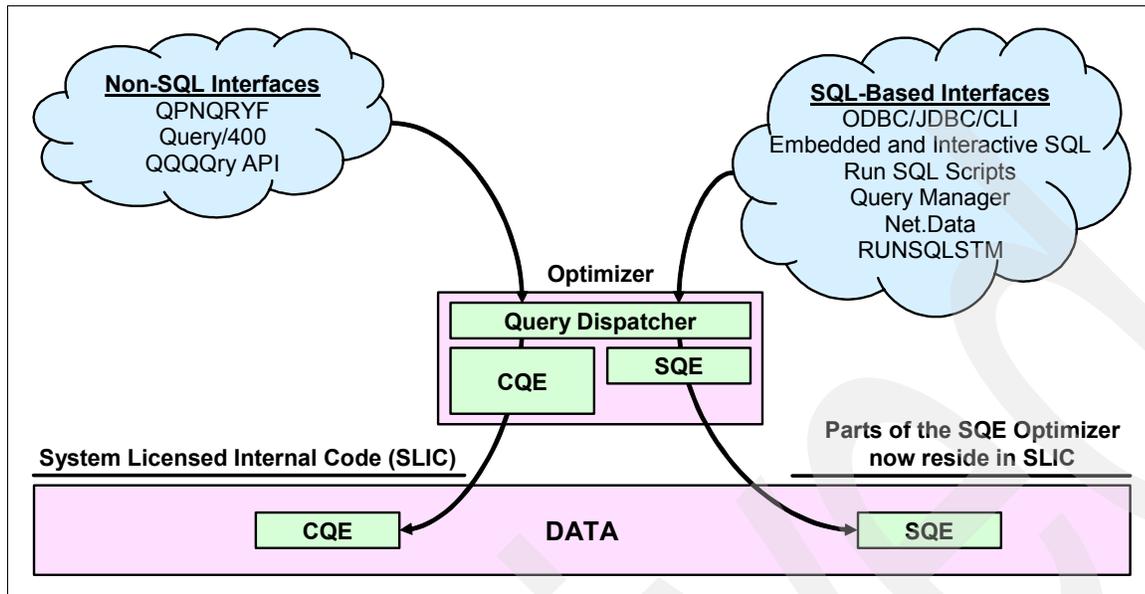


Figure 2-6 Query Dispatcher routing database requests to the query engines

The staged implementation of SQE enabled a limited set of queries to be routed to SQE in V5R2. In general, read-only single-table queries with a limited set of attributes were routed to SQE. With the V5R2 PTF applied (PTF SI07650), the dispatcher routes many more queries through SQE. More single table queries and a limited set of multi-table queries can take advantage of the SQE enhancements. Queries with OR and IN predicates might be routed to SQE with the enabling PTF as are SQL queries with the appropriate attributes on systems with symmetric multiprocessing (SMP) enabled. V5R3 enhanced the V5R2 set of queries by adding, Views, subselects, Common Table Expressions, Derived Table, Unions, UPDATEs/DELETEs/INSERTs and Star Schema joins without QAQQINI overrides.

Note: For more information about PTF SI07650, see Informational APAR II13486 on the Web at:

http://www-912.ibm.com/n_dir/nas4apar.nsf/042d09dd32beb25d86256c1b004c3f9a/61ffe88d56a943ed86256c9b0041fbeb?openDocument

In i5/OS V5R4, a much larger set of queries is implemented in SQE, including those implemented with V5R2 and V5R3, plus many queries with the following types of attributes:

- ▶ LIKE predicates
- ▶ LOB columns
- ▶ ALWCPYDTA(*NO)
- ▶ Sensitive cursor
- ▶ Recursive Common Table Expressions
- ▶ Online analytical processing (OLAP) functions (RANK, ROW NUMBER)

SQL queries that continue to be routed to CQE in i5/OS V5R4 have the following attributes:

- ▶ Table Functions or Lateral correlation
- ▶ Scalar Functions: UPPER, LOWER, or TRANSLATE
- ▶ Scalar Functions: CHARACTER_LENGTH, POSITION, or SUBSTRING with a UTF-8 or UTF-16 argument

- ▶ CCSID character conversion
- ▶ Sort sequence other than *HEX
- ▶ DB2 Multisystem (for example, Distributed Table)
- ▶ References to logical files
- ▶ Tables with select or omit logical files over them

Note: The QAQQINI option IGNORE_DERIVED_INDEX allows SQE to process the query even when a derived key or select/omit index exists over a table in the query. If allowed to run, SQE ignores the derived and select/omit indexes.

Derived keys occur when either the Create Logical File (CRTLF) command's data description specification (DDS) specifies keys that are derivations, for example Substring, or if an NLSS sort sequence is active when the CREATE INDEX SQL statement is performed. By default, if one of these indexes exists over a table in the query, SQE is not allowed to process the query.

The dispatcher also has the built-in capability to reroute an SQL query to CQE that was initially routed to SQE. A query typically reverts to CQE from SQE whenever the optimizer processes table objects that define any of the following logical files or indexes:

- ▶ Logical files with the SELECT/OMIT DDS keyword specified
- ▶ Logical files built over multiple physical file members
- ▶ Nonstandard indexes or derived keys, such as logical files specifying the DDS keywords RENAME or Alternate Collating Sequence (ACS) on a field referenced in the key
- ▶ Sort sequence NLSS specified for the index or logical file

Note: SQL requests that are passed back to CQE from SQE might experience an overhead of up to 10 to 15% in the query optimization time. However, that overhead is not generated every time that an SQL statement is run. After the access plan is built by CQE, the Query Dispatcher routes the SQL request to CQE on subsequent executions. The overhead appears when the access plan is built the first time or rebuilt by the optimizer.

Table 2-2 contains a summary of the information contained in this section. You can find a detailed description by i5/OS version of how the Query Dispatcher routes queries to SQE and CQE in *The Query Dispatcher* section in chapter 2 of the latest version of *Preparing for and Tuning the SQL Query Engine on DB2 for i5/OS*, SG24-6598.

Table 2-2 SEQ Integration Timeline

	V5R2	V5R2*	V5R3	V5R4
Single Table	X	X	X	X
Grouping	X	X	X	X
Distinct	X	X	X	X
Ordering	X	X	X	X
Joins		X	X	X
OR & IN Predicates		X	X	X
SMP			X	X
STAR_JOIN INI Option			X	X
Views/Nested Table Expressions(NTEs) /Common Table Expressions(CTEs)			X	X
Update/Delete			X	X
Subqueries			X	X
Unions			X	X
LIKE Predicates				X
LOBs (Referenced in queries)				X
Sensitive Cursors (ALWCPYDTA(*NO))				X
Lateral Correlations NTEs				X
CCSID/Sort Sequences/Translation				
Select/Omit and Derived Key Index Support (Ignore option via QAQQIN file)				
User Defined Table Functions				
Read Triggers				
Distributed Queries via DB2 Multisystem				
Native Database Logical File references				
Non-SQL Queries (QQQqry API, Query/400, OPNQRYF)				
Note: * V5R2 + SI07650				

2.2.4 Statistics Manager

In releases before V5R2, the retrieval of statistics was a function of the CQE Optimizer. When the optimizer needed to know information about a table, it looked at the table description to retrieve the row count and table size. If an index was available, the optimizer might then extract further information about the data in the table. Figure 2-7 illustrates how CQE relies on indexes for statistics.

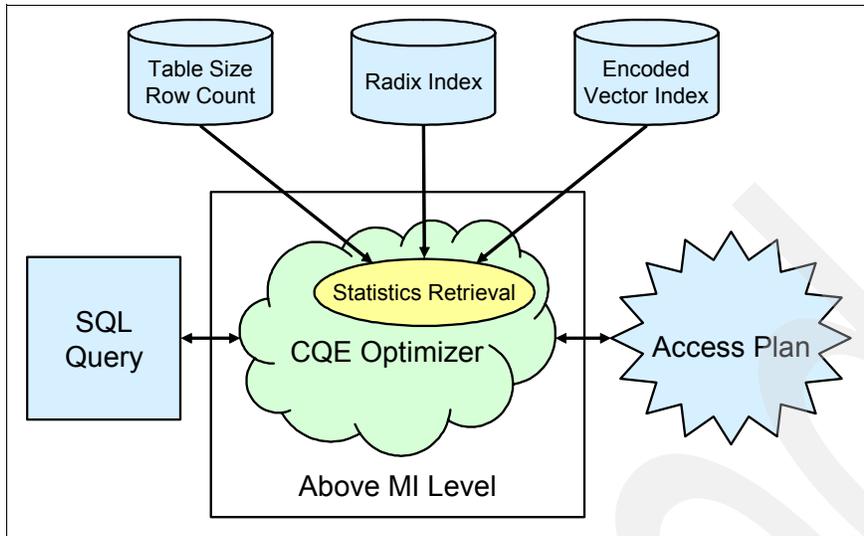


Figure 2-7 CQE Optimizer and Statistics Retrieval

In V5R2, the collection of statistics was removed from the optimizer and is now handled by a separate component called the *Statistics Manager*. The Statistics Manager does not actually run or optimize the query. It controls the access to the metadata and other information that is required to optimize the query. It uses this information to answer questions posed by the query optimizer. The Statistics Manager always provides answers to the optimizer. In cases where it cannot provide an answer based on actual existing statistics information, it is designed to provide a predefined answer.

Note: This new statistical information is used only by the SQE. Queries that are dispatched to the CQE do not benefit from available statistics, nor do they trigger the collection of statistics.

Figure 2-8 shows the new design with Statistics Manager.

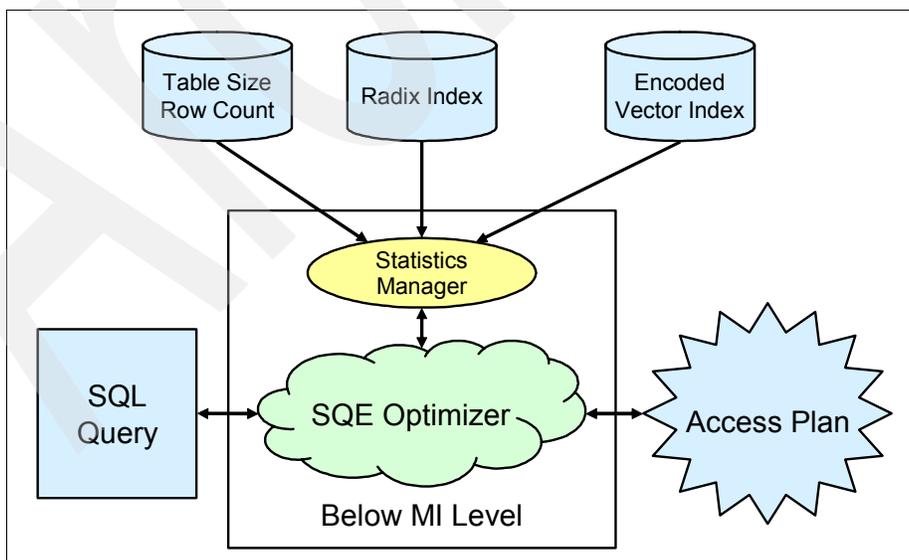


Figure 2-8 SQE Optimizer and Statistics Manager

The Statistics Manager controls the access to the metadata that is required to optimize the query. It uses this information to provide answers to the questions posed by the query optimizer. The Statistics Manager typically gathers and keeps track of the following information:

► **Cardinality of values**

This is the number of unique or distinct occurrences of a specific value in a single column or multiple columns of a table.

► **Selectivity**

Also known as a *histogram*, this information is an indication of how many rows will be selected by any given selection predicate or combination of predicates. Using sampling techniques, it describes the selectivity and distribution of values in a given column of the table.

► **Frequent values**

This is the top *nn* most frequent values of a column together with a count of how frequently each value occurs. This information is obtained by using statistical sampling techniques. Built-in algorithms eliminate the possibility of data skewing. For example, NULL values and default values that can influence the statistical values are not taken into account.

► **Metadata information**

This includes the total number of rows in the table, which indexes exist over the table, and which indexes are useful for implementing the particular query.

► **Estimate of I/O operation**

This is an estimate of the number of I/O operations that are required to process the table or the identified index.

You can obtain the majority of this information from existing binary-radix indexes or encoded-vector indexes. An advantage of using indexes is that the information is available to the Statistics Manager as soon as the index is created or maintained.

2.2.5 SQE Optimizer

Like the CQE Optimizer, the SQE Optimizer controls the strategies and algorithms that are used to determine which data access methods should be employed to retrieve the required data. Its purpose is to find the best method to implement a given query.

A fundamental characteristic distinguishes the SQE Optimizer from the CQE Optimizer. The SQE Optimizer gains access to the statistical data collected by the Statistics Manager, by simply asking questions related to the system and the tables used in the query. Based on this information, the access method is determined based on the lowest cost of CPU utilization and I/O utilization.

Because most of the SQE Optimizer functionality is implemented beneath the MI and is consequently closer to the data, the database management system provides greater flexibility and increased performance. In addition the SQE is able to utilize the constraint definitions from the database to rewrite queries in a more efficient way. For example Referential Integrity constraints may allow the optimizer to rewrite queries to minimize the number of joins required to satisfy the SQL request.

The CQE Optimizer uses a clock-based time-out algorithm. The CQE Optimizer re-sequences the priority of the indexes, based on the number of matching index columns and operators used in the WHERE clause of the SQL statement. This approach ensures that the most efficient indexes are optimized first, before the set time limit expires.

In contrast, the amount of time that the SQE Optimizer spends optimizing an access plan is unlimited. A check is done to determine if any indexes exist on the table, with keys built over the columns specified in WHERE or SELECT clauses of the SQL statement. These indexes are then re-sequenced so that the most appropriate indexes are processed first and reorganized further based on index-only access, index probe selectivity, total index selectivity, and the size of the index keys.

Note: For SQE, the indexes are ordered for each query's selection so that the indexes that access the smallest number of entries are examined first. For CQE, the indexes are generally ordered from mostly recently created to oldest.

2.2.6 Data Access Primitives

The basic function of SQE Data Access Primitives is to implement the query. Using the data access methods derived from the object-oriented, tree-based architecture, Data Access Primitives provide the implementation plan of the query.

The SQE Data Access Primitives have a more aggressive approach to the use of System i resources, such as, main storage and subsystem resources. Because the SQE data access algorithms are more in tune with i5/OS single-level storage, SQE data access plans use table scans more frequently than CQE.

Although the SQE Data Access Primitives use a number of the data access methods found in CQE, the data access methods have undergone dramatic re-engineering to take advantage of the System's architectural strengths.

2.2.7 Access plan

An access plan is the method used for a specific SQL statement to get access to the data. If the access plan does not exist, it is created the first time that an SQL statement is executed. If the access plan already exists, it is compared with the information provided by the Statistics Manager (SQE) or by the query optimizer in CQE. If the optimizer decides to use another access path, the access plan is updated.

If you use SQL statements in programs, there are different ways to embed, prepare, and execute your SQL statements. These different methods affect the creation time of the access plan for the specified SQL statements. All executable SQL statements must be prepared before they can be executed. The result of preparation is the executable or operational form of the statement. We can differentiate between the three methods:

► Static SQL

Static SQL statements are embedded in the source code of a host application program. These host application programs are written in high-level languages, such as COBOL or RPG. The host application source code must be processed by an SQL pre-compiler before compiling the host application program itself. The SQL pre-compiler checks the syntax of the embedded SQL statements and turns them into host language statements that interface with the database manager upon execution of the program. This process is often referred as *binding*.

The SQL statements are therefore prepared before running the program and the associated access plan persists beyond the execution of the host application program.

In static SQL, the SQL statements that must be executed are already known at compile time. The precompiler checks the syntax and converts the SQL statement into an executable form, as well as creates an access plan that is embedded into the program object. If the access plan is changed because of an altered memory pool or the creation of

new indexes, the access plan is updated in the program object. In this way, a program object can grow over time, even if no modifications are performed.

► **Dynamic SQL**

Dynamic SQL statements are prepared at the time an SQL application is executed. The SQL statements are passed to the database manager in the form of a character string. This string uses interfaces with the PREPARE and EXECUTE statements or an EXECUTE IMMEDIATE type of statement.

Programs that contain embedded dynamic SQL statements must be precompiled like those that contain static SQL. Unlike static SQL, the dynamic SQL statements are checked, constructed, and prepared at run time. The source form of the statement is a character or graphic string that is passed to the database manager by the program that is using the static SQL PREPARE or EXECUTE IMMEDIATE statement. The operational form of the statement persists for the duration of the connection or until the last SQL program leaves the call stack. Access plans associated with dynamic SQL might not persist after a database connection or job is ended.

► **Extended dynamic SQL**

An extended dynamic SQL statement is neither fully static nor fully dynamic. The Process Extended Dynamic SQL (QSQPRCED) API provides users with extended dynamic SQL capability. Like dynamic SQL, statements can be prepared, described, and executed using this API. Unlike dynamic SQL, the plan is stored in the package by this API and it persists until the package or statement is explicitly dropped.

The iSeries Access ODBC driver and toolbox JDBC driver both have extended dynamic SQL options available. They interface with the QSQPRCED API on behalf of the application program.

2.2.8 SQL packages

SQL packages are permanent objects with the object type *SQLPKG used to store information related to prepared, extended dynamic SQL statements. They can be used by the iSeries Access for Windows® ODBC driver and the IBM Toolbox for Java JDBC driver. They are also used by applications which use the QSQPRCED API interface.

Note: SQL Packages are not used for static SQL on i5/OS like in other DB2 databases.

The SQL package contains all the necessary information to execute the prepared statement. This includes registry of the statement name, the statement text, the internal parse tree for the statement, definitions of all the tables and fields involved in the statement, and the query access plan needed to access the tables at run time.

Note: When using embedded SQL, no separate SQL package is created, but the access plan is integrated into the program or service program object.

Creation time of SQL packages

In the case of ODBC and JDBC, the existence of the package is checked when the client application issues the first prepare of an SQL statement. If the package does not exist, it is created at that time, even though it might not yet contain any SQL statements. In the case of QSQPRCED, creation of the package occurs when the application calls QSQPRCED specifying function 1.

Advantages of SQL packages

Because SQL packages are a shared resource, the information built when a statement is prepared is available to all the users of the package. This saves processing time, especially in an environment when many users are using the same or similar statements. Because SQL packages are permanent, this information is also saved across job initiation or termination and across initial program loads (IPLs). In fact, SQL packages can be saved and restored on other systems. By comparison, dynamic SQL requires that each user go through the prepare processing for a particular statement and do this every time the user starts the application.

SQL packages also allow the system to accumulate statistical information about the SQL statements. Accumulating such information results in better decisions about how long to keep cursors open internally and how to best process the data needed for the query. As indicated previously, this information is shared across users and retained for future use. In the case of dynamic SQL, every job and every user must relearn this information.

2.2.9 SQE Plan Cache

The SQE Plan Cache is a repository that contains query implementation plans for queries optimized by the SQE Optimizer. Query access plans generated by CQE are not stored in the Plan Cache. The architecture of DB2 for i5/OS allows for only one Plan Cache per iSeries server or logical partition (LPAR).

Access plans generated by CQE are not stored in the SQE Plan Cache; instead, they are stored in SQL Packages, the system-wide statement cache, and job cache. The purposes of the SQE Plan Cache are to:

- ▶ Facilitate the reuse of a query access plan when the same query is re-executed
- ▶ Store runtime information for subsequent use in future query optimizations

Once an access plan is created, it is available for use by all users and all queries, regardless of where the query originates. Furthermore, when an access plan is tuned, when creating an index for example, all queries can benefit from this updated access plan. This eliminates the need to reoptimize the query, resulting in greater efficiency.

Before optimizing an incoming query, the optimizer looks for the query in the plan cache. If an equivalent query is found, and the associated query plan is found to be compatible with the current environment, the already-optimized plan is used, avoiding full optimization.

Figure 2-9 shows the concept of reusability of the query access plans stored in the SQE Plan Cache. The SQE Plan Cache is interrogated each time a query is executed using SQE.

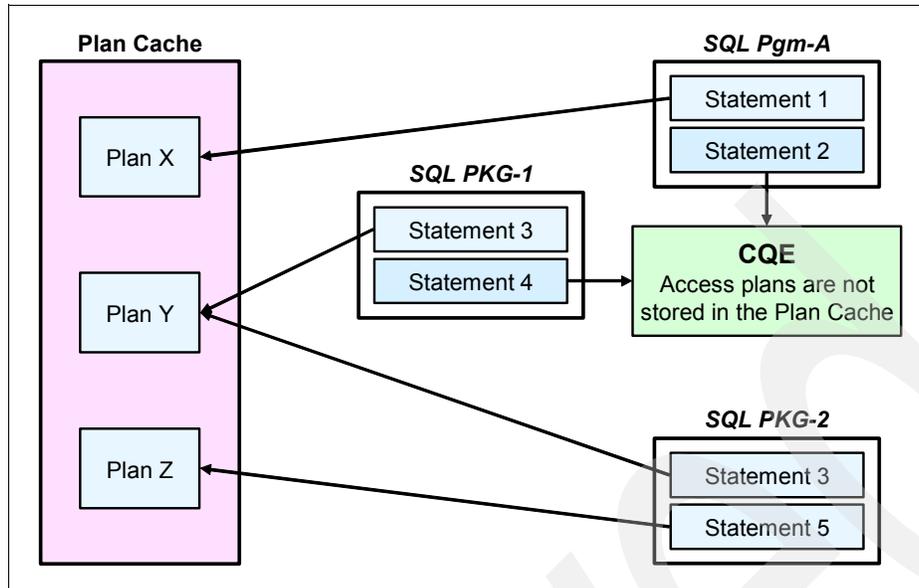


Figure 2-9 Plan Cache

In addition, unlike CQE, SQE can save multiple different plans for the same query. This method is useful in more dynamic environments where the plan changes depending on user inputs, available memory, and so on. If a valid access plan is found, it is used to implement the query. Otherwise a new access plan is created and stored in the SQE Plan Cache for future use. However, access plans generated by CQE are not stored in the SQE Plan Cache; instead, they are stored in SQL Packages, the system-wide statement cache, and job cache.

To illustrate this concept, assume that Statement 2 in Figure 2-9 is executed for the first time by SQE. The access plan for Statement 2 is stored in the SQE Plan Cache. Statement 4 is issued by CQE. It is not stored in the SQE Plan Cache. It can, however, be stored in the SQL Package.

The SQE Plan Cache is automatically updated with new query access plans as they are created, or is updated for an existing plan (the next time the query is run) when new statistics or indexes become available. The SQE Plan Cache is also automatically updated by the database with runtime information as the queries are run. It is created with an overall size of 512 Megabytes (MB). Each SQE Plan Cache entry contains the original query, the optimized query access plan and cumulative runtime information gathered during the runs of the query. In addition, several instances of query runtime objects are stored with a SQE Plan Cache entry. These runtime objects are the real executable and temporary storage containers (hash tables, sorts, temporary indexes, and so on) used to run the query. All systems are currently configured with the same size SQE Plan Cache, regardless of the server size or the hardware configuration.

When the SQE Plan Cache exceeds its designated size, a background task is automatically scheduled to remove plans from the SQE Plan Cache. Access plans are deleted based upon the age of the access plan, how frequently it is being used and how much cumulative resources (CPU/IO) were consumed by the runs of the query. The total number of access plans stored in the SQE Plan Cache depends largely upon the complexity of the SQL statements that are being executed. In certain test environments, there have been typically between 10,000 to 20,000 unique access plans stored in the SQE Plan Cache. The SQE Plan Cache is cleared when a system Initial Program Load (IPL) is performed.

Multiple access plans can be maintained for a single SQL statement. Although the SQL statement itself is the primary hash key to the SQE Plan Cache, different environmental settings can cause different access plans to be stored in the SQE Plan Cache. Examples of these environmental settings include:

- ▶ Different SMP Degree settings for the same query
- ▶ Different library lists specified for the query tables
- ▶ Different settings for the job's share of available memory in the current pool
- ▶ Different ALWCPYDTA settings

Currently, the SQE Plan Cache can maintain a maximum of three different access plans for the same SQL statement. As new access plans are created for the same SQL statement, older access plans are discarded to make room for the new access plans. There are, however, certain conditions that can cause an existing access plan to be invalidated. Examples of these include:

- ▶ Specifying REOPTIMIZE_ACCESS_PLAN(*YES) or (*FORCE) in the QAQQINI table or in the SQL Script
- ▶ Deleting or recreating the table that the access plan refers to
- ▶ Deleting an index that is used by the access plan

Note: There is a separate SQE Plan Cache for each IASP for the system, so varying the IASP also causes the SQE Plan Cache to be cleared.

2.2.10 Open Data Path

When a full Open is required, a path is created at runtime to get the data in and out. This path, called an *open data path*, is the path through which all of the read and write operations for the table and file are performed. ODPs provide a way for more than one program in the same job to share the same file status information (I/O feedback areas), file pointer positions, and storage area.

A full open creates an Open Data Path (ODP) that will be then be used to fetch, update, delete, or insert rows. Since there will typically be many fetch, update, delete, or insert operations for an ODP, as much processing of the SQL statement as possible is done during the ODP creation so that the same processing does not need to be done on each subsequent I/O operation. An ODP may be cached at close time so that if the SQL statement is run again during the job, the ODP will be reused. Such an open is called a pseudo-open and is much less expensive than a full open. ODPs are quite useful as they can improve performance, reduce the amount of main storage needed by the job, and reduces files opens/closes.

For more information about SQE and CQE, see the latest version of the IBM Redbook *Preparing for and Tuning the SQL Query Engine on DB2 for i5/OS*, SG24-6598.

Also refer to DB2 Universal Database for iSeries Database Performance and Query Optimization, which is available in the iSeries Information Center on the Web at:

<http://publib.boulder.ibm.com/infocenter/series/v5r4/topic/rzajq/rzajq.pdf>

Archived

Gathering, analyzing, and querying database performance data

In this part of the book, we describe and discuss the different ways to gather database performance data. After we explain how to gather this data, we describe the different ways to analyze it with the tools that DB2 for i5/OS has. Later, we show you how to query the Database performance data and how to tie this data to one of the preferred tools called *Visual Explain*. We also dedicate some chapters to the new tools provided by i5/OS in V5R4. At the end, we present a methodology to do Performance Analysis.

This part contains the following chapters:

- ▶ Chapter 3, “Overview of tools to analyze database performance” on page 33
- ▶ Chapter 4, “Gathering SQL performance data” on page 89
- ▶ Chapter 5, “Analyzing SQL performance data using iSeries Navigator” on page 117
- ▶ Chapter 6, “Custom Database Monitor Analysis” on page 173
- ▶ Chapter 7, “SQE Plan Cache and SQE Plan Cache Snapshots” on page 237
- ▶ Chapter 8, “Analyzing database performance data with Visual Explain” on page 275
- ▶ Chapter 9, “Index Advisor” on page 319
- ▶ Chapter 10, “SQL performance analysis: a methodology” on page 363

Archived



Overview of tools to analyze database performance

Database performance is a high priority in any system. The objective is to maximize system resource utilization, while achieving maximum performance throughput. Therefore, analyzing your queries is the most important step to ensure that they are tuned for optimal performance.

You must select the proper tools for collecting and analyzing the database performance data first to ensure that your queries are tuned for optimal performance.

V5R4 DB2 for i5/OS has new tools to collect and analyze database performance. Furthermore, some existing tools were enhanced in order to offer you more accuracy and productivity during the analysis, troubleshooting or optimization of your queries.

In this chapter, we introduce and provide information about the tools for monitoring and analyzing the database performance data of your SQL queries.

3.1 Introduction to the tools

By using the following analysis tools to obtain information through the monitoring processes, you should be able to take the appropriate corrective actions:

- ▶ SQE Plan Cache (V5R4)
- ▶ SQE Plan Cache Snapshots (V5R4)
- ▶ Detailed Database Monitor(enhanced in V5R4)
- ▶ Summary Database Monitor
- ▶ Visual Explain - (enhanced in V5R4)
- ▶ Index Advisor(V5R4)
- ▶ Evaluators
- ▶ Current SQL for a Job
- ▶ Debug Messages
- ▶ PRTSQLINF

Every time an SQL statement goes through the query optimization process, there is feedback from the optimizer. In Figure 3-1 we illustrate the different types of feedback produced by the optimizer. We also show how to get more details about specific queries by drilling down directly to Visual Explain, from SQE Plan Cache Snapshot and from Detailed Database Monitor as well. We can also drill down to Visual Explain.

Note: Access plans generated by CQE are not stored in the SQE Plan Cache. Instead, they are stored in SQL Packages, program objects, the system-wide statement cache, and job cache.

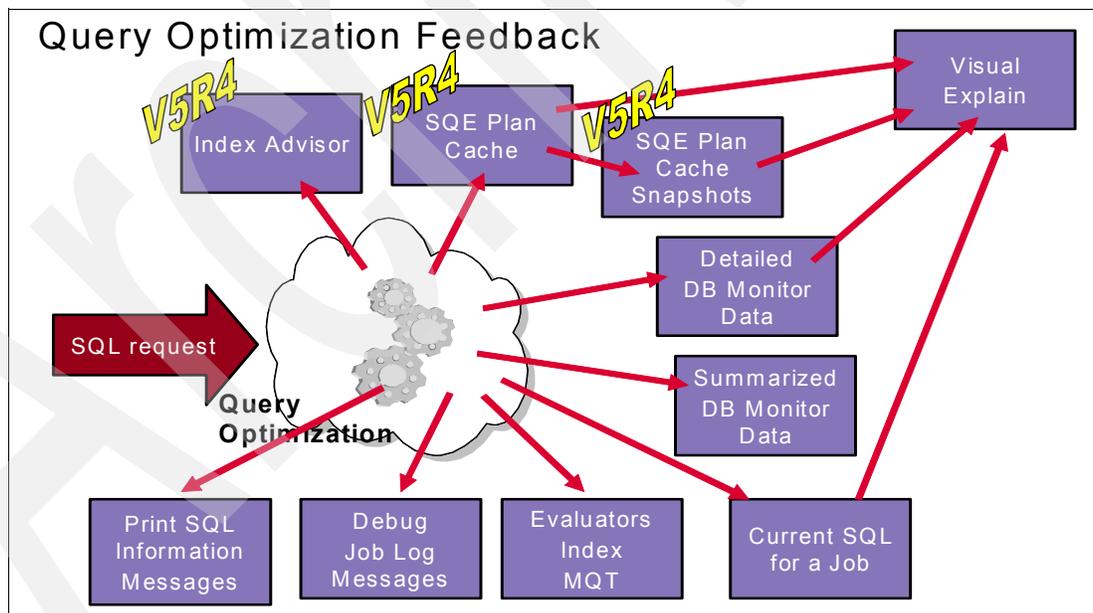


Figure 3-1 Query Optimization feedback

The tools for monitoring and analyzing the database performance data of your SQL queries are based on the above types of feedback.

3.2 SQE Plan Cache

Since the new re-engineered SQL Query Engine (SQE) was introduced in DB2 UDB for iSeries V5R2, it has been enhanced with additional capabilities in every subsequent release. In V5R3, SQE maintains an internal cache of all access plans (called SQE Plan Cache) created for all SQL statements that SQE serves. The Plan Cache helps improve SQL runtime when statements are executed repeatedly. The SQE Plan Cache is always on, automatic and has no Database Monitor Overhead. Although the SQE Plan Cache is activated automatically after the iSeries finishes an IPL, it cannot be accessed by users in V5R3 and is cleared at every system restart.

Note: Access plans generated by CQE are not stored in the SQE Plan Cache. Instead, they are stored in SQL packages, program objects, the system-wide statement cache, and job cache.

The purposes of the SQE Plan Cache are to:

- ▶ Facilitate the reuse of a query access plan when the same query is re-executed.
- ▶ Store runtime information for subsequent use in future query optimizations.
- ▶ Supply most of the information that a database monitor has without the overhead and disk space that is used by the database monitor.
- ▶ Locate long running queries and drill down into Visual Explain in order to analyze specific access plans.

Once an access plan is created, it is available for use by all users and all queries regardless of where the query originates. Furthermore, when an access plan is tuned, after creating an index for example, all queries can then benefit from this updated access plan. This eliminates the need to re-optimize the query, resulting in greater efficiency.

In V5R4, you are now able to access through iSeries Navigator the SQE Plan Cache. This feature enables you to identify areas for performance improvement of your SQL statements that are served by SQE.

iSeries Navigator V5R4 provides you with a newly-designed graphical user interface (GUI) that accommodates your efforts to analyze detailed performance-related information from SQE as well as its SQL Performance Monitor. You will still need to use SQL Performance Monitor (available since V5R1) for the analysis of SQL performance data maintained by the Classic Query Engine (CQE).

In order to see the properties of the SQE Plan Cache in iSeries Navigator, expand **Databases** → **The Database name**. Select and right-click **SQL Plan Cache Snapshot** icon. Select **SQL Plan Cache** → **Properties** menu items as shown in Figure 3-2.

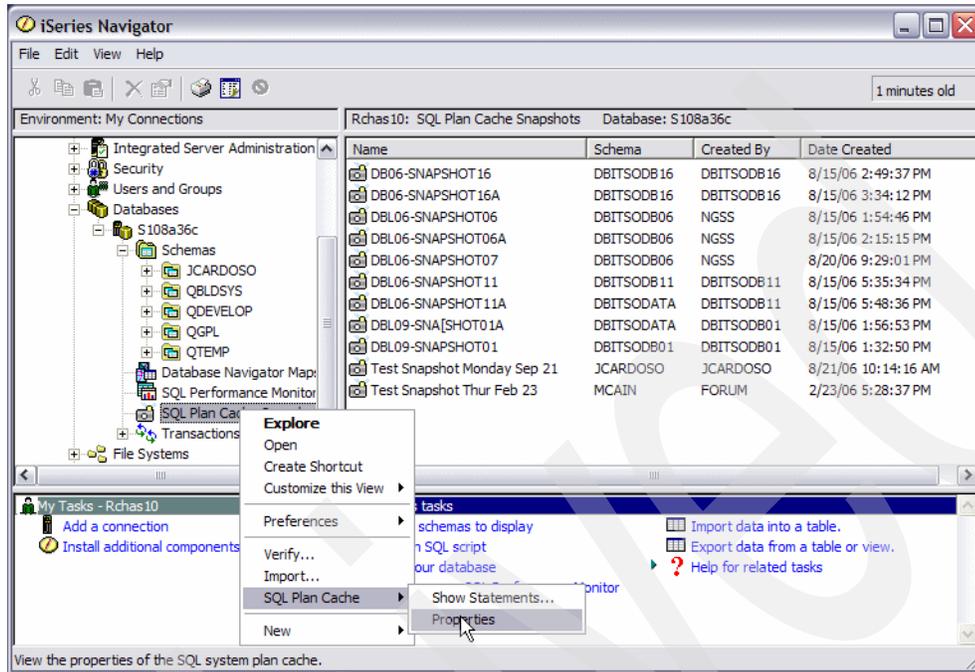


Figure 3-2 How to get SQE Plan Cache Properties

The Properties window shown in Figure 3-3 opens.

As shown in Figure 3-3 SQE Plan Cache Properties window shows the following:

- ▶ Time of Summary
- ▶ Active Query Summary
 - Number of Currently Active Queries
 - Number of Queries Run Since Start
 - Number of Query Full Opens Since Start
- ▶ Plan Usage Summary
 - Current Number of Plans in Cache
 - Current SQE Plan Cache Size
 - SQE Plan Cache Size Threshold

Description	Value
Time Of Summary	2006-08-21-10.23.07.66551
Active Query Summary	
Number of Currently Active Queries	43
Number of Queries Run Since Start	36230
Number of Query Full Opens Since Start	7186
Plan Usage Summary	
Current Number of Plans in Cache	716
Current Plan Cache Size	104 MegaBytes
Plan Cache Size Threshold	512 MegaBytes

Figure 3-3 SQE Plan Cache Properties

Important: As you can see in the SQL Plan Cache Properties window, the Plan Cache size Threshold is 512 MegaBytes. Therefore, keep checking the Current Plan Cache Size.

The SQE Plan Cache maintains the access plans with an algorithm that ensures that frequently used access plans are kept for efficient reuse. The least-used access plans may be wiped out if its storage is nearly full to make room for those which are used more often. Therefore, you may not always see all access plans that SQE serves in its plan cache. The number of access plans maintained in the Plan Cache may be much lower than the number of SQE SQL statements run on the system because many statements can reuse the same access plans.

The Plan Cache clears with IPL.

You can View the Contents of the SQE Plan Cache by right-clicking on the **SQL Plan Cache Snapshot** option and selecting **SQL Plan Cache** → **Show Statements** as shown in Figure 3-4.

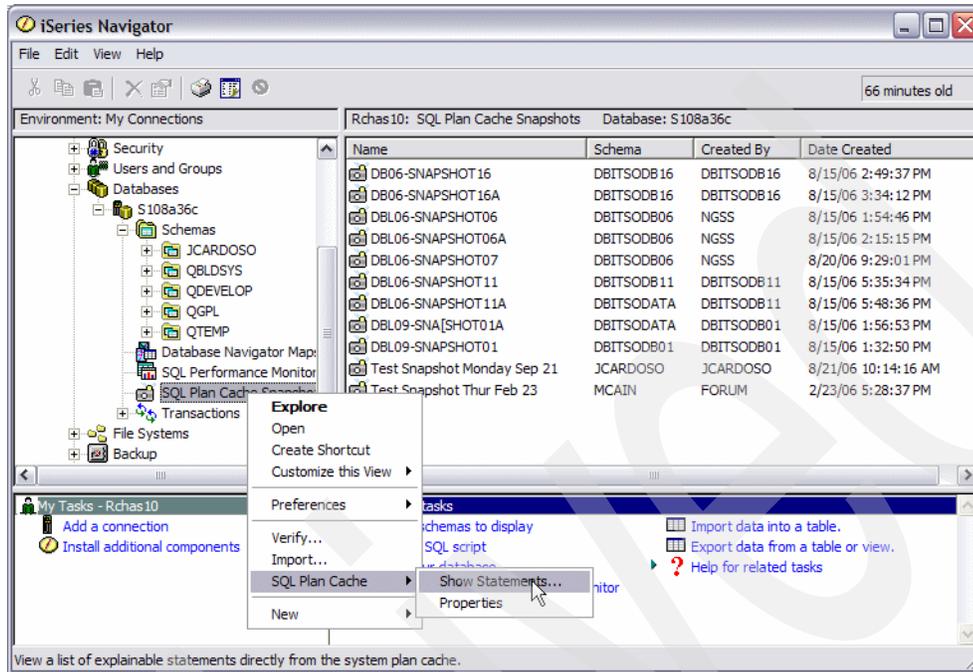


Figure 3-4 How to show statements of SQE Plan Cache

The SQL Plan Cache Statements window opens as shown in Figure 3-5.

Note: The list of statements is initially empty. Read the Notice message in the right window.

The list will appear after you choose the filters and press the **Retrieve** button.

In the new SQL Plan Cache Statements window that appears, the left pane shows the following filtering options that you can use:

- ▶ Minimum runtime for the longest execution
- ▶ Queries run after a specified date and time
- ▶ Top 'n' most frequently run queries
- ▶ Top 'n' queries with the largest total accumulated runtime
- ▶ Queries ever run by a specified user
- ▶ Queries currently active
- ▶ Queries with index advised
- ▶ Queries with statistics advised
- ▶ Include queries initiated by operating system
- ▶ Queries that use or reference specified tables
- ▶ SQL statements that contain a specified text string

You can use any combination of these filtering options that serve your interest.

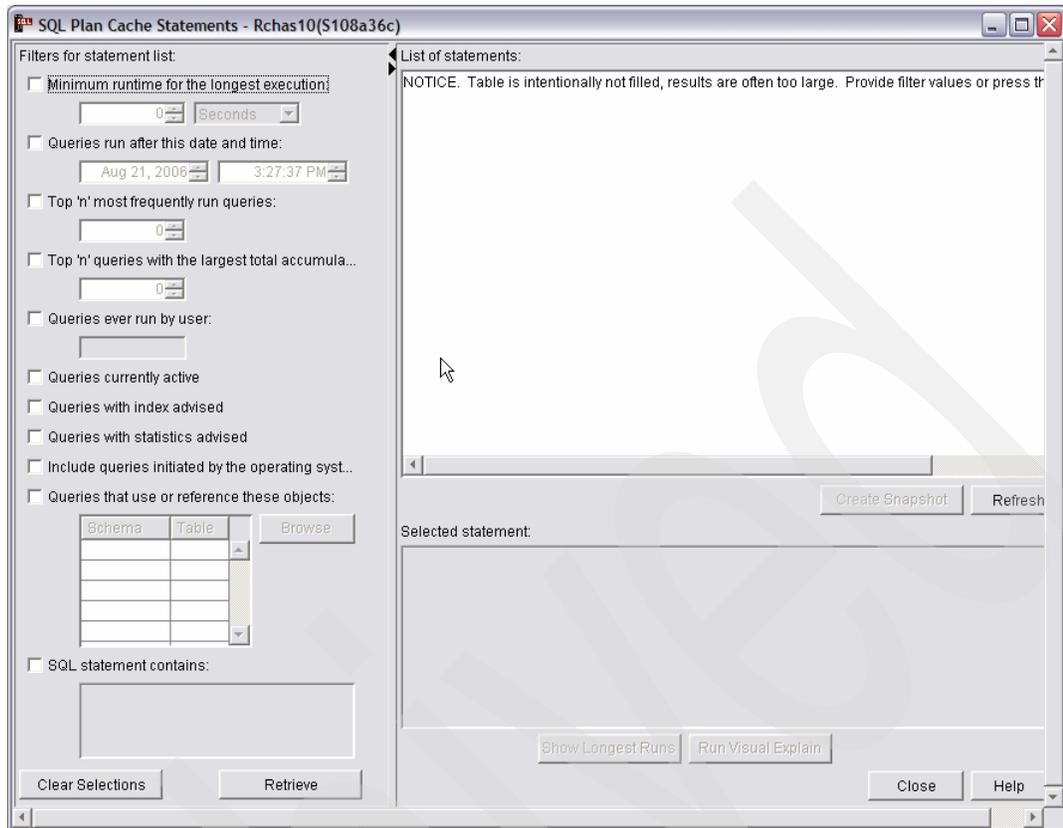


Figure 3-5 SQL Plan Cache Statements window

Important: SQE Plan Cache exists on V5R2, V5R3 and V5R4 release versions. The SQE Plan Cache GUI Interfaces in iSeries Navigator are available beginning in V5R4.

Restriction: Access plans generated by CQE are not stored in the SQE Plan Cache. Instead, they are stored in SQL Packages, program objects, the system-wide statement cache, and job cache.

Refer to Chapter 7, “SQE Plan Cache and SQE Plan Cache Snapshots” on page 237 for more details.

3.3 SQE Plan Cache Snapshots

The SQE Plan Cache is cleared when you do an IPL. This means that once you IPL the system you lose all the contents of the SQE Plan Cache. For this reason it is nice to have a method to save a copy of the contents of the SQE Plan Cache in a separate place. SQE Plan Cache snapshots are images of the contents of the SQE Plan Cache. Its data is not Volatile. SQE Plan Cache Snapshots cause no overhead. Its data capture is part of normal query optimization and execution. The data of an SQE Plan Cache Snapshot is placed into a single DB2 table.

Also keep in mind that the contents of the SQE Plan Cache are volatile and dynamic. Up to 3 access plans of the same SQL statement are kept in the Plan Cache. For this reason if lots of

concurrent SQL users are executing the same SQL Statement, an SQL statement executed by an user 15 minutes ago may no longer have an entry in the Plan Cache. This is another reason why the Plan Cache Snapshots are important.

You can create an SQE Plan Cache Snapshot from the following interfaces:

- ▶ SQL Plan Cache Snapshot icon of iSeries Navigator
 - In this interface, you have the filtering capability when you are doing the dump of the Plan Cache.
- ▶ SQL interface (CALL QSYS2/DUMP_PLAN_CACHE)
 - Be aware that there is no filtering when using this stored procedure to the Plan Cache dump.

SQE Plan Cache Snapshots have very interesting analysis methods such as “before” and “after” comparisons (For details, refer to Chapter 7, “SQE Plan Cache and SQE Plan Cache Snapshots” on page 237).

Note: SQE Plan Cache Snapshots support only SQE SQL Statements. Remember that SQE Plan Cache Snapshots are based on the SQE Plan Cache which does not support CQE SQL Statements.

As shown in Figure 3-6, create a snapshot of an SQE Plan Cache by performing the following steps:

1. From the main iSeries Navigator window, expand the **Databases** container. Then, move down to right-click **SQL Plan Cache Snapshots** and select **New** → **Snapshot** menu items.

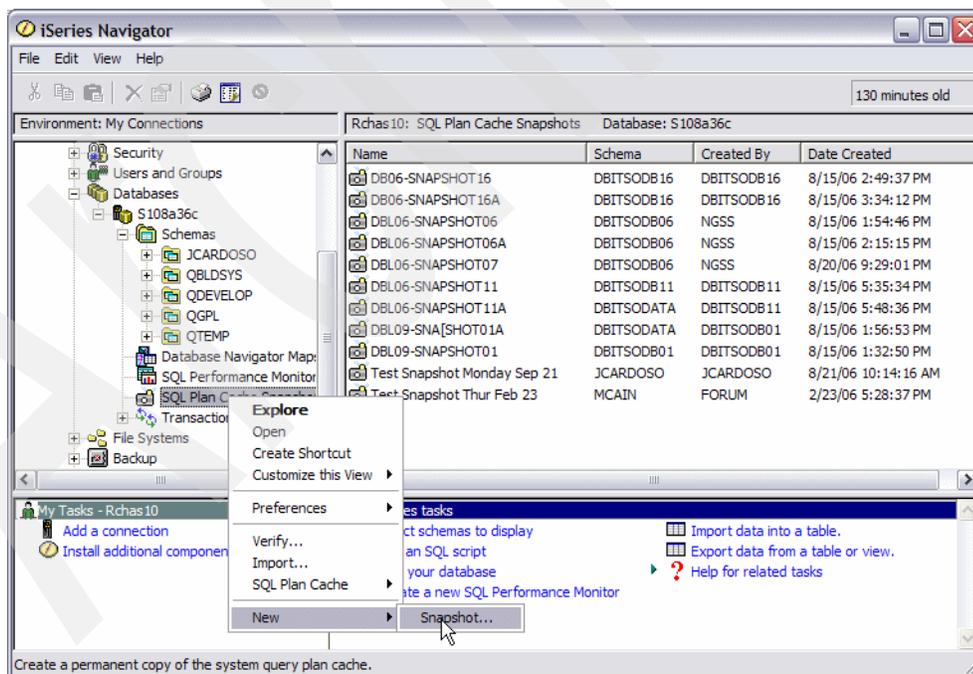


Figure 3-6 Creating a snapshot of SQE Plan Cache

A New Snapshot of SQL Plan Cache window appears as shown in Figure 3-7.

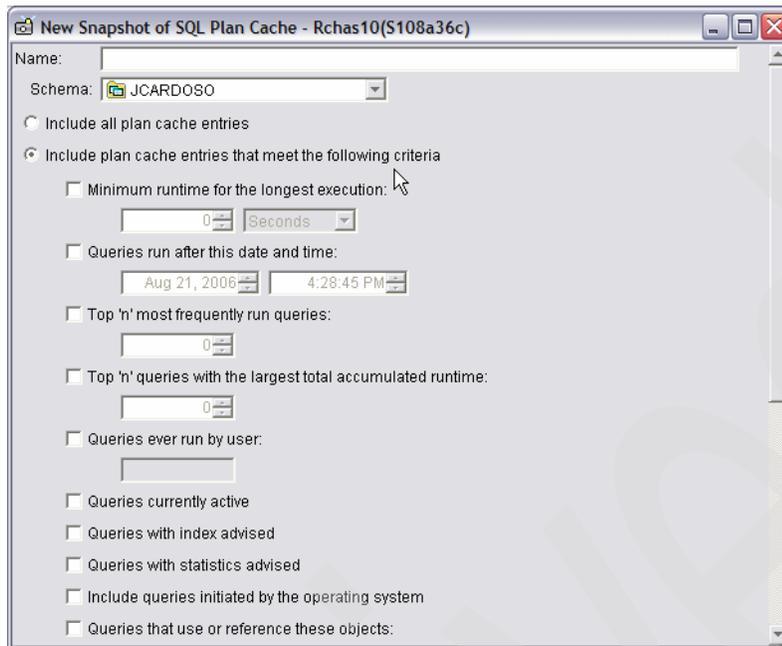


Figure 3-7 New Snapshot of SQL Plan Cache window

Note: Filtering criteria for SQE Plan Cache snapshot creation:

In many occasions, you may want to create a snapshot with only some specific filtering criteria. You can see that the option named “Include plan cache entries that meet the following criteria” provides up to 11 filtering criteria for you to use:

- ▶ Minimum runtime for the longest execution - for filtering out fast statements
- ▶ Queries run after a specific date and time
- ▶ Top ‘n’ most frequently run queries
- ▶ Top ‘n’ queries with the largest total accumulated runtime
- ▶ Queries ever run by a specific user
- ▶ Queries currently active
- ▶ Queries with index advised
- ▶ Queries with statistics advised
- ▶ Include queries initiated by the operating system
- ▶ Queries that use or reference specific tables
- ▶ SQL statements that contain specific syntax

Remember that these criteria only apply to those SQL statements that are served by SQE only. For SQL statements served by CQE, you need to start SQL Performance Monitor that is also enhanced in V5R4 with filtering criteria specification.

- After making your filtering choices, click the **OK** button. To analyze your SQE Plan Cache Snapshot, right-click and select **Analyze** as shown in Figure 3-8.

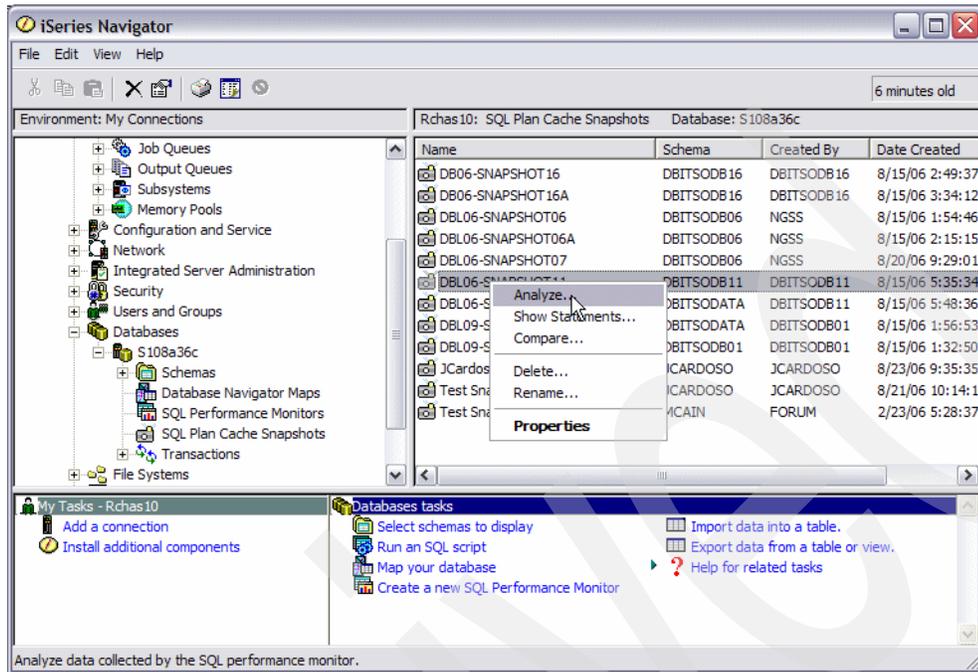


Figure 3-8 Selecting Analyze menu item of an SQE Plan Cache Snapshot

When you select **Analyze**, the dashboard will appear as shown in Figure 3-9.

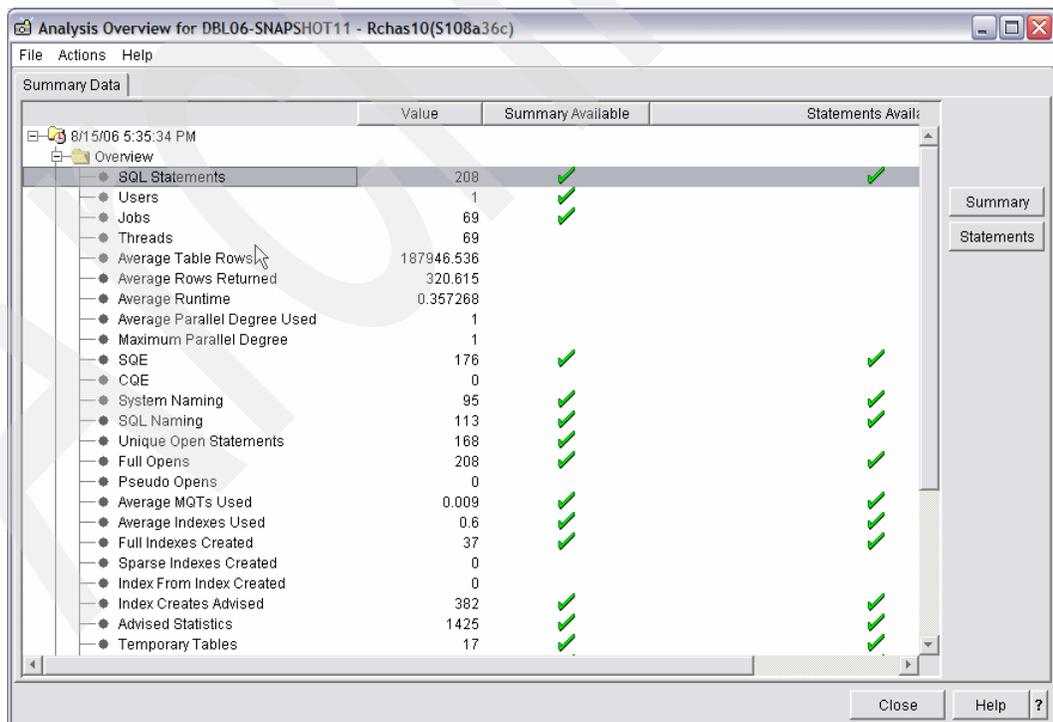


Figure 3-9 The dashboard

Refer to Chapter 7, “SQE Plan Cache and SQE Plan Cache Snapshots” on page 237 for more details.

3.4 The Database Performance Monitors

The *Database Performance Monitors* are integrated tools used to collect database-specific performance information for SQL requests being run on the iSeries. They let you keep track of resources that SQL statements use. The collected data is output to a database table or tables. Then reports or queries are run against the collected data from these output tables to analyze query optimization and the performance behavior of the database request. This analysis helps to identify and tune performance problem areas.

Note: The Database Performance Monitor name is also known as *SQL Performance Monitor*. From this point forward in the book, we use “SQL Performance Monitor” to refer to the iSeries Navigator function. We use “Database Monitors” when using a green screen and accessing the tool by running the Start Database Monitor (STRDBMON) CL command.

The SQL Performance Monitors can collect information for SQL-based interfaces such as embedded SQL, Open Database Connectivity (ODBC), and Java Database Connectivity (JDBC).

The SQL Performance Monitors provide all the information that the STRDBG or PRTSQLINF commands provide plus additional information such as:

- ▶ System and job name
- ▶ SQL statement text
- ▶ Start and End time stamp
- ▶ Estimated processing time
- ▶ Total rows in table queried
- ▶ Number of rows selected
- ▶ Estimated number of rows selected
- ▶ Estimated number of joined rows
- ▶ Key columns for advised index
- ▶ Total optimization time
- ▶ Join type and method
- ▶ Open data path (ODP) implementation
- ▶ QAQQINI settings

The collected information and statistics can be analyzed later to determine information such as:

- ▶ The number of queries that rebuild access plans
- ▶ The number of temporary indexes that have been created over a particular table
- ▶ The queries that are the most time-consuming
- ▶ The user that is has the longest-running queries
- ▶ The queries that were implemented using reusable ODPs
- ▶ Whether the implementation of a particular query changed with the application of a PTF or a new release (uses a before and after comparison of monitor data)

Note: Monitor data is not volatile. The information from the optimizer and engine is *captured* at a point in time.

Database monitors have existed on DB2 UDB for iSeries for many releases, but the usability of these monitors takes a big step forward in *V5R4 DB2 for i5/OS* with addition of both *pre-* and *post-filters*. These filters are very similar to the filters that were available with the *SQL Plan Cache viewer*. Database monitor filters are important for a couple of reasons. First, the filters can minimize the overhead and disk space consumed by a database monitor collection. As a case in point, a customer recently ran a database monitor collection on all jobs on their system for 15 minutes. An SQL-based ERP application that was running on the system at the time resulted in almost 3 million rows of detailed monitor data. Applying one of the filters available in V5R4 (refer to figure Figure 3-12 on page 46) such as only collecting monitor data for a certain user or only for SQL statements referencing a specified table could have significantly reduced the amount of data that had to be collected by DB2 for i5/OS.

Note: These filters are made available when you start a new SQL Performance Monitor collection via the graphical interface or the STRDBMON CL command.

Two types of SQL Performance Monitors are available with OS/400 and i5/OS, which are both packaged with the operating system:

- ▶ Detailed Monitor
- ▶ Summary Monitor, also known as the *Memory-based Database Performance Monitor*

Note: Both *Detailed Database Monitor* and *Summary Database Monitor* have CQE and SQE supports.

We discuss both monitors in the following sections.

3.4.1 Detailed Monitor

The Detailed Monitor gathers information about SQL requests and non-SQL queries, such as OPNQRYF. It has details about optimization and runtime behavior. The monitor data is dumped into a single output table.

Important: Use care when running the Detailed Monitor for long periods of time. The collection and output of performance data can consume both disk and CPU resources with its collection, resulting in system overhead.

Working with the Detailed SQL Performance Monitor interface of iSeries Navigator

As shown in Figure 3-10, you can to create a Detailed SQL Performance Monitor from the iSeries Navigator interface by performing the following steps:

1. Expand **Database** → **Database Name**.

2. Select and right-click **SQL Performance Monitors**.

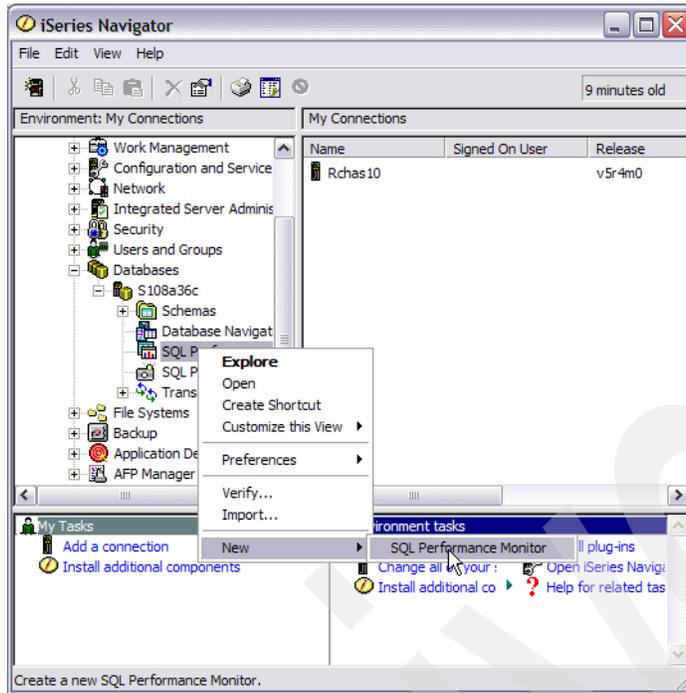


Figure 3-10 Creating a new SQL Performance Monitor

3. Select **New** → **SQL Performance Monitor** menu items. SQL Performance Monitor Wizard window opens as shown in Figure 3-11.

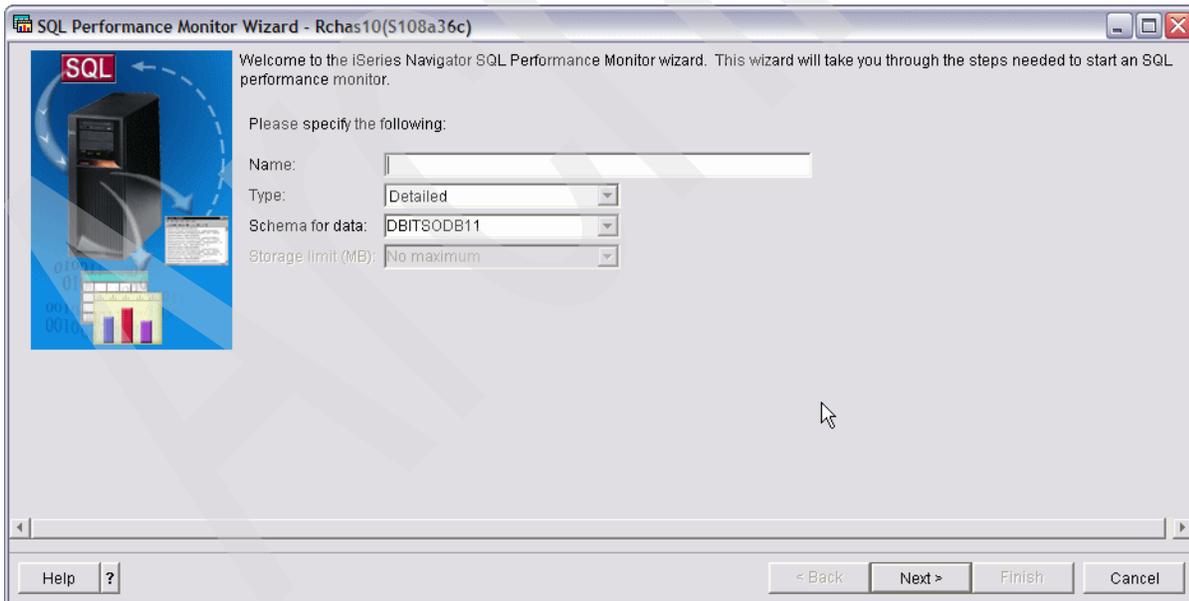


Figure 3-11 SQL Performance Monitor Wizard

4. On the SQL Performance Monitor window, perform the following steps:
 - a. Type a name for your SQL Performance Monitor.
 - b. Choose a type (Detailed or Summary).

- c. Choose a Schema for SQL Performance Monitor data.
5. Click the **Next** button. The SQL Performance Monitor Wizard window opens. As shown in Figure 3-12, from the SQL Performance Monitor Wizard window, select one or more of the following filters:
 - Minimum estimated query runtime
 - Job Name
 - Job user
 - Current user
 - Internet Address
 - Queries that access these tables
6. Choose one of the following Activity to monitor options:
 - Only collect monitor output for user activity
 - Collect monitor output for user and system activity

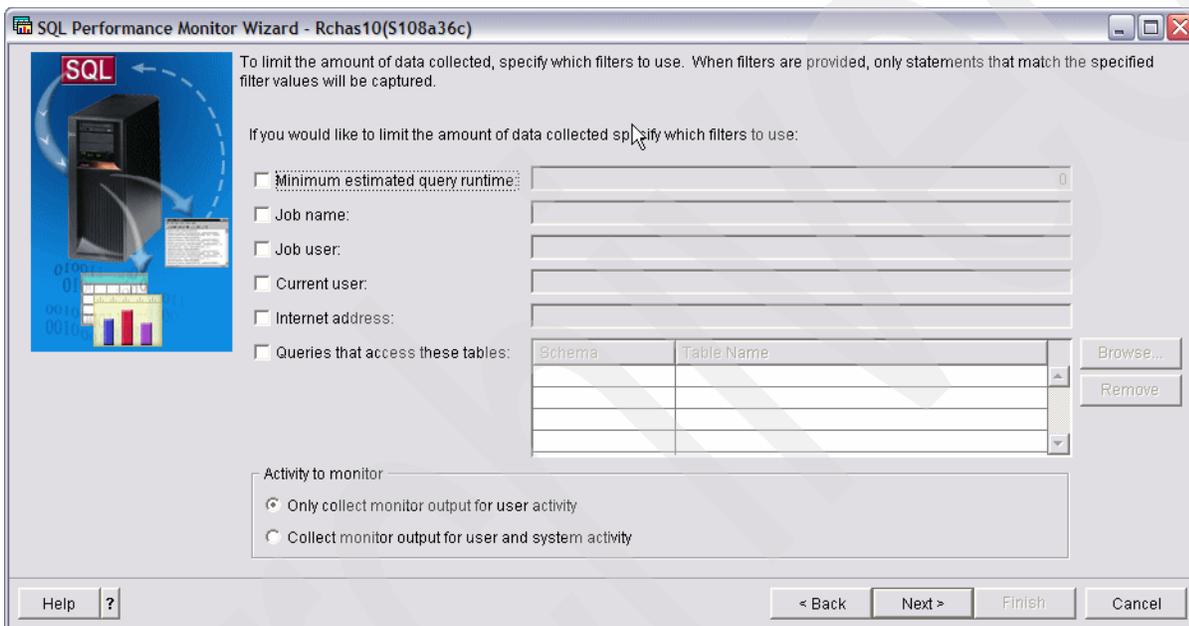


Figure 3-12 SQL Performance Monitor Wizard - filters

7. Click **Next** button. The SQL Performance Monitor Wizard window opens showing the Details of your choices as shown in Figure 3-13.

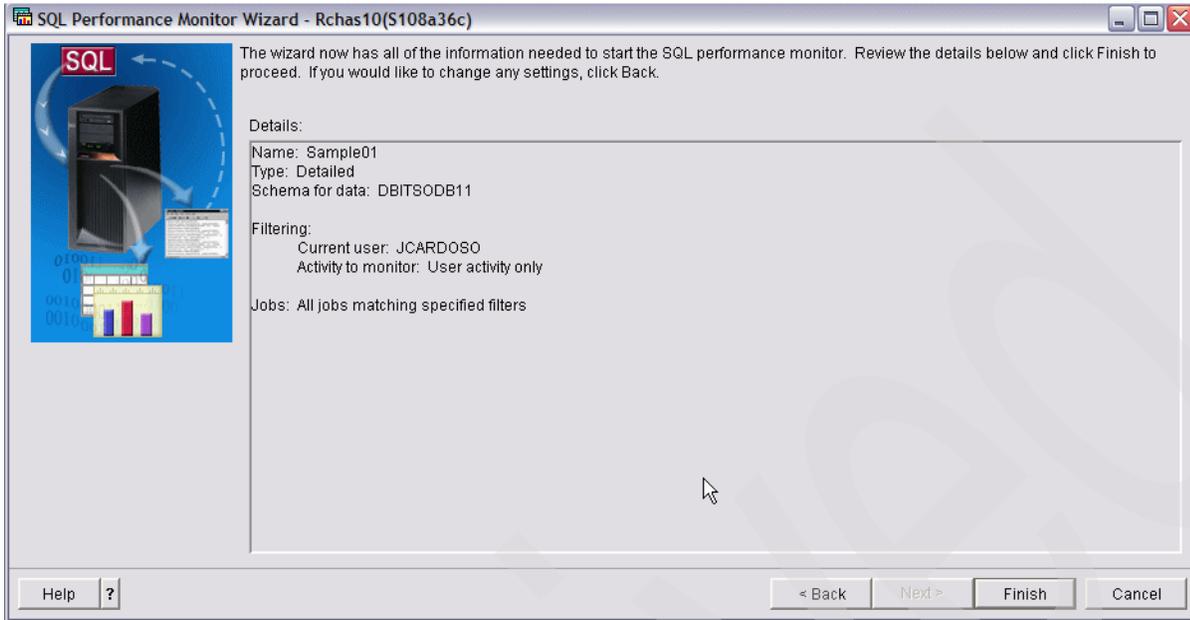


Figure 3-13 SQL Performance Monitor Wizard window - Details

8. Click the **Finish** button.

The aforementioned filters eliminate the number of rows in a database monitor collection, however, there are still hundreds of fields in a single database monitor row and many monitor row types that have to be sorted through to identify and fix performance problems. Previously, this maze of data made it a challenge to quickly look at a database monitor collection and determine if there were any issues, unless you were a DB2 performance expert.

Thus, a *dashboard summary* was added in *V5R4* to address this issue. On the first analysis of a database monitor collection, DB2 for i5/OS will automatically summarize some of the key performance indicators in the collection and then present the results in the summary window (the dashboard summary) as shown in Figure 3-14. This summary will allow an analyst to

quickly scroll through some high-level DB2 performance indicators to determine if more detailed analysis is required.

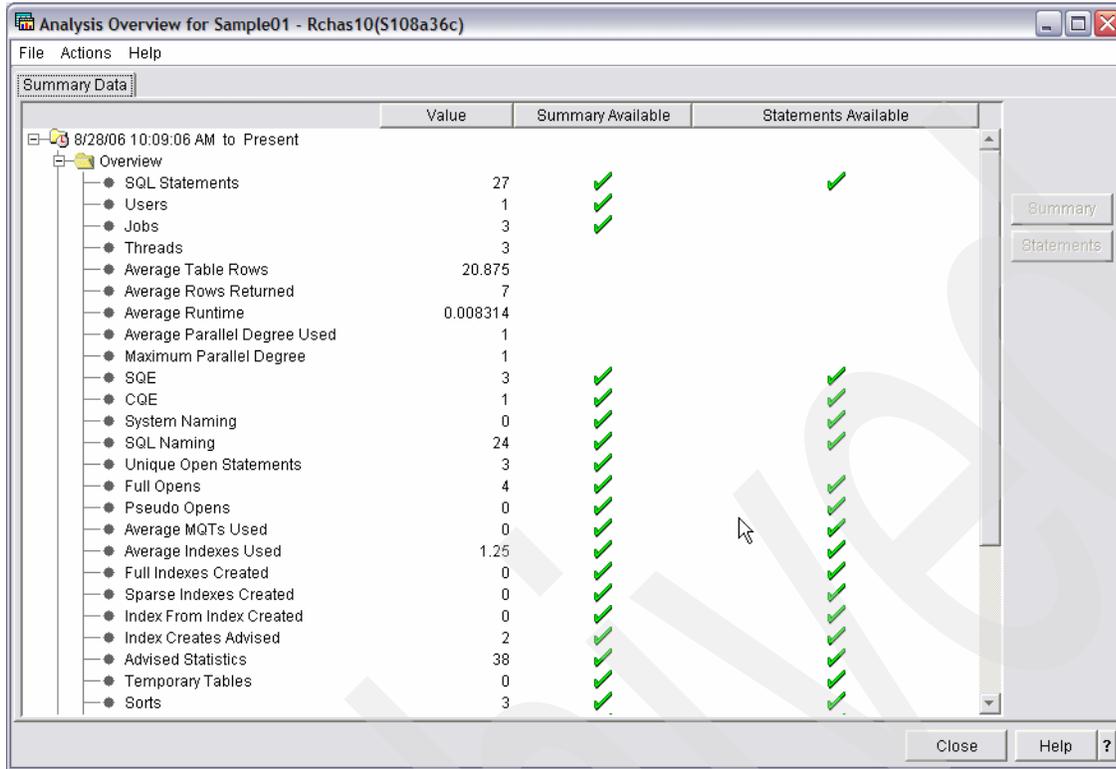


Figure 3-14 Analysis Overview window: the dashboard

V5R4 also addresses another efficiency issue with the database monitor analysis reports by providing drill-through analysis. In the past, the reports could be used to find an SQL statement with performance issues. However, if more detailed research of this statement was needed with a tool such as *Visual Explain*, the user had to exit out of the tool and then manually copy that SQL statement for *Visual Explain* analysis. As shown in Figure 3-15, an analyst can easily right-click to drill into more detail with *Visual Explain*. This is a great enhancement for improving the efficiency of your database analysts and administrators.

Once you get to the Analysis Overview window the different indicators may have a check mark under Summary Available and also under Statements Available. If you click the SQL

Statement line, the Summary and Statements buttons will be activated. Click the **Statements** button and a window like the one shown in Figure 3-15 is displayed.

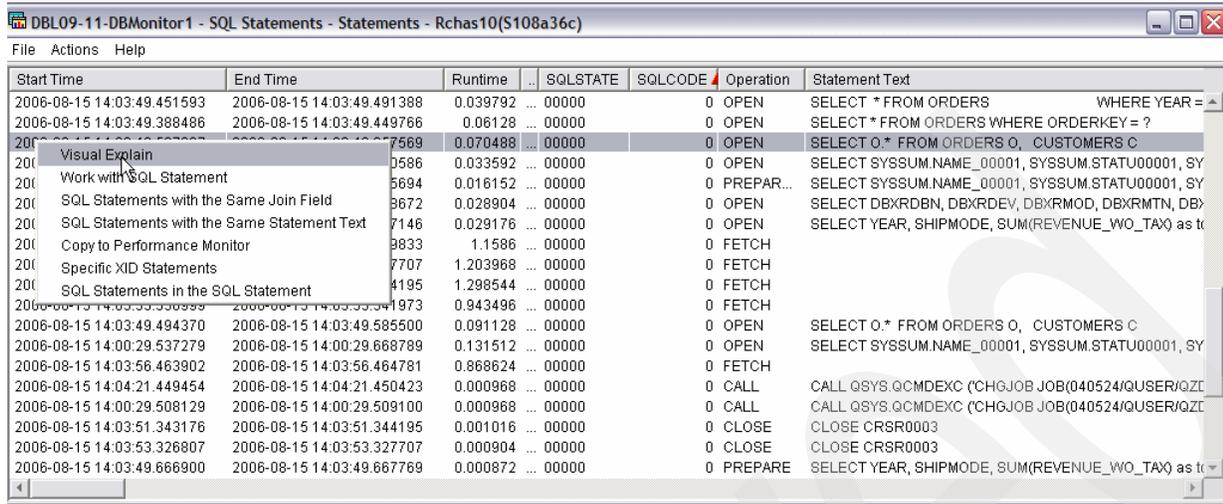


Figure 3-15 Drilling down to Visual Explain from SQL Performance Monitor

Figure 3-16 shows the Visual Explain graphic of the query selected as shown in Figure 3-15.

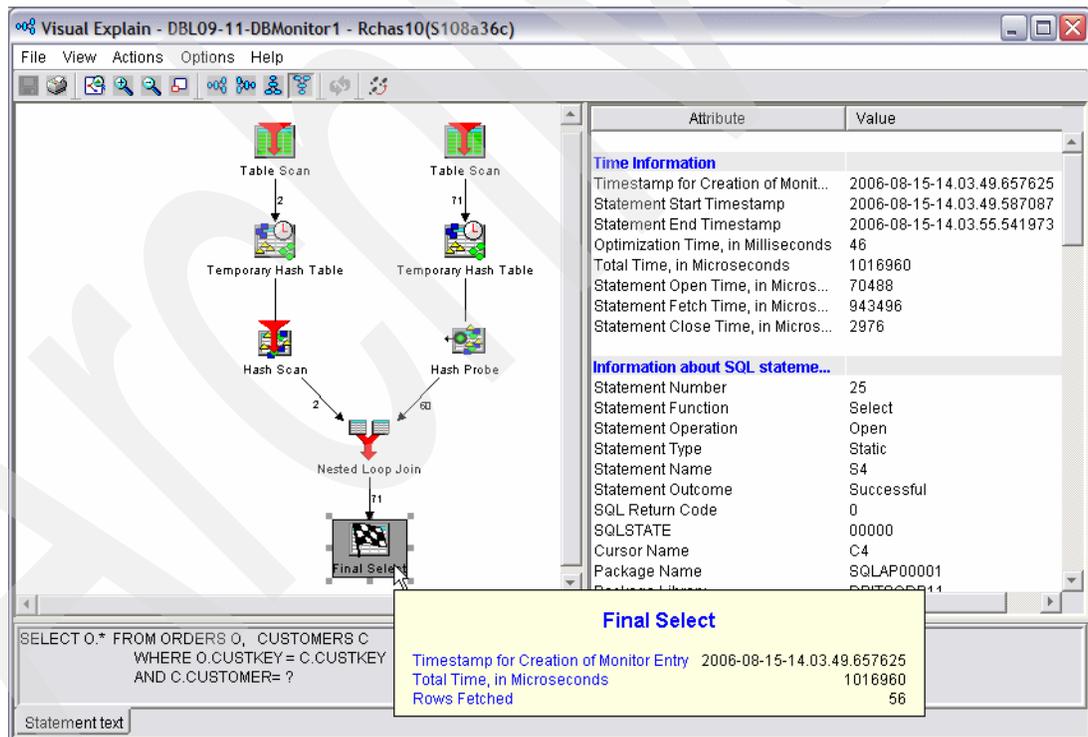


Figure 3-16 Visual Explain graphic started from the SQL Statements Interface

STRDBMON and ENDDBMON CL commands

The CL commands, Start Database Monitor (STRDBMON) and End Database Monitor (ENDDBMON), provide the interface for the Detailed Performance Monitors. The commands start and end the collection of database performance statistics.

The STRDBMON command gathers information about a query in real-time and stores this information in an output table. The information can help you determine whether your system and your queries are performing as they should or whether they need fine-tuning. The monitor can be specified for a specific job or all jobs on system. Collected information and statistics are placed in a single output database table made up of different record types.

Note: Be aware that many of the parameters of the STRDBMON CL Command are new. Therefore, they are not available on pre-V5R4 releases.

For a complete description of the parameters refer to “Starting a Detailed Database Monitor using the command interface” on page 90.

The ENDDBMON command ends the Database Monitor for a specific job or all jobs on the server. If an attempt to end the monitor on all jobs is issued, there must have been a previous STRDBMON issued for all jobs. If a particular job is specified on this command, the job must have the monitor started explicitly and specifically on that job.

When collecting information for all jobs, the Database Monitor collects on previously started jobs or new jobs that are started after the monitor starts. Each job in the system can be monitored concurrently by two monitors:

- ▶ One monitor started specifically on that job, and another started for all jobs in the system.
- ▶ When monitored by two monitors and each monitor is logging to a different output file, monitor information is written to both files.

Database Monitors can generate significant CPU and disk storage overhead when in use. You can gather performance information for a specific query instead of every query to reduce this overhead on the system. You can also gather only specific monitoring data for smaller monitor collection since large monitor database files can slow analysis.

Consider the following guidelines:

- ▶ If possible, try collecting data only for the job that you want.
- ▶ Collect monitor data only for long-running SQL statement based on the optimizer’s estimated runtime.
- ▶ Eliminate the SQL statement generated by DB2.
- ▶ By using the table filter function, collect only statements that reference certain tables such as the one shown in the following example:

```
STRDBMON OUTFILE(o) COMMENT('TABLEFILTER(lib1/tab1,lib2/tab2)')
```

Note: To use the TABLE FILTER function, apply the following required PTFs in advance.

- ▶ *V5R2 PTFs:* SI15035, SI15139, SI15140, SI15142, SI15143, SI15154, and SI15155
- ▶ *V5R3 PTFs:* SI15955, SI16331, SI16332, and SI16333

For more information about valid parameter settings for the QAQQINI table, refer to “Monitoring your queries using Start Database Monitor (STRDBMON)” in *DB2 Universal Database for iSeries Database Performance and Query Optimization*, which is available in the iSeries Information Center on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseres/v5r3/index.jsp?topic=/rzajq/rzajqmst.htm>

3.4.2 Summary Monitor or Memory Resident Database Monitor

The Memory Resident Database Monitor is a tool that provides another method for monitoring database performance. This tool is intended only for the collection of performance statistics in SQL queries. To collect performance statistics for non-SQL queries, you should start a Detailed Database Monitor as explained in previous sections. The Memory-based Database Monitor or Summary Monitor, with the help of a set of APIs, manages database monitoring information for the user in memory. This memory-based monitor reduces CPU overhead and resulting table sizes.

The Monitor data is placed into multiple DB2 tables. You can configure pre-filtering and post-filtering to collect summarized Database Monitor data as well.

Working with the Summary SQL Performance Monitors interfaces of iSeries Navigator

In order to create a Summary SQL Performance Monitor, perform the following steps:

1. Follow the first steps you use to Create a Detailed SQL Performance Monitor. When the SQL Performance Monitor wizard window asks you for the Monitor type, choose the Summary option instead of Detailed as shown in Figure 3-17. Click the **Next** Button.

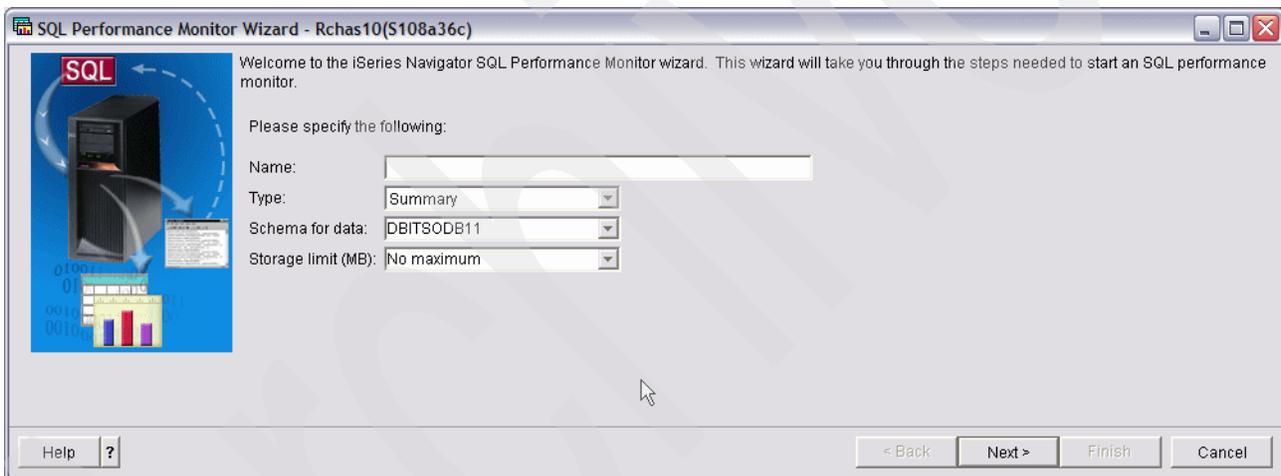


Figure 3-17 SQL Performance Monitor Wizard window - Summary

- The SQL Performance Monitor Wizard window asks you to choose the optional information you want to collect as shown in Figure 3-18.

Note: You can choose one or more of the following:

- ▶ Table scans and arrival sequences
- ▶ Indexes used
- ▶ Index creation
- ▶ Data sorts
- ▶ Temporary file use
- ▶ Indexes considered
- ▶ Subselect processing

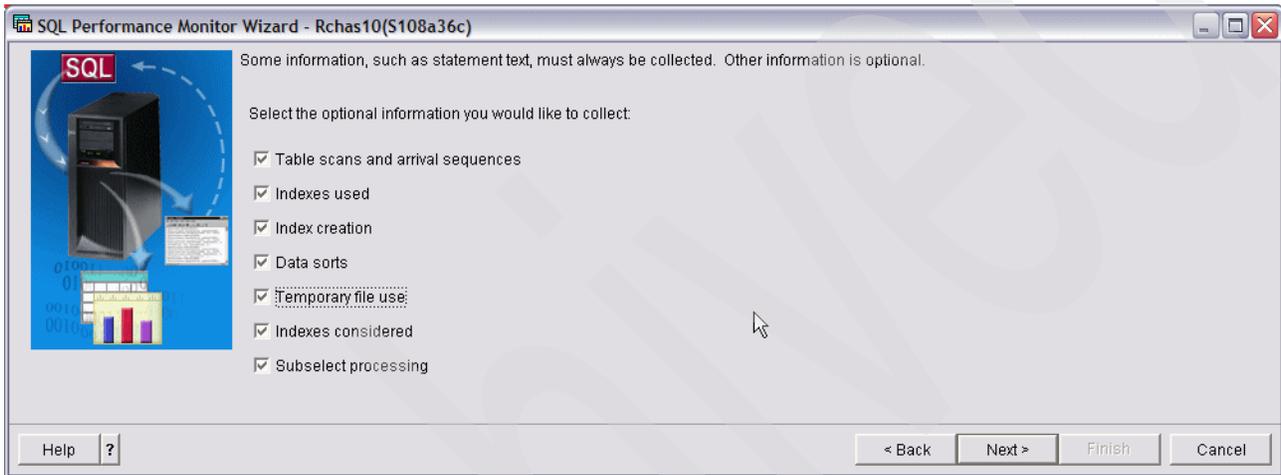


Figure 3-18 SQL Performance Monitor Wizard window - Summary Optional information

- After you make your choices, click the **Next** button. The SQL Performance Monitor Wizard permits you to select the jobs you would like to monitor as shown in Figure 3-19. You can select to monitor All jobs or Specific jobs. Click the **Next** button.

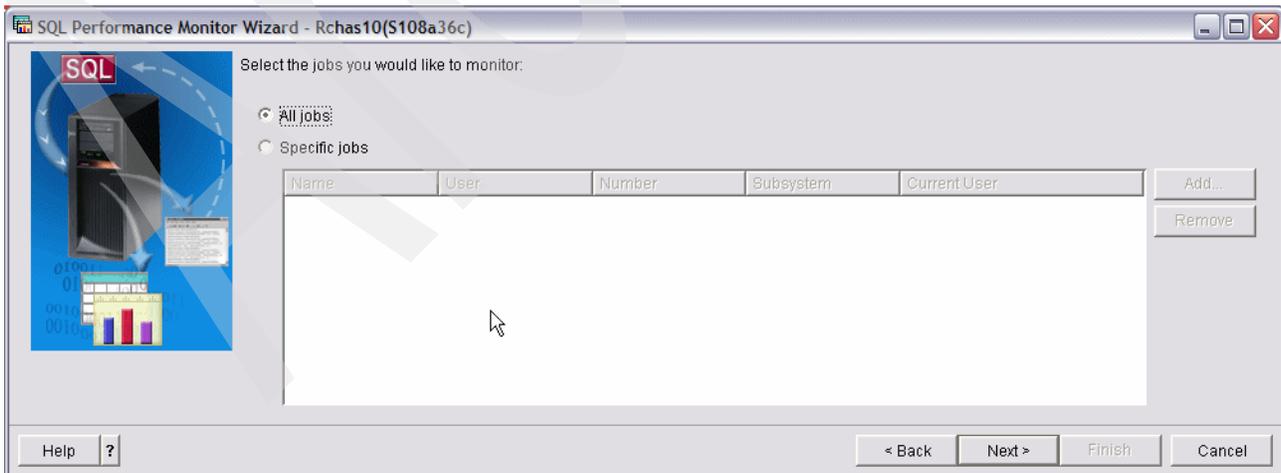


Figure 3-19 Selecting the jobs you would like to monitor

- As shown in Figure 3-20, the SQL Performance Monitor Wizard window lists your choices. Review the details and click the **Finish** button. The SQL Performance Monitor starts to collect data.

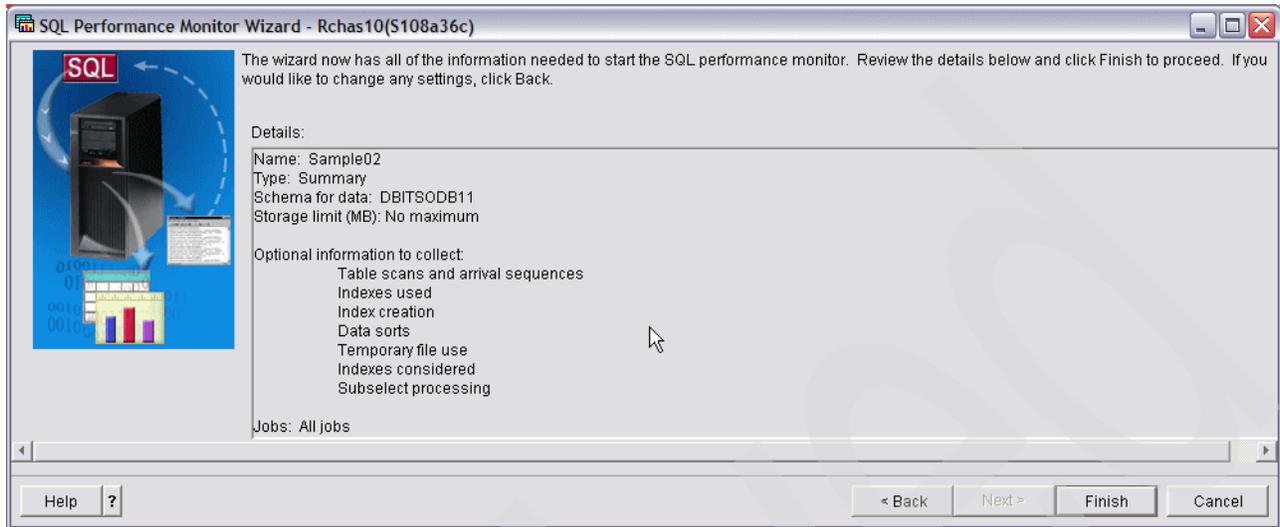


Figure 3-20 Review the details of SQL Performance Monitor Wizard - summary

To analyze the Summary SQL Performance Monitor data, perform the following steps:

- Click **Analyze** menu item as shown in Figure 3-21.

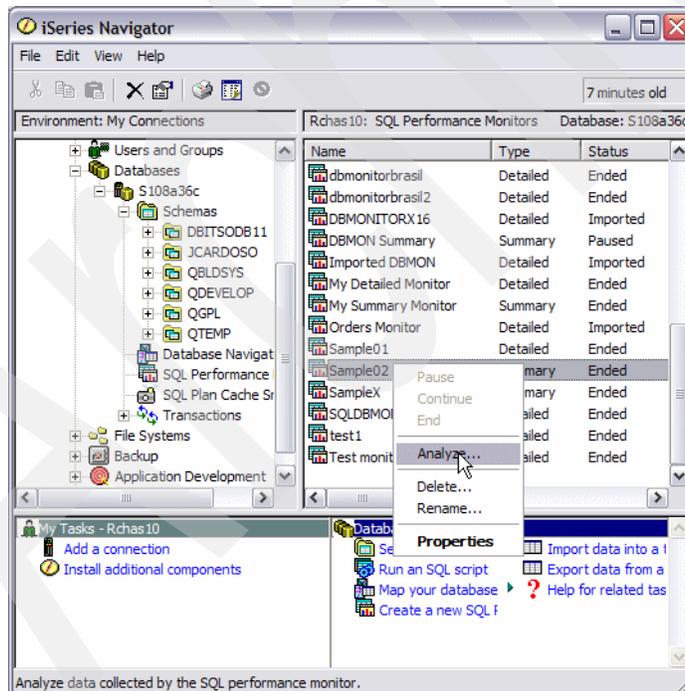


Figure 3-21 How to analyze SQL Performance Monitor - summary

2. A Result window to monitor the Summary SQL Database Monitor opens as shown in Figure 3-22.

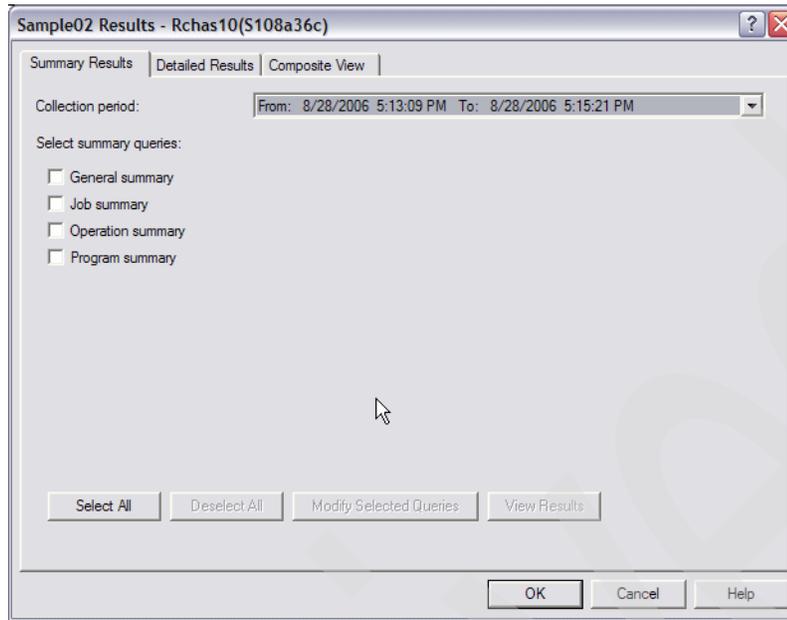


Figure 3-22 Result window to monitor the Summary SQL Database Monitor

Refer to Chapter 5, “Analyzing SQL performance data using iSeries Navigator” on page 117 for more details.

API support for the Memory Resident Database Monitor

A set of application programming interfaces (APIs) provides support for the Summary Monitor or Memory-Resident Database Monitor that allow you to perform the activities listed in Table 3-1.

Table 3-1 External API description

API	Description
QQQSSDBM	This API starts the Memory-based Database Monitor. Database Monitor data is collected in the threaded process but summarized at the job level.
QQQCSDBM	This API clears and frees the associated memory area of the SQL monitor.
QQQDSDBM	This API dumps the contents of the SQL monitor table. The API does not force a clear operation (QQQCSDBM) of the memory. Data continues to be added to memory until the QQQCSDBM or QQQESDBM API is called.
QQQESDBM	This API ends the memory-based SQL monitor.
QQQQSDBM	This API queries the status of the Database Monitor, and returns information about the activity of the SQL and the original Database Monitor.

Figure 3-23 illustrates the different events in the Memory Resident Database Monitor life cycle and the APIs associated with each event.

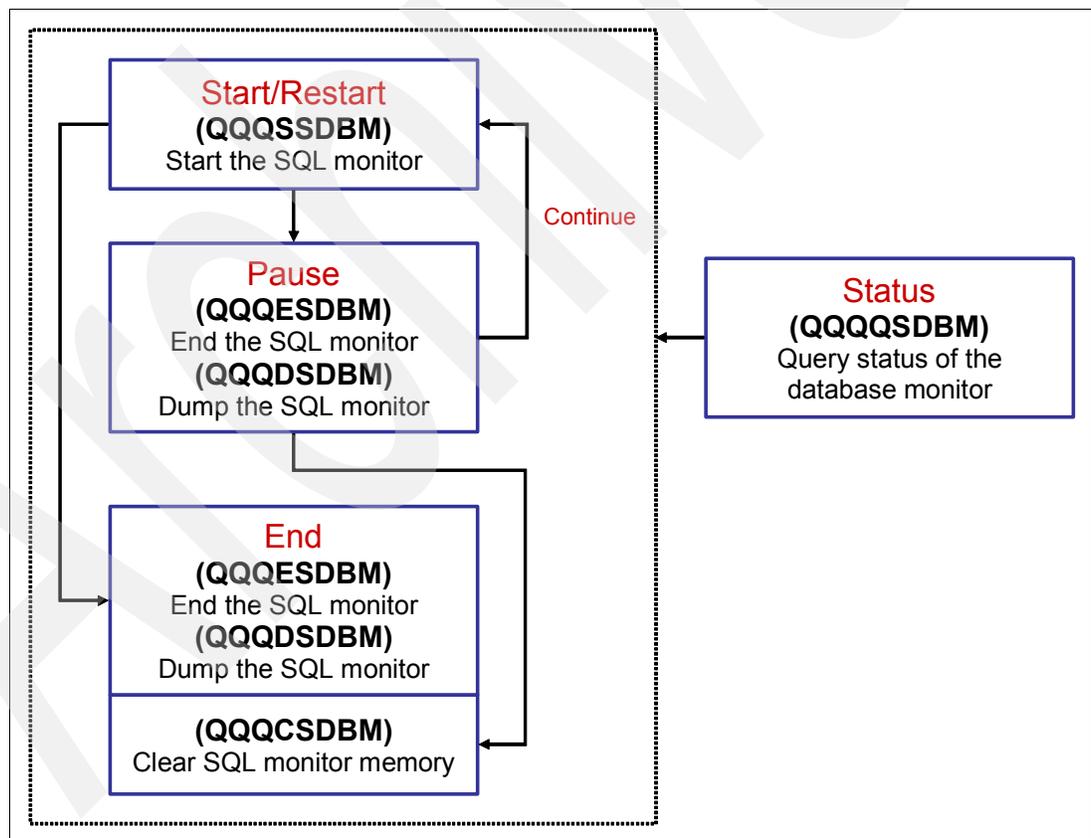


Figure 3-23 Memory Resident Database Monitor events and their APIs

For more information, search with Memory Resident Database Monitor external API description in the V5R3 iSeries Information Center on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseres/v5r3/index.jsp>

Note:

- ▶ iSeries Navigator provides a graphical interface for these APIs, through the Summary SQL Performance Monitor to administer the memory-based collection mode and to run analytical database performance reports from the information collected.
- ▶ Unlike the Detailed Monitor, the Memory-Resident Database Monitor or Summary Monitor outputs or dumps the collected information into 10 separate, categorized output tables. To get the consolidated view of information collected for a single statement, you must run join queries.

Memory Resident Database Monitor external table description

The Memory Resident Database Monitor uses its own set of tables instead of using the single table with logical files that the STRDBMON monitor uses. The Memory Resident Database Monitor tables closely match the suggested logical files of the STRDBMON monitor. The tables are:

QAQQRYI	Query (SQL) information
QAQQTEXT	SQL statement text
QAQQ3000	Table scan
QAQQ3001	Index used
QAQQ3002	Index created
QAQQ3003	Sort
QAQQ3004	Temporary table
QAQQ3007	Optimizer time-out/all indexes considered
QAQQ3008	Subquery
QAQQ3010	Host variable values
QAQQ3030	Materialized Queries Tables considered

For more information about the definitions of these tables, search on Memory Resident Database Monitor: DDS in the V5R3 iSeries Information Center:

<http://publib.boulder.ibm.com/infocenter/series/v5r3/index.jsp>

Detailed versus Summary Monitor

The STRDBMON command can constrain server resources when collecting performance information. This overhead is mainly attributed to the fact that performance information is written directly to a database table as the information is collected. The memory-based collection mode reduces the server resources consumed by collecting and managing performance results in memory. This allows the monitor to gather database performance statistics with a minimal impact to the performance of the server as a whole (or to the performance of individual SQL statements).

The Summary Monitor collects much of the same information as the Detailed Database Monitor, but the performance statistics are kept in memory. At the expense of some detail, information is summarized for identical SQL statements to reduce the amount of information collected. The objective is to get the statistics to memory as fast as possible while deferring any manipulation or conversion of the data until the performance data is dumped to a result table for analysis.

The Summary Monitor is not meant to replace the Detailed Database Monitor. There are circumstances where the loss of detail in the summary Monitor is not sufficient to fully analyze an SQL statement. In such cases, we recommend that you use the Detailed Database Monitor. Also keep in mind that the Summary monitor information cannot be analyzed using

Visual Explain as it is shown in Figure 3-1 which in some cases it could be a very big disadvantage.

3.4.3 Importing a Database Monitor to SQL Performance Monitor

You can import monitor data that has been collected using Start Database Monitor (STRDBMON) command or some other interface by using iSeries Navigator. This is a very useful feature because this can be used to import Detailed SQL Performance monitor data that was collected on a system that was at a previous release of the operating system. This will allow you to use the new analysis tools to analyze data that was collected on a system at an earlier release.

To import monitor data, perform the following step:

As shown in Figure 3-24, right-click **SQL Performance monitors** and select **Import**. Once you have imported a monitor, you can analyze the data.

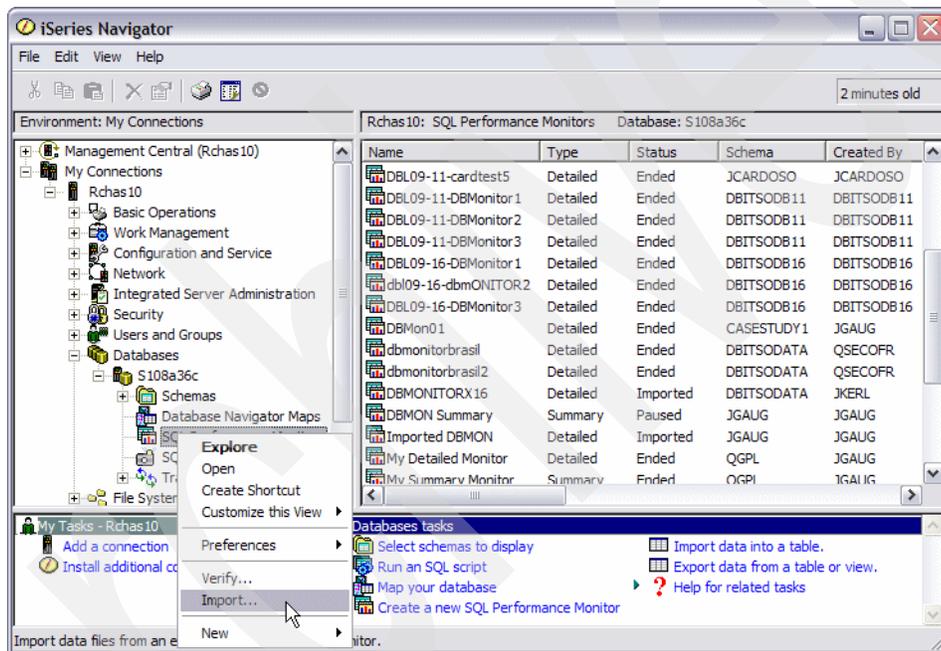


Figure 3-24 Importing a Database Monitor

3.4.4 The Monitor Comparison feature of SQL Performance Monitor

With the new monitor comparison utility, users can utilize database monitor collections to help determine what change is causing performance issues. This technique requires that a customer collects and saves database monitor collections when a critical DB2 workload or report is performing and running well on their server. Once performance of this workload or report changes, a new database monitor collection can be compared with the “good performance” monitor collection to help identify the differences causing performance problems. This can be a long-running process with large database monitor collections, so plan accordingly.

As shown in Figure 3-25, in order to compare SQL Performance Monitor data via iSeries Navigator, perform the following steps:

1. Expand **System** → **Databases**.
2. Select **SQL Performance Monitor** icon.
3. Right click the SQL Performance Monitor which you want to compare.
4. Click **Compare** menu item.

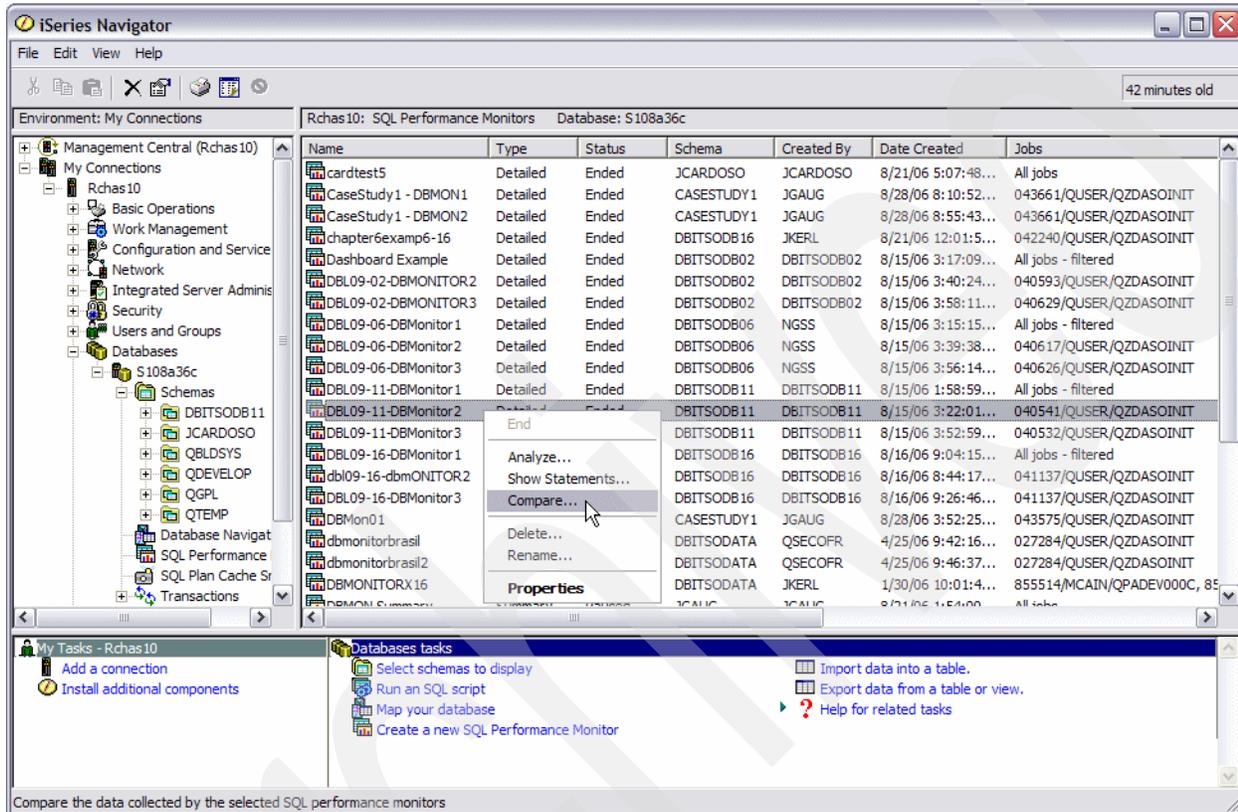


Figure 3-25 Compare menu item of SQL Performance Monitors

5. As shown in Figure 3-26 the Compare SQL Performance Data window opens. Choose the SQL Performance Monitor you want to compare as shown in Figure 3-26.

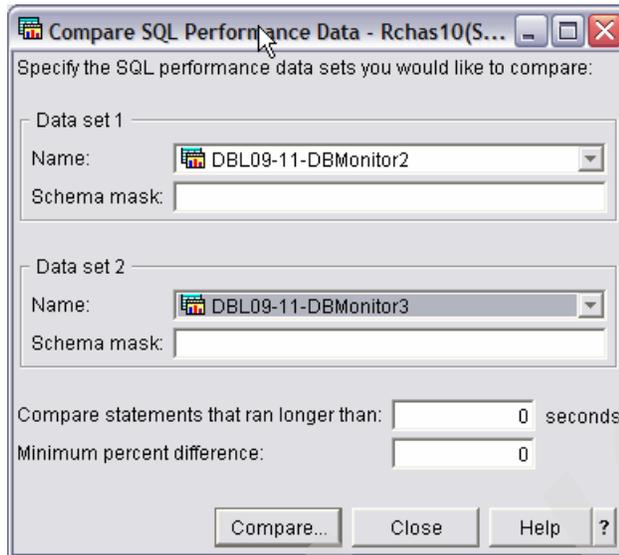


Figure 3-26 Compare SQL Performance Data window

6. Click **Compare** button. The Compare SQL Performance Data window opens as shown in Figure 3-27.

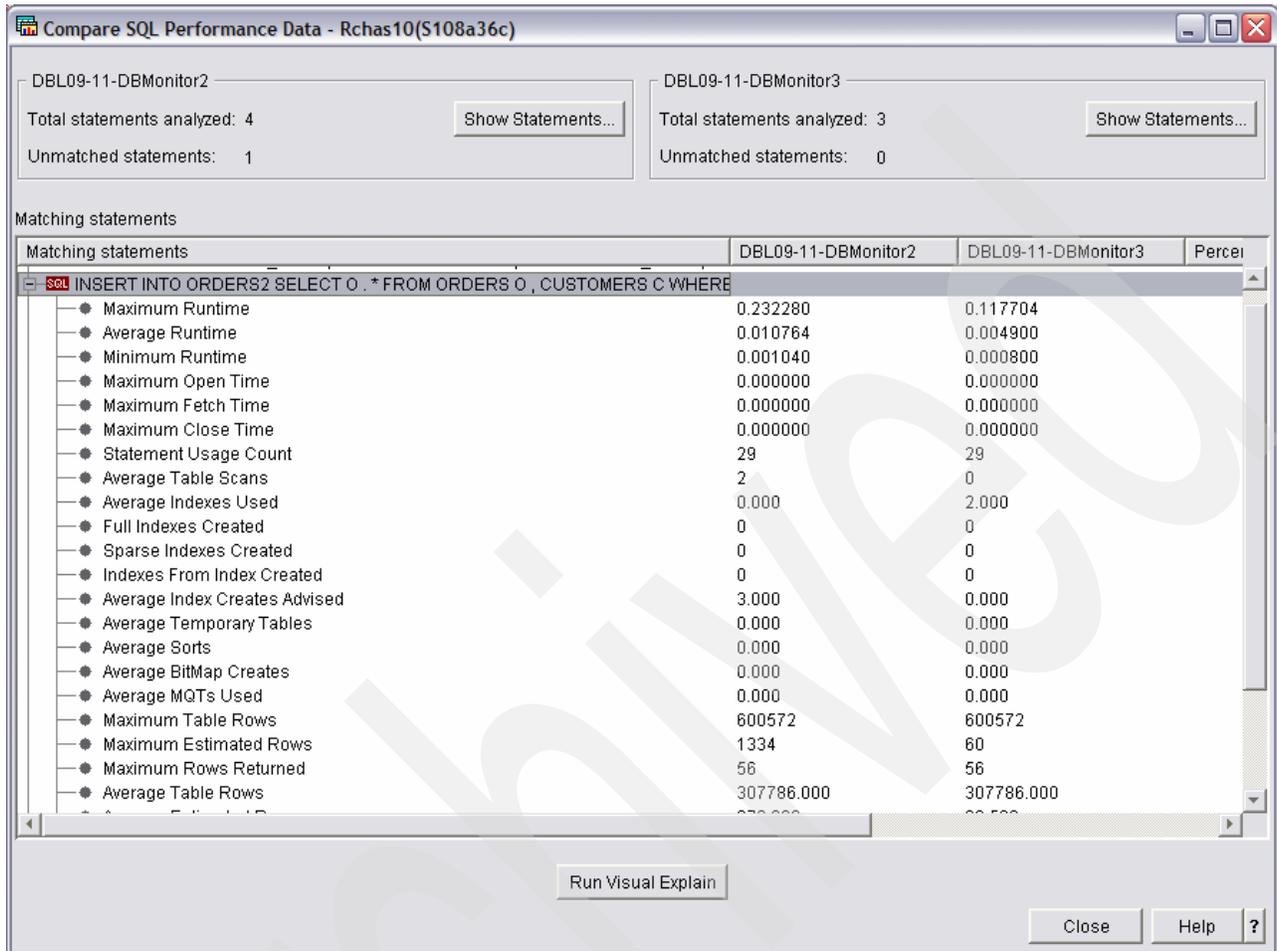


Figure 3-27 Compare SQL Performance Data window

3.5 Visual Explain

Visual Explain provides a graphical representation of the optimizer implementation of a query request. The query request is broken down into individual components with icons that represent each unique component. Visual Explain also includes information about the database objects that are considered and chosen by the query optimizer. Visual Explain's detailed representation of the query implementation makes it easier to understand where the greatest cost is incurred.

Visual Explain shows the job-run environment details and the levels of database parallelism that were used to process the query. It also shows the access plan in diagram form, which allows you to zoom to any part of the diagram for further details as shown in Figure 3-28.

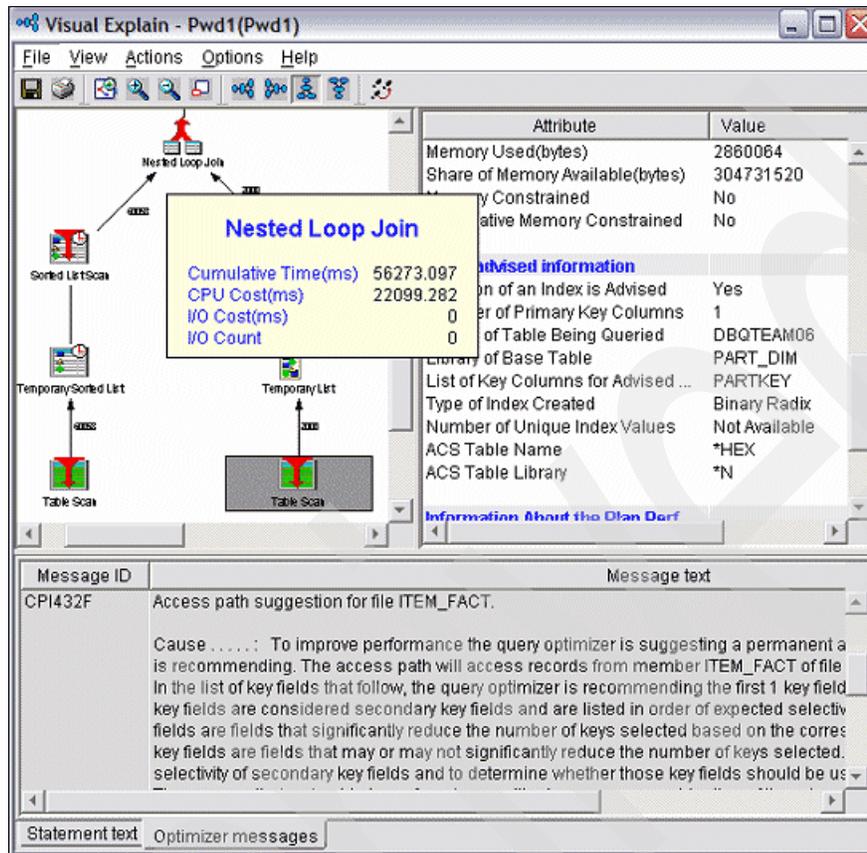


Figure 3-28 Visual Explain diagram

Visual Explain is a component of iSeries Navigator. From the Run SQL Script Center, you can access Visual Explain directly, either from the menu or from the toolbar.

Note: The Run SQL Script interface always deletes ODPs to force full optimization.

Visual Explain is based on *detailed* optimizer information. As shown in Figure 3-29, the optimizer sends detailed feedback information to SQE Plan Cache and to the Detailed DB Monitor. Therefore, in V5R4 you can also drill down into Visual Explain from the following interfaces:

- ▶ SQE Plan Cache
- ▶ SQE Plan Cache Snapshots which are based on SQE Plan Cache information
- ▶ Detailed Database Monitor

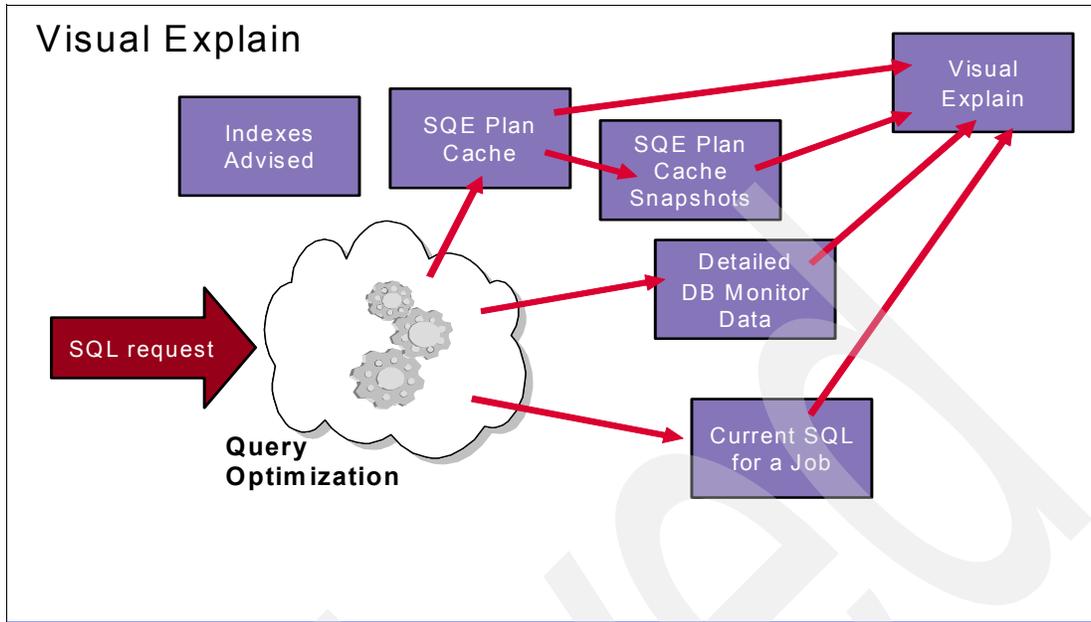


Figure 3-29 Feedback Mechanisms integrated with Visual Explain

Note: Visual Explain supports both SQE and CQE as described below:

- ▶ SQE
 - SQE Plan Cache
 - SQE Plan Cache Snapshot
 - Detailed Database Monitor
 - Current SQL for a Job
 - Run SQL Script
- ▶ CQE
 - Detailed Database Monitor
 - Run SQL Script

Also, from the Current SQL for a Job window we can drill down into Visual Explain by clicking the **Visual Explain** button as shown in Figure 3-30.

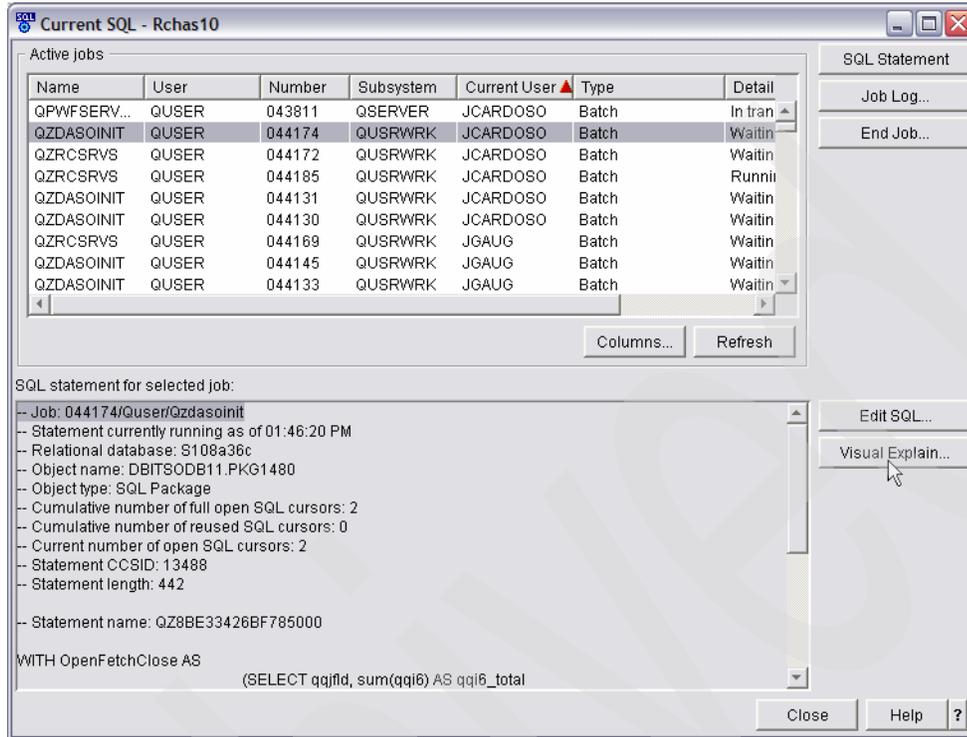


Figure 3-30 Drilling down into Visual Explain from Current SQL for a Job window

SQL Performance Monitor is used to create Database Monitor data and to analyze the monitor data with predefined reports. For more information about the SQL Performance Monitor, refer to Chapter 5, “Analyzing SQL performance data using iSeries Navigator” on page 117.

Visual Explain works with the monitor data that is collected by SQL Performance Monitor on that system or by the Start Database Monitor (STRDBMON) command. Visual Explain can also analyze Database Monitor data that is collected on other systems after data is restored on the iSeries server.

For more information about the Visual Explain, refer to Chapter 8, “Analyzing database performance data with Visual Explain” on page 275.

3.6 Index Advisor

The optimizer has always provided index advised information in the past. On V5R4 the optimizer provides new and better ways to get at the advice. System-wide Index Advisor is a new feature which started in V5R4. It is autonomic, causes no overhead and is always on. If the optimizer determines that a permanent index might be beneficial, it returns the key columns necessary to create the suggested index. The data of the System-wide Index Advisor is placed into SYSIXADV table in the QSYS2 schema.

SQE and CQE can take advantage of the System-wide Index Advisor as described below:

- ▶ SQE provides complex advice based on all parts of the query
- ▶ CQE only provides basic advice based on local selection predicates

iSeries Navigator has an interface where Index advised is listed for the System, or Schema or Table. Also, you can create indexes directly from the GUI of iSeries Navigator.

Note: Additional indexing analysis might be required.

The system only adds rows, the user must manage the data. There are options to clear or prune the SYSIXADV table in the QSYS2 schema.

As shown in Figure 3-31, from the Index Advisor menu item, you can choose the following:

- ▶ Index Advisor
- ▶ Clear all advised indexes
- ▶ Prune advised indexes

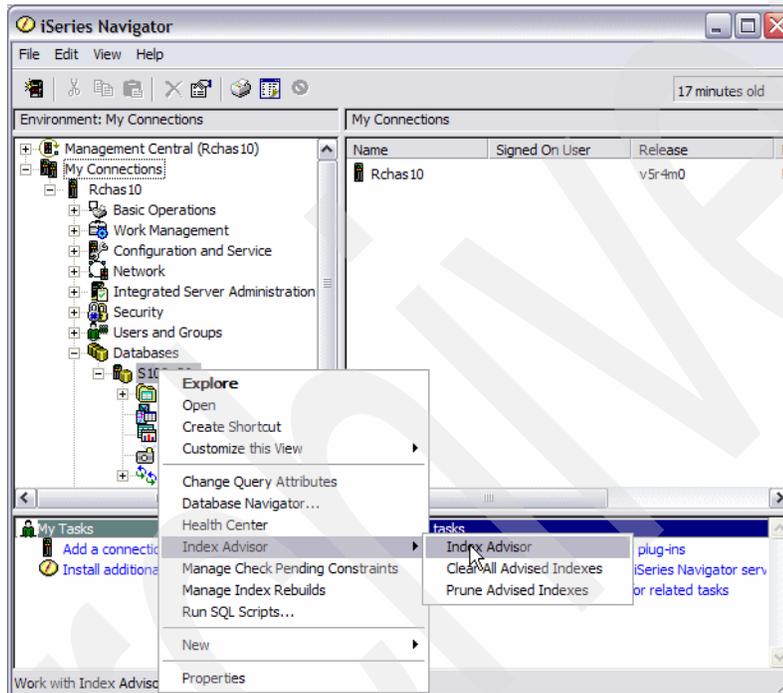


Figure 3-31 How to access System-wide Index Advised

The Index Advisor window opens as shown in Figure 3-32.

Note: The result list of the Index Advisor depends on what icon you right-click. As mentioned before you can access the Index Advisor menu item when you right-click the following icons:

- ▶ System
- ▶ Schema
- ▶ Table

Table for Which Index was Advised	Schema	Short Name	Partition	Keys Advised	Leading Keys Order Independent	Index Type Advised	Last Advised for Query Use	Times Advised	Estimated Index Creation Time
QASZRAIRD	QUSRSYS	QASZRAIRD		FULLPATH, COM...	FULLPATH, COM...	Binary Radix	8/8/06 9:39:37 AM	12940	00:00:16
ACCOUNTJE8	TRADE6DB	ACCOUNTJE8		PROFILE_USERI...	PROFILE_USERID	Binary Radix	2/13/06 4:53:16 AM	5892	00:00:01
QAUGDBPMD2	QUSRSYS	QAUGDBPMD2		"Save Data File...	"Save Data File...	Binary Radix	8/22/06 9:01:38 AM	1568	00:00:01
QAUGDBPMD2	QUSRSYS	QAUGDBPMD2		"Save Library", "...	"Save Library", "...	Binary Radix	8/22/06 9:01:32 AM	746	00:00:01
QAYPSYGRP	QMGTG	QAYPSYGRP		OWNER, CHANG...	OWNER	Binary Radix	6/27/06 11:00:18...	601	00:00:01
QAYPSJDT	QMGTG	QAYPSJDT		OWNER, CHANG...	OWNER	Binary Radix	6/27/06 11:00:18...	526	00:00:01
QAYPSJDS	QMGTG	QAYPSJDS		OWNER, CHANG...	OWNER	Binary Radix	6/27/06 11:00:18...	526	00:00:01
QAYPSJDFN	QMGTG	QAYPSJDFN		OWNER, CHANG...	OWNER	Binary Radix	6/27/06 11:00:18...	526	00:00:01
QAYPSJEVT	QMGTG	QAYPSJEVT		OWNER, CHANG...	OWNER	Binary Radix	6/27/06 11:00:18...	526	00:00:01
QAYPSJET	QMGTG2	QAYPSJET		OWNER, CHANG...	OWNER	Binary Radix	6/27/06 11:00:18...	526	00:00:01
QAYPSJES	QMGTG2	QAYPSJES		OWNER, CHANG...	OWNER	Binary Radix	6/27/06 11:00:18...	526	00:00:01
QAYPSJDT	QMGTG	QAYPSJDT		OWNER, STATUS	OWNER	Binary Radix	6/27/06 11:00:18...	525	00:00:01
QAYPSJDS	QMGTG	QAYPSJDS		OWNER, STATUS	OWNER	Binary Radix	6/27/06 11:00:18...	525	00:00:01
QAYPSJET	QMGTG2	QAYPSJET		OWNER, STATUS	OWNER	Binary Radix	6/27/06 11:00:18...	525	00:00:01
QAYPSJES	QMGTG2	QAYPSJES		OWNER, STATUS	OWNER	Binary Radix	6/27/06 11:00:18...	525	00:00:01
QASZRAIRC	QUSRSYS	QASZRAIRC		VENDOR, VERSI...	VENDOR, VERSI...	Binary Radix	1/28/06 4:25:17 PM	437	00:00:01
QAUGDBPMD2	QUSRSYS	QAUGDBPMD2		"Save Library", "...	"Save Library", "...	Binary Radix	8/21/06 5:09:04 PM	322	00:00:01
ORDERS	DBITSOD...	ORDERS		CUSTKEY		Encoded vec...	8/15/06 11:55:15...	312	00:00:03
SALES	COUNTR...	SALES		TYPE, SALESDAT...	TYPE, SALESDATE	Binary Radix	4/24/06 7:50:17 AM	243	00:00:01
SALES	COUNTR...	SALES		TYPE, SALESDAT...	TYPE, SALESDATE	Binary Radix	4/24/06 7:50:17 AM	232	00:00:01
QASZRAIRC	QUSRSYS	QASZRAIRC		VERSION, PROD...	VERSION, PROD...	Binary Radix	8/8/06 9:39:37 AM	211	00:00:01
ORDERS	DBITSOD...	ORDERS		SHIPDATE		Encoded vec...	8/15/06 11:55:13...	210	00:00:03
CUSTOMERS	DBITSOD...	CUSTOMERS		CUSTOMER, CU...	CUSTOMER	Binary Radix	8/15/06 11:55:14...	180	00:00:01
ORDERS	DBITSOD...	ORDERS		CUSTKEY		Binary Radix	8/15/06 11:55:14...	177	00:00:05
QASZRAIRC	QUSRSYS	QASZRAIRC		VERSION, FEAT...	VERSION, FEAT...	Binary Radix	8/8/06 9:14:23 AM	172	00:00:01
QAUGDBPMD2	QUSRSYS	QAUGDBPMD2		"Free Storage M...	"Free Storage M...	Binary Radix	8/22/06 9:01:18 AM	166	00:00:01

Figure 3-32 Index Advisor window

The Index Advisor window has a list of Indexes advised whose columns attributes are explained with more details in Chapter 9, “Index Advisor” on page 319.

You also can access Index Advised information from the following interfaces:

- ▶ SQL Plan Cache Statements window - See Chapter 7, “SQE Plan Cache and SQE Plan Cache Snapshots” on page 237
- ▶ SQL Plan Cache Snapshots - See Chapter 7, “SQE Plan Cache and SQE Plan Cache Snapshots” on page 237
- ▶ Detailed SQL Performance Monitor of iSeries Navigator - See Chapter 5, “Analyzing SQL performance data using iSeries Navigator” on page 117
- ▶ Summary SQL Performance Monitor of iSeries Navigator - See Chapter 5, “Analyzing SQL performance data using iSeries Navigator” on page 117
- ▶ Visual Explain - See Chapter 8, “Analyzing database performance data with Visual Explain” on page 275
- ▶ Querying the Database Monitor view 3020 - Index advised (SQE) - See Chapter 6, “Custom Database Monitor Analysis” on page 173
- ▶ Print SQL Information
- ▶ Debug messages in job log

Note: The PRTSQLINF and Debug messages do not have the same level of index advised as those from Database Monitors and SQE Plan Cache.

Enhancements to the Index Advisor

In V5R3, the Index Advisor assists you more in suggesting indexes because the index advised code was improved to recommend more useful indexes. To take advantage of this enhancement, you should apply the MF34412 program temporary fix (PTF).

Note: The improved Index Advisor is only for SQL requests that are routed to SQE.

Index Advisor offers the following improvements:

- ▶ Advanced Radix Index advice
 - It provides advice for a more optimal index, even when a suboptimal index exists and is potentially used. For example, the query has two predicates, but an index exists on only one of the predicates.

Note: This advanced function can increase the possibility of recommending an unnecessary index where a suboptimal index is sufficient. Therefore, you must analyze the recommendations that the advisor makes.

- It improves the handling of advice regarding join predicate, grouping, ordering, and distinct clauses. A more complex combination of the record selection criteria can be advised.
- Advice is based on a high-level view of the query rather than the implementation chosen.

Attention: The CQE optimizer when suggesting indexes only considers the selection criteria and does not include join, ordering, and grouping criteria. The SQE optimizer includes selection, join, ordering, and grouping criteria when suggesting indexes.

- ▶ Advice for an encoded-vector index (EVI)
 - EVI recommendations are made for certain grouping queries.
 - Recommendations for the EVI are made when Lookahead Predicate Generation (LPG) predicates are generated.

In the past, you might have received the CPI432F message, which advised the use of an index when an index already existed in key columns. Now the SQE Optimizer does not advise an index if one already exists, even when the existing index differs from the advised index. This occurs as a result of the one of the following reasons:

- ▶ The keys of the existing index match the keys of the advised index.
- ▶ The keys of the existing index are in a different, but acceptable, order than the advised index.

Note: There are times when the keys have to be in a specified order, such as in sorting, so the SQE Optimizer advises an index if the existing one does not match the correct order.

The order of the keys in the existing index are opposite of the advised index. For example, the existing index is in descending order, and the advised index wants the keys in ascending order.

For more information, refer to Chapter 9, “Index Advisor” on page 319.

3.7 Evaluators

In this section, we cover two very useful tools that we have named the *Evaluators*. The tools are:

- ▶ Index Evaluator
- ▶ MQT Evaluator

3.7.1 Index Evaluator

Prior to V5R3, there was no easy way to determine if there were any unnecessary indexes over a physical file or table. No information specified whether an index was used by a query or when an index was used to give statistics to the query optimizer. The only information available was from the Last Used Date. Unfortunately, customers erroneously used the date to determine whether an index was used recently, and if it was not used, they deleted the index.

After the indexes were deleted, performance problems resulted because the optimizer no longer had the necessary indexes over the table to use in making a good decision. In V5R3 iSeries Navigator and V5R3 i5/OS, the Index Evaluator was added to give statistics about index usage.

Note: To activate this feature, apply the PTFs for APAR SE14709, iSeries Access Service Pack 2 for V5R3, and PTF S114782.

The new statistics fields support the following characteristics:

- ▶ There are two counters, one for when the index is used in the query implementation and one for when the index was used to gather statistics.
- ▶ Both counters are set by the two query engines, SQE and Classic Query Engine (CQE). The statistic fields are updated regardless of the query interface, such as Query/400, SQL/400®, OPNQRYF, or QQQQRY API, that is used.
- ▶ The statistics survive IPLs.
- ▶ Save/Restore does not reset the statistics on an index if an index is restored over an existing index.

Note: If an index is restored that does not exist on the system, the statistics are reset.

- ▶ In V5R3, the statistics start counting after the PTFs are applied and active.
- ▶ For each counter, there is a corresponding time stamp to go with it from the last time the counter was bumped.
- ▶ The data is stored internally in the index object. At this time, there is no way to query this. To determine usage of an index over a specific time frame, look at each time stamp on each individual index.

The new statistics are query statistics only. Native RPG, COBOL, and similar OPEN operations that are *not* queries are not covered by the new statistic fields. However, we have had native OPEN usage statistics for many years. If a user wants to determine whether the index is used via these nonquery interfaces, look at the existing statistics via fields such as Object Last Used.

Note: You can also use the QUSRMBRD API to return statistics.

To find this information, in iSeries Navigator, right-click the desired table and then click **SHOW INDEXES** as shown in Figure 3-33.

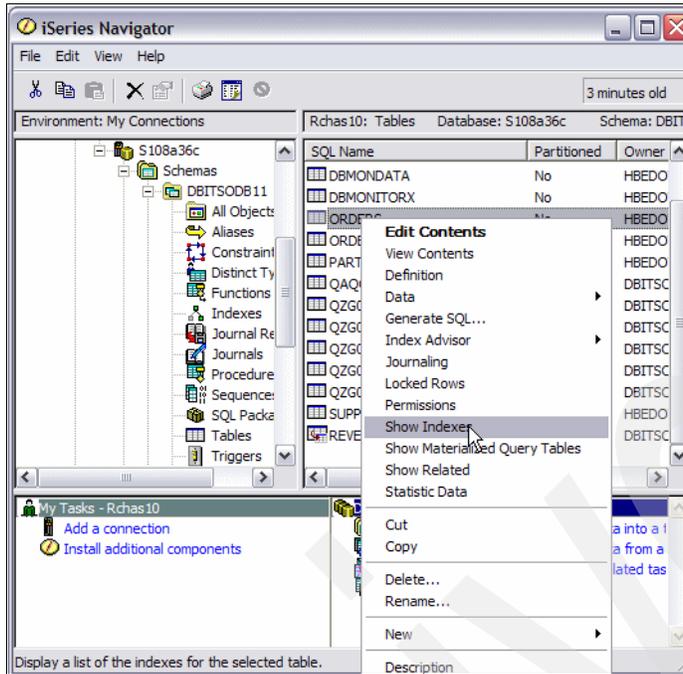


Figure 3-33 Getting index information in iSeries Navigator

In this example, you can see two indexes in the ITEM_FACT table as shown in Figure 3-34.

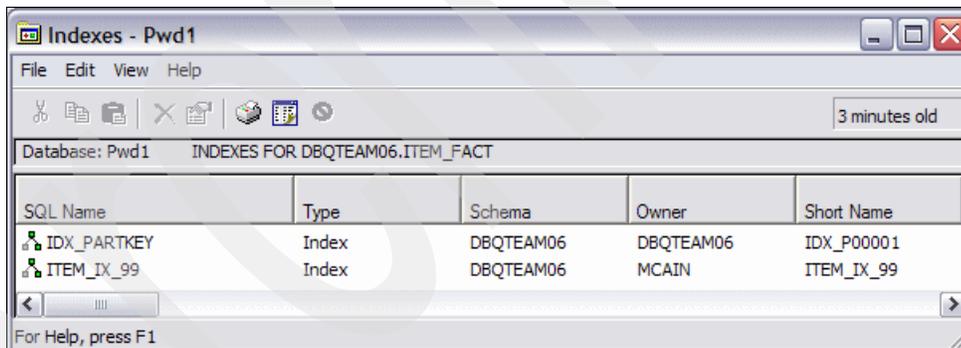


Figure 3-34 Two indexes in the ITEM_FACT table

The two counters update the columns as shown in Figure 3-35:

- ▶ Last Query Use
- ▶ Last Query Statistics Use
- ▶ Query Use Count
- ▶ Query Statistics Use

Last Query Use shows the time stamp when the index was last used to access tables in a query, and Query Use Count is updated with a corresponding count. Last Query Statistics Use is updated when the optimizer uses the index to gather statistical information about a

table, and Query Statistics Use is updated with a corresponding count as shown in Figure 3-35.

Note that the user has to scroll several screens to the right in order to get the usage statistics.

SQL Name	Type	LAST QUERY USE	LAST QUERY STATISTICS USE	QUERY USE COUNT	QUERY STATISTICS USE	CURR VALL
IDX_PARTKEY	Index	2005-03-10 13:22:21	2005-03-10 13:22:...	4	4	6005
ITEM_IX_99	Index			0	0	6005

Figure 3-35 New information about indexes in V5R3 iSeries Navigator

To see the other information, such as an index definition and description, right-click the index. Figure 3-36 shows an example of an index definition.

Order	Column Name	Short Name	Data Type	Length	Null...	Default Value
Ascending	ORDERKEY	ORDERKEY	DECIMAL	16,0	No	No default

Figure 3-36 Index Definition example

Figure 3-37 shows an example of a detailed index description.

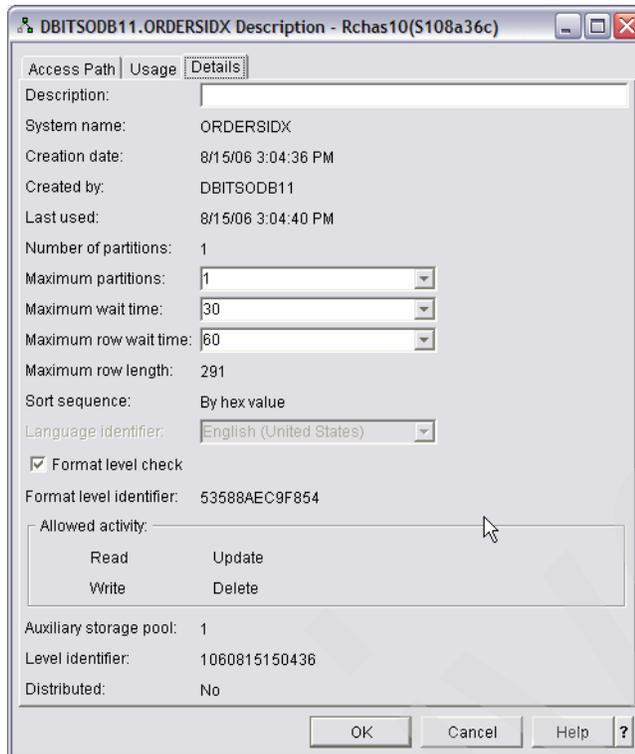


Figure 3-37 The detailed index descriptions

Note: You can also view the index information by using the Display File Description (DSPFD) command on the table. But the information does not include the newly-added fields in V5R3.

3.7.2 MQT (Materialized Query Tables) Evaluator

Materialized Query Table (MQT) support is a new feature introduced in V5R3 via a group of program temporary fixes (PTFs) delivered as of April 29, 2005. MQTs are built directly into V5R4.

An MQT is a DB2 table that contains the results of a query, along with the query's definition. An MQT can also be thought of as a materialized view or automatic summary table that is based on an underlying table or set of tables. These underlying tables can be referred to as the base tables. By running the appropriate aggregate query once using the base tables, storing the results and accessing the results on subsequent requests, data processing and query performance can be significantly enhanced.

MQT provides the most benefit when you use queries that frequently aggregate or summarize similar data from many rows that results in a few final groups (the ratio of base table rows to distinct groups is many-to-one).

The functionality of an MQT is similar to the role of an index. Both objects provide a path to the data that the user is normally unaware of. Unlike an index, a user might directly query the MQT just like a table or view. However, adapting queries to use an MQT directly might not be a trivial exercise for those who are general query users.

Note: MQTs are not automatically maintained by the system. MQTs are User maintained only.

The MQT Optimization and Use has the following characteristics:

- ▶ The query must use SQE
- ▶ The query must match or partially match the MQT
- ▶ Using the MQT is costed like any other strategy
- ▶ Only 1 MQT will be used per query in V5R3
- ▶ The MQT data integrity is the user's responsibility
- ▶ RI between base tables can help in V5R4

As shown in Figure 3-38 the SQE Optimizer can rewrite the user's query to use MQTs in situations such as the following:

- ▶ Query uses MQTs pre-aggregated, pre-calculated, pre-joined data
- ▶ Can significantly increase query performance
- ▶ Can significantly decrease resource utilization

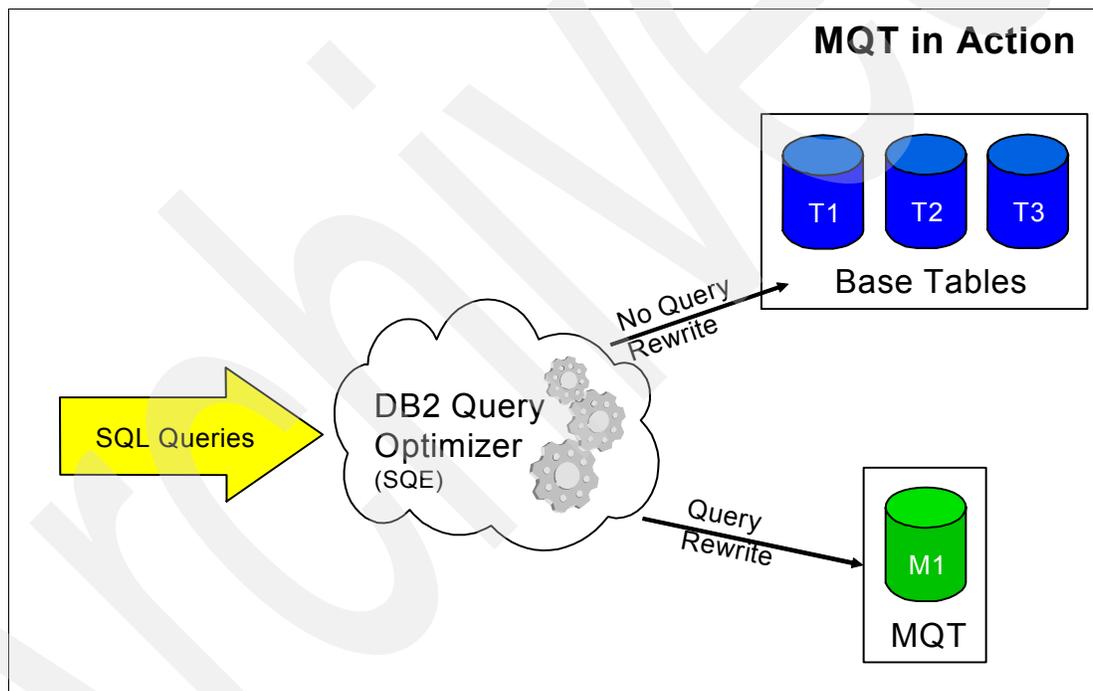


Figure 3-38 MQT as a choice for DB2 Query Optimizer (SQE)

Figure 3-39 shows a Run SQL Script window with an SQL statement to create the MQT REVENUESUMMARY in schema DBITSODB11.

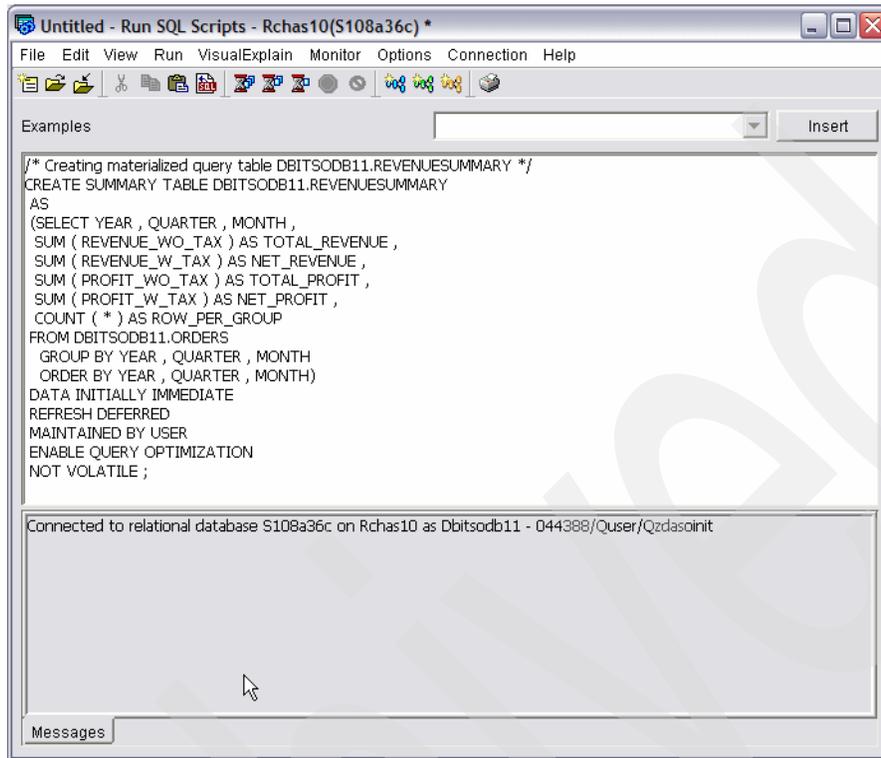


Figure 3-39 Creating an MQT

Consider an application configured to use MQTs and an SQL statement with aggregation as shown in figure Figure 3-40.

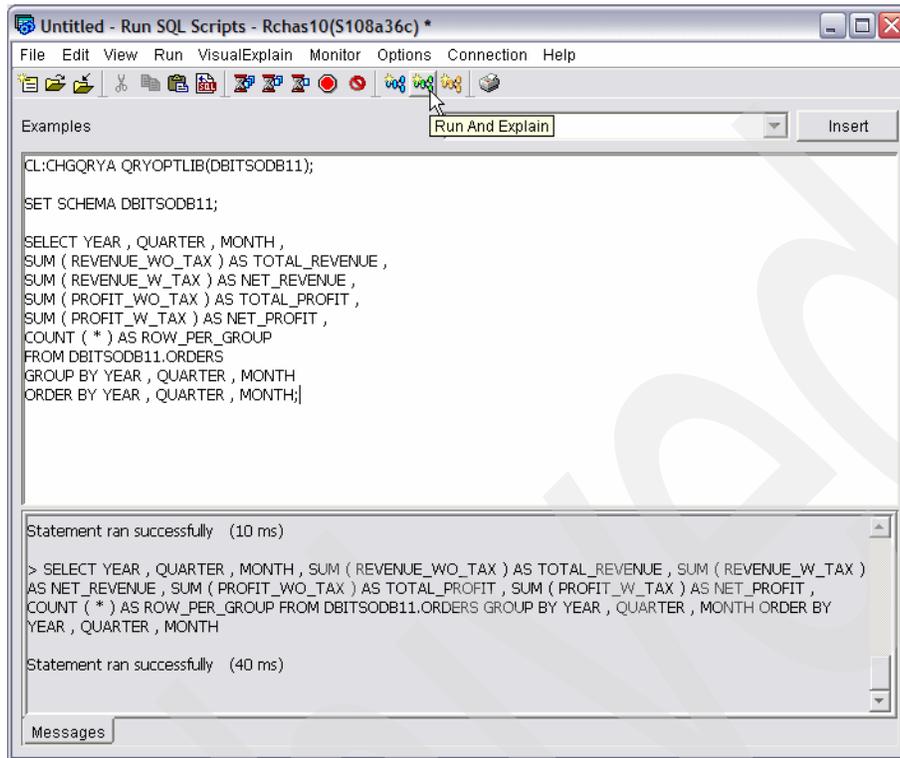


Figure 3-40 SQL statement with aggregation

We executed and explained the statements above. As you can see in Figure 3-41 the Visual Explain shows that the optimizer used the MQT DBITSODB11/REVENUESUMMARY.

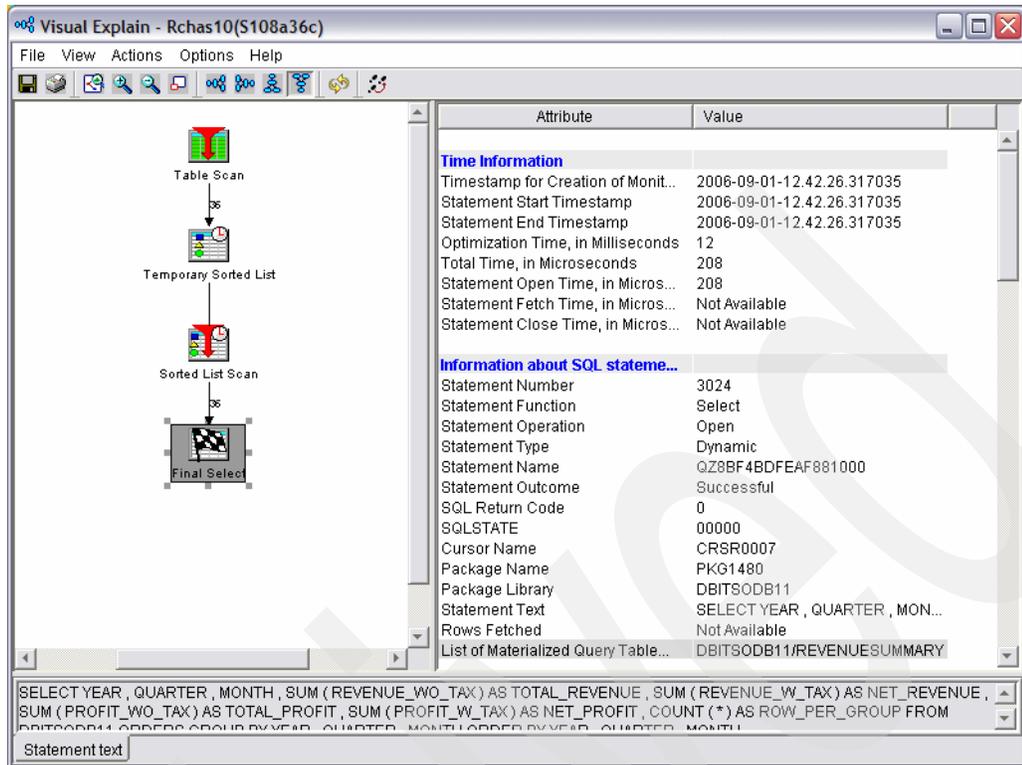


Figure 3-41 Visual Explain Showing the use of MQT

Attention: The Default behavior is not to use MQTs. You have to configure QAQQINI file options to enable/disable MQTs support. In the example above we used a QAQQINI file as shown in Figure 3-42.

QQPARAM	QQVAL	QQTEXT
APPLY_REMOTE	*DEFAULT	Specifies for database queries involving distributed file...
PARALLEL_DEGREE	*DEFAULT	Specifies the parallel processing option that can be us...
ASYNC_JOB_USAGE	*DEFAULT	Specifies the circumstances in which asynchronous (te...
QUERY_TIME_LIMIT	*DEFAULT	Specifies a time limit for database queries allowed to b...
UDF_TIME_OUT	*DEFAULT	Specifies the amount of time, in seconds, that the dat...
MESSAGES_DEBUG	*DEFAULT	Specifies whether query optimizer debug messages th...
PARAMETER_MARKER_CONVERSION	*DEFAULT	For dynamic SQL queries, specifies whether or not to ...
OPEN_CURSOR_THRESHOLD	*DEFAULT	Specifies the threshold to start full close of pseudo clo...
OPEN_CURSOR_CLOSE_COUNT	*DEFAULT	Specifies the number of cursors to full close when thre...
OPTIMIZE_STATISTIC_LIMITATION	*DEFAULT	Specifies limitations on query optimizer's statistics gath...
OPTIMIZATION_GOAL	*DEFAULT	Specifies the goal that the query optimizer should use ...
FORCE_JOIN_ORDER	*DEFAULT	Specifies that the join of tables is to occur in the order...
COMMITMENT_CONTROL_LOCK_LIMIT	*DEFAULT	Specifies the maximum number of records which can b...
REOPTIMIZE_ACCESS_PLAN	*DEFAULT	For queries with a saved access plan, this option spec...
SQLSTANDARDS_MIXED_CONSTANT	*DEFAULT	For SQL queries, this parameter specifies whether or ...
SYSTEM_SQL_STATEMENT_CACHE	*DEFAULT	Specifies for dynamic SQL queries that are not stored i...
IGNORE_LIKE_REDUNDANT_SHIFTS	*DEFAULT	Specifies whether redundant shift characters are igno...
STAR_JOIN	*DEFAULT	Specifies whether or not to enable EVI Star Join opti...
SQL_SUPPRESS_WARNINGS	*DEFAULT	For SQL statements, this parameter provides the abilit...
SQL_TRANSLATE_ASCII_TO_JOB	*DEFAULT	When using DRDA to connect to an iSeries as the appli...
NORMALIZE_DATA	*DEFAULT	Specifies whether normalization will be performed on U...
LOB_LOCATOR_THRESHOLD	*DEFAULT	Specifies either *DEFAULT or an Integer Value -- the t...
MATERIALIZED_QUERY_TABLE_USAGE	*ALL	This parameter provides the ability to control the usag...
MATERIALIZED_QUERY_TABLE_REFRESH_AGE	*ANY	This parameter provides the ability to examine which ...
ALLOW_TEMPORARY_INDEXES	*ONLY_REQUIRED	This option allows the user to indicate if temporary ind...
VARIABLE_LENGTH_OPTIMIZATION	*DEFAULT	Allows aggressive optimization techniques(Including I...
IGNORE_DERIVED_INDEX	*DEFAULT	Allows SQE to process the query even when a mapp...
CACHE_RESULTS	*DEFAULT	For SQE queries involving temporary results (e.g. sor...
LIMIT_PREDICATE_OPTIMIZATION	*DEFAULT	Indicates that the query optimizer can only use simple...
STORAGE_LIMIT	*DEFAULT	Specifies a temporary storage limit for database queri...

Figure 3-42 QAQQINI file configured to use MQTs

Characteristics of MQT usage feedback:

- ▶ Visual Explain will show MQT in place of one or more base tables
- ▶ Visual Explain does not indicate explicit MQT usage in V5R3
- ▶ Visual Explain supports “highlight MQT” in V5R4
- ▶ iSeries Navigator “Show MQTs” reports all MQTs on a table in V5R4
- ▶ SQL Performance Monitor contains information about MQT optimization and use
- ▶ Detailed Database Monitor only (that is, STRDBMON)
 - 3030 Record
 - 3000, 3001, 3002 Records
 - 3014 Record
 - 1000/3006 Records

The MQT evaluator was introduced in V5R4. The MQT evaluator shows usage statistics to help you to determine if MQTs are being used as you expected.

In order to determine that MQTs are being used as you expected right-click the table and select **Show Materialized Query Tables** menu item as shown in Figure 3-43.

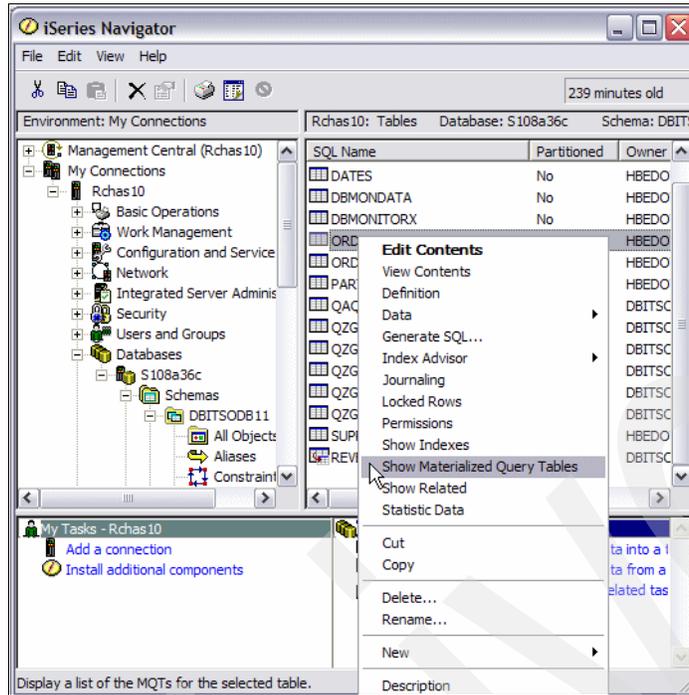


Figure 3-43 Selection Show Materialized Query Tables menu item

When you select the **Show Materialized query Tables** menu item the MQTs window appears. As shown in Figure 3-44, some columns of the MQTs window such as *Last Query Use* and *Query Use Count*, will show if the MQT has been used. Also, you can select **Definition** or **Description** menu items to see more details about the selected MQT.

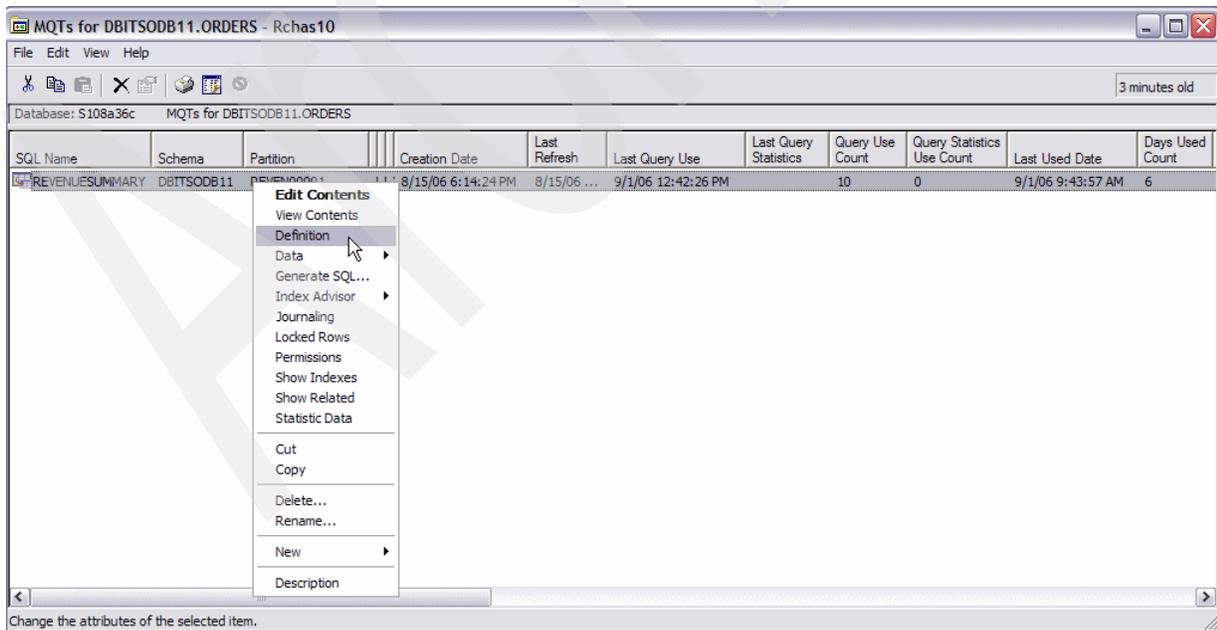


Figure 3-44 List of MQTs of a Table

Select **Definition** menu item and Definition window will appear as shown in Figure 3-45. You can see the definition of the MQTs of the selected table.

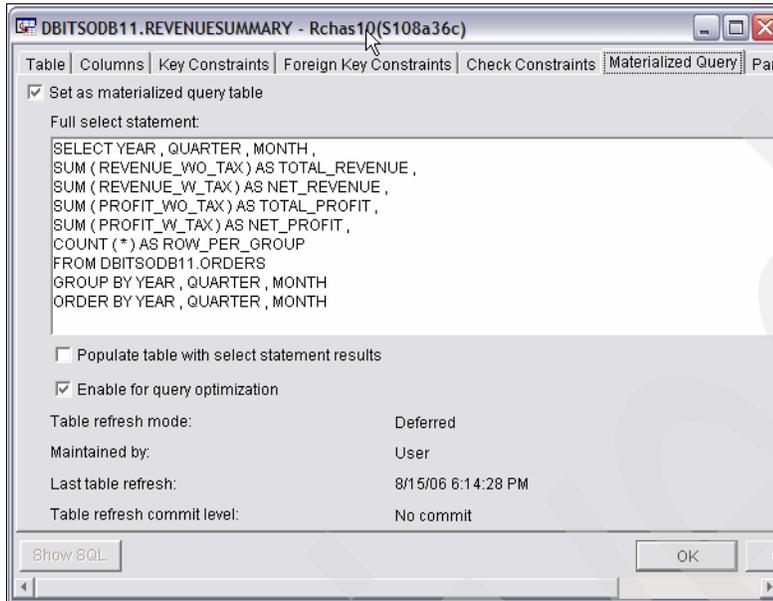


Figure 3-45 Materialized Query Table Definition

Select **Description** menu item and Description window will appear as shown in Figure 3-46. You can see the description of the MQTs of the selected table.

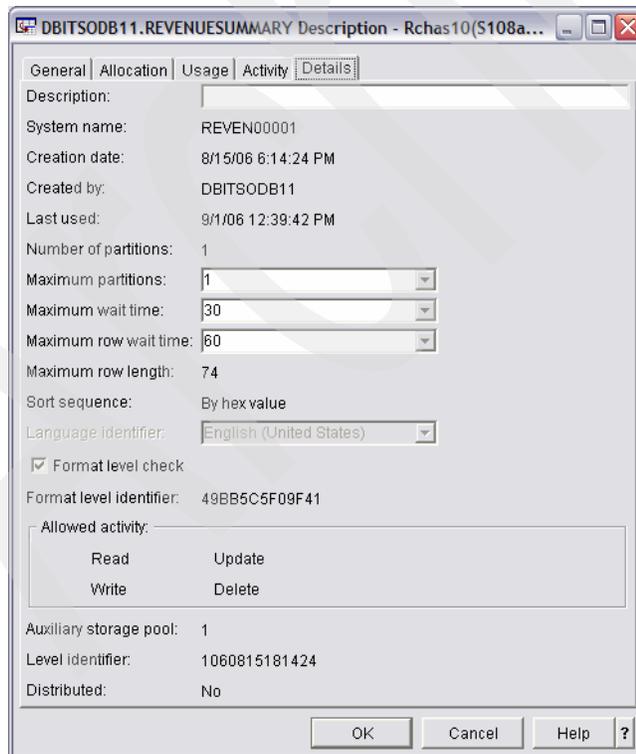


Figure 3-46 MQT Description

For detailed information about MQTs, refer to a 50-page IBM white paper named *The creation and use of materialized query tables within IBM DB2 UDB for iSeries* by Michael Cain, published in April 2005. To download a PDF file of this white paper, perform the following steps:

1. Go to the following Web address:
<http://www.ibm.com/series/db2>
2. Click **Getting Started** tab.
3. In the Web page, locate and click the hot link: White Papers.
4. Locate and click the hot link line for the article named *The creation and use of materialized query tables within IBM DB2 UDB for iSeries*.
5. Provide your information to register and download the paper.

Alternatively, you can download the paper from the Web at:

http://www-03.ibm.com/servers/enable/site/education/abstracts/438a_abs.html

Also, read MQTs topics in the Manual IBM Systems - iSeriesDB2 Universal Database for iSeries Database Performance and Query Optimization Version 5 Release 4 at the Information Center Web site:

<http://publib.boulder.ibm.com/infocenter/series/v5r4/index.jsp>

3.8 Current SQL for a Job function

You can use the Current SQL for a Job function to select any job running on the system and display the current SQL statement being run, if any. In addition to displaying the last SQL statement being run, you can edit and rerun it through the Run SQL Script option (linked automatically) and display the actual job log for the selected job or, even end the job. You can also use this function for database usage and performance analysis with the Visual Explain tool as explained in Chapter 8, "Analyzing database performance data with Visual Explain" on page 275.

Important: Current SQL only has Visual Explain capabilities for SQL statements processed by SQE.

To start the Current SQL for a Job function, in iSeries Navigator left pane, right-click **Databases** and select **Current SQL for a Job** as shown in Figure 3-47.

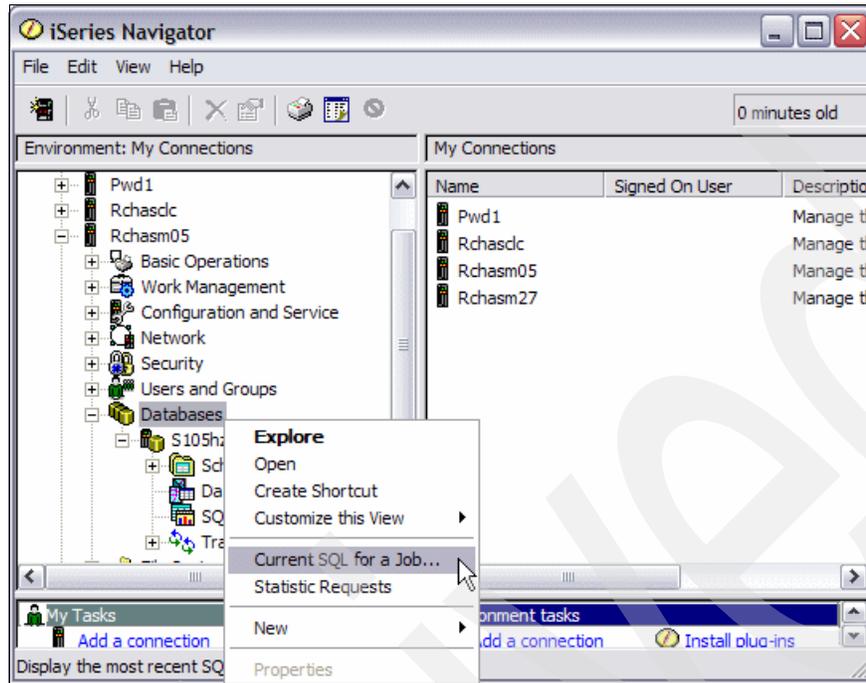


Figure 3-47 Selecting the Current SQL for a Job function in iSeries Navigator

Then the Current SQL window (Figure 3-48) opens. This window displays the name, user, job number, job subsystem, and current user for the available jobs on your system. You can select a job and display its job log, the SQL statement currently being run (if any), decide to reuse this statement in the Run SQL Script Center, or end the job, provided that you have sufficient authority.

Note: See the line "-- Statement currently running as of 05:22:37 PM". If the line "Last statement to finish as of..." appears, it is the time that the SQL statement button was selected in iSeries Navigator. This is *not* an indication of when the SQL ran. The SQL statement displayed might be the current SQL the job is running or the last SQL statement ran in the job.

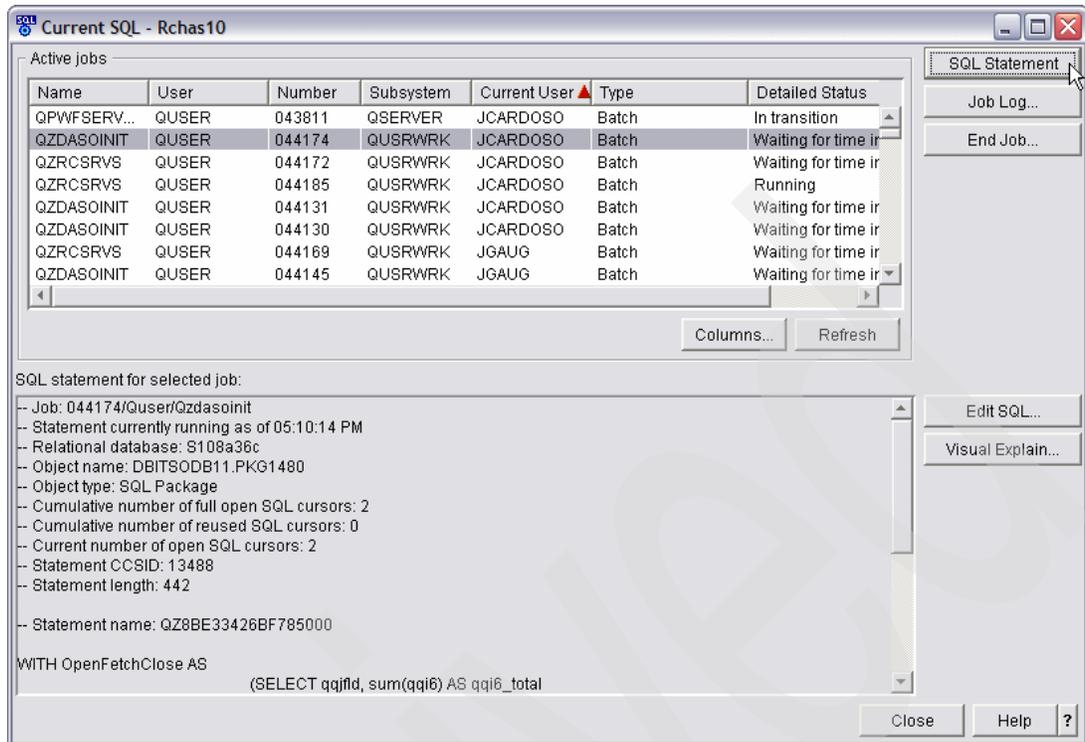


Figure 3-48 Current SQL for a Job

In our example, we selected a Java Database Connectivity (JDBC) job and clicked the SQL Statement button to view the last SQL statement that it ran, in the bottom part of the panel. To go to its job log, we can click the Job Log button. After the SQL statement is displayed in the bottom part of the panel, we can click the Edit SQL button to work on this same statement with the Run SQL Script center as shown in Figure 3-50.

Important: In V5R4 the Current SQL for a Job returns more attributes about SQL requests to help you more quickly identify the issue.

We recommend that users right-click the **Database** icon and select **Current SQL for a Job** menu option. Also, with the proper authority, you can expand the **System Name** icon then expand **Work Management** icon and select **Active Jobs** icon or **Server Jobs** icon. As shown in figure Figure 3-49 a list of jobs will appear. Right-click a job and select **Details** menu item and then **Last SQL Statement** menu item. The Run SQL Script Center similar the one shown in figure Figure 3-50 will appear. If you need to drill down into Visual Explain, the Current SQL for a Job is the preferred option.

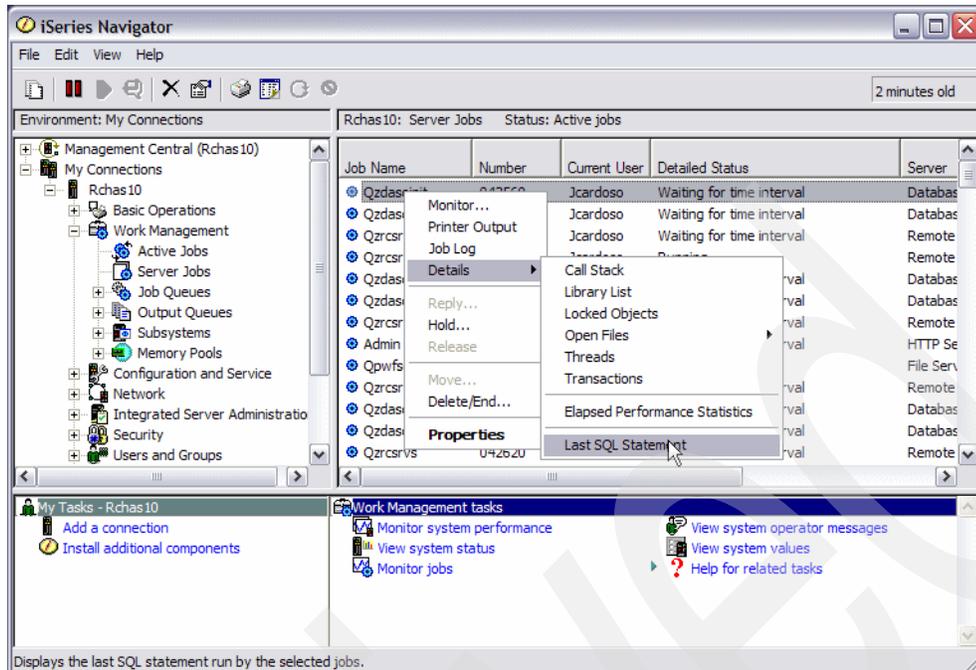


Figure 3-49 Last SQL Statement for a Job

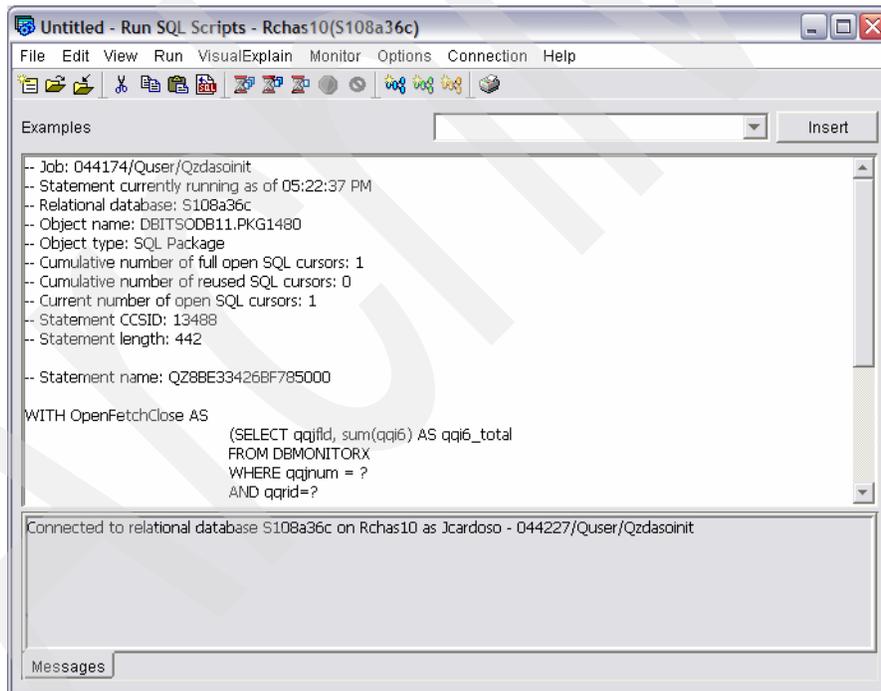


Figure 3-50 Run SQL Scripts with Current SQL for a job

Click the Run Visual Explain button to perform direct analysis of the SQL statement as shown in Figure 3-51.

Refer to Chapter 8, “Analyzing database performance data with Visual Explain” on page 275, for a more detailed explanation about how to analyze a query and use the functionality in Visual Explain within iSeries Navigator.

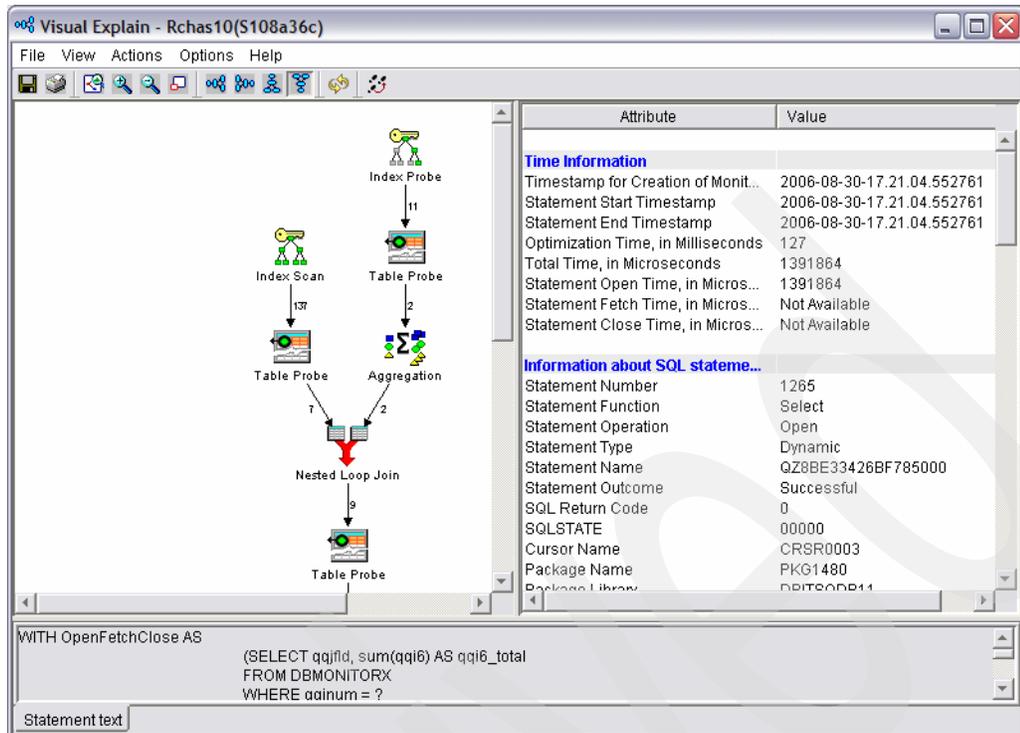


Figure 3-51 Running Visual Explain in Current SQL for a job

3.9 Debug messages

Analyzing debug messages is another important tool for monitoring and tuning queries. When doing so, keep in mind the following points:

- ▶ The debug messages are no longer being enhanced (that is, no new messages are being added for queries that go through SQE).
- ▶ No new messages are being added for queries that go through SQE, for this reason the SQE optimizer uses the closest CQE message and that message may or may not accurately reflect the access plan built by the SQE optimizer.
- ▶ It is hard to tie a message to an SQL statement.
- ▶ It is difficult to search through all of the job log messages. It is text-based only.
- ▶ There is low overhead.

Important: Starting in V5R4, IBM is treating Debug Messages as a non-strategic feedback mechanism to analyze database Performance. There is no enhancement in V5R4.

There are multiple methods of directing the system to generate debug messages while executing your SQL statements such as:

- ▶ Selecting the option in the Run SQL Scripts interface of iSeries Navigator
- ▶ Using the Start Debug (STRDBG) CL command
- ▶ Setting the QAQQINI table parameter
- ▶ Using Visual Explain

Note: We can generate debug messages from the Visual Explain interface. On the other hand, we cannot drill down to Visual Explain from Debug Messages interfaces.

You can choose to write only the debug messages for one particular job to its job log. If you want to use iSeries Navigator to generate debug messages, perform the following steps:

1. In the Run SQL Scripts window, click **Options** → **Include Debug Messages in Job Log** as shown in Figure 3-52.

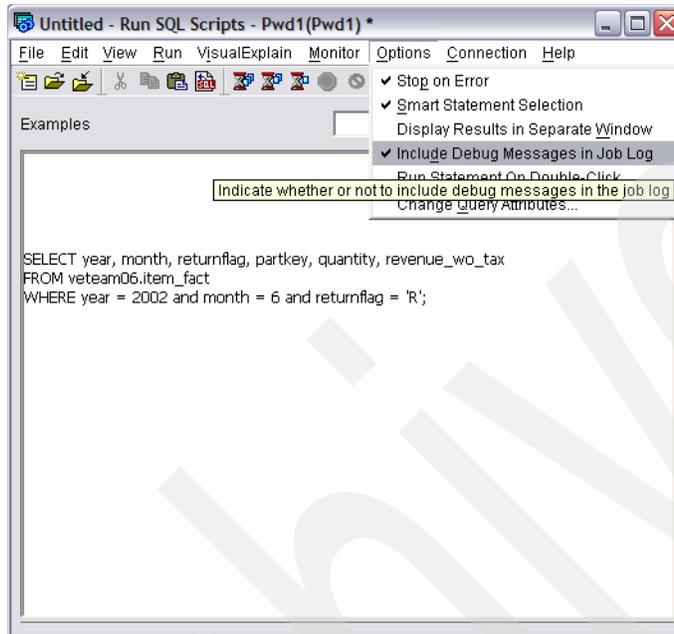


Figure 3-52 Enabling debug messages for a single job

2. After you run your query, in the Run SQL Scripts window, select **View** → **Joblog** to view the debug messages in the job log. In our example, we used the SQL statement shown in Example 3-1.

Example 3-1 Example SQL statement

```
SELECT year, month, returnflag, partkey, quantity, revenue_wo_tax
FROM veteam06.item_fact
WHERE year = 2002 and month = 6 and returnflag = 'R';
```

The detailed job log describes information that you can use to identify and analyze potential problem areas in your query such as:

- Indexes
- File join order
- Temporary result
- Access plans
- Open data paths (ODPs)

All of this information is written to the job log when under debug using the STRDBG command.

Figure 3-53 shows an example of the debug messages contained in the job log after you run the previous query.

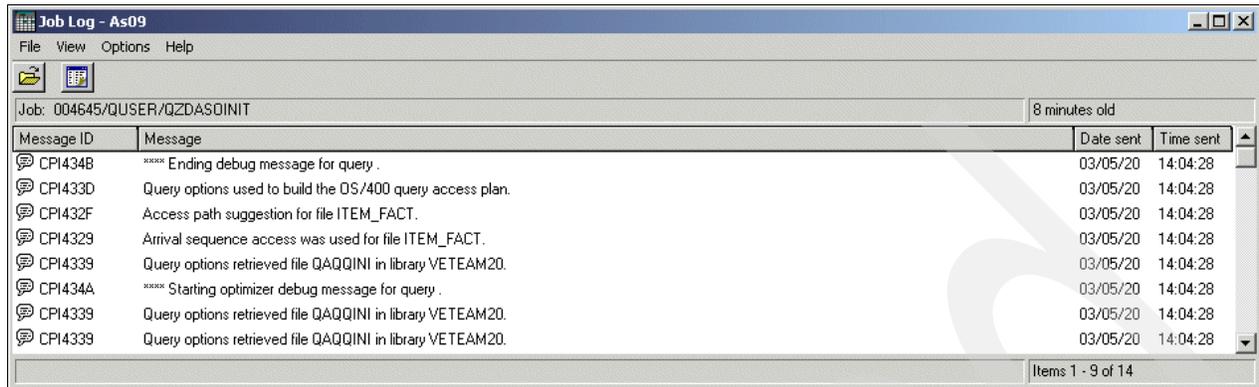


Figure 3-53 Job Log debug messages

After you enable these settings for a particular job, only debug messages relating to queries running in that job are written to the job log. You see the same debug messages with this option as those explained later when the QAQQINI parameter `MESSAGES_DEBUG` is set to `*YES`. You also see additional SQL messages, such as “SQL7913 - ODP deleted and SQL7959 - Cursor CRSRxxxx closed”, in the job log.

3. Select any of the debug messages displayed in the job log. Click **File** → **Details** to obtain more detailed information about the debug message. Figure 3-54 shows an example of a detailed debug message that is displayed.

By looking at the messages in the job log and reviewing the second-level text behind the messages, you can identify changes that might improve the performance of the query such as:

- Why an index was or was not used
- Why a temporary result was required
- Join order of the file
- Index advised by the optimizer

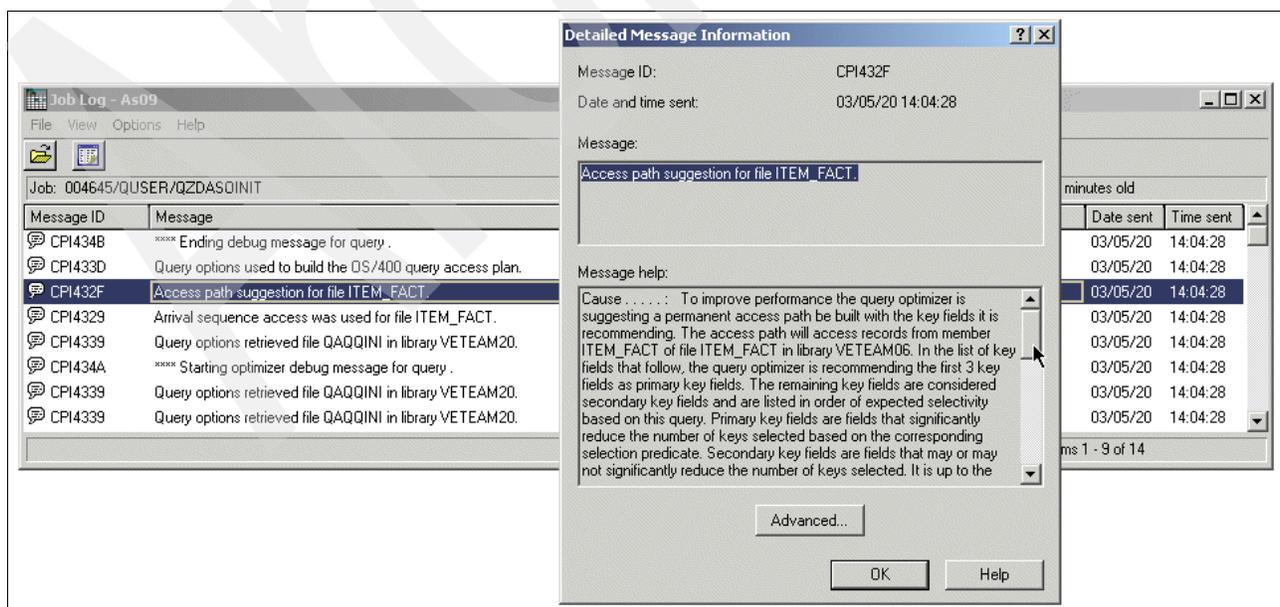


Figure 3-54 Detailed debug message information

When running SQL interactively, either through a 5250-session or via the Run SQL Scripts window in iSeries Navigator, you can also use the STRDBG CL command to generate debug messages in your job log.

Remember: You also must run the Start Server Job (STRSRVJOB) CL command if your query runs as a batch job.

By setting the value of the QAQQINI parameter MESSAGES_DEBUG to *YES, you can direct the system to write detailed information about the execution of your queries into the job's job log.

To activate this setting through the Run SQL Scripts interface of iSeries Navigator, perform the following steps:

1. Select the **QAQQINI** table that you want to use as shown in Figure 3-55.
2. Select the **MESSAGES_DEBUG** parameter and change the parameter value as shown in Figure 3-55. After you make the appropriate change, close the window.
3. In the Change Query Attributes window, click **OK** to save your changes.

QQPARAM	QQVAL	QQTEXT
APPLY_REMOTE	*DEFAULT	Specifies for database queries involving distributed files, whether or not the CHGQRY...
PARALLEL_DEGREE	*OPTIMIZE	Specifies the parallel processing option that can be used when running database que...
ASYNC_JOB_USAGE	*DEFAULT	Specifies the circumstances in which asynchronous (temp writer) jobs can be used to ...
QUERY_TIME_LIMIT	*DEFAULT	Specifies a time limit for database queries allowed to be started based on the estimat...
UDF_TIME_OUT	*DEFAULT	Specifies the amount of time, in seconds, that the database will wait for a User Define...
MESSAGES_DEBUG	*YES	Specifies whether query optimizer debug messages that would normally be issued if t...
PARAMETER_MARKER_CONVERSION	*DEFAULT	For dynamic SQL queries, specifies whether or not to allow literals to be implemented ...
OPEN_CURSOR_THRESHOLD	*DEFAULT	Specifies the threshold to start full close of pseudo closed cursors. QQVAL: *DEFAULT...
OPEN_CURSOR_CLOSE_COUNT	*DEFAULT	Specifies the number of cursors to full close when threshold is encountered. QQVAL: *...
OPTIMIZE_STATISTIC_LIMITATION	*DEFAULT	Specifies limitations on query optimizer's statistics gathering. QQVAL: *DEFAULT-Th...
OPTIMIZATION_GOAL	*ALLIO	Specifies the goal that the query optimizer should use when making costing decisions...
FORCE_JOIN_ORDER	*DEFAULT	Specifies that the join of tables is to occur in the order specified in the query. QQVAL: *

Figure 3-55 Enabling debug messages in QAQQINI

Important: Changes made to the QAQQINI table are effective immediately. They affect all users and queries that use this table. For example, if you set the MESSAGES_DEBUG parameter to *YES in a particular QAQQINI table, all queries that use that QAQQINI table write debug messages to their respective job logs.

The analysis of optimizer debug messages was made easier with the addition of a *Predictive Query Governor*. By specifying a time limit of zero in the Predictive Query Governor, you can generate query optimizer debug messages in the job log without running the query. The query time limit is checked against estimated query time before initiating your query since the optimization cost and access plan are determined prior to execution in cost-based optimization.

The time limit is set on a per-job basis such as:

- ▶ The QRYTIMLMT parameter on the CHGQRYA CL command
- ▶ The QUERY_TIME_LIMIT parameter in the QAQQINI file
- ▶ The QQRRTIMLMT system value (CHGSYSVAL QQRRTIMLMT)

That is, you can analyze a query, which might take 16 hours to run, in only a few seconds. Some changes can be made to the query or to the database. The effect can be modeled on

the query in a few minutes. The query is then run when the optimum implementation is achieved.

One of the most important debug messages to look for is advice about the creation of indexes, since the objective of creating indexes is to improve the performance of your queries. The query optimizer analyzes the record selection in the query and determines, based on the default estimate, whether the creation of an index can improve performance. If a permanent index is beneficial, it returns the key fields necessary to create the suggested index. You can find Index Advisor information in the CPI432F debug message. This takes us to the next tool, which is the Index Advisor.

3.10 Print SQL information

The information contained in SQL packages, service programs, and embedded SQL statements can also assist in identifying potential performance problems in your queries.

To view the information pertaining to the implementation and execution of your query, select an SQL package from iSeries Navigator. Right-click the **SQL package name** and select **Explain SQL**, as shown in Figure 3-56.

This is equivalent to using the Print SQL Information (PRTSQLINF) CL command that extracts the optimizer access method information from the object and places that information in a spool file. The spool file contents can then be analyzed to determine if any changes are required to improve performance.

Important: Starting in V5R4, IBM is treating Print SQL Information as a less strategic tool to analyze database Performance.

The information in the SQL package is comparable to the debug messages discussed in 3.9, “Debug messages” on page 82. However, there is more detail in the first-level SQLxxxx messages.

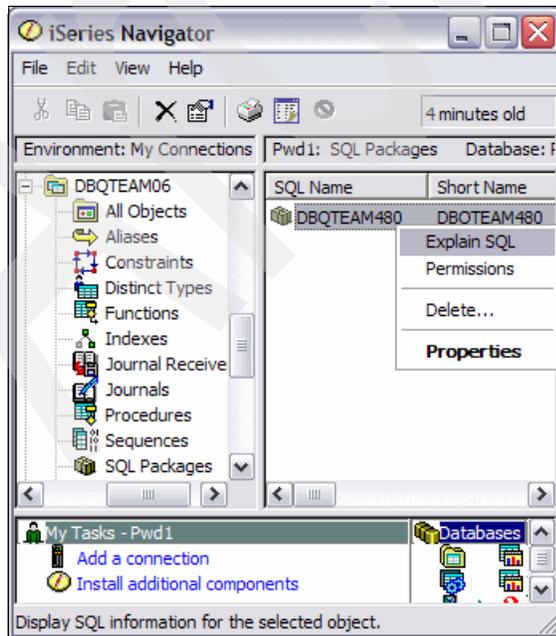


Figure 3-56 Selecting to print SQL information

Figure 3-57 shows an example of the SQL package information that can be displayed. The information pertaining to the join order of the tables, the access methods used in the implementation of the query, and runtime statistics are available. Notice the messages that indicate which QAQQINI query options file was used for executing the query and whether this query implementation used symmetric multiprocessing (SMP).

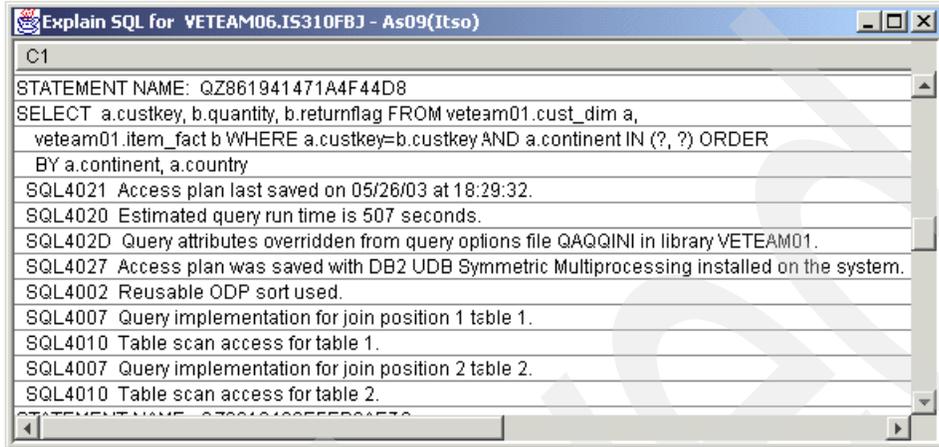


Figure 3-57 Viewing SQL package information

You can also obtain this information by using the following PRTSQLINF CL command:

```
PRTSQLINF OBJ(library_name/program_name or package_name) OBJTYPE(*SQLPKG)
```

Note: If you are unsure of the program or package name, look at the program name above the QSQ* module in the call stack when you use option 11 (Display call stack, if active) of the Work with Active Jobs (WRKACTJOB).

The PRTSQLINF CL command directs the output data to a spooled file, from which you can display or print the information.

Note: PRTSQLINF may not return the access plan last used for both program and package objects. For example, if circumstances at execution time cause the query to be re-optimized and the package or program is locked, then a new access plan is dynamically generated and placed in the SQE Plan Cache (for SQL Query Engine (SQE) use). The version stored in the package or program is not updated.

Archived



Gathering SQL performance data

After you identify a possible Structured Query Language (SQL) performance problem, you must start gathering SQL performance data. In this chapter, we discuss the different tools that are available to gather this data. Among these tools are the Detailed SQL Performance Monitor and the Summary SQL Performance Monitor.

We introduce the SQL Performance Monitor Wizard, new to V5R4, which enables you to start a Performance Monitor collection using several new collection filters to set criteria on what data is to be collected.

4.1 Types of SQL Performance Monitors

Database Monitors have been part of the OS/400 operating system since V3R6. Database Monitors and SQL Performance Monitors are used to gather information about queries run in DB2 for i5/OS. This information can help you determine whether your system and your queries are performing as they should, or whether they require fine-tuning. There are two types of SQL Performance Monitors:

- ▶ Detailed SQL Performance Monitors
- ▶ Summary SQL Performance Monitors (also known as *Memory Resident Database Monitors*)

The following sections describe the different ways that you can enable these two types of monitors to gather database performance data.

Note: Summary SQL Performance Monitor and Memory Resident Monitor are synonymous terms. In this book, we will refer to this type of monitor only as the *Summary SQL Performance Monitor*.

A Detailed Database Monitor and a Detailed SQL Performance Monitor also refer to the same type of monitor. Generally, when starting the monitor using the STRDBMON command, we refer to the *Database Monitor*, and when started using iSeries Navigator, we refer to it as an *Detailed SQL Performance Monitor*.

4.2 Collecting Detailed SQL performance data

There are different ways to start collecting detailed SQL performance data, depending on what interface you are using. In this section, we look at several different ways to start an SQL Performance Monitor, along with the various new parameters which can be used to filter what monitor data is captured. The monitor can be started from any of the following:

- ▶ The CL command line (that is, the green-screen)
- ▶ The new SQL Performance Monitor Wizard
- ▶ Open Database Connectivity (ODBC) clients
- ▶ Object Linking and Embedding (OLE) DB clients
- ▶ Java Database Connectivity (JDBC) clients
- ▶ .NET clients
- ▶ An exit program

4.2.1 Starting a Detailed Database Monitor using the command interface

You can start a Detailed Database Monitor from the command line interface by using the STRDBMON command. At V5R3 and earlier releases, you were limited to either specifying a single job or *ALL jobs on the system. At V5R4, you can be more precise on what jobs you wish to monitor. There are also parameters available on the STRDBMON command to set the collection filters introduced in the previous section.

When you start a new monitor with the STRDBMON command, you must specify the file to which the SQL performance information is to be written. If the file or member does not exist, one is created based on the QAQQDBMN file in QSYS library. If the file or member does exist, the record format of the file is checked to see if it is the same.

Note: Be aware that many of the parameters of the CL Command STRDBMON are new. Therefore, they are not available on pre-V5R4 releases.

Consider the following parameters before you start database monitoring:

Basic parameters

The STRDBMON command uses the following basic parameters:

► **OUTFILE**

- The file name is required, the library name is optional.
- It is created if it does not exist or is reused if it does exist.

► **OUTMBR**

- Member to receive output: Defaults to first member in file(*FIRST)
- Replace or add records: Defaults to replace member in file (*REPLACE)

► **JOB**

This parameter defaults to the job issuing the STRDBMON command. The user can specify a single job, specify *ALL for all jobs, or specify some subset of jobs. A subset of jobs can be specified in several ways:

- All jobs for a particular user can be monitored with the following command:
`STRDBMON OUTFILE(MYLIB/DBMON) JOB(*ALL/JOEUSER/*ALL)`
- All of the same named jobs can be monitored for multiple users with the following command:
`STRDBMON OUTFILE(MYLIB/DBMON) JOB(*ALL/*ALL/QZDAS0INIT)`
- Generic job names and user names can also be specified to provide for further flexibility with the following command:
`STRDBMON OUTFILE(MYLIB/DBMON) JOB(*ALL/J*/QPADEV*)`

Note: A monitor that is started over multiple jobs or all jobs on the system is known as a *public* monitor.

If the monitor is started on all jobs or a subset of all jobs, any jobs waiting on job queues or any jobs started during the monitoring period have statistics gathered from them after they begin. If the monitor is started on a specific job, that job must be active in the server when the command is issued.

► **TYPE**

This parameter allows the user to specify the type of data to be collected:

- *BASIC provides all necessary analysis data.
- *DETAIL collects the same data as *BASIC plus the 3019 row. This type of collection causes a little more overhead on the system but will log statistics regarding database I/O operations.
- *SUMMARY collects the same data as *BASIC.

Important: The *BASIC parameter value is new and is the default at V5R4. This collects the same level of detail as the *SUMMARY value did at V5R3 and earlier releases. This change was made to avoid confusion with the Summary SQL Performance Monitor, also known as the *memory resident monitor*. *SUMMARY is still an option for the TYPE parameter, but is only provided for consistency with prior releases and will provide the same detail as *BASIC. *It is not possible to start the memory resident Summary SQL Performance Monitor using the STRDBMON command.*

► FRCRCD

This parameter allows you to control the number of rows that are kept in the row buffer of each job being monitored before forcing the rows to be written to the output table. The default value is *CALC.

By specifying a force row write value of FRCRCD(1), monitor rows are displayed in the log as soon as they are created. FRCRCD(1) also ensures that the physical sequence of the rows is most likely, but not guaranteed, to be in time sequence. However, FRCRCD(1) causes the most negative performance impact on the jobs being monitored. By specifying a larger number for the FRCRCD parameter, the performance impact of monitoring can be reduced.

► INCSYSSQL

- This parameter uses the default value of *NO.
- *NO - No monitor records will be created for system-generated SQL statements. Monitor records will only be created for user-specified SQL statements.
- *YES - Monitor records will be created for both user-specified and system-generated SQL statements.
- *INI - Monitor records will be generated based on the value of the SQL_DBMON_OUTPUT option in the current INI file. A value of *USER or *DEFAULT creates monitor records for only user-specified SQL statements. A value of *SYSTEM creates monitor records for only system-generated SQL statements. A value of *ALL creates monitor records for both user-specified and system-generated SQL statements. If no INI file exists, then a default value of *NO will be used for the INCSYSSQL option.

► RUNTHLD

- This Parameter uses the default value of *NONE.
- If you type a runtime-threshold, the Monitor records will be created for all SQL statements whose estimated runtime meets or exceeds this value.

► COMMENT

This parameter allows you to add a meaningful description to the collection. It specifies the description that is associated with the Database Monitor record whose ID is 3018.

Filter parameters

The new V5R4 collection filters can be applied using the STRDBMON command as well, as shown in Figure 4-1. This is where you can specify to filter the data based on file name, user name, or internet IP address.

```

Start Database Monitor (STRDBMON)
Type choices, press Enter.
Filter by database file . . . . *NONE
-----
Library . . . . . Name, generic*
+ for more values -----
-----
Filter by user profile . . . . *NONE Name, generic*, *NONE...
Filter by internet address . . . *NONE
Comment . . . . . *BLANK
-----
-----
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
Bottom
  
```

Figure 4-1 New Database Monitor filters

► FTRFILE

- This parameter uses the default value of *NONE
- *ALL - Monitor records will be created for any SQL statement that uses any file in the specified library. If none of the files used in the SQL statement come from the specified library, no monitor records will be created for the SQL statement.
- File Name - Monitor records will be created only for those SQL statements that use the specified file. Monitor records will not be created for any SQL statements that do not use the specified file.
- Generic File-name - Monitor records will be created only for those SQL statements that use a file that matches the generic prefix. If none of the files used in the SQL statement match the specified prefix, no monitor records will be created for the SQL statement.
- Library Name - Monitor records will be created only for those SQL statements that use a file from the specified library. Monitor records will not be created if none of the files used in the SQL statement come from the specified library.
- Generic Library Name - Monitor records will be created only for those SQL statements that use a file from a library that matches the generic prefix. If none of the files used in the SQL statement come from the generic library, no monitor records will be created for the SQL statement.

You could use the following CL command to start a monitor to collect data only for operations against files with the prefix CUST across any library beginning with the character B:

```
STRDBMON OUTFILE(MYLIB/CUST_DBMON) JOB(*ALL) FTRFILE(B*/CUST*)
```

► FTRUSER

- This parameter uses the default value of *NONE.
- *CURRENT - Monitor records will be created only for those SQL statements that are executed by the user who is invoking the STRDBMON command. Monitor records will not be created for SQL statements executed by a different user.

- User name - Monitor records will be created only for those SQL statements that are executed by the specified user. Monitor records will not be created for SQL statements executed by a different user.
- Generic-user-name - Monitor records will be created only for those SQL statements that are executed by a user whose name starts with the specified prefix. Monitor records will not be created for SQL statements executed by a different user.

In the following example, the filter will limit the capture to only those statements run by JOEUSER in a QZDASOINIT job:

```
STRDBMON OUTFILE(MYLIB/USERDBMON) JOB(*ALL/*ALL/QZDASOINIT) FTRUSER(JOEUSER).
```

► FTRINTNETA

- This parameter uses the default value of *NONE.
- Internet-address - The internet address is specified in the form nnn.nnn.nnn.nnn, where nnn is a decimal number ranging from 0 through 255, without the leading zeros. (An internet address having all binary ones or zeros in the bits of the network or host identifier portions of the address is not valid.)

This filter allows you to log data for only SQL requests started from a particular remote system. For example:

```
STRDBMON OUTFILE(MYLIB/IPDBMON) JOB(*ALL) FTRINTNETA('9.10.20.30')
```

The filters may be used together in any combination. By using this new filters, you can reduce both CPU overhead and disk consumption by only logging the information that will be of interest.

The monitor ID

At V5R4, the system generates a 10-digit monitor ID when the STRDBMON command is successfully processed. In cases where you may be running multiple monitors, you should make a note of this unique ID so that the desired monitor can later be ended. The system issues message CPI436A which gives you the monitor ID:

```
Database monitor started for job *ALL/*ALL/*ALL, monitor ID 0416572016.
```

If you forget to record the monitor ID and later need to retrieve it, you can do so by querying the QQC101 column of the 3018 record in the monitor output file:

```
select qqc101 from mylib.dbmon where qqrid = 3018
```

Ending a Detailed Database Monitor

To end the collection of database performance data started using the STRDBMON command you must use the End Database Monitor (ENDDDBMON) command. The following parameters are available for this command:

► JOB

You can specify to end a monitor that was started for that named job, as long as that is the only monitor running for the given job name. If multiple monitors are running for the same job name, then you must end the monitor by using the monitor ID.

Use the value *ALL for this parameter to end a database monitor started against *ALL jobs.

► MONID

Use this parameter to specify the monitor ID of the monitor you wish to end. If you have multiple monitors running that were started with the same values for the JOB parameter,

then you should use the MONID parameter to uniquely identify which monitor you want to end.

Note: If you start a monitor collection and specify JOB(*ALL), you cannot end that monitor for individual jobs. You must specify *ALL jobs on the ENDDDBMON parameter to end monitoring all jobs at once.

A very helpful new feature of V5R4 is that database monitors started using the STRDBMON command are now automatically presented in the SQL Performance Monitor GUI as a started monitor. This allows you to keep track of all started monitors from one central location. It also eliminates the need to manually import a monitor that was started on the same system using the native interface.

For example, you can start a monitor using the STRDBMON command, then end it or analyze it, or both, using the iSeries Navigator interface, without having to import it. A database monitor started using STRDBMON will have the naming convention “USER TABLE DBMONID” for its SQL Performance Monitor name, where USER is the user who started the database monitor, TABLE is the name of the database monitor output file, and DBMONID is the 10 digit monitor ID. Figure 4-2 shows a database monitor through the SQL Performance Monitor interface that was started using the STRDBMON command.

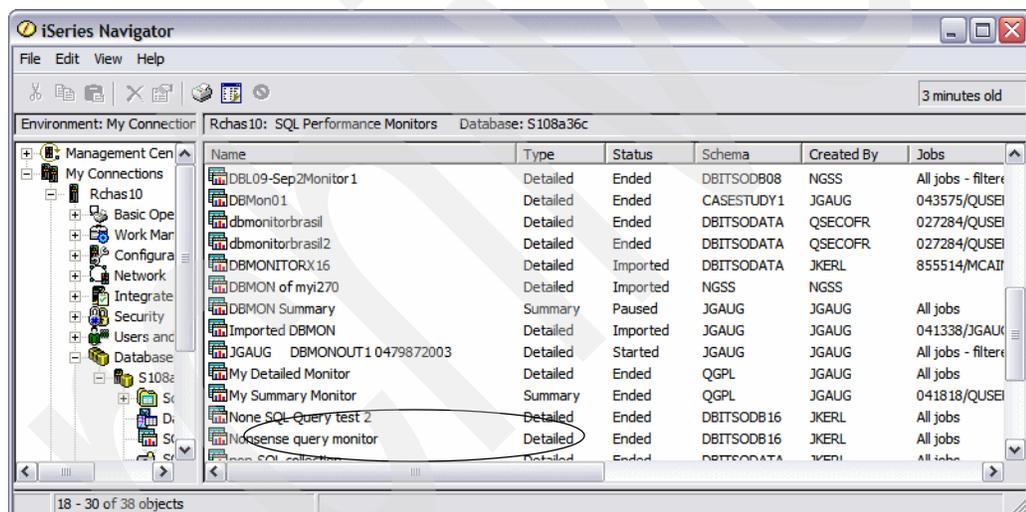


Figure 4-2 View of a monitor started using STRDBMON

Important: You need the following V5R4 PTFs applied in order for a database monitor started using STRDBMON to be automatically visible in the SQL Performance Monitor GUI:

- ▶ SI25041
- ▶ SI24847

If for some reason you do not have access to the SQL Performance Monitor GUI interface, there is a trick which you can use to determine what monitors are active system-wide. From the command line interface, enter the following command:

```
STRDBMON OUTFILE(QTEMP/DUMMY) COMMENT(DISPLAY_ACTIVE_MONITORS)
```

The name specified for the outfile does not matter, but the comment value “DISPLAY_ACTIVE_MONITORS” has a special meaning and it will cause a CPF436C to be

issued in your joblog for each active monitor on the system. The Monitor ID will be listed in the second level text of the CPF436C, allowing you to end any active public monitors by Monitor ID.

4.2.2 The SQL Performance Monitor Wizard

In V5R4 iSeries Navigator provides a new SQL Performance Monitor Wizard to start an SQL performance monitor. This wizard has very complete prefiltering capabilities. To start the wizard, right-click **SQL Performance Monitors** → **New** → **SQL Performance Monitor**, as shown in Figure 4-3.

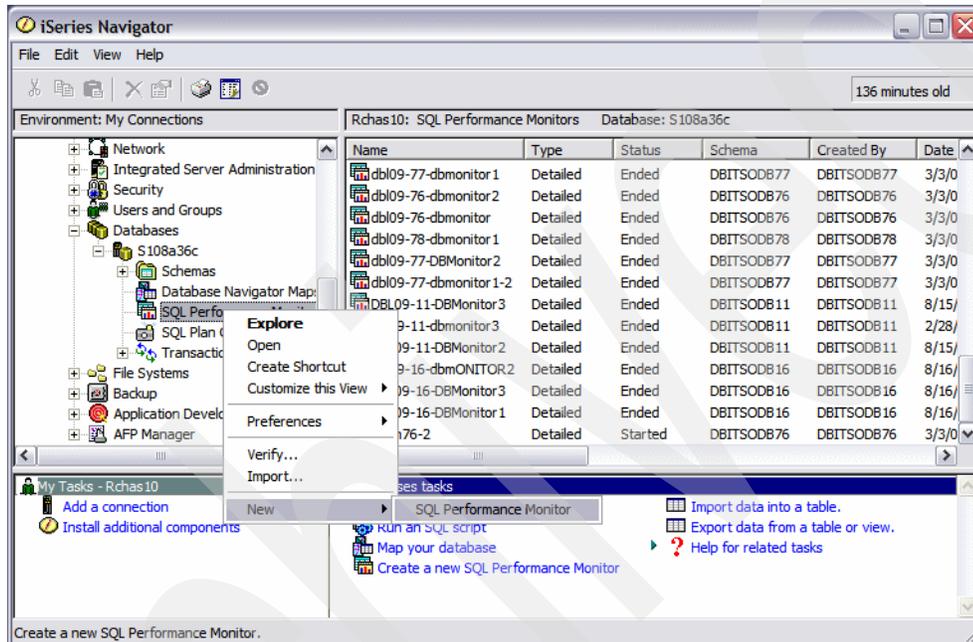


Figure 4-3 Starting a new SQL Performance Monitor

The SQL Performance Monitor Wizard window is displayed, Figure 4-4. From this window, you can choose to start a Summary SQL Performance Monitor or a Detailed SQL Performance Monitor.

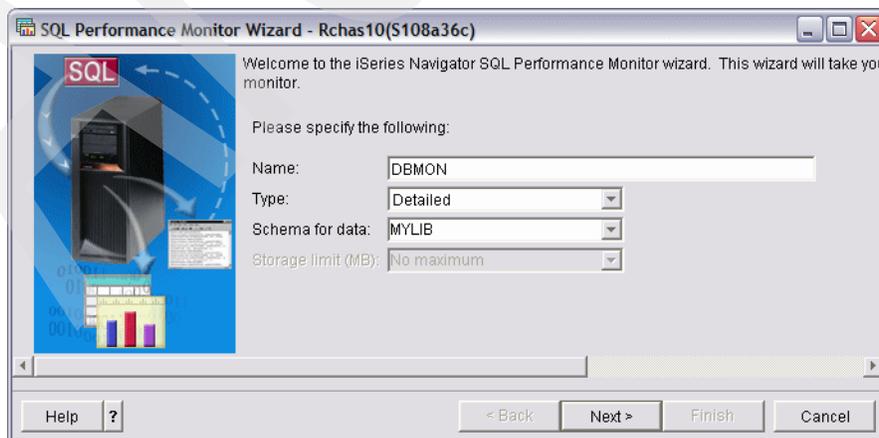


Figure 4-4 Welcome to the new SQL Performance Monitor Wizard

4.2.3 Starting a Detailed SQL Performance Monitor

The Detailed SQL Performance Monitor in iSeries Navigator provides the same level of detail as running the STRDBMON command with *DETAIL in the TYPE parameter, which was explained in 4.2.1, “Starting a Detailed Database Monitor using the command interface” on page 90. To start the Detailed SQL Performance Monitor, perform the following steps:

1. From the SQL Performance Monitor Wizard, give your Detailed monitor a name and select **Detailed** from the Type box (this is the default), Figure 4-5.

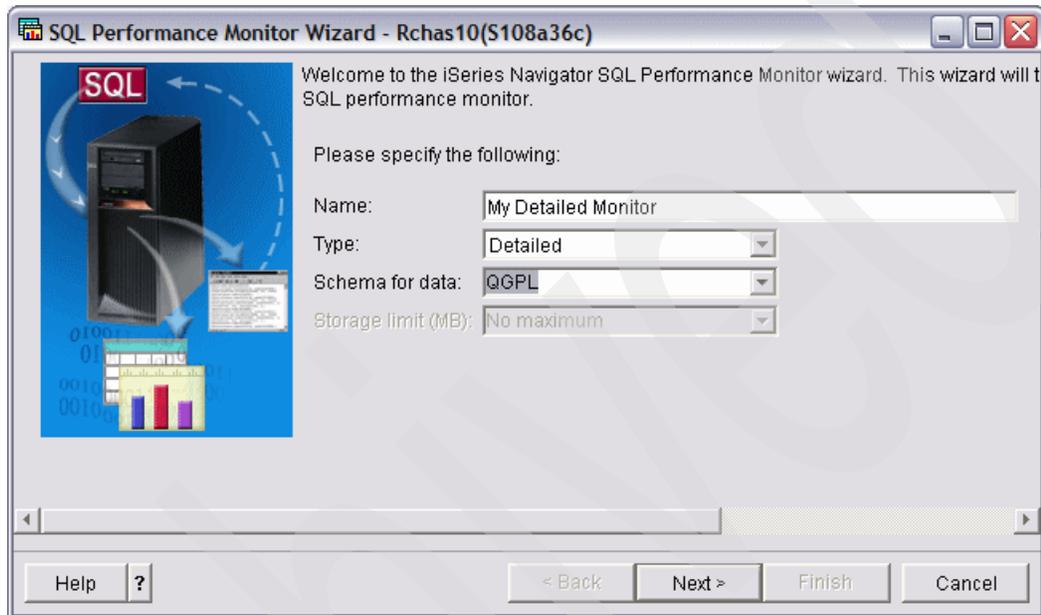


Figure 4-5 Starting a detailed performance monitor.

2. The next window presented allows you to set filters on the data that is to be collected. There are several filtering options which are new to V5R4. You can see these in Figure 4-6. Specify any combination of the filters to limit the data that is collected by the

monitor, which in turn will reduce CPU consumption and disk usage by only capturing the data you are most interested in.

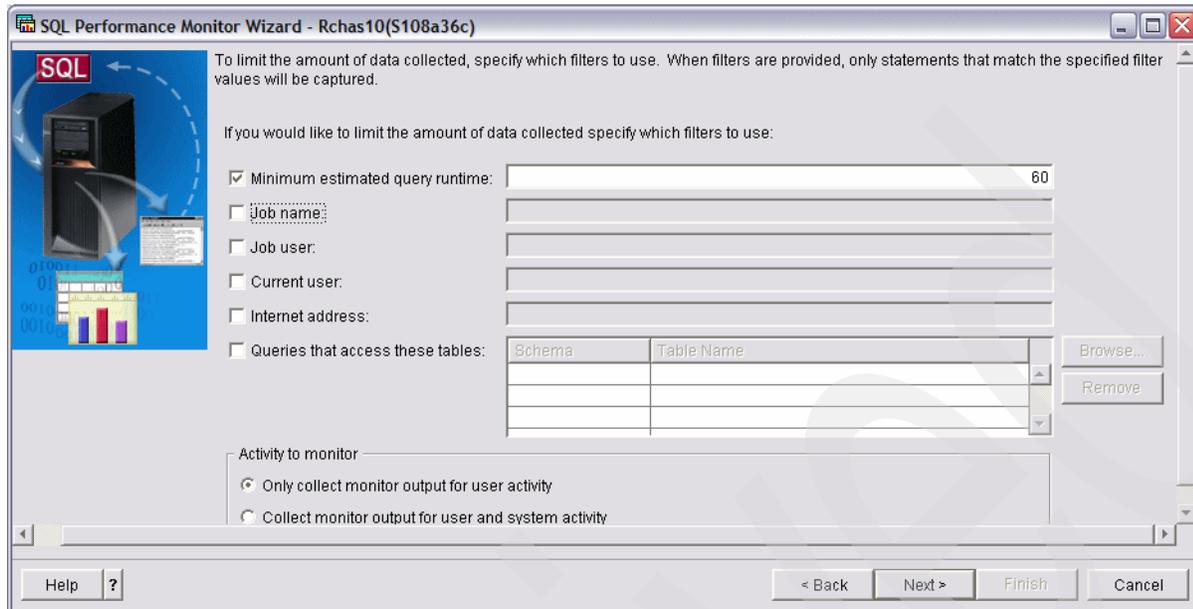


Figure 4-6 Specify filtering options

The filtering options are as follows:

– Minimum estimated query runtime

Specify a value in seconds. If any query's runtime is estimated to be less than the number of seconds specified here, information about that query will not be collected. This option can allow you to greatly reduce the amount of data collected by only gathering data for long-running queries. In this example, we are setting the minimum filter time to 60 seconds.

– Job name

Specify a job name here to limit the collection to only activity from a particular named job. For example, specify QZDASOINIT as the job name if you only want to collect data for jobs with that name. You can use a wildcard to specify a generic name.

– Job user

Specify a job user to collect monitor data only for jobs in which the specified user name appears in the fully qualified job name. Wildcards may be used.

– Current user

When this parameter is specified, only information for queries run by the named user will be collected. Current user differs from job user in that the current user refers to the current user for the SQL statement, and not necessarily the user name that appears on the fully qualified job name. This filter is especially useful in a client/server environment where you have all server jobs running under the same user profile such as QUSER. Filtering on the current user will allow you to monitor only data for that SQL run by a particular user, even though all jobs are serviced by the QUSER profile.

– Internet address

This filter allows you to log data for only SQL requests started from a particular remote system. Specify the IP address in the format xxx.xxx.xxx.xxx.

- Queries that access certain tables

Checking this filter and then clicking the **Browse** button will bring up a dialog in which you can select, by schema, a table or list of tables. The monitor will only collect information pertaining to SQL activity on the selected tables.

- User activity only

The default is to collect statements only for user activity. If you have a need to collect data for systems activity as well, then check that option.

Note: At V5R3, this feature was supported using SQL_DBMON_OUTPUT in the QAQQINI query options file.

3. Once you set any desired filters, click **Next** to move on to the job selection window. If you want to limit the collection further by monitoring only a specific job or jobs, then click the **Specific Jobs** option and you will be presented with a list of active jobs in a new window from which you can select **Add** to add specific jobs. In this example, we have selected to monitor all jobs, Figure 4-7.

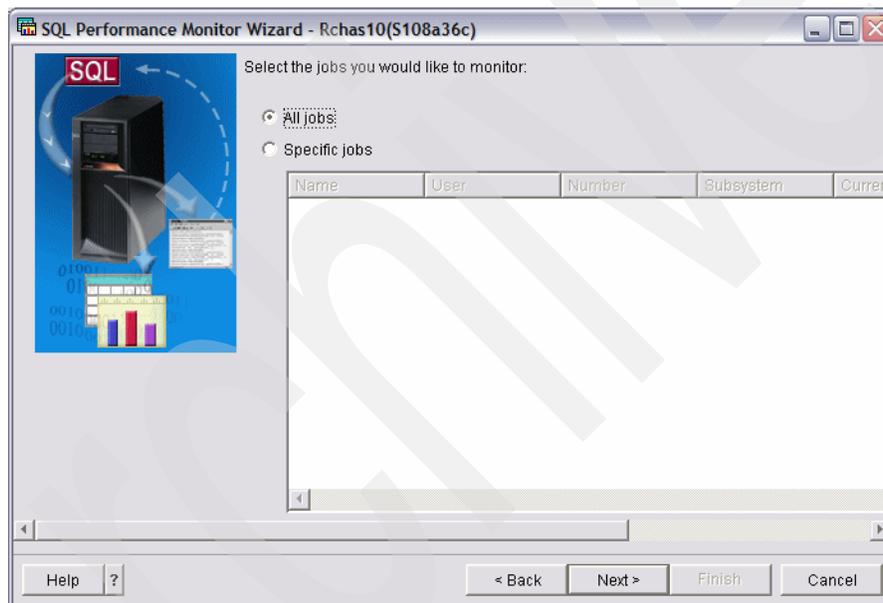


Figure 4-7 Select jobs to monitor

- You are presented with an overview on the selections you've made, as shown in Figure 4-8. Use this opportunity to review your choices, and if satisfied, select **Finish** to begin the monitor.



Figure 4-8 Review your Detailed SQL Performance Monitor selections

Figure 4-9 shows that your monitor is now listed in the list of the system's SQL Performance monitors with a status of started.

Name	Type	Status	Schema	Created By	Date Created	Jobs
DBL09-06-DBMonitor3	Detailed	Ended	DBITSODB06	NGSS	8/15/06 3:5...	040626/QUSER/QZD
DBL09-11-DBMonitor1	Detailed	Ended	DBITSODB11	DBITSODB11	8/15/06 1:5...	All jobs - filtered
DBL09-11-DBMonitor2	Detailed	Ended	DBITSODB11	DBITSODB11	8/15/06 3:2...	040541/QUSER/QZD
DBL09-11-DBMonitor3	Detailed	Ended	DBITSODB11	DBITSODB11	8/15/06 3:5...	040532/QUSER/QZD
DBL09-16-DBMonitor1	Detailed	Ended	DBITSODB16	DBITSODB16	8/16/06 9:0...	All jobs - filtered
dbl09-16-dbmONITOR2	Detailed	Ended	DBITSODB16	DBITSODB16	8/16/06 8:4...	041137/QUSER/QZD
DBL09-16-DBMonitor3	Detailed	Ended	DBITSODB16	DBITSODB16	8/16/06 9:2...	041137/QUSER/QZD
dbmonitorbrasil	Detailed	Ended	DBITSODATA	QSECOFR	4/25/06 9:4...	027284/QUSER/QZD
dbmonitorbrasil2	Detailed	Ended	DBITSODATA	QSECOFR	4/25/06 9:4...	027284/QUSER/QZD
My Detailed Monitor	Detailed	Started	QGPL	JGAUG	8/18/06 11:...	All jobs
My Summary Monitor	Summary	Ended	QGPL	JGAUG	8/18/06 8:5...	041818/QUSER/QZD
SAMPLEDB06 db MON...	Detailed	Ended	SAMPLEDB06	DBNAV06	2/28/06 4:2...	017129/QUSER/QZD
SAMPLEDB06 MONIT...	Summary	Ended	SAMPLEDB06	DBNAV06	2/28/06 4:3...	016854/QUSER/QZD
SAMPLEDB06 MONIT...	Detailed	Ended	SAMPLEDB06	DBNAV06	2/28/06 4:3...	017134/QUSER/QZR
SAMPI FDR07	Detailed	Ended	SAMQIFDR07	DBNAV07	2/28/06 4:5...	017154/OI ISFR/QZD

Figure 4-9 Started Detailed SQL Performance Monitor

Note: The data collection in an SQL Performance Monitor is done inline within the job instead of in a separate job.

Optimization records and data do not appear for queries that are already in reusable open data path (ODP) mode when the monitor is started. To ensure the capture of this data for a batch job, start the monitor before the job starts and collect it over the entire length of the job or as much as needed.

4.2.4 Enabling Database Monitors in ODBC clients

You can start a database monitor before a client initiates a connection to the server by using the STRDBMON command to monitor the client job. Other options are available for you to start a monitor. The two ways to start a Detailed Database Monitor in ODBC clients are:

- ▶ Enabling the Database Monitor option in the data source name
- ▶ Using an ODBC connection keyword to start the Database Monitor

ODBC data source name

To start a monitor using the data source name (DSN), perform the following steps:

1. In Microsoft® Windows XP, click **Start** → **Programs** → **IBM iSeries Access for Windows** and then select **ODBC administration**.
2. From the ODBC Data Source Administrator window (Figure 4-10), select the desired data source name and click the **Configure** button.

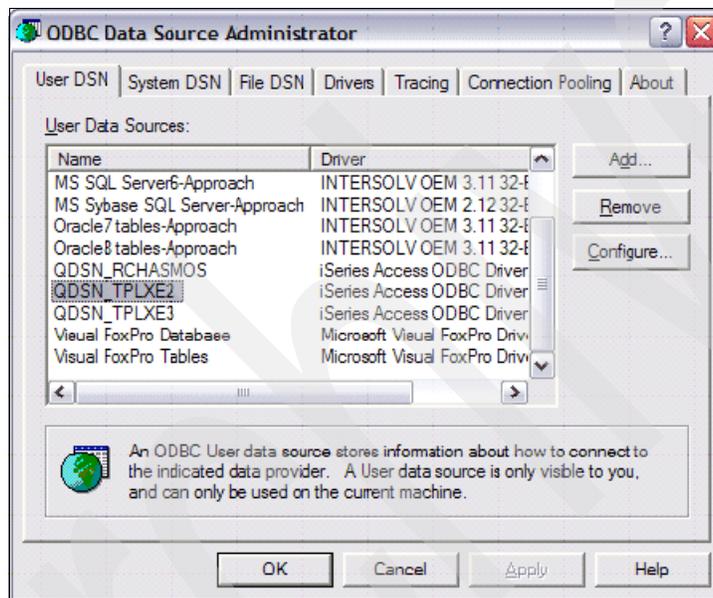


Figure 4-10 ODBC Data Source Administrator window

3. In the iSeries Access for Windows ODBC setup window (Figure 4-11) that opens, click the **Diagnostic** tab and then select the **Enable Database Monitor** option. Click **OK**.

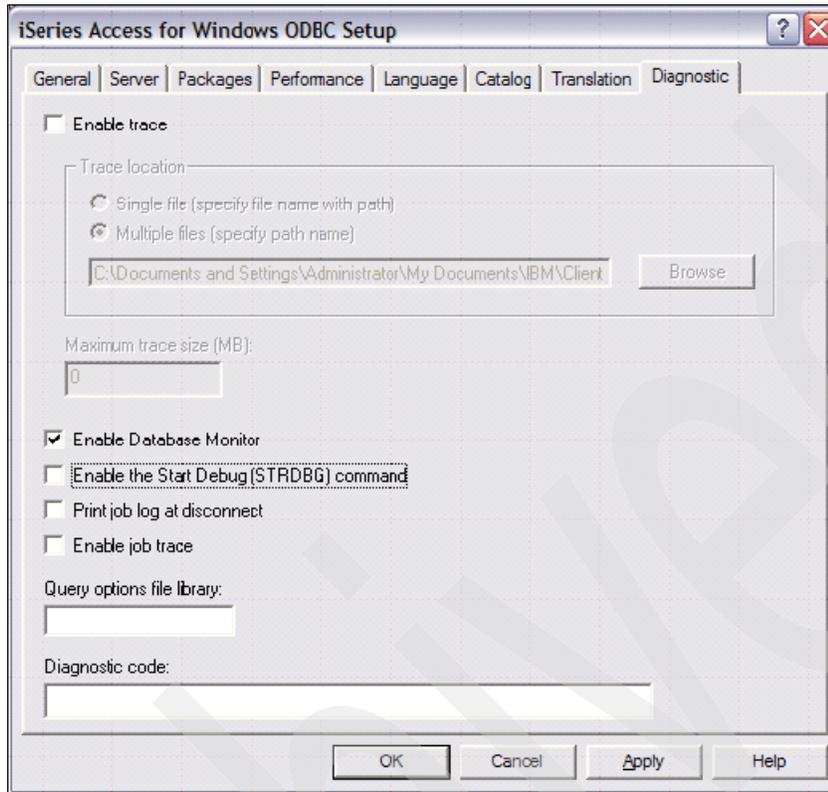


Figure 4-11 iSeries Access for Windows ODBC Setup window

The Enable Database Monitor option causes the ODBC driver to issue a call to STRDBMON for the job connecting to this data source name. The output file is created in the QUSRSYS library starting with the prefix QODB and ending with the job number, for example QUSRSYS/QODB337344.

The ODBC driver attempts to end the monitor when the application disconnects. If the application ends abnormally without issuing a disconnect, the monitor might continue to run. In this case, you must manually end the monitor.

ODBC connection keywords

One potential problem with redistributing an application that uses ODBC is that a data source might need to be created on each user's PC. Data sources are normally created using ODBC C APIs. However, this interface might be difficult to use for some programming languages.

An alternative for this potential problem is for a client to connect to the server without using an ODBC data source and to use connection keywords instead. The iSeries Access ODBC driver has many connection string keywords that can be used to change the behavior of the ODBC connection. These same keywords and their values are also stored when an ODBC data source is setup.

Tip: When an ODBC application makes a connection, any keywords specified in the connection string override the values that are specified in the ODBC data source.

The connection keyword to enable the monitor is the TRACE keyword. The TRACE keyword supports the following options:

- ▶ 0 = No tracing
- ▶ 1 = Enable internal driver tracing
- ▶ 2 = Enable Database Monitor
- ▶ 4 = Enable the Start Debug (STRDBG) command
- ▶ 16 = Enable job trace
- ▶ 32 = Enable database host server trace

To specify multiple trace options, add together the values for the options that you want. For example, if you want to activate the Database Monitor (2) and the STRDBG command (4) on the server, then the value that you specify is 6 (2+ 4=6).

Important: Use these options only when debugging problems because they can adversely affect performance.

The part of a Microsoft Excel® macro in Example 4-1 uses a data source name and the TRACE connection keyword to enable the database monitor and to enable the STRDBG command. Because this example uses both a data source name and a connection keyword, the value in the TRACE keyword overrides the values specified in the data source name.

Example 4-1 Microsoft Excel macro using a data source name and TRACE connection keyword

```
'ODBC connection.  
  'The system is specified in the B1 cell in the Settings worksheet  
  ConnectStr = connectStr & "DSN=QDSN_TPLXE2;System=" &  
Worksheets("Settings").Range("B1").Value & ";TRACE=6"  
  CnMYAS400.Open connectStr
```

The TRACE keyword, including the value to Enable Database Monitor, causes the ODBC driver to issue a call to STRDBMON for the current job. The output file is created in the QUSRSYS library, starting with the prefix QODB, and ending with the job number, for example QUSRSYS/QODB337344.

4.2.5 Enabling Database Monitors in OLE DB clients

There are two ways you can start a Detailed Database Monitor in OLE DB clients:

- ▶ Using an OLE DB connection property
- ▶ Using an OLE DB connection keyword

OLE DB connection properties

A set of custom properties (IBMDA400, IBMDARLA, and IBMDASQL) is available for the OLE DB providers shipped with iSeries Access for Windows. The trace property (available in V5R3) is used to enable diagnostic traces when troubleshooting errors. It is an integer property, and several numeric constants are defined for various trace options.

To determine the value this property should contain, select the desired trace options and add the constant values. The resulting number is the value that should be specified. The constants are:

- ▶ 0 = No trace options (the default value)
- ▶ 1 = Enable Database Monitor
- ▶ 2 = Enable the STRDBG command
- ▶ 4 = Print Job Log at disconnect

- ▶ 8 = Enable Job trace via the Start Trace (STRTRC) command

Example 4-2 shows how to enable the Database Monitor using Visual Basic®.

Example 4-2 Enabling the Database Monitor using Visual Basic

```
Dim cnAS400 as ADODB.Connection
  Dim strJobName as String

  Set cnAS400 = New ADODB.Connection

  'Set the provider to Client Access
  cnAS400.Provider = "IBMDA400"

  'Set custom properties.
  cnAS400.Properties("Block Fetch") = True
  cnAS400.Properties("Catalog Library List") = "LIBRARY1, LIBRARY2"
  cnAS400.Properties("Convert Date Time To Char") = "FALSE"
  cnAS400.Properties("Default Collection") = "MYLIB"
  cnAS400.Properties("Force Translate") = 0
  cnAS400.Properties("Cursor Sensitivity") = 0
  cnAS400.Properties("Data Compression") = True
  cnAS400.Properties("Hex Parser Option") = 0
  cnAS400.Properties("Initial Catalog") = "*SYSBAS"
  cnAS400.Properties("Maximum Decimal Precision") = 31
  cnAS400.Properties("Maximum Decimal Scale") = 31
  cnAS400.Properties("Minimum Divide Scale") = 0
  cnAS400.Properties("Query Options File Library") = "QUSRSYS"
  cnAS400.Properties("SSL") = "FALSE"
  cnAS400.Properties("Trace") = 1 'Enable Database Monitor

  'Open the connection
  cnAS400.Open "Data Source=MySystem;", "USERID", "PWD"
  strJobName = cnAS400.Properties("Job Name")
```

OLE DB connection keywords

In addition to using the trace property to enable database monitor in OLE DB, you can use the Trace connection keyword. In Example 4-3, we illustrate an Excel macro that uses the Trace connection keyword to enable database monitor at connection time.

Example 4-3 Excel macro using the Trace connection keyword

```
'OLE DB Connection
  connectStr = connectStr & "provider=IBMDA400;data source=" &
Worksheets("Settings").Range("B1").Value & ";TRACE=1"
  cnMYAS400.Open connectStr
```

The trace property or the trace connection keyword causes the OLE DB Provider to issue a call to STRDBMON for the current job. As with ODBC, the output file is created in the QUSRSYS library, starting with the prefix QODB and ending with the job number.

4.2.6 Enabling Database Monitors in JDBC clients

The JDBC driver shipped with the Developer Kit for Java (commonly known as the *native JDBC driver*) and the IBM Toolbox for Java JDBC driver support a server trace connection property. Among other options, it includes an option to start a database monitor.

In Example 4-4, the Java code uses a properties object to enable tracing. The example uses native JDBC and the IBM Toolbox for Java JDBC concurrently.

Example 4-4 Java code using JDBC

```
// Register both drivers.
try {
    Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    Class.forName("com.ibm.as400.access.AS400JDBCdriver");
} catch (ClassNotFoundException cnf) {
    System.out.println("ERROR: One of the JDBC drivers did not load.");
    System.exit(0);
}

Connection conn1, conn2;
Properties props = new Properties();
props.setProperty("user", "MYUSER");
props.setProperty("password", "MYPASSWORD");
props.setProperty("server trace", "n");

try {
    // Obtain a connection with each driver.
    conn1 = DriverManager.getConnection("jdbc:db2://localhost", props);
    conn2 = DriverManager.getConnection("jdbc:as400://localhost", props);

    conn1.close();
    conn2.close();
} catch (SQLException e) {
    System.out.println("ERROR: " + e.getMessage());
}
```

In this example, *n* is one of the following options:

- ▶ 0 = No trace
- ▶ 2 = Database monitor
- ▶ 4 = STRDBG
- ▶ 8 = Save the job log
- ▶ 16 = Job trace
- ▶ 32 = Save SQL information
- ▶ 64 = Database host server tracing

As with the ODBC and OLE DB connection properties, you can add these values together to set multiple trace options. The database monitor option causes the drivers to issue a call to STRDBMON for the current job. The output file is created in the QUSRSYS library, starting with the prefix QJT for the toolbox driver and the prefix QSQL for the native driver, and ending with the job number for example, QUSRSYS/QJT042174 or QUSRSYS/QSQL041477.

4.2.7 Enabling Database Monitors in .NET clients

The IBM.Data.DB2.iSeries .NET provider supports a `ConnectionString` attribute called *Trace* that enables you to turn on various server-side traces. When the `Trace` property is specified in the `ConnectionString`, the provider sends commands to the iSeries to enable the specified traces.

The trace options are `DatabaseMonitor`, `StartDebug`, `PrintJoblog`, and `TraceJob`. When the `DatabaseMonitor` option is specified, the output file will be created in `QUSRSYS` starting with the prefix `QNET` and ending with the job number, for example, `QUSRSYS/QNET052800`.

Several server traces can be turned on at the same time by placing them in the connection string. Example 4-5 shows a sample connection string where we are turning on a job trace and database monitor.

Example 4-5 Trace property specified in the connection string

```
iDB2Connection cn = new iDB2Connection();  
cn.ConnectionString = "DataSource=myiSeries; Trace=TraceJob, DatabaseMonitor;";
```

4.2.8 Enabling Database Monitors using an exit program

An exit program provides another way to start a database monitor for clients using the `QZDASOINIT`, `QZDASSINIT`, or `QZDAINIT` prestart jobs, such as `ODBC`, `OLE DB`, or `IBM Toolbox for Java` clients. The database server has five different exit points defined. `QIBM_QZDA_INIT` is one of those exit points and is called when the prestart jobs are started.

By using the `Work with Registration Info (WRKREGINF)` command, you can add or remove the exit program, which has a `STRDBMON` command, to the exit point. You must end and restart the prestart jobs using the `Start Prestart Jobs (STRPJ)` and `End Prestart Jobs (ENDPJ)` commands for the change to take effect. To learn more about exit programs, search on `register exit programs` in the `V5R4 iSeries Information Center` at:

<http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp>

4.3 Collecting Summary SQL Performance Monitor data

Although the Summary Monitor itself has not changed between `V5R3` and `V5R4`, the interface to start it from within `iSeries Navigator` has been moved to the wizard. To start the Summary SQL Performance Monitor using the wizard, simply select **Summary** rather than **Detail** in the `Type` drop-down box.

This monitor resides in memory and only retains a summary of the data collected. When the monitor is paused or ended, the data is written to disk and can then be analyzed. Because the

monitor stores its information in memory, the performance impact to your system is minimal. Now perform the following steps:

1. From the next window (shown in Figure 4-12), specify the types of information you want collected into the Summary Monitor.

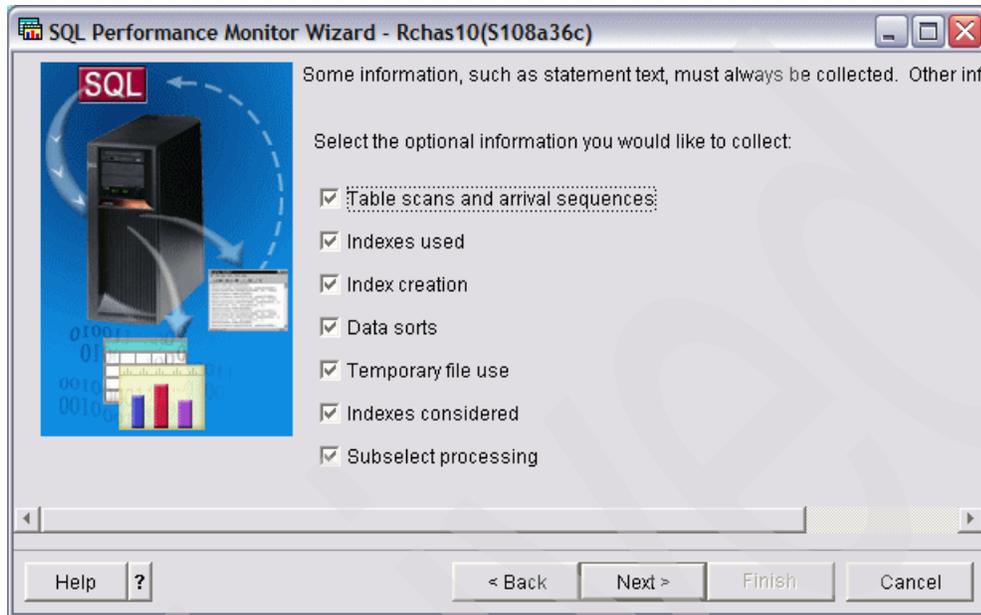


Figure 4-12 Specify summary information

The options you can check to include are:

- **Table scans and arrival sequences:** Contains the table scan data for the monitored jobs.
- **Indexes used:** Contains details of how indexes are used by monitored jobs.
- **Index creation:** Contains details of the creation of indexes by monitored jobs.
- **Data sorts:** Contains details of data sorts that monitored jobs perform.
- **Temporary file use:** Contains details about temporary files that monitored jobs created.
- **Indexes Considered:** Contains information about the index considered for each SQL statement.
- **Subselect processing:** Contains information about each subselect in an SQL statement.

- After you have made your selections, the next window, shown in Figure 4-13, allows you to specify which jobs to monitor. You can select to monitor all jobs, or, you can choose which specific jobs you want to monitor by clicking the **Add** button.

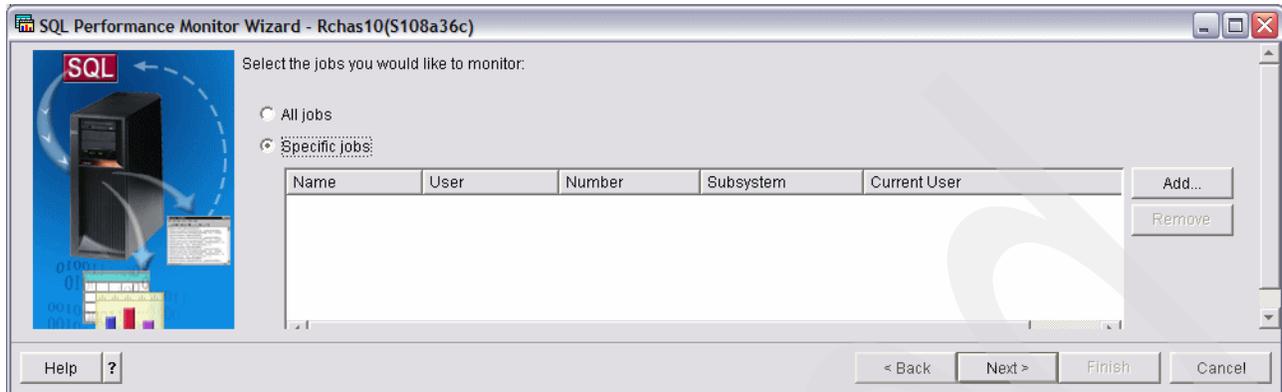


Figure 4-13 Select to monitor all jobs, or specific jobs

- When you click **Add**, this will open a new window listing all jobs currently in the system (Figure 4-14) and from here you can select individual jobs from the list.

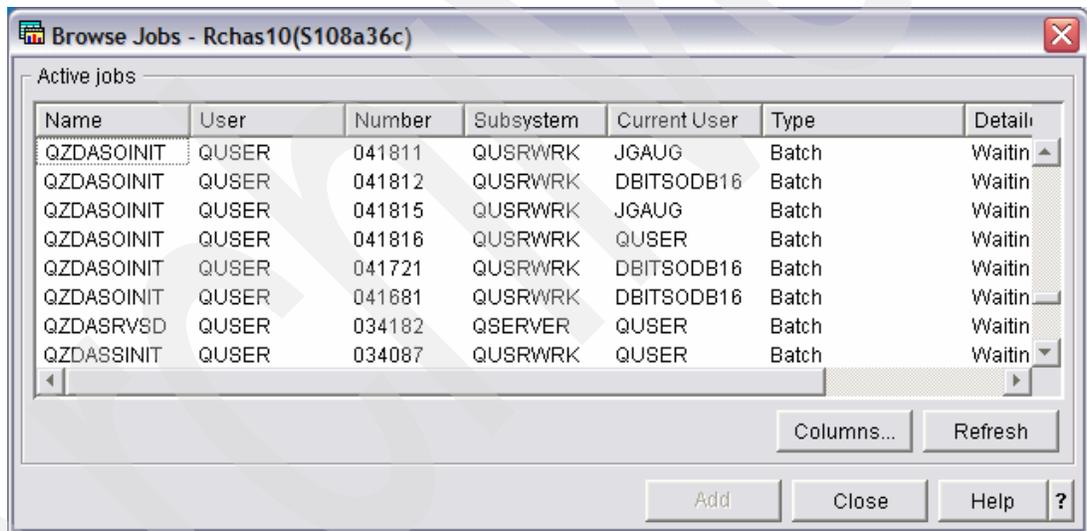


Figure 4-14 Choose individual jobs from the list

You can have multiple instances of summary monitors running on your system at one time. However, only one monitor instance can monitor all jobs. Additionally, you cannot have two monitors monitoring the same job. When collecting information for all jobs, the monitor collects information about previously started jobs or new jobs started after the monitor is created.

4. After making your job selections, click the **Next** button and the wizard will present you with a final summary of the selections you have made (Figure 4-15). If you are satisfied with everything, select **Finish** to start the Summary SQL Performance Monitor.

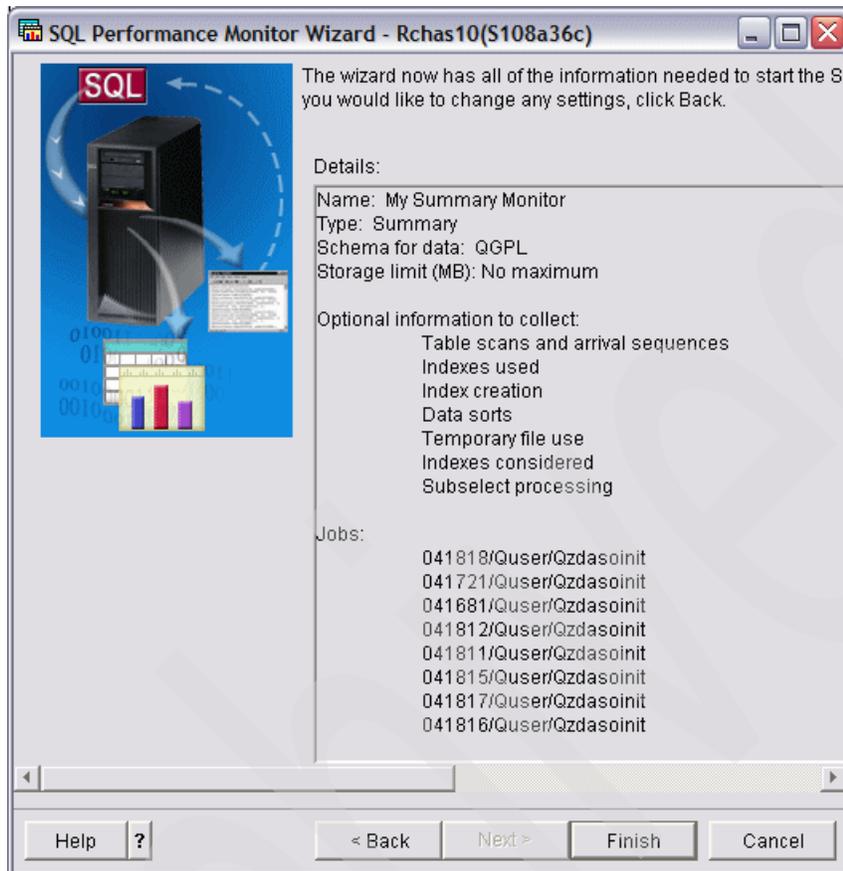


Figure 4-15 Final summary window for the Summary monitor

The new monitor will be listed in the right pane of the iSeries Navigator window with the type *Summary* and a status of *Started*.

You can pause or end a summary monitor, or you can continue a monitor that was previously paused. However, you cannot continue a monitor that was previously ended. When the monitor is paused or ended, the monitored data that resides in memory is written into several database tables that can be queried and analyzed using the predefined reports that come with the summary monitor analysis tool, which are briefly discussed in 5.3, “Summary SQL Performance Monitor analysis overview” on page 143.

4.4 Importing Database Monitors into iSeries Navigator

In the Import SQL Performance Data window, you can incorporate data from monitors that were collected on a system other than the one you are performing analysis on.

To access the import window, from the main iSeries Navigator window, right-click **SQL Performance Monitors** and select **Import**. You are presented with the Import SQL Performance Data window (shown in Figure 4-16).

4.4.1 Importing detailed monitor data

In most import situations, you will likely be importing detailed monitor data that was collected from another system. This is shown in Figure 4-16.

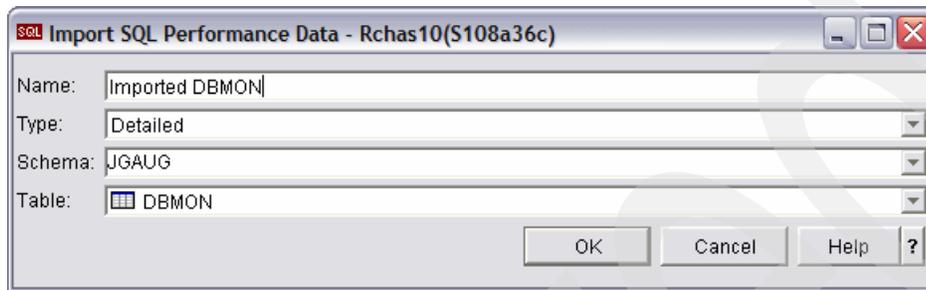


Figure 4-16 Import SQL Performance Monitor window

You can give your imported monitor any descriptive name you like. This will then show up on the main iSeries Navigator window with a status of *Imported*, as shown in Figure 4-17.

Name	Type	Status	Schema	Created By	Date Created	Jobs
DBL09-11-DBMonitor2	Detailed	Ended	DBITSODB11	DBITSODB11	8/15/06 3:2...	040541/QUSER/Q...
DBL09-11-DBMonitor3	Detailed	Ended	DBITSODB11	DBITSODB11	8/15/06 3:5...	040532/QUSER/Q...
DBL09-16-DBMonitor1	Detailed	Ended	DBITSODB16	DBITSODB16	8/16/06 9:0...	All jobs - filtered
dbl09-16-dbmONITOR.2	Detailed	Ended	DBITSODB16	DBITSODB16	8/16/06 8:4...	041137/QUSER/Q...
DBL09-16-DBMonitor3	Detailed	Ended	DBITSODB16	DBITSODB16	8/16/06 9:2...	041137/QUSER/Q...
dbmonitorbrasil	Detailed	Ended	DBITSODATA	QSECOFR	4/25/06 9:4...	027284/QUSER/Q...
dbmonitorbrasil2	Detailed	Ended	DBITSODATA	QSECOFR	4/25/06 9:4...	027284/QUSER/Q...
DBMONITORX16	Detailed	Imported	DBITSODATA	JKERL	1/30/06 10:...	855514/MCAIN/Q...
Imported DBMON	Detailed	Imported	JGAUG	JGAUG		
My Detailed Monitor	Detailed	Started	QGPL	JGAUG	8/18/06 11:...	All jobs
My Summary Monitor	Summary	Ended	QGPL	JGAUG	8/18/06 8:5...	041818/QUSER/Q...
SAMPLEDB06 db MON...	Detailed	Ended	SAMPLEDB06	DBNAV06	2/28/06 4:2...	017129/QUSER/Q...
SAMPLEDB06 MONIT...	Summary	Ended	SAMPLEDB06	DBNAV06	2/28/06 4:3...	016854/QUSER/Q...
SAMPLEDB06 MONIT...	Detailed	Ended	SAMPLEDB06	DBNAV06	2/28/06 4:3...	017134/QUSER/Q...
SAMPI FDR07	Detailed	Ended	SAMPI FDR07	DBNAV07	2/28/06 4:5...	017154/QUSER/Q...

Figure 4-17 Imported monitor

4.4.2 Importing detailed monitor data from a previous release

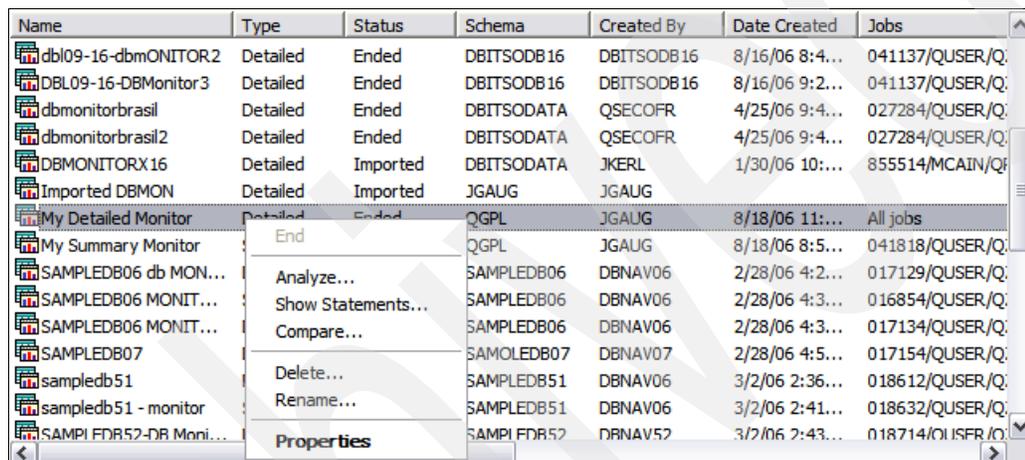
A very useful feature is that you can import Detailed SQL Performance Monitor data that was collected on a system that was at a previous release of the operating system. This allows you to use the new analysis tools to analyze data that was collected on a system at an earlier release. At analysis time, the system automatically detects that the file is from a previous release and converts the imported database monitor file from a pre-V5R4 format to the V5R4 format that is required for the analysis tools. For example, any "More Text" (MT) type records are removed from the file since the V5R4 format places all statement text in one column (QQ1000L). This conversion is transparent to the user and only occurs the first time analysis is performed. The new, converted file replaces the file that was originally imported.

4.4.3 Importing summary monitor data

If you are importing data collected by a Summary monitor on another system, select **Summary** for type of monitor. You then can specify any one of the several output files from the Summary monitor. The system will automatically recognize the other output files from the same summary collection, and ask if you would like to import them as well.

4.5 SQL Performance Monitors properties

At any time, you can display the properties of a monitor in iSeries Navigator. Right-click the monitor for which you want to display the properties and select **Properties** as shown in Figure 4-18.



Name	Type	Status	Schema	Created By	Date Created	Jobs
dbi09-16-dbmONITOR2	Detailed	Ended	DBITSODB16	DBITSODB16	8/16/06 8:4...	041137/QUSER/Q...
DBL09-16-DBMonitor3	Detailed	Ended	DBITSODB16	DBITSODB16	8/16/06 9:2...	041137/QUSER/Q...
dbmonitorbrasil	Detailed	Ended	DBITSODATA	QSECOFR	4/25/06 9:4...	027284/QUSER/Q...
dbmonitorbrasil2	Detailed	Ended	DBITSODATA	QSECOFR	4/25/06 9:4...	027284/QUSER/Q...
DBMONITORX16	Detailed	Imported	DBITSODATA	JKERL	1/30/06 10:...	855514/MCAIN/Q...
Imported DBMON	Detailed	Imported	JGAUG	JGAUG		
My Detailed Monitor	Detailed	Ended	QGPL	JGAUG	8/18/06 11:...	All jobs
My Summary Monitor			QGPL	JGAUG	8/18/06 8:5...	041818/QUSER/Q...
SAMPLEDB06 db MON...			SAMPLEDB06	DBNAV06	2/28/06 4:2...	017129/QUSER/Q...
SAMPLEDB06 MONIT...			SAMPLEDB06	DBNAV06	2/28/06 4:3...	016854/QUSER/Q...
SAMPLEDB06 MONIT...			SAMPLEDB06	DBNAV06	2/28/06 4:3...	017134/QUSER/Q...
SAMPLEDB07			SAMPLEDB07	DBNAV07	2/28/06 4:5...	017154/QUSER/Q...
sampledb51			SAMPLEDB51	DBNAV06	3/2/06 2:36...	018612/QUSER/Q...
sampledb51 - monitor			SAMPLEDB51	DBNAV06	3/2/06 2:41...	018632/QUSER/Q...
SAMPI FDR52-DR Moni...			SAMPI FDR52	DRNAV52	3/2/06 2:43...	018714/QUSER/Q...

Figure 4-18 Select to view properties

4.5.1 Detailed monitor properties

Figure 4-19 shows the detailed monitor properties window. From this window, you are able to see the time period that monitor data was collected for, the table name of the underlying monitor output file, as well as the various filters, if any, that were applied to the collection

when the monitor was started. As with the summary monitor, you are given the list of jobs that were monitored - in this example, all jobs.

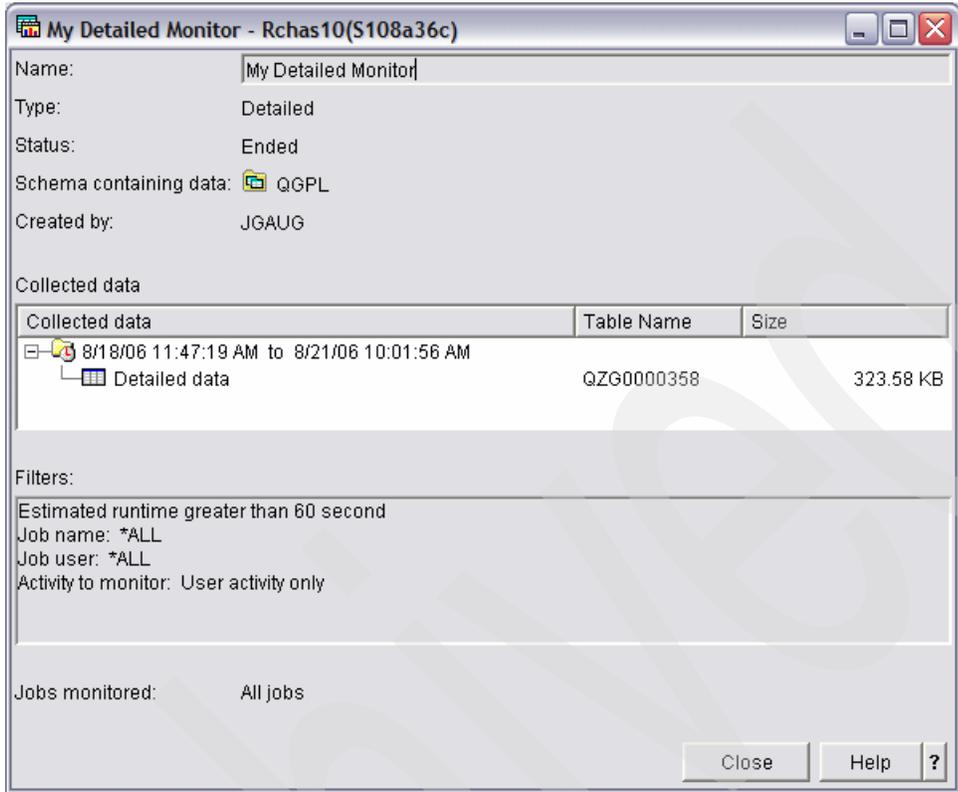


Figure 4-19 Detailed Monitor properties

4.5.2 Summary monitor properties

When you select to view the Summary Monitor properties, you are presented with the window shown in Figure 4-20. From the Summary Monitor properties window, you can see the names

of the various output files which constitute the Summary monitor, as well as the list of jobs that the monitor was run against.

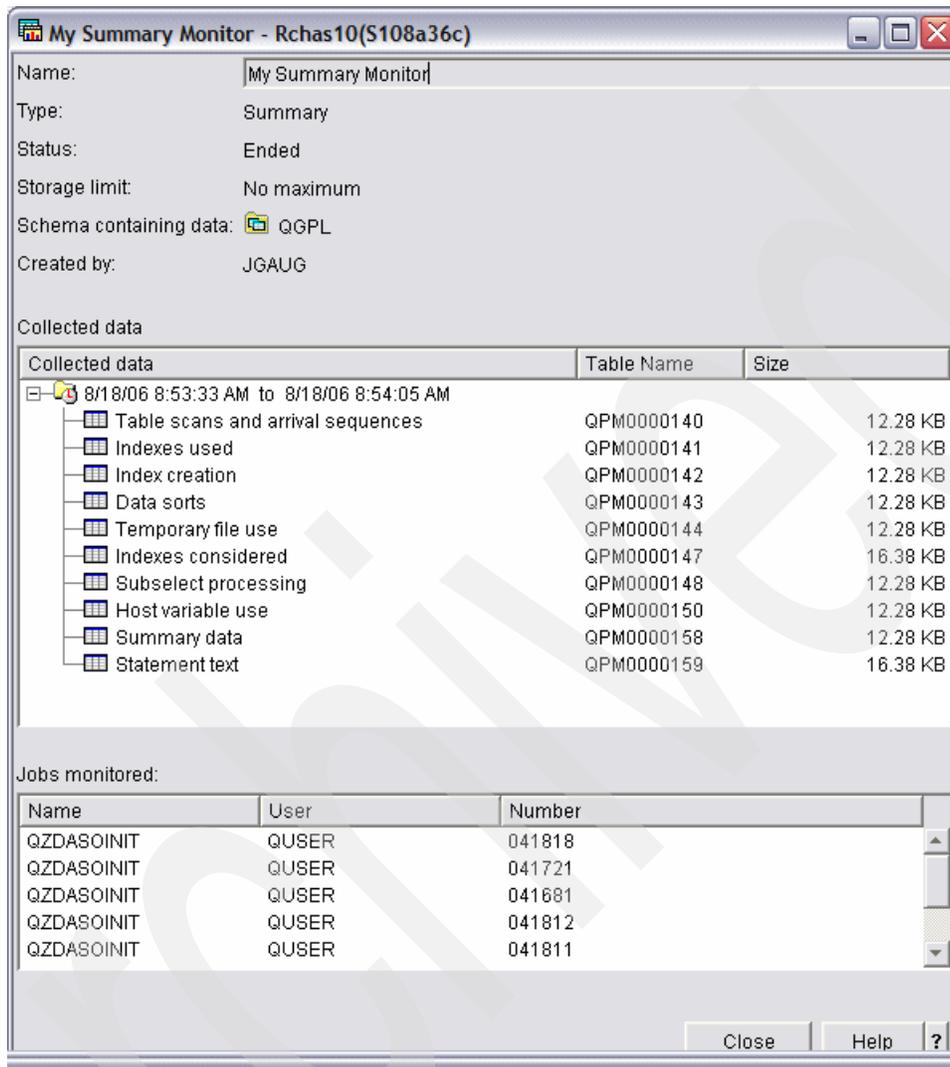


Figure 4-20 Summary properties

Note: To see the data description specification (DDS) for the output files, see the *DB2 Universal Database for iSeries Database Performance and Query Optimization* book under the section entitled Memory Resident Database Monitor: DDS, which is available on the Web at:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/rzajq/rzajq.pdf>

4.5.3 Imported monitor properties

The properties window for an imported detailed monitor is shown in Figure 4-21. One thing to note is that for an imported monitor, the list of jobs is not automatically filled in. Because this monitor was imported, the system does not know what jobs were monitored without querying the data. To have the system query the data to return the list of monitored jobs, click the **Retrieve** button.

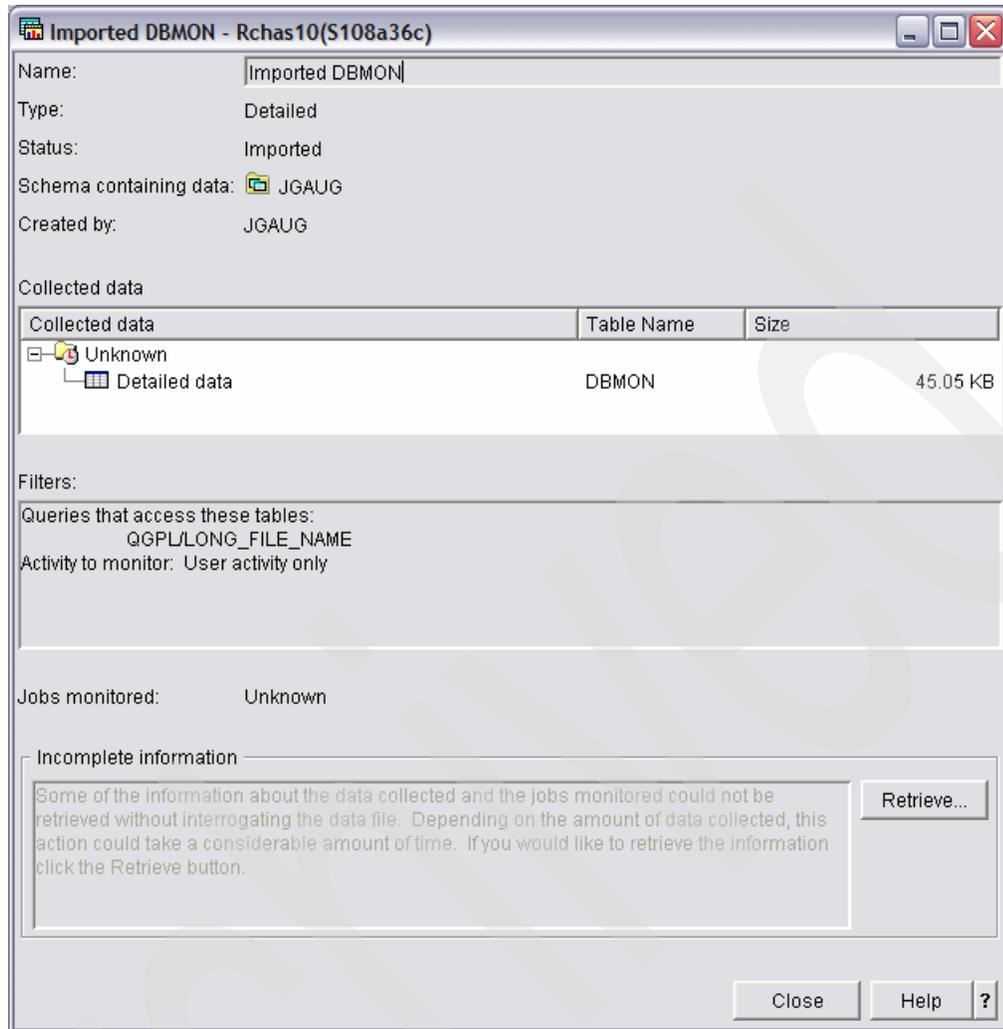


Figure 4-21 Imported monitor properties

4.6 Summary or Detailed SQL Performance Monitor

A question you may ask yourself when approaching an SQL performance problem is whether you should collect a Summary or Detailed SQL Performance Monitor. In this section, we'll explain when one type of monitor may be more useful than the other.

For some investigations, the Summary SQL Performance Monitor may have certain advantages over the Detailed SQL Performance Monitor. It may make more sense to use the Summary Monitor in situations:

- ▶ When you are not sure what is causing the performance issue that you might be seeing
You can collect a lot of data over the entire system, in a short amount of time, using the detailed monitor. If disk usage is a concern, it may be better to collect a summary monitor over the entire system and analyze that. However, with the new filters introduced in V5R4, disk usage for a detailed database monitor may not be the concern that it was at previous releases.

- ▶ When you want to monitor the system over a period of time to compare, for example, the results of one week against another.

This helps with a proactive approach to SQL performance and with observing trends in the way the database is performing.

In most circumstances, however, a Detailed SQL Performance Monitor will be more useful than a summary monitor. Specifically, when:

- ▶ You need more detailed information about your queries and not just summary data.
- ▶ You want to use the robust new set of query analysis tools to investigate your queries.
- ▶ You want to investigate any queries that are returning errors.
- ▶ You need to collect a monitor to send to IBM Support.

Remember, the Summary SQL Performance Monitor can be very useful in helping to narrow down a SQL Performance problem. However, a Detailed SQL Performance Monitor will likely be required to provide the level of detail needed to resolve most SQL Performance problems. Furthermore, the new analysis tools introduced in V5R4 can only be used against a Detailed SQL Performance Monitor. These new analysis tools will be introduced in Chapter 5, “Analyzing SQL performance data using iSeries Navigator” on page 117. The tools for analyzing Summary SQL Performance Monitor data are unchanged between V5R3 and V5R4.

Tip: When choosing between the two types of monitors, our recommendation is to make the Detailed SQL Performance Monitor your standard for SQL Performance analysis, and use the Summary monitor only if you have a specific reason *not* to use the Detailed monitor.

Archived



Analyzing SQL performance data using iSeries Navigator

In Chapter 4, “Gathering SQL performance data” on page 89, you learned about the different Database and SQL Performance Monitors and how to collect SQL performance data. After you collect the SQL performance data, you are ready to analyze it.

V5R4 introduces a new, robust set of feedback tools that can be used to analyze Detailed SQL Performance Monitor data. These new tools provide a powerful way to quickly make sense out of your collected SQL Performance Monitor data.

5.1 Detailed SQL Performance Monitor Analysis overview

Version 5 Release 4 of i5/OS provides a brand new GUI interface for analysis of Detailed SQL Performance monitor data collections. The hallmark of the new interface is its drill-down capability, in which each report presents you with additional actions to take or links you can take to another related report. This same new interface is used for analysis of the SQE Plan Cache as well as the SQE Plan Cache snapshots. (Plan Cache topics are covered in Chapter 7, “SQE Plan Cache and SQE Plan Cache Snapshots” on page 237.)

To begin your analysis, right-click the Detailed SQL Performance monitor you wish to analyze. From the options that appear, you can choose from the following functions for a Detail monitor:

- ▶ **End:** Ends the current collection, if active.
- ▶ **Analyze:** The new SQL Performance Monitor GUI, known as the Analysis Overview *dashboard*, is opened.
- ▶ **Show Statements:** This function opens a window that lists the SQL statements for which the Detailed SQL Performance Monitor has collected data and for which a Visual Explain diagram can be produced. This interface is also new at V5R4. We will take a look at it in 5.4, “Show Statements” on page 149.
- ▶ **Properties:** This function opens a window which provides the properties of the database monitor collection. The window will differ depending on if the monitor is for summary or detailed data. For examples, see 4.5, “SQL Performance Monitors properties” on page 111.
- ▶ **Delete:** Select this to remove the SQL Performance Monitor. You will be given the option to remove the underlying data file as well.
- ▶ **Rename:** Use this option to rename the SQL Performance Monitor. This will not rename the underlying data file.

Let’s start by taking the **Analyze** option. (Figure 5-1)

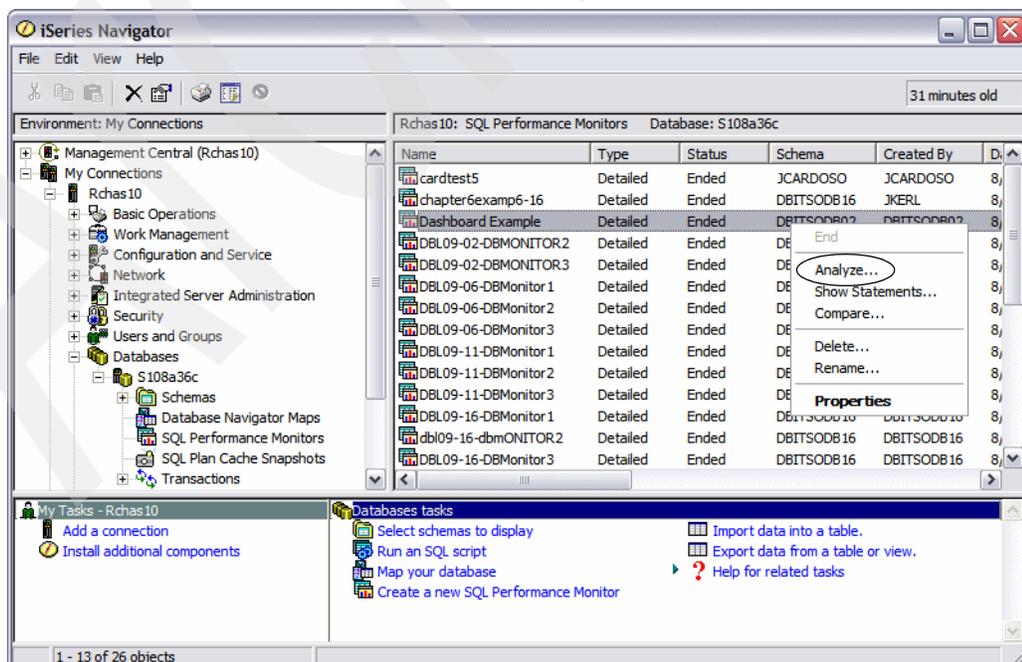


Figure 5-1 Begin analysis of a Detailed SQL Performance monitor

V5R4 provides an Analysis Overview “*dashboard*” which will give you a quick overview of the information captured in the monitor collection. This analysis overview window breaks items of interest into several categories and serves as a starting point for you to begin in-depth analysis of those areas. In this section, we will take a look at the information that is presented in the Analysis Overview dashboard (Figure 5-2), as well as explore several of the detailed reports that you can launch from the overview information.



Figure 5-2 Initial Analysis Overview

The initial window shows several folders, or categories, of analysis overview items. When you click to expand one of the folders, you will be presented with several overview analysis items from which you can begin analysis. There are two columns after each analysis report to indicate *Summary Available* and *Statement Available*. A checkmark ✓ is placed in each column to indicate if the summary or statement information is available for that particular analysis item. On the right side of the window are the **Summary** and **Statements** buttons which can be used to retrieve the information. Clicking on those buttons will give you a lot of information regarding the queries that make up the particular report. We’ll take a look at the detailed information retrieved when you click these buttons in 5.2, “In-depth analysis reports” on page 128.

Notes on terminology: From this point on in this chapter, when we refer to the SQL Performance Monitor, we are talking about the *Detailed SQL Performance Monitor*.

The Analysis Overview *dashboard* refers to the entire graphical user interface (GUI) window that is presented when selecting to analyze a Detailed SQL Performance Monitor.

An Analysis Overview *item* refers to a single line in the dashboard which may have zero, one or two checkmarks after it.

An analysis *report* refers to the report generated when you select either summary or statements for an overview item.

5.1.1 Analysis overview

Let us start by taking a look at all the useful information that can be investigated from the initial analysis overview window. Figure 5-3 shows the first folder, Overview, expanded. This folder may already be expanded for you when you first enter the dashboard. As you can see, there is a wealth of data contained therein. This section provides a information about the overview analysis items available from the Overview folder.

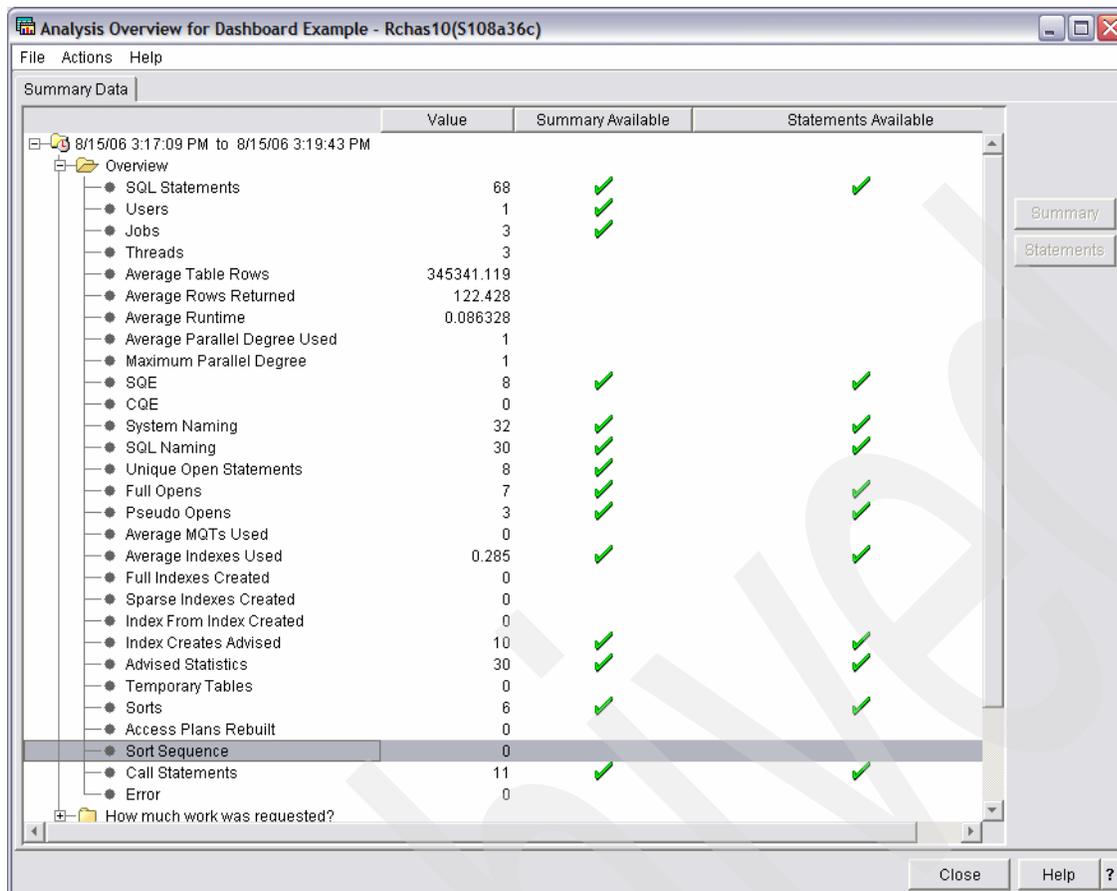


Figure 5-3 Overview Analysis items in the dashboard

► SQL Statements

This item will provide you with information about all the SQL statements whose information has been collected in the SQL Performance Monitor. Information returned includes the statement text, the run time of the statement, the statement's unique count, the user name, and number of the job running the statement. By sorting on the run time you can very quickly find the longest running query and operation captured by the SQL Performance Monitor.

Note: The count returned here is equivalent to the number of records in the underlying database monitor table where the QQRID field is equal to 1000.

► Users

This analysis overview item can be used to get a list of users who were running SQL during the collection period, and how much runtime can be attributed to each. This is especially useful in a client/server environment where you may have server jobs being serviced by the same user profile, such as QUSER. This report returns information for the actual user profile that was connecting to the server jobs.

► Jobs and Threads reports

Use these analysis overview items to quickly identify the number of jobs, and if applicable, threads, that were monitored during the collection period. From this item, you can quickly determine if a particular job or jobs is responsible for most of the run time.

- ▶ Averages reports

The next several items give you averages for the number of rows returned, the size of the queried tables, and the average runtime for all the queries in the database monitor considered as a whole.

- ▶ Parallel degree information

This gives you a quick place to check to what degree Symmetric Multiprocessing (SMP) may have been involved with the queries monitored.

- ▶ SQE/CQE

Here you can find the count of statements that were processed by the SQE optimizer and by the CQE optimizer. Note that only those statements whose implementation info was captured in the database monitor file are included in this count. For queries whose implementation info was captured, this count includes both full opens and pseudo opens.

- ▶ Naming convention information

These reports give you a count of queries that use SQL naming convention and system naming convention. Note that the sum of the values for SQL and system naming convention may not necessarily add up to the total number of SQL statements seen in the first report in this category, since hard close operations do not have an associated naming convention.

- ▶ Open information

The next three items give you information about the number of opens captured in the database monitor and if those opens were full opens or pseudo opens:

- Unique open statements gives you a count of all unique open statements captured in the monitor, where they are full opens or pseudo-opens.
- Full opens returns the number of opens logged for which implementation info has been captured, thus indicating a full open.
- Pseudo-opens give the number of pseudo-opens, regardless of whether the full open for the statement was captured in the monitor.

A quick check of these values can give you a good idea of whether or not you may have a full open problem. For example, if the number of unique and pseudo-open statements is relatively low but the number of full opens is high, then this could be an area that warrants further investigation.

Note: It is important to note that ODPs can only be reused within a job or connection and this overview is aggregating that information across the jobs.

- ▶ Average MQTs used

You can check this item to quickly discover if any Materialized Query Tables were used for the SQL statements captured in the monitor.

- ▶ Index information

The next several overview items provide information about index usage for the queries captured in the monitor. If indexes are created, then the counts are logged here. Separate counts are returned for full indexes, sparse indexes, and index-from-index creations. Also returned here is a count of indexes advised by the index advisor. If the same index is advised multiple times, it is included in this count each time it is advised. A quick look at these numbers may inform you if an improvement to your indexing strategy is needed. For example, if the number of average indexes used is low, while the number of indexes created or advised is high, then some investigation of your indexing strategy may be in order.

► **Advised Statistics**

Information about queries that have had statistics advised for them is found in this item. This count is incremented for both full opens and pseudo-opens of queries for which a statistic has been advised. A high count here could warrant a visit to the statistics advisor.

Note: If the system is running with the default system value for statistics collection the statistics would probably have been already collected and this value is of very little interest.

► **Additional analysis reports**

The remaining reports in the overview category return additional information about the queries logged in the database monitor:

– **Temporary tables and sorts.**

For queries which had implementation info captured, the respective count is incremented if a temporary table or a sort table is used for both full opens and pseudo-opens.

– **Sort sequence**

If an NLSS sort sequence is used for any query, then this count will be incremented.

– **Call statements**

The number of stored procedure calls and further information about the procedures can be found here.

– **Errors**

If any statements captured in the database monitor were not successful, that is, had a negative SQL return code, they will be logged here. This is an easy place to check for any functional problems with any of the statements in your database monitor collection.

5.1.2 Amount of work requested

The next category of reports, Figure 5-4 answers the question “How much work was requested?”.

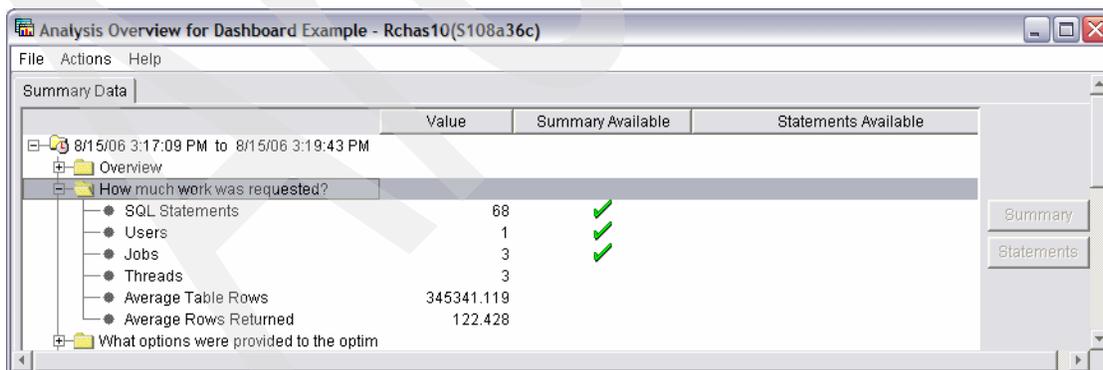
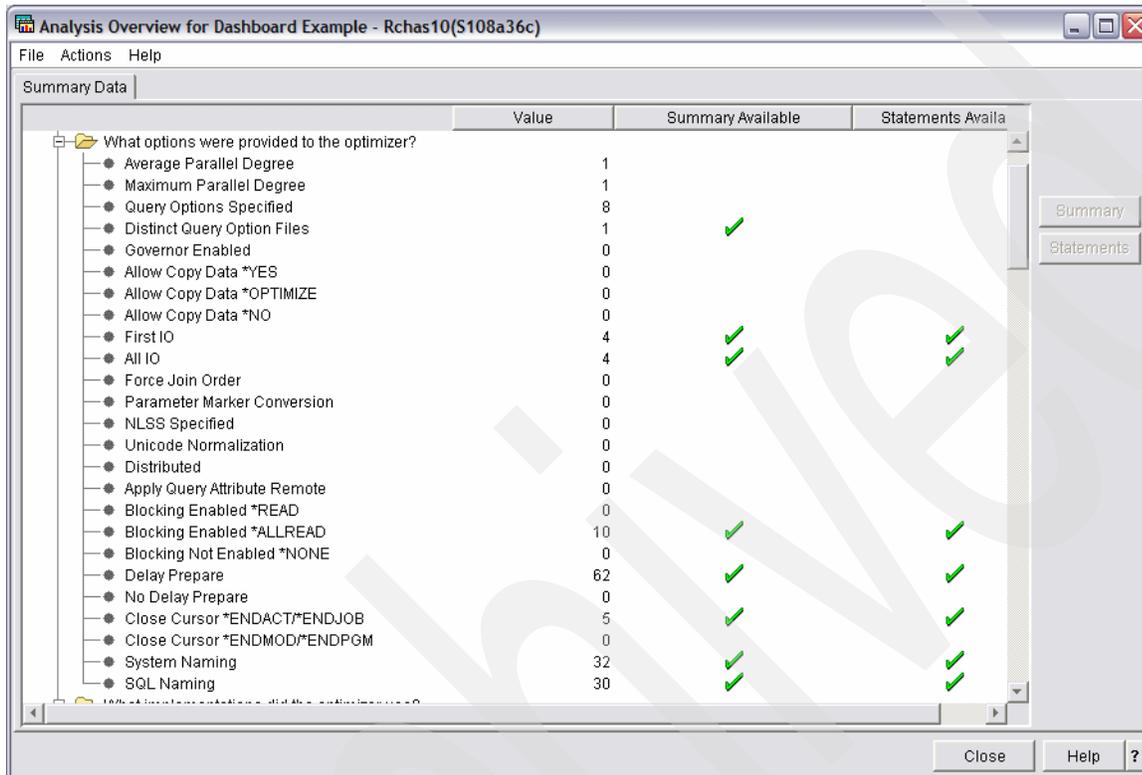


Figure 5-4 Amount of work requested

You may notice that the overview items in this folder look familiar. They were also included in the Overview folder and are placed here to give you a quick place to check to get an idea of the amount of work that was collected in the SQL Performance Monitor collection.

5.1.3 Environmental factors

There are many environmental factors that go into the optimization of a query. Because changes in these factors can lead to different query implementations, it is important to understand the environmental settings that were in place at the time the query was run. The next folder shows what options were provided to the optimizer (Figure 5-5).



	Value	Summary Available	Statements Available
What options were provided to the optimizer?			
● Average Parallel Degree	1		
● Maximum Parallel Degree	1		
● Query Options Specified	8		
● Distinct Query Option Files	1	✓	
● Governor Enabled	0		
● Allow Copy Data *YES	0		
● Allow Copy Data *OPTIMIZE	0		
● Allow Copy Data *NO	0		
● First IO	4	✓✓	✓✓
● All IO	4	✓✓	✓✓
● Force Join Order	0		
● Parameter Marker Conversion	0		
● NLSS Specified	0		
● Unicode Normalization	0		
● Distributed	0		
● Apply Query Attribute Remote	0		
● Blocking Enabled *READ	0		
● Blocking Enabled *ALLREAD	10	✓	✓
● Blocking Not Enabled *NONE	0		
● Delay Prepare	62	✓	✓
● No Delay Prepare	0		
● Close Cursor *ENDACT*ENDJOB	5	✓	✓
● Close Cursor *ENDMOD*ENDPGM	0		
● System Naming	32	✓✓	✓✓
● SQL Naming	30	✓✓	✓✓

Figure 5-5 Options provided to the optimizer

Several of these environmental factors are explained in detail in Chapter 11, “Environmental settings that affect SQL performance” on page 379. We will briefly go over a few of the most useful ones here.

- ▶ Parallel degree info

On a multiprocessor system with SMP enabled, the database engine can use multiple processors for certain aspects of query processing. If multiple processing is used, we will indicate here the average and maximum degree of parallelism. Otherwise, if multiple processing is not used, these values will be set to one.

- ▶ Query options file info

If a QAQQINI query options file was used for the optimization of any of the queries collected in the database monitor, that will be indicated here. We give a count of queries that had query options specified for them, as well as the number of distinct query options files used.

- ▶ Governor enabled

The query governor can be used to stop a query from running if its estimated runtime after optimization exceeds a threshold set by the user. If any queries were run with the governor enabled, that will be indicated here.

- ▶ Allow Copy Data information

This item gives you the count of the number of statements that went each of the three ALWCPYDTA options, YES, NO, or OPTIMIZE.

- ▶ Optimization goal information

These two items give a count of the optimization goal for each of the queries in the database monitor collection. The two choices for optimization goal are First IO, which means that the optimizer should optimize the query with the goal of bringing back the first page of records as quickly as possible, or All IO, which means the query should be optimized to bring back the entire result set as quickly as possible.

- ▶ Additional options provided to the optimizer

There are several additional options that can be provided to the optimizer. We list them here but will not go into additional details:

- Force join order specified
- Parameter marker conversion
- Sort sequence table specified
- Unicode normalization
- Distributed query info
- Blocking info
- Delay Prepare info
- Close cursor info
- System or SQL naming used (also available in the overview folder)

5.1.4 Implementation information

The next category of analysis overview items in the dashboard, shown in Figure 5-6, are the implementation items. These items will give you information about the specific

implementation methods that were used by the queries represented in the SQL Performance Monitor collection.

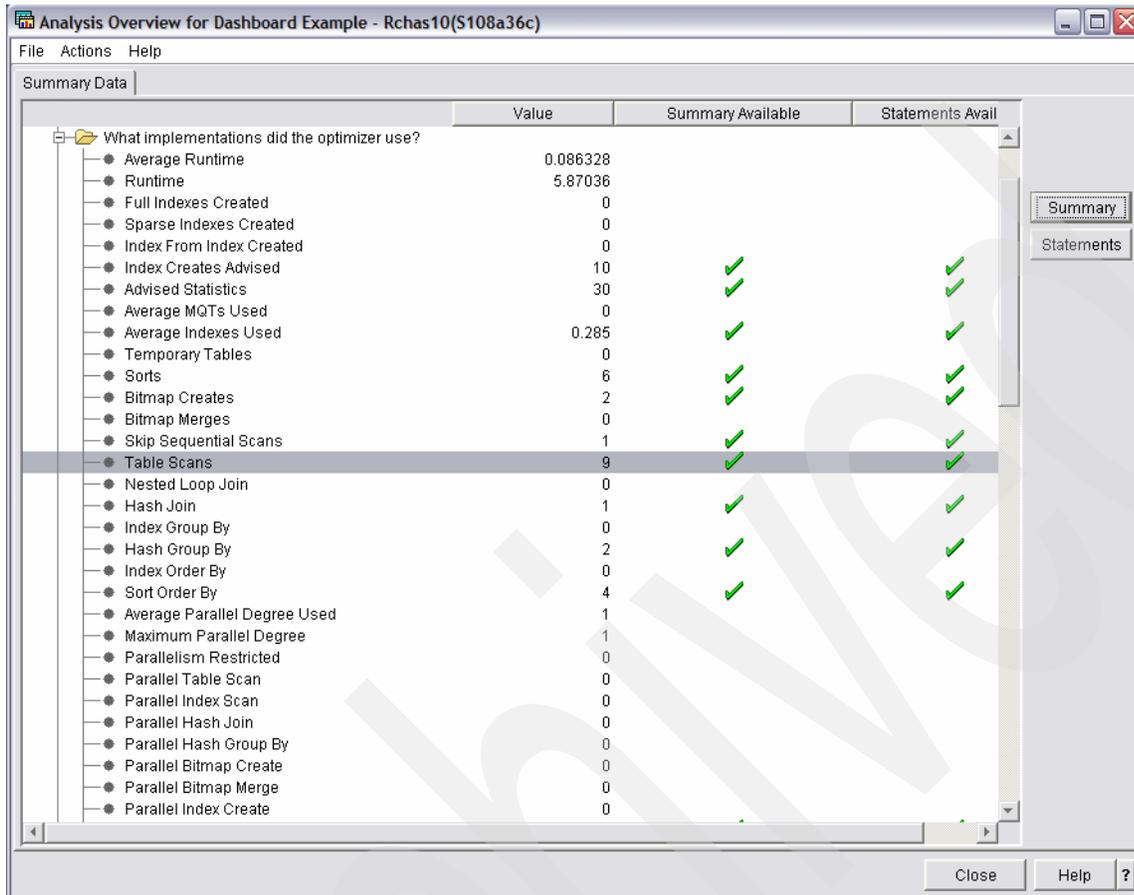


Figure 5-6 Implementation information

You will notice that several of the items in this section have appeared elsewhere in the dashboard, such as runtime, index created, and statistic information. In addition to those overview items, there are items for specific implementation info.

► **Bitmap information**

If the optimizer creates a dynamic bitmap to implement the query, these reports will contain a count of bitmaps created and bitmaps merged and give you details of the queries that used bitmaps.

► **Table scans**

Here is where we can find information about table scans performed by the queries collected in the database monitor data. A high number here in relation to the total number of queries may indicate a problem.

► **Join implementation information**

If any of the queries in the collected monitor data are join queries, we provide information about which join algorithm was used: nested loop join or hash join.

► **Grouping and ordering implementation information**

The types of grouping and ordering implementations are indicated here. For grouping, the two possibilities are index and hash table. For ordering, the optimizer can choose to use an index or create a temporary sort table. The count of each method used is listed here.

- ▶ Parallel degree info

Several overview items are provided here to indicate if certain optimizer operations were performed using parallelism.

- ▶ Estimated row information

The average and maximum estimated row counts for the database monitor collection are found here.

5.1.5 Types of statements

The next grouping of overview items in the report gives information and counts on the different types of statements that were run in the SQL Performance Monitor collection. This is shown in Figure 5-7.

What types of SQL statements were requested?	Value	Summary Available	Statements Avail
Row Selection	0		
Subquery	0		
Group By	2	✓	✓
Order By	4	✓	✓
Inner Join	2	✓	✓
Outer Join	0		
Exception Join	0		
Union	3	✓	✓
Except	0		
Intersect	0		
Distinct	3	✓	✓
Variables Specified	8	✓	✓
Call Statements	11	✓	✓
Select Statements	38	✓	✓
Update Statements	0		
Insert Statements	0		
Delete Statements	0		
Data Definition Statements	0		
Other Statements	19	✓	✓
Static SQL	15	✓	✓
Extended Dynamic	0		
System-wide Cache Dynamic	5	✓	✓
Dynamic	22	✓	✓
No Commit	62	✓	✓
Uncommitted Read	0		
Cursor Stability	0		
Cursor Stability KEEP LOCKS	0		
Read Stability	0		
Repeatable Read	0		
Not Applicable	6	✓	✓
Data Conversions	3	✓	✓
Success	67	✓	✓

Figure 5-7 Types of statements run

- ▶ SQL statement elements

From these items, you can very quickly get an overview of the number of queries that contained subqueries, had grouping or ordering, had unions, used the distinct clause, and so on.

- ▶ Statement functions

In these items we report the total number of select, delete, insert, update, data definition language, and other (that is, prepare, describe, set, and so on) statements in the database monitor collections.

Note: For these reports, the tool groups all statements contained in the monitor into one of the six statement function categories.

► **Static / dynamic processing information**

The extent to which static and dynamic processing occurred can be determined by reviewing these overview items. For dynamic processing, there are two additional items which further indicate if the statement was extended dynamic or system wide cache dynamic. Extended dynamic statements are stored in SQL packages, and system-wide cache dynamic statements are stored in the SQL system-wide statement cache.

► **Commitment control information**

The queries in the SQL Performance Monitor are put into different overview items based on the commitment control level used by the statement.

► **Data conversion information**

If data conversion was required for the query, it is indicated by this item. Data conversion may occur, for example, when fetching column into a host variable and the host variable and column have differing data types. A small amount of overhead is associated with data conversion, so if this number is large, then you may want to take steps to eliminate data conversions where possible.

► **Success, warning, or error information**

Separate counts are maintained for the number of SQL statements that return a successful return code of 0, a positive return code which represents a warning, or a negative return code to indicate an error.

5.1.6 Miscellaneous information

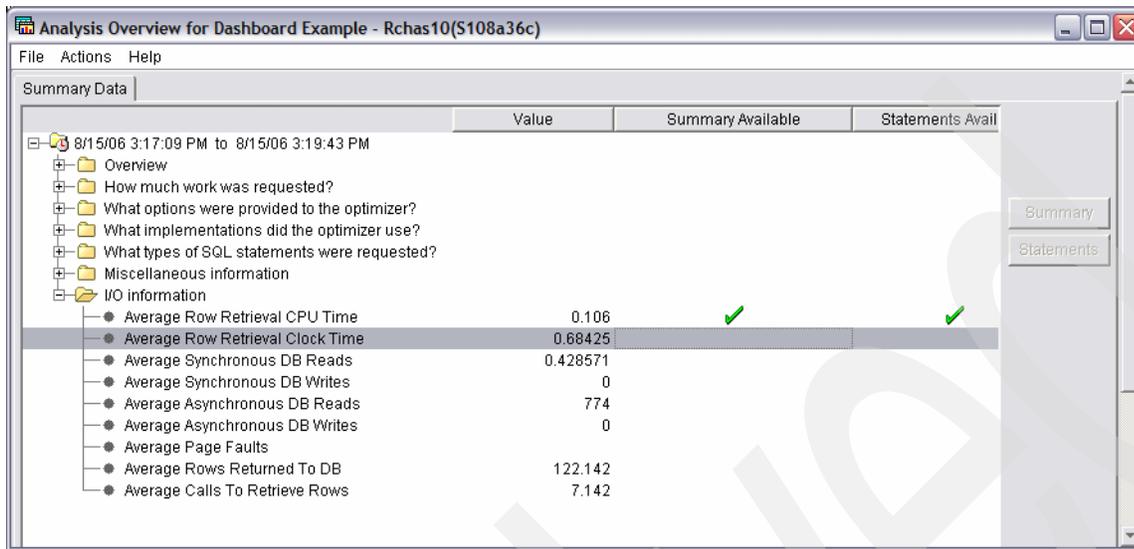
The miscellaneous information category is a place where you can quickly check various maximum values for individual operations such as open, fetch, and close. These are shown in Figure 5-8.

	Value	Summary Available	Statements Avail
8/15/06 3:17:09 PM to 8/15/06 3:19:43 PM			
Overview			
How much work was requested?			
What options were provided to the optimizer?			
What implementations did the optimizer use?			
What types of SQL statements were requested?			
Miscellaneous Information			
● Maximum Runtime	1.273808		✓
● Maximum Open Time	0.08756		✓
● Maximum Fetch Time	1.273808		✓
● Maximum Close Time	0.002728		✓
● Maximum Other Time	0.000856		✓
● Maximum Table Rows	600572		✓
● Maximum Rows Returned	582		✓
IO information			

Figure 5-8 Miscellaneous Information

5.1.7 I/O information

The final category in the dashboard display is the I/O information, shown here in Figure 5-9.



The screenshot shows a window titled "Analysis Overview for Dashboard Example - Rchas10(S108a36c)". The window has a menu bar with "File", "Actions", and "Help". Below the menu bar is a "Summary Data" tab. The main area displays a table with columns "Value", "Summary Available", and "Statements Avail". The table lists various I/O metrics for the period "8/15/06 3:17:09 PM to 8/15/06 3:19:43 PM". The "Average Row Retrieval Clock Time" row is highlighted and has green checkmarks in both the "Summary Available" and "Statements Avail" columns. On the right side of the window, there are two buttons: "Summary" and "Statements".

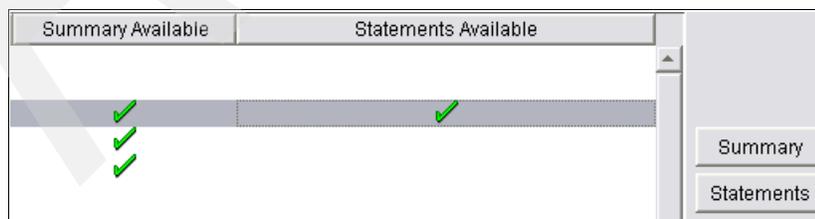
	Value	Summary Available	Statements Avail
8/15/06 3:17:09 PM to 8/15/06 3:19:43 PM			
Overview			
How much work was requested?			
What options were provided to the optimizer?			
What implementations did the optimizer use?			
What types of SQL statements were requested?			
Miscellaneous information			
I/O information			
Average Row Retrieval CPU Time	0.106	✓	✓
Average Row Retrieval Clock Time	0.68425	✓	✓
Average Synchronous DB Reads	0.428571		
Average Synchronous DB Writes	0		
Average Asynchronous DB Reads	774		
Average Asynchronous DB Writes	0		
Average Page Faults			
Average Rows Returned To DB	122.142		
Average Calls To Retrieve Rows	7.142		

Figure 5-9 Input / Output information

As you can see, this category lists several averages related to input/output operations, such as average CPU and clock time for the queries, as well as average synchronous and asynchronous database reads and writes. The information displayed here is derived from the detailed information logged in the 3019 records, which are collected by default when using the SQL Performance Monitor wizard, or by specifying *DETAIL on the STRDBMON command.

5.2 In-depth analysis reports

Now that you are familiar with the look and feel of the new dashboard interface, you are ready to begin focusing in on the details behind the overview analysis items. For each entry in the analysis overview, you may have the option of getting a summary report, a statement report, or both. The system will place a checkmark under the appropriate column if the respective report is available for that overview. The appropriate buttons will also be activated on the right side of your window (Figure 5-10). You can also right-click the desired overview analysis report and get a menu from which you can select Summary or Statements.



This close-up view shows the "Summary Available" and "Statements Available" columns. The "Average Row Retrieval Clock Time" row is highlighted and has a green checkmark in the "Summary Available" column and a green checkmark in the "Statements Available" column. On the right side, the "Summary" and "Statements" buttons are visible and appear to be active.

Summary Available	Statements Available
✓	✓
✓	
✓	

Figure 5-10 Summary and Statement options

5.2.1 Getting detailed reports from Summary and Statement buttons

When you select the *Summary* button, or right-click and select the Summary menu option, the system aggregates the data in the underlying database monitor collection that relates to the particular analysis area. When you select the *Statement* button, or right-click and select the *Statement* menu option, the systems returns info for the individual statements, therefore, you will generally get a row back for each statement relating to the particular analysis area. In Figure 5-11, we are requesting a Summary report for the SQL Statements analysis overview item.

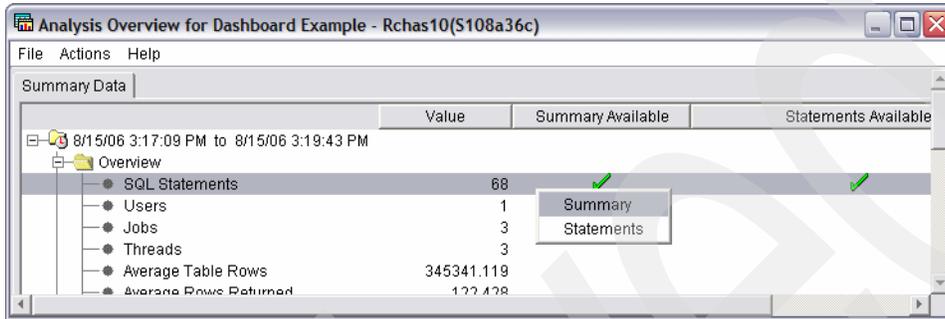
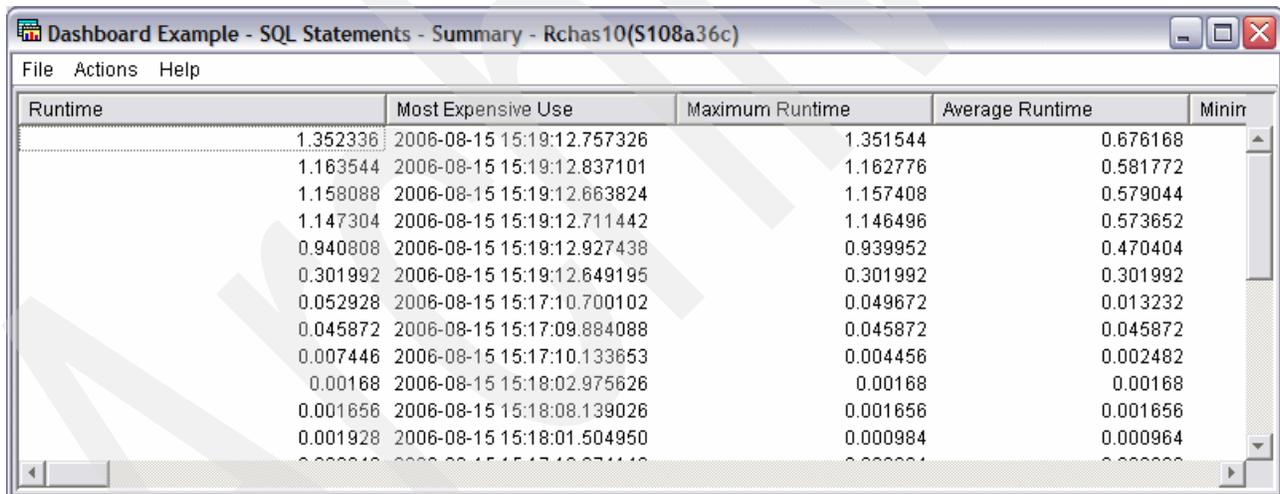


Figure 5-11 Right-clicking to select a summary report

For the summary report, the system groups identical statements together and totals the associated run time for each. Figure 5-12 shows the window that is initially returned.

The screenshot shows a window titled "Dashboard Example - SQL Statements - Summary - Rchas10(S108a36c)". It has a menu bar with "File", "Actions", and "Help". Below the menu bar is a table with the following columns: "Runtime", "Most Expensive Use", "Maximum Runtime", "Average Runtime", and "Minin". The table contains 15 rows of data. The first row is highlighted. The data in the first row is: Runtime: 1.352336, Most Expensive Use: 2006-08-15 15:19:12.757326, Maximum Runtime: 1.351544, Average Runtime: 0.676168, Minin: 0.676168.

Runtime	Most Expensive Use	Maximum Runtime	Average Runtime	Minin
1.352336	2006-08-15 15:19:12.757326	1.351544	0.676168	0.676168
1.163544	2006-08-15 15:19:12.837101	1.162776	0.581772	0.581772
1.158088	2006-08-15 15:19:12.663824	1.157408	0.579044	0.579044
1.147304	2006-08-15 15:19:12.711442	1.146496	0.573652	0.573652
0.940808	2006-08-15 15:19:12.927438	0.939952	0.470404	0.470404
0.301992	2006-08-15 15:19:12.649195	0.301992	0.301992	0.301992
0.052928	2006-08-15 15:17:10.700102	0.049672	0.013232	0.013232
0.045872	2006-08-15 15:17:09.884088	0.045872	0.045872	0.045872
0.007446	2006-08-15 15:17:10.133653	0.004456	0.002482	0.002482
0.00168	2006-08-15 15:18:02.975626	0.00168	0.00168	0.00168
0.001656	2006-08-15 15:18:08.139026	0.001656	0.001656	0.001656
0.001928	2006-08-15 15:18:01.504950	0.000984	0.000964	0.000964
0.000000	2006-08-15 15:17:00.000000	0.000000	0.000000	0.000000

Figure 5-12 Initial SQL Statements report

Since there are several columns of data returned, you will need to use the scroll bar to move left and right to view all the columns. You can also resize the column by clicking and dragging the column separator bar, or you can rearrange the columns to a different order by clicking the column heading and dragging it to the left or right. (Note that to move the column more than a single window width left or right, you may have to pick it up and drop it multiple times).

You can sort a column in ascending order by clicking its heading once. Click the column heading a second time to sort in descending order, and a third time to remove the sort. You can also set up secondary and tertiary sorting columns by right-clicking the primary sort column and selecting **Sort** → **Edit sort**. When you have selected a sort order, an up or down arrow appears next to the column name to indicate the sort order. In Figure 5-13, for

demonstration purposes we have rearranged the columns to include on a single window the statement type, statement text, total run time, and number of uses, and we have sorted by descending run time.

Operation	Runtime	Statement Usage Count	Statement Text
OPEN...CLOSE (HARD)	1.352336	2	SELECT O.* FROM ORDERS O, CUSTOMERS C
OPEN...CLOSE (HARD)	1.163544	2	SELECT O.* FROM ORDERS O, CUSTOMERS C
OPEN...CLOSE (HARD)	1.158088	2	SELECT * FROM ORDERS WHERE ORDERKEY = ?
OPEN...CLOSE (HARD)	1.147304	2	SELECT * FROM ORDERS WHERE YEAR = ?
OPEN...CLOSE (HARD)	0.940808	2	SELECT YEAR, SHIPMODE, SUM(REVENUE_WO_TAX) as total_reve
CALL	0.301992	1	call dbitsodb02/sqlapplication1
OPEN...FETCH	0.052928	4	SELECT SYSSUM.NAME_00001, SYSSUM.STATU00001, SYSSUM.JC
OPEN...CLOSE (HARD)	0.045872	1	SELECT SYSSUM.NAME_00001, SYSSUM.STATU00001, SYSSUM.JC
OPEN...FETCH	0.007446	3	SELECT DBXRDBN, DBXRDEV, DBXRMOD, DBXRMTN, DBXRLCL, D
CALL	0.002646	3	CALL QSYS.QCMDEXC ('CHKOBJ OBJ(QUSRSYS/QAUGDBPMD) OB
CALL	0.002295	3	CALL QSYS.QCMDEXC ('CHKOBJ OBJ(QUSRSYS/QAUGDBPMD2) O
CALL	0.001928	2	CALL QSYS.QCMDEXC ('CHGJOB JOB(040546/QUSER/QZDASOINIT

Figure 5-13 Columns rearranged and sorted by descending runtime

Using this report, you can very quickly determine which SQL statement collectively took the most amount of time. In the previous release, this was as far as you could take the analysis using the GUI. But, in V5R4, you now have additional options. From this window, you can continue the analysis to focus in on the particular SQL statement that you have identified. As shown in Figure 5-14, you can right-click the row that interests you and you will be given two more options.



Figure 5-14 Options when right clicking the row of interest

Selecting work with SQL statement will open a new SQL Scripts window into which will be populated with the chosen statement, Figure 5-15.

Note: When you right click a particular record within a report, the options that you have in the menu that pops up are sensitive to the type of report you are looking at. For example, in the indexes advised report, which we will look at in the next section, you are given the option to create the advised index. In the table scan report, you are given the option to view all other SQL statements using the same base table. Explore!

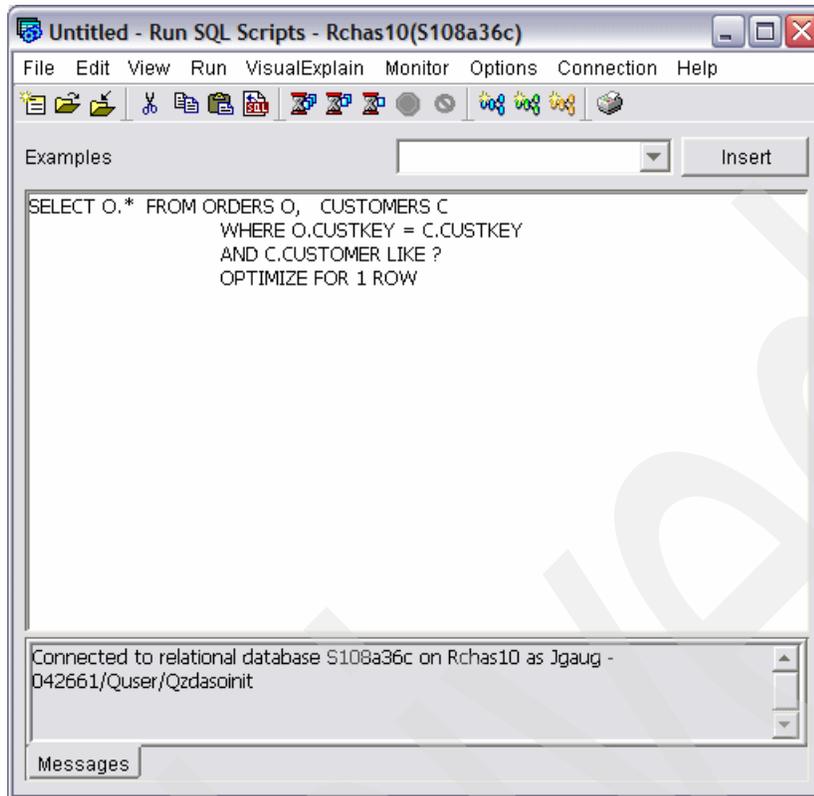


Figure 5-15 Run SQL Scripts window populated with selected statement.

Now, you have all the functionality of SQL Scripts available to you. From here you can run Visual Explain to explain the subject SQL statement by clicking on the  icon, as we have shown in Figure 5-16.

Note: In the SQL Scripts window, literals will likely be replaced by parameter markers, which are indicated by question marks. In order to run Visual Explain, you will need to replace the parameter markers with some real values. The actual host variable values can be obtained by checking the *Variable Values* column in the analysis report.

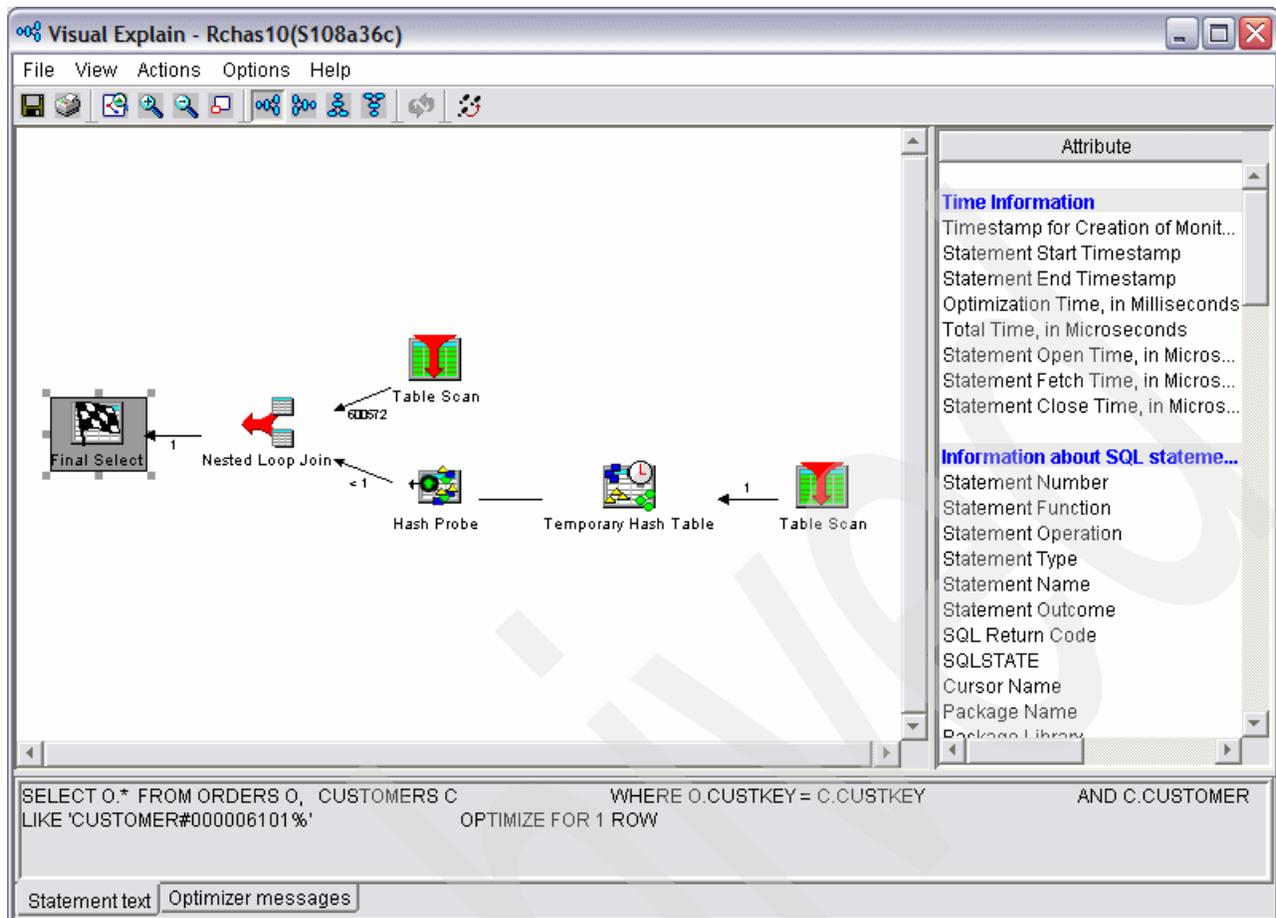


Figure 5-16 Visual Explain Diagram

Based on those results, you can make changes in real time, such as making a modification to the statement or creating an index, and then re-run your query from the same script session. Refer to Chapter 8, “Analyzing database performance data with Visual Explain” on page 275, for a full discussion on Visual Explain and how it can graphically show you a query implementation.

Since starting with the Detailed SQL Performance monitor collection we have been able to drill down to determine a single statement for investigation, and then make real-time modifications and test them. To review, we took the following steps to get here:

1. We selected the Analyze option from the iSeries Navigator performance monitor window.
2. We reviewed the various items available in the Overview Analysis dashboard.
3. We clicked the Summary button to get the list of SQL statements, runtimes, and a host of other information about the statements in the collection.
4. We sorted the statements by descending runtime to determine the longest-running statement.
5. We right-clicked the statement at the top of the list to work with it in a new SQL Scripts window.
6. We ran Visual Explain to examine the implementation of the query.

5.2.2 Additional information reports

In the previous section, we looked in-depth at the SQL Statement summary information report. The SQL Statements item is just one of the many dashboard items that can be analyzed further, as indicated by the checkmarks in the Summary Available and Statements columns. In this section, we'll take a look at several of the other reports that you may find useful in your query performance analysis.

SQE or CQE information reports

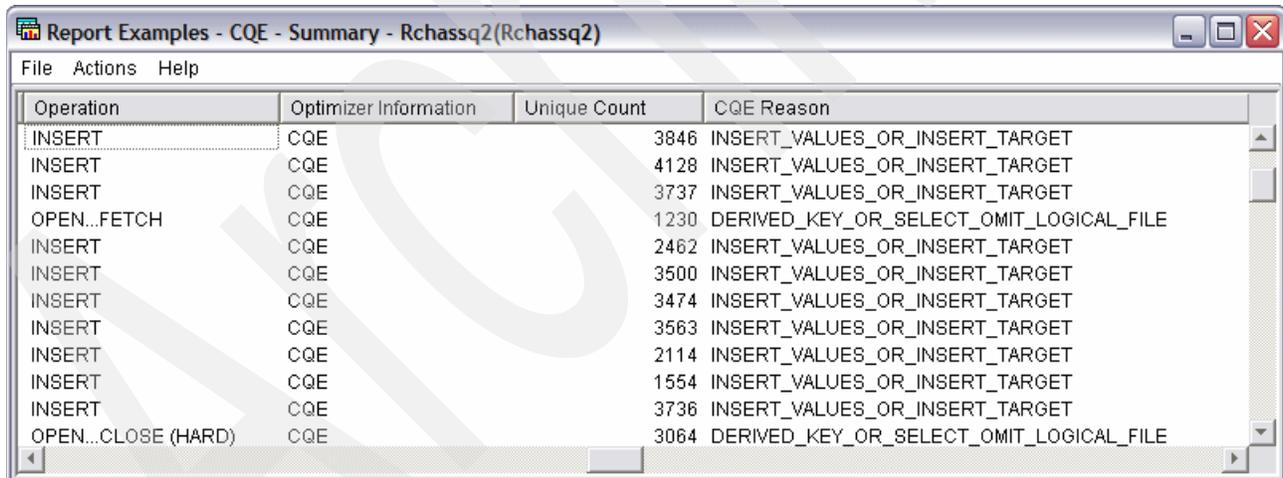
The SQE and CQE information reports give you details on which statements are going down each of the two query engines.

<input checked="" type="radio"/> SQE	118989	✓	✓	
<input checked="" type="radio"/> CQE	12335	✓	✓	

Figure 5-17 SQE or CQE information reports available

At V5R4, you should find that most queries are using the SQL Query Engine. However, there are valid reasons as to why a statement may still be using the Classic Query Engine. Selecting the CQE information report will give you the statements using CQE and the reason they are unable to use SQE. Figure 5-18 shows a sample Summary report.

Note: For display purposes, we have rearranged the columns in the report to a different order than they are first displayed by default. We will do the same for all subsequent sample reports in this chapter.



Operation	Optimizer Information	Unique Count	CQE Reason
INSERT	CQE	3846	INSERT_VALUES_OR_INSERT_TARGET
INSERT	CQE	4128	INSERT_VALUES_OR_INSERT_TARGET
INSERT	CQE	3737	INSERT_VALUES_OR_INSERT_TARGET
OPEN...FETCH	CQE	1230	DERIVED_KEY_OR_SELECT_OMIT_LOGICAL_FILE
INSERT	CQE	2462	INSERT_VALUES_OR_INSERT_TARGET
INSERT	CQE	3500	INSERT_VALUES_OR_INSERT_TARGET
INSERT	CQE	3474	INSERT_VALUES_OR_INSERT_TARGET
INSERT	CQE	3563	INSERT_VALUES_OR_INSERT_TARGET
INSERT	CQE	2114	INSERT_VALUES_OR_INSERT_TARGET
INSERT	CQE	1554	INSERT_VALUES_OR_INSERT_TARGET
INSERT	CQE	3736	INSERT_VALUES_OR_INSERT_TARGET
OPEN...CLOSE (HARD)	CQE	3064	DERIVED_KEY_OR_SELECT_OMIT_LOGICAL_FILE

Figure 5-18 CQE summary report showing reasons why CQE was used

Remember, the real value-add of the new interface is that you can continue to drill down into your statement from here. You can right-click the statement, select **Work with SQL Statement** to bring up a Run SQL Scripts window, make any changes to the SQL statement, and from there run Visual Explain to see how the changes affect the query implementation.

Open information reports

The open information reports contain a row of information regarding open activity for each SQL statement.

● Unique Open Statements	781	✓	✓
● Full Opens	17730	✓	✓
● Pseudo Opens	161310	✓	✓

Figure 5-19 Open information reports available

If an open statement occurs for the first time for a specific statement in a job, it is a *full open*. A full open creates an (Open Data Path) ODP that is used to fetch, update, delete, or insert rows. Since there are typically many fetch, update, delete, or insert operations for an ODP, as much processing of the SQL statement as possible is done during the ODP creation, so that the same processing is not done on each subsequent I/O operation. An ODP may be cached at close time so that, if the SQL statement is run again during the job, the ODP is reused. Such an open is called a *pseudo-open*, which is much less expensive than a full open.

Using the full open reports, you can verify that an SQL statement run multiple times within the same job is taking advantage of a reusable ODP. To obtain the sample report in Figure 5-20, we first selected the Statements report for full opens. We then found an instance of the statement which we wanted to investigate further, right-clicked it, and chose the option to show SQL Statements with the Same Statement Text (Figure 5-20).

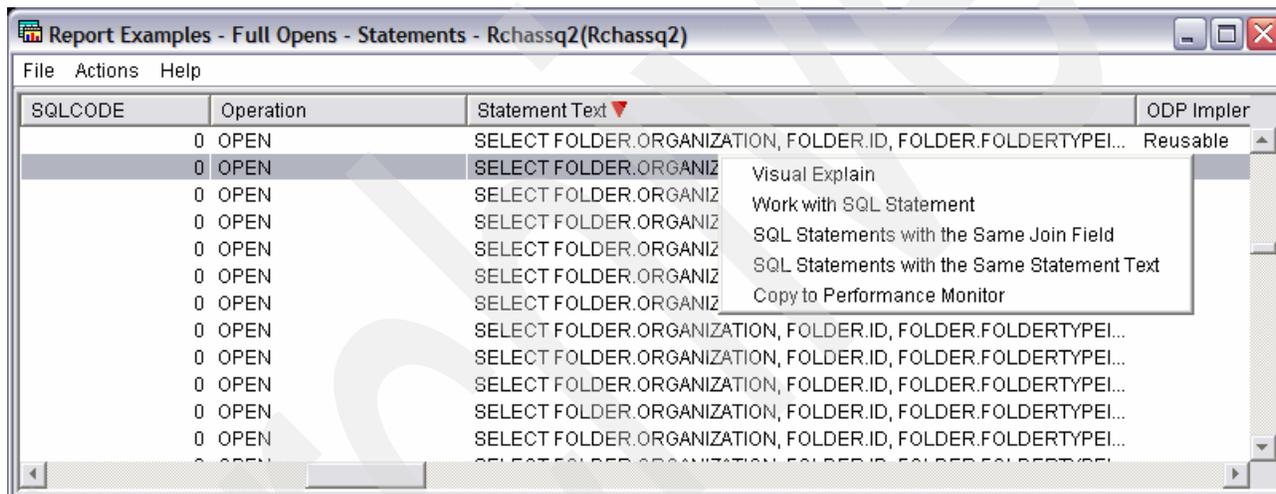


Figure 5-20 Right click to select SQL statements with same text

In the new window containing records for just the selected statement text, we sorted on start time in ascending order (Figure 5-21).

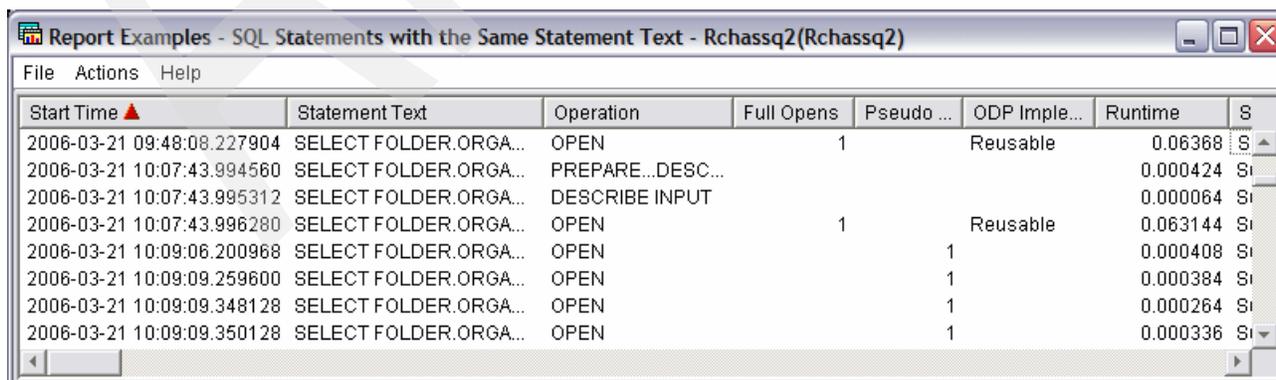


Figure 5-21 Report demonstrating reuse of open data path

You can see that a full open was logged for the first and second opens of the statement and a pseudo-open was logged for subsequent opens. After two full opens of the same SQL statement, the optimizer will generally perform a pseudo-close of the cursor which will maintain the open data path, thus allowing ODP re-use on the third and subsequent runs. Note that the runtime logged for the pseudo-opens are less than the run times of the full opens by a factor of more than 100. This demonstrates the performance benefit of the optimizer being able to re-use an open data path.

Table scan information report

The table scan information report indicates which entire tables are scanned in arrival sequence without using an index.

Table Scans	1130	✓	✓
-------------	------	---	---

Figure 5-22 Table scan information reports available

A table scan is generally acceptable in cases where a large portion of the table is selected or the selected table contains a small number of records. A table scan is not an efficient method when only a small percentage of the rows are being returned. Therefore, if you are investigating a long-running SQL statement that is doing a table scan, creating an appropriate index usually provides better performance.

A sample table scan Statements report is shown in Figure 5-23.

Reason Code	Rows in Base Table	Estimated Rows Selected	Advised Index	Advised Index Leading
No Indexes Exist	323	0	Yes	UNEMPLFLAPID,UNE
No Indexes Exist	323	0	Yes	UNEMPLFLAPID,UNE
No Indexes Exist	323	0	Yes	UNEMPLFLAPID,UNE
Optimizer chose table scan over indexes	7	7	No	
Optimizer chose table scan over indexes	7	7	No	
Optimizer chose table scan over indexes	7	7	No	
Optimizer chose table scan over indexes	7	7	No	
Optimizer chose table scan over indexes	7	7	No	
Optimizer chose table scan over indexes	7	7	No	
Optimizer chose table scan over indexes	7	7	No	
No Indexes Exist	2	1	No	
No Indexes Exist	2	1	No	

Figure 5-23 Table Scan Information report

We have rearranged the column order to show the following columns in the window.

- ▶ Reason code: This column shows why the arrival sequence was chosen by the optimizer.
- ▶ Rows in base table: This column indicates the number of rows in the table.
- ▶ Estimate rows selected: From this value, you can determine if the table scan was done for a significant number or rows.
- ▶ Advised index: This column indicated whether an index was advised.
- ▶ Advised index keys: This column suggests keys for an index. The Advised index keys column can contain both primary and secondary keys. The advised primary index keys column indicates the number of keys that are considered primary. The other keys are considered less selective (secondary).

In the above example, you can see that a table scan is being done for the first three statements because no indexes exist. Creating the advised index may improve performance of this query. For the next several statements the optimizer chooses to do a table scan, which is probably a good implementation, due to the small number of records in the file.

Index advised information report

Although it is possible to see the advised index keys from the table scan information report, you can see the same information from the index advised information report.

● Index Creates Advised	758	✓	✓
-------------------------	-----	---	---

Figure 5-24 Index advised information report available

You can use this report to get advised index information for any jobs that were monitored in the SQL Performance Monitor collection. From the Statements report, you can even right-click the line of interest to create the index, as shown in Figure 5-25.

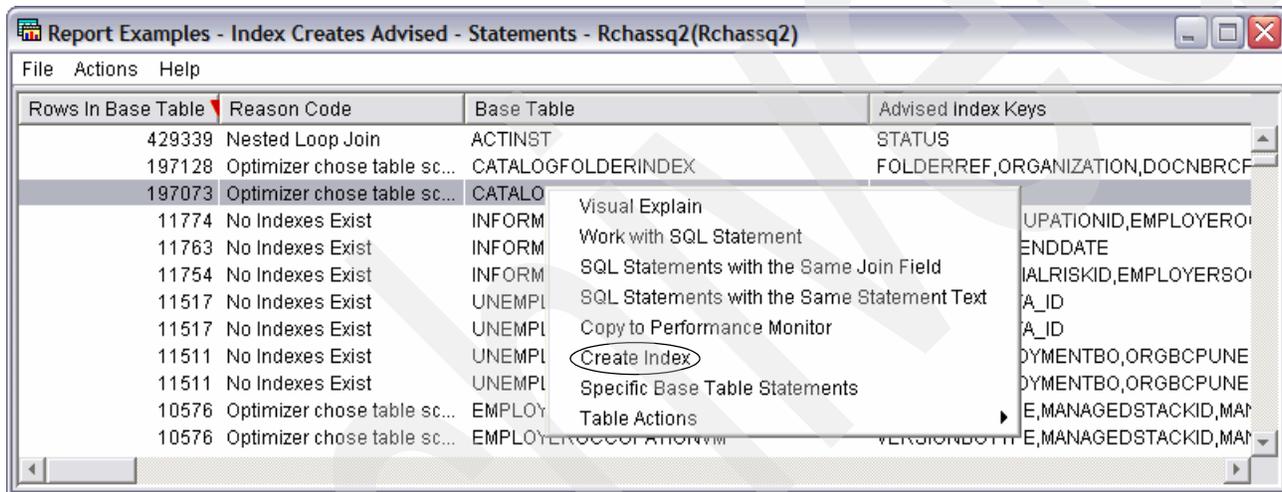


Figure 5-25 Right-click the desired row to create the index advised

Note that at V5R4, there is another new option to get system-wide index advised information and create the advised indexes. The new system-wide index advisor is introduced in Chapter 9, "Index Advisor" on page 319.

Indexes created reports

The Index Created information report contain information for each SQL statement that required an index to be created.

● Full Indexes Created	2	✓	✓
● Sparse Indexes Created	4	✓	✓
● Index From Index Created	4	✓	✓

Figure 5-26 Indexes created reports available

Temporary indexes might need to be created for several reasons such as to perform a join, to support scrollable cursors, or to implement ordering or grouping. The indexes that are created might contain keys only for rows that satisfy the query (such indexes are known as *sparse indexes*). In many cases, index creation might be perfectly normal and the most efficient way to perform the query. However, if the number of rows is large, or if the same index is

repeatedly created, you can create a permanent index to improve performance of this query. This might be true regardless of whether an index was advised.

In CQE temporary index build does *not* always mean that the ODP is nonreusable. The database optimizer tries to reuse the temporary index for each execution of the specific query if possible.

For example, if a host variable value is used to build selection into a temporary index (that is, sparse), then ODP is not reusable because temporary index selection can be different on every execution of the query. In this case, the optimizer tends to avoid creating *sparse* indexes if the statement execution history shows it to be a frequently executed statement. But if the temporary index build is done during repeated query run times and the query goes into reusable ODP mode, then the temporary index *is* reusable.

SQE in V5R4 supports temporary indexes as well. The advantage SQE has over CQE concerning temporary indexes is that an SQE temporary index is maintained as the underlying table data is changed allowing the index to be used by the SQE optimizer for other SQL requests and connections. This SQE temporary index is known as an autonomic index or a Maintained Temporary Index (MTI).

Temporary indexes are never built for selection alone. They always involve a join or a group by or order by clause. Furthermore, since no name is given to the temporary index, the prefix *TEMP is used in the monitor data.

Figure 5-27 shows the Statements report for Full Indexes Created. In this example, you can see that an index with the same key is being created multiple times by the optimizer. Therefore, this index may be a good candidate to be created as a permanent index.

Table/View Name	Entries In Created Index	Reason Code	Created Index Keys	Created Index Name	Created I
ENTITY	488	Nested Loop Join	ID ASCEND	*TEMPX0003	Binary Ra
ENTITY	488	Nested Loop Join	ID ASCEND	*TEMPX0003	Binary Ra
ENTITY	488	Nested Loop Join	ID ASCEND	*TEMPX0003	Binary Ra
ENTITY	488	Nested Loop Join	ID ASCEND	*TEMPX0003	Binary Ra
ENTITY	488	Nested Loop Join	ID ASCEND	*TEMPX0003	Binary Ra

Figure 5-27 Temporary index created report

Access plans rebuilt report

● Access Plans Rebuilt	4904	✓	✓
------------------------	------	---	---

Figure 5-28 Access plan rebuilt report available

The Access plan rebuilt information report contains a row of information for each SQL statement that requires the access plan to be rebuilt. Reoptimization is occasionally necessary for one of several reasons such as a new index being created or dropped, applying a program temporary fix, significant change in the size of a memory pool, and so on. However, excessive access plan rebuilds might indicate a problem.

Figure 5-29 shows an example of the Statements Access Plan Rebuilt report.

Last Access Plan Rebuilt	Unique Count	Access Plan Rebuild Reason	Operation	Statement Text
2006-03-21 09:32:25.974656	480	First run with variable values	UPDATE	UPDATE TARGET
2006-03-21 09:35:31.806816	490	Different storage pool or paging option	OPEN	SELECT GASMTIT
2006-03-21 09:36:51.859752	499	Different storage pool or paging option	OPEN	SELECT DOCASS
2006-03-21 09:36:52.264496	500	Different storage pool or paging option	OPEN	SELECT BOXNBR
2006-03-21 09:36:58.677320	501	Different storage pool or paging option	OPEN	SELECT T.unique
2006-03-21 09:37:04.076400	503	Different storage pool or paging option	OPEN	SELECT IMAGEKE
2006-03-21 09:37:24.877664	510	Different storage pool or paging option	UPDATE	UPDATE CATALO
2006-03-21 09:40:31.021344	524	Different storage pool or paging option	OPEN	SELECT DECISIO
2006-03-21 09:40:33.583176	532	Different storage pool or paging option	OPEN	SELECT LABEL.S
2006-03-21 09:41:11.392960	536	Different storage pool or paging option	OPEN	SELECT GASMTIT
2006-03-21 09:41:19.167896	548	Significant change in the number of rows	OPEN	SELECT C.organiz
2006-03-21 09:41:35.263984	557	System programming change	OPEN	SELECT ZG53S5,
2006-03-21 09:41:40.042432	558	Different storage pool or paging option	OPEN	SELECT decision.

Figure 5-29 Access plan rebuild reasons

From this report, the most important information is in the Access Plan Rebuild Reason column. Some of the possible reasons are:

- ▶ Different File Or Member(change in library list or default schema)
- ▶ More Than Ten Percent Change In Number Of Rows
- ▶ New Access Path Found(Index created)
- ▶ Access Path No Longer Found Or Valid(Index deleted)
- ▶ Different CCSID
- ▶ Different Date Or Time Format
- ▶ Different Sort Sequence Table
- ▶ Different Storage Pool or Paging Option(Optimizer's fair share of memory pool change)
- ▶ Symmetric Multi Processing Change
- ▶ QAQQINI Change
- ▶ Different Isolation Level or Scroll Option
- ▶ New Release

Temporary table report information

The Temporary Table Information reports contains information for SQL statements that required a temporary result as shown in Figure 5-30.

● Temporary Tables	1721	✓	✓
--------------------	------	---	---

Figure 5-30 Temporary table reports available

Temporary results are sometimes necessary based on the SQL statement. If you find that a lot of time is being consumed by creation of temporary results, you might want to investigate why the temporary result is necessary. Figure 5-31 shows a sample Statements report for temporary results.

Unique Count	Temporary Create Time	Number Of Rows In Tempo...	Reason Code
1221	1.72	1	Static cursor specified
1221	1.72	1	Static cursor specified
1221	1.72	1	Static cursor specified
1221	1.72	1	Static cursor specified
1193	0.521	1	Static cursor specified
532	0	0	Optimizer chose sort to minimize I/O wait time
532	0	0	Optimizer chose sort to minimize I/O wait time
533	0	0	Optimizer chose sort to minimize I/O wait time
586	0	1	Optimizer chose sort to minimize I/O wait time

Figure 5-31 Temporary results statement report

In our example, you can see that several of the temporary files were created due to a static, or insensitive, cursor. Insensitive means that once opened, the cursor is not sensitive to changed data, so a temporary file must be created. If you find that a lot of time is being spent creating a temporary file due to a static cursor then you may want to examine your application to see if a static cursor is really necessary. If not, you may be able to eliminate the temporary file builds by declaring the cursor as `ASENSITIVE` instead.

CPU retrieval time report

This report can be found under the I/O information report. This report provides a quick way to review some of the I/O statistics for the statements recorded in the monitor file as shown in Figure 5-32.

● Average Row Retrieval CPU Time	0.00202	✓	✓
----------------------------------	---------	---	---

Figure 5-32 Average Row Retrieval CPU time

At a glance, you can determine which statement was responsible for the most time spent in the CPU as well as the most clock time, check the number of synchronous or asynchronous I/O operations as well as the average number of rows returned, Figure 5-33. A large retrieval time for a relatively small number of rows may indicate a lot of unnecessary I/O is occurring and that the query needs to be tuned.

Row Retrieval CPU Time	Row Retrieval Clock Time	Synchronous Database...	Asynchronous Database Reads	Calls To Retrieve Rows	Rows Returned	Statement
7.252	23.948	177.5	12208.5	12	350	SELECT
6.552	6.724	0	1.5	1	1	SELECT
2.179	9.041	67	6856	3	50	select de
0.896	0.944	0.632	0	2.532	0.632	with a1 a
0.828	1.98	24	3144	2	29	select de
0.582	1.466	4.666	2294	2.666	30.666	select de
0.269	5.601	175	207	100	3141	SELECT
0.248	3.64	178	185	100	3159	SELECT
0.14	0.304	6.948	9.024	346.621	346.621	SELECT
0.087	0.451	9	377	11	309	SELECT
0.083	0.821	6	386	3	43	SELECT
0.065	0.373	87.5	29.255	1.792	9.473	SELECT

Figure 5-33 CPU retrieval time report

In this example, the statement responsible for the most CPU time consumed about 7 seconds of CPU time and 24 seconds of clock time to return. You could right-click that record to view the statement and check its implementation using Visual Explain.

Note: The I/O information reports are based on the 3019 records, so for this information to be available you need to have collected *DETAIL records if you used STRDBMON to start the collection. A detailed collection started through the SQL Performance Monitor wizard will always collect the 3019 records.

Many more reports available

In this section we have gone over a few of the reports which you may find most helpful in your query analysis. However, there are more which may contain important information for your particular situation. Therefore, feel free to explore.

5.2.3 Action menu items

In addition to the reports available when a checkmark is indicated in the Summary Available or Statements Available column, there are a few additional high-level summary reports and other features available from the Actions menu of the main dashboard window, Figure 5-34. Let us take a look at the available options.

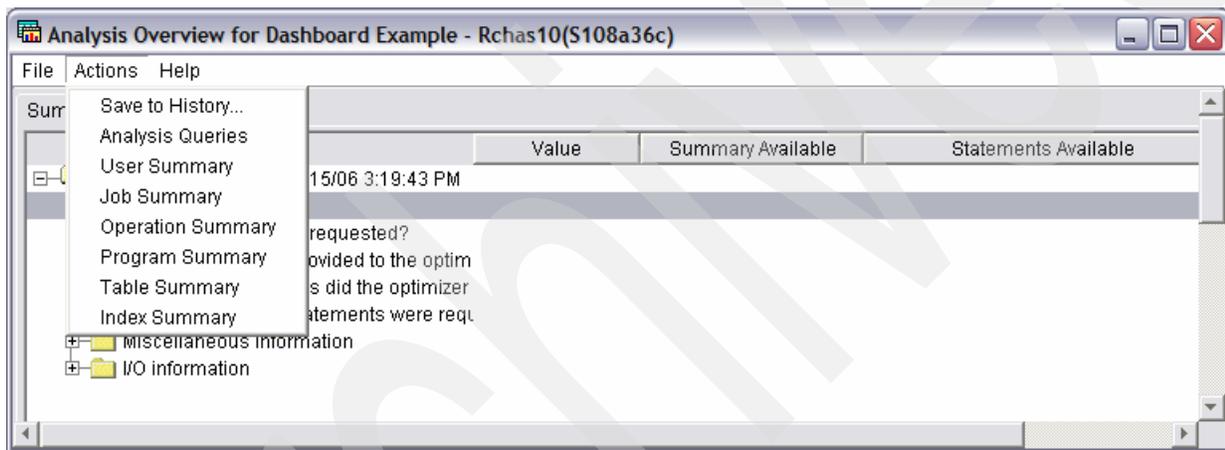


Figure 5-34 Actions menu options

Save history

The save history options allows you to save the analysis overview information into a history file. When you take this option, you will be asked to name a history file to save the information in.

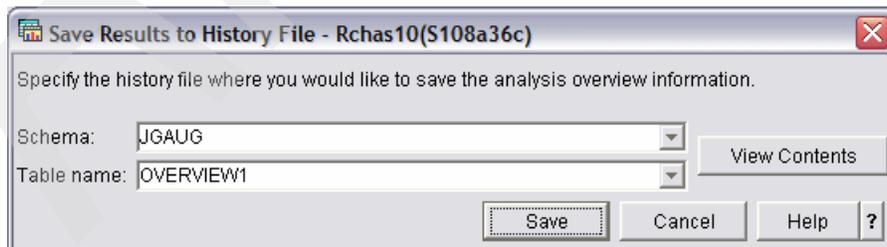
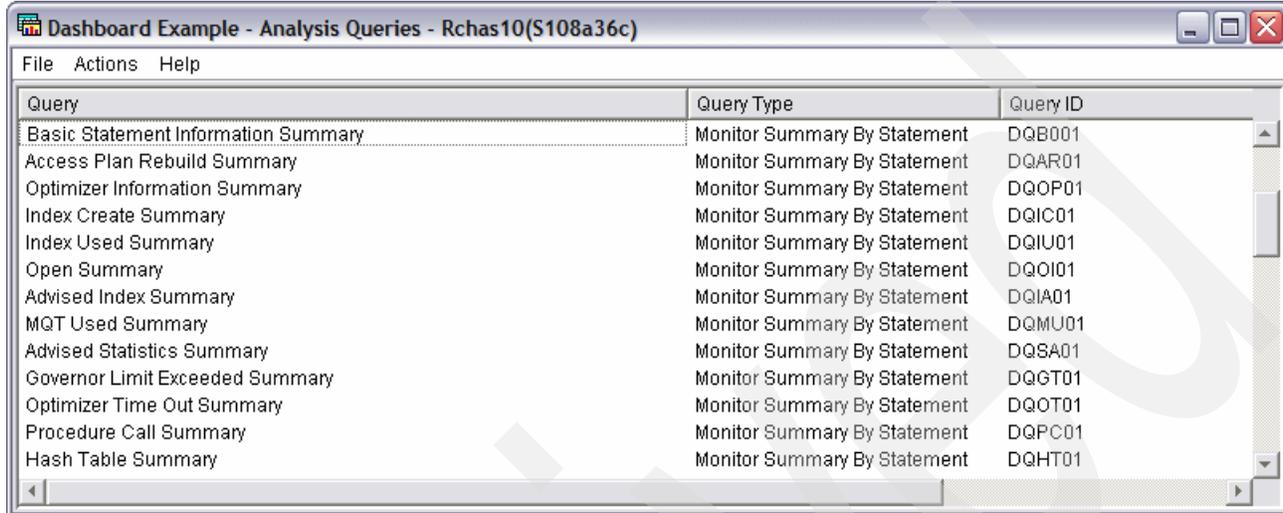


Figure 5-35 Saving the overview information to a history file.

If the history file does not exist, it will be created. The high-level analysis overview information will be saved into a single record in the history file. If the file already exists, then a record will be added. The history file allows you to maintain a repository of the analysis overview information for multiple SQL Performance monitor collections.

Analysis Queries

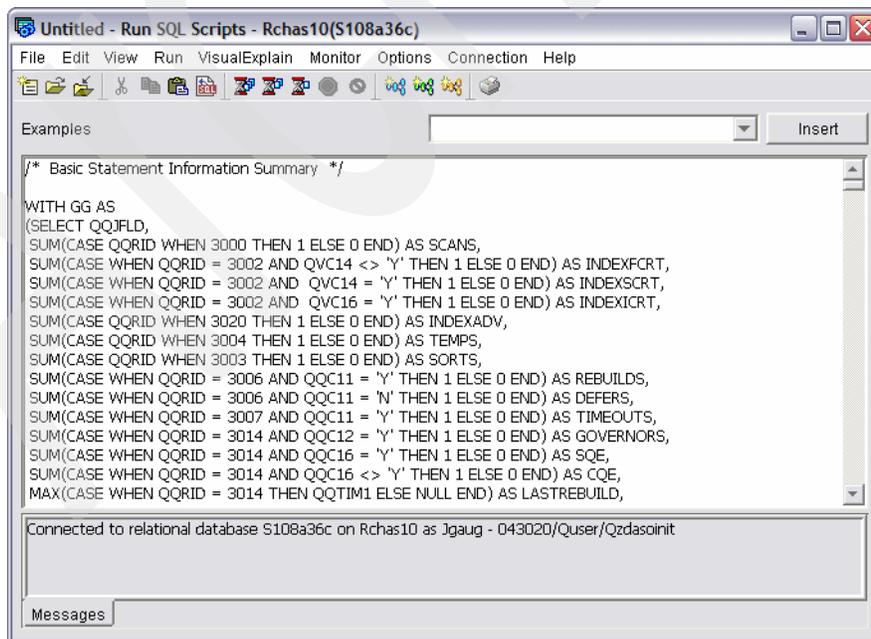
Select this option if you want to see, and optionally modify, the underlying queries that are processed to create the Summary and Statements reports that are available for the Analysis Overview items. The window that comes up for this option lists each of the queries with some descriptive text and a query ID as shown in Figure 5-36.



Query	Query Type	Query ID
Basic Statement Information Summary	Monitor Summary By Statement	DQB001
Access Plan Rebuild Summary	Monitor Summary By Statement	DQAR01
Optimizer Information Summary	Monitor Summary By Statement	DQOP01
Index Create Summary	Monitor Summary By Statement	DQIC01
Index Used Summary	Monitor Summary By Statement	DQIU01
Open Summary	Monitor Summary By Statement	DQOI01
Advised Index Summary	Monitor Summary By Statement	DQIA01
MQT Used Summary	Monitor Summary By Statement	DQMU01
Advised Statistics Summary	Monitor Summary By Statement	DQSA01
Governor Limit Exceeded Summary	Monitor Summary By Statement	DQGT01
Optimizer Time Out Summary	Monitor Summary By Statement	DQOT01
Procedure Call Summary	Monitor Summary By Statement	DQPC01
Hash Table Summary	Monitor Summary By Statement	DQHT01

Figure 5-36 Analysis Queries

When you right-click any of these queries, you have the option to execute it and view the report, or to modify the report in some fashion to suit your specific needs. When you select to modify the report, it will bring up an SQL Scripts window populated with the SQL statement behind the report (as shown in Figure 5-37). Modification of the query will not affect the original query statement for this report, but, you could save the modified query to a new SQL script file for future use.



```
/* Basic Statement Information Summary */
WITH GG AS
(SELECT QQIFLD,
SUM(CASE QQRID WHEN 3000 THEN 1 ELSE 0 END) AS SCANS,
SUM(CASE WHEN QQRID = 3002 AND QVC14 <> 'Y' THEN 1 ELSE 0 END) AS INDEXFCRT,
SUM(CASE WHEN QQRID = 3002 AND QVC14 = 'Y' THEN 1 ELSE 0 END) AS INDEXSCRT,
SUM(CASE WHEN QQRID = 3002 AND QVC16 = 'Y' THEN 1 ELSE 0 END) AS INDEXICRT,
SUM(CASE QQRID WHEN 3020 THEN 1 ELSE 0 END) AS INDEXADV,
SUM(CASE QQRID WHEN 3004 THEN 1 ELSE 0 END) AS TEMPS,
SUM(CASE QQRID WHEN 3003 THEN 1 ELSE 0 END) AS SORTS,
SUM(CASE WHEN QQRID = 3006 AND QQC11 = 'Y' THEN 1 ELSE 0 END) AS REBUILDS,
SUM(CASE WHEN QQRID = 3006 AND QQC11 = 'N' THEN 1 ELSE 0 END) AS DEFERS,
SUM(CASE WHEN QQRID = 3007 AND QQC11 = 'Y' THEN 1 ELSE 0 END) AS TIMEOUTS,
SUM(CASE WHEN QQRID = 3014 AND QQC12 = 'Y' THEN 1 ELSE 0 END) AS GOVERNORS,
SUM(CASE WHEN QQRID = 3014 AND QQC16 = 'Y' THEN 1 ELSE 0 END) AS SQE,
SUM(CASE WHEN QQRID = 3014 AND QQC16 <> 'Y' THEN 1 ELSE 0 END) AS CQE,
MAX(CASE WHEN QQRID = 3014 THEN QQTIM1 ELSE NULL END) AS LASTREBUILD,

```

Figure 5-37 Run SQL Scripts window where you can modify the query if desired

A particularly SQL-savvy user may wish to set a preference to have the buttons displayed in the dashboard which allows you easy access to modify a particular report. To activate this option, from the main dashboard window, select **File** → **Preferences** → **Show Modify Query Buttons**.

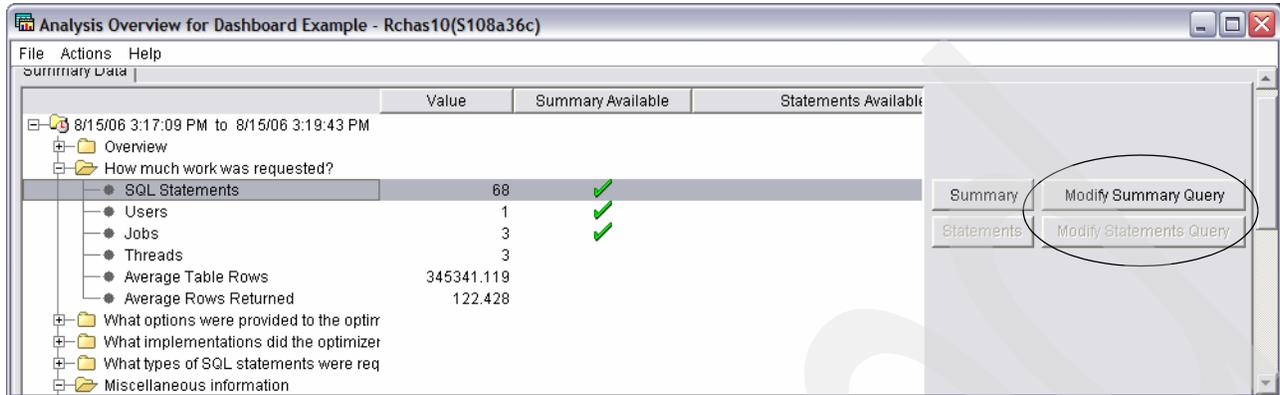


Figure 5-38 Modify Query buttons

As shown in Figure 5-38, the buttons appear next to the Summary and Statement options. Clicking the modify button brings up the SQL Scripts window with the report's query.

User and Job Summary

This is simply another place for you to access the user and job summary queries. These queries will run the same reports as are found in the user and job summary analysis overview items.

Operation Summary

The Operation Summary report will give you a quick overview of the different types of operations (fetch, open, close, and so on) that were captured in the monitor (Figure 5-39).

Operation	SQL Stateme...	Runtime	Maximum Runtime	Average Runtime	Maximum Open
FETCH	10	5.487	1.273	0.548	
OPEN	10	0.368	0.087	0.036	
CALL	11	0.312	0.301	0.028	
CLOSE	10	0.008	0.002	0	
PREPARE	5	0.003	0	0	
PREPARE...DESCRIBE	5	0.001	0	0	
SET CURRENT SCHEMA	1	0	0	0	
SET VARIABLE	9	0	0	0	
CLOSE (Hard)	6	0	0	0	
SET RESULTS	1	0	0	0	

Figure 5-39 Operation Summary report

From this report, you can easily see what types of operations were run most frequently along with their accumulated runtimes. Furthermore, you can drill-down from this report by right-clicking the desired record. You can choose to view all the statements behind that particular operation, or get the operations summary for that specific operation.

Program Summary

The Programs Summary report is very useful if you want to determine what statements your specific SQL programs were running. The report will give you the list of programs along with the number of jobs running those programs and the runtime attributable to each program. SQL statements not associated with any program will be grouped together in a row with a blank program name (as shown in Figure 5-40).

Program Library	Program	Jobs Running Program	Runtime	Maximum Runtime	Average Runtime
DBITSODB02	SQLAP00001	1	0.28	0.087	0.014
		3	5.902	1.273	0.122

Figure 5-40 Program Summary report

As with the Operation Summary report, you can right-click to drill down further. You can choose to view the particular SQL statements that were being run by the program, either summarized or individually, or you can select **Program Actions** → **Explain SQL**, which will bring up a new window with the output from PRTSQLINF for the program.

Table and Index Summary

The Table Summary will give you a quick overview of all the tables accessed in the collected monitor data (Figure 5-41), while the Index Summary gives reports on all the indexes used to implement the queries logged in the monitor.

SQL Statements	Base Table Library	Base Table	Rows In Base Table	Table Size	System Base Table Library
5	QUSRSYS	QAUGDBPMD2	81	140032	QUSRSYS
5	DBITSODB02	ORDERS	600572	160958192	DBITSODB02
3	QUSRSYS	QAUGDBPMD	7	7344	QUSRSYS
2	DBITSODB02	CUSTOMERS	15000	3315096	DBITSODB02

Figure 5-41 Table Summary report

5.3 Summary SQL Performance Monitor analysis overview

To begin viewing the results for the Summary SQL Performance Monitor, right-click the paused or ended monitor. You will have the following options for a Summary monitor as shown in Figure 5-42:

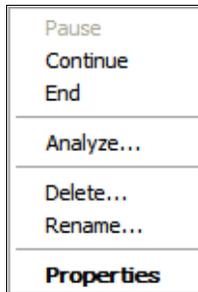


Figure 5-42 Options for a summary monitor

- ▶ **Pause:** This function stops the current collection of statistics and writes the current statistics into several database files or tables that can be queried by selecting the Analyze Results action. The monitor remains ready to collect more statistics, but requires the Continue action to restart collection.
- ▶ **Continue:** This function restarts the collection of statistics for a monitor that is currently paused.
- ▶ **Analyze:** The Summary Monitor analysis window is opened. This is unchanged between V5R3 and V5R4.

Note that the option to Show Statements and Compare are missing, since those options only apply to Detailed monitor.

To begin analysis of the Summary monitor, select **Analyze**.

Figure 5-43 shows the first results panel that groups analysis options according to three tabs:

- ▶ Summary Results
- ▶ Detailed Results
- ▶ Composite View

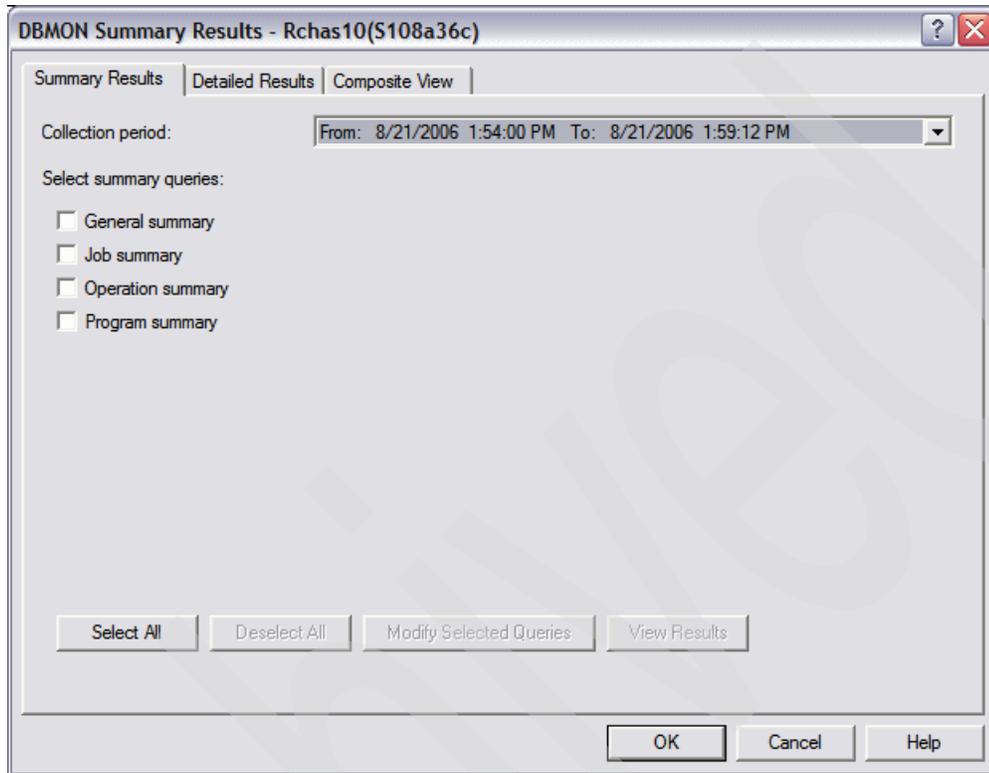


Figure 5-43 Access SQL Performance Monitor results

On the Summary Results page, you can select individual summary queries or use the **Select All** button. After you select the queries that you want to run, click the **View Results** button.

You can also choose to modify the predefined queries and run the new queries by clicking the **Modify Selected Queries** button.

The Summary SQL Performance Monitor uses its own set of tables instead of using the single table that the Detailed SQL Performance Monitor uses. The Summary monitor tables closely match the suggested logical file formats of the Detailed SQL Performance Monitor.

5.3.1 SQL performance report information from summary data

To begin viewing the summary reports from Summary SQL Performance Monitor, click **View Results** in the Summary Results tab.

Because all reports from the summary results are summarized at an output table level, you can get a high-level perspective of what was happening during the monitoring such as:

- ▶ Analysis of the type of SQL operations
- ▶ Identification of the jobs that consumed the most DB2 resources
- ▶ Information regarding the programs that ran the monitored SQL

Summary reports of the Summary monitor

The Summary SQL Performance Monitor collects some of the same information as the Detailed SQL Performance Monitor, but the performance statistics are kept in memory. At the expense of detail, information is summarized for identical SQL statements to reduce the amount of information collected. The objective is to get the statistics to memory as fast as possible, while deferring any manipulation or conversion of the data until the performance data is dumped to a result table for analysis.

In the first tab, you can select any of the four summary reports, in any combination, and then click the **View Results** button. If multiple reports were selected, then multiple windows will be brought up, one for each report. To produce Figure 5-44, we selected all four reports. You can move the scroll bar left and right to view the several columns returned for each row.

	Total Runtime (sec)	Maximum Runtime	Average Runtime	Maximum Open Time	Maximum Fetch Time
1	92.528	5.406	.214	4.915	.335

	Job	Job User	Job Number	Total Runtime (sec)	Maximum Runtime	Average Runtime	Maximum Ot
1	QRWTSRVR	QUSER	308498	25.361	5.406	3.623	0
2	QSQSRVR	QUSER	304301	22.323	5.251	1.888	4.915
3	QSQSRVR	QUSER	304300	12.052	1.227	.077	.707
4	QZDASOINIT	QUSER	306217	7.607	2.380	.865	2.068
5	QDBSRV05	QSYS	303960	5.176	4.242	2.588	0

	Operation	Total Statements	Total Runtime (sec)	Maximum Runtime	Average Runtime	Maximum Oper
1	OPEN	5712	42.907	5.251	.211	4.915
2	INSERT	262	28.982	5.406	.180	0
3	UPDATE	18	7.098	4.242	.515	0
4	DELETE	87	5.221	.335	.131	0
5	SELECT INTO	11	3.773	1.615	.343	0
6	COMMIT	208	2.080	.285	.010	0

	Program Library	Program	Total Jobs Running Program	Total Runtime	Maximum Runtime	Average Runtime
1			16	64.019	5.251	.182
2	QSYS	QSCFNTRC	1	25.361	5.406	3.623
3	QSYS	Q1PCNT4	1	1.615	1.615	1.615
4	QSYS	QS9UTIL	1	.768	.444	.096
5	QPFR	QVPESVGN	1	.765	.631	.255

Figure 5-44 Summary Monitor summary reports

We'll briefly describe each of the four summary reports:

- ▶ **General summary:** This report contains information in a single row that summarizes all SQL activity, such as the amount of SQL used in the application, whether the SQL statements are mainly short-running or long-running, and whether the number of results returned is small or large.
- ▶ **Job summary:** This report contains a row of information for each job. The information can be used to tell which jobs on the system are the heaviest users of SQL, and therefore, which ones are perhaps candidates for performance tuning.

- ▶ Operation summary: This report contains a row of summary information for each type of SQL operation. A quick review of this report can tell you if you have a specific operation type that is consuming most of the SQL runtime.
- ▶ Program summary: This report contains a row of information for each program that performed SQL operations. This information can be used to identify which programs use the most or most expensive SQL statements. Note that a program name is available only if the SQL statements are embedded inside a compiled program. SQL statements that are issued through Open Database Connectivity (ODBC), Java Database Connectivity (JDBC), or OLE DB have a blank program name unless they result from a procedure, function, or trigger.

Detail reports of the Summary monitor

The Detailed reports tab is shown in Figure 5-45.

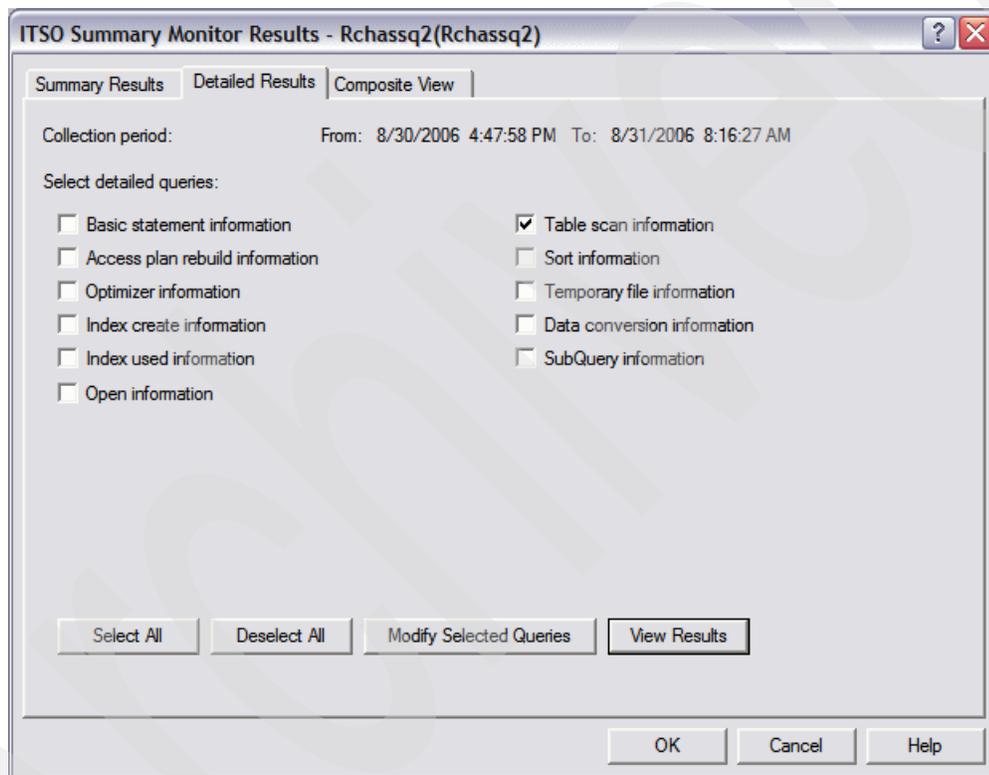


Figure 5-45 Detailed results tab of the summary monitor

In this example, Figure 5-46, we have selected the table scan report. The report will give you a row for each query that was implemented using a table scan, along with runtime information and information about the base table that the query was run over.

	Time	Estimated Processing Time	Reason Code	Total Rows In Base Table	Estimated Rows Selected
1	2006-08-30 16:57:24.440491	.019	T1	120275	120
2	2006-08-30 16:57:32.893567	.019	T1	120275	120
3	2006-08-31 05:13:57.199509	.001	T1	0	0
4	2006-08-31 00:21:07.483143	.001	T1	10	10
5	2006-08-31 00:21:08.127248	.001	T1	0	0
6	2006-08-31 05:18:12.854745	.001	T3	4	0
7	2006-08-31 05:18:12.854745	.001	T3	166	0
8	2006-08-31 02:46:59.627787	.001	T2	1226	123
9	2006-08-31 00:21:05.683032	.001	T1	0	0
10	2006-08-31 00:50:45.773197	.001	T2	20	2
11	2006-08-31 00:21:06.605906	.001	T1	0	0
12	2006-08-31 02:45:27.231970	.001	T3	1	1
13	2006-08-31 02:45:26.706256	.001	T3	2	1

Figure 5-46 Detailed table scan report of the summary monitor

A review of the information contained in this report will let you know if you may have an excessive number of table scans occurring on your system.

Composite view reports of the Summary monitor

The composite view tab provides the same report options as the detailed tab, but instead of returning each report in a separate window, a single composite report is returned featuring data from each of the selected report areas.

5.3.2 Examples and application of Summary SQL Performance Monitor

For further applications and examples of using the Summary SQL Performance Monitor, please refer to the following URL and click **Database Memory Monitor Command**.

<http://www-03.ibm.com/servers/eserver/series/db2/db2code.html>

5.3.3 Limitations of the Summary monitor

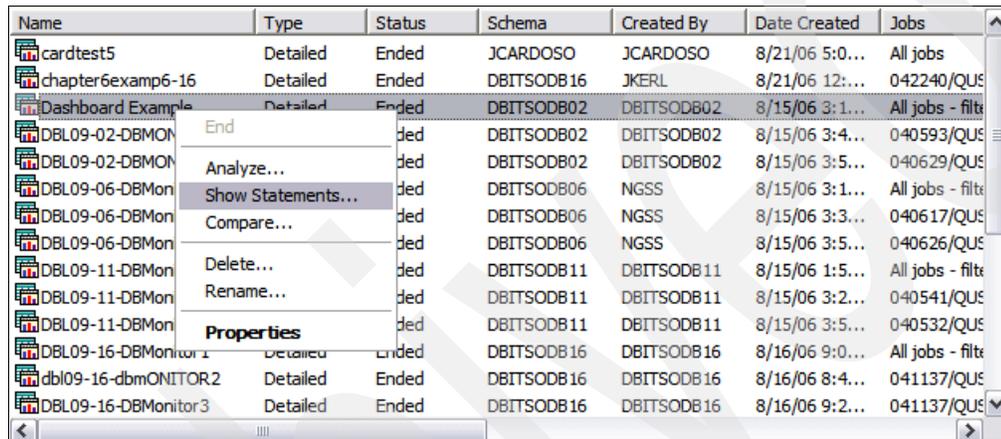
It is important to mention that while the Summary monitor can be a valid feedback mechanism, it is currently not the strategic direction for SQL performance monitoring in i5/OS. Current development efforts are focused on tools for analysis of the Detailed SQL Performance Monitor. For example, as you learned in 3.6, “Index Advisor” on page 63, the index advisor is improved in recent releases to provide better index advice. However, the Summary monitor was not updated to support the improved recommendations. Therefore, you need to use the Detailed monitor to take full advantage of the index advice improvements.

Additionally, the Summary monitor does not feature the drill-down capability of the new Detailed SQL Performance Monitor tools, nor does it offer the ability to sort on individual columns from within the report.

5.4 Show Statements

At V5R4, the new Show Statements option replaces the List Explainable Statements option from V5R3 and earlier releases. The Show Statements interface features filtering capabilities that allow you to reduce the list of statements displayed to a more manageable size. This new Show Statements interface is also used to display statements from the SQE Plan Cache and SQE Plan Cache Snapshots, which will be introduced in Chapter 7, “SQE Plan Cache and SQE Plan Cache Snapshots” on page 237.)

To access the Show Statements interface, right-click the desired monitor and choose **Show Statement**, as shown in Figure 5-47.



Name	Type	Status	Schema	Created By	Date Created	Jobs
cardtest5	Detailed	Ended	JCARDOSO	JCARDOSO	8/21/06 5:0...	All jobs
chapter6examp6-16	Detailed	Ended	DBITSODB16	JKERL	8/21/06 12:...	042240/QUS
Dashboard Example	Detailed	Ended	DBITSODB02	DBITSODB02	8/15/06 3:1...	All jobs - filte
DBL09-02-DBMON		ded	DBITSODB02	DBITSODB02	8/15/06 3:4...	040593/QUS
DBL09-02-DBMON		ded	DBITSODB02	DBITSODB02	8/15/06 3:5...	040629/QUS
DBL09-06-DBMon		ded	DBITSODB06	NGSS	8/15/06 3:1...	All jobs - filte
DBL09-06-DBMon		ded	DBITSODB06	NGSS	8/15/06 3:3...	040617/QUS
DBL09-06-DBMon		ded	DBITSODB06	NGSS	8/15/06 3:5...	040626/QUS
DBL09-11-DBMon		ded	DBITSODB11	DBITSODB11	8/15/06 1:5...	All jobs - filte
DBL09-11-DBMon		ded	DBITSODB11	DBITSODB11	8/15/06 3:2...	040541/QUS
DBL09-11-DBMon		ded	DBITSODB11	DBITSODB11	8/15/06 3:5...	040532/QUS
DBL09-16-DBMonitor1	Detailed	Ended	DBITSODB16	DBITSODB16	8/16/06 9:0...	All jobs - filte
dbl09-16-dbmONITOR,2	Detailed	Ended	DBITSODB16	DBITSODB16	8/16/06 8:4...	041137/QUS
DBL09-16-DBMonitor3	Detailed	Ended	DBITSODB16	DBITSODB16	8/16/06 9:2...	041137/QUS

Figure 5-47 Select Show Statements option

5.4.1 Filtering options

The list of statements window is initially empty as shown in Figure 5-48, which allows you the opportunity to set filters on the data before the monitor file is interrogated for a list of statements. You can filter by any one or multiple of the following criteria:

- ▶ Minimum runtime for the longest execution: Enter a whole number of seconds. Any statement with an actual runtime of less than the chosen value will not be displayed.
- ▶ Queries run after this data and time: This allows you to subset the data based on a specific time period.
- ▶ Queries that reference these objects: Select only those queries that are referencing objects in the listed tables and schemas.
- ▶ SQL statement contains: Enter a string of text which must be contained in the statements retrieved.

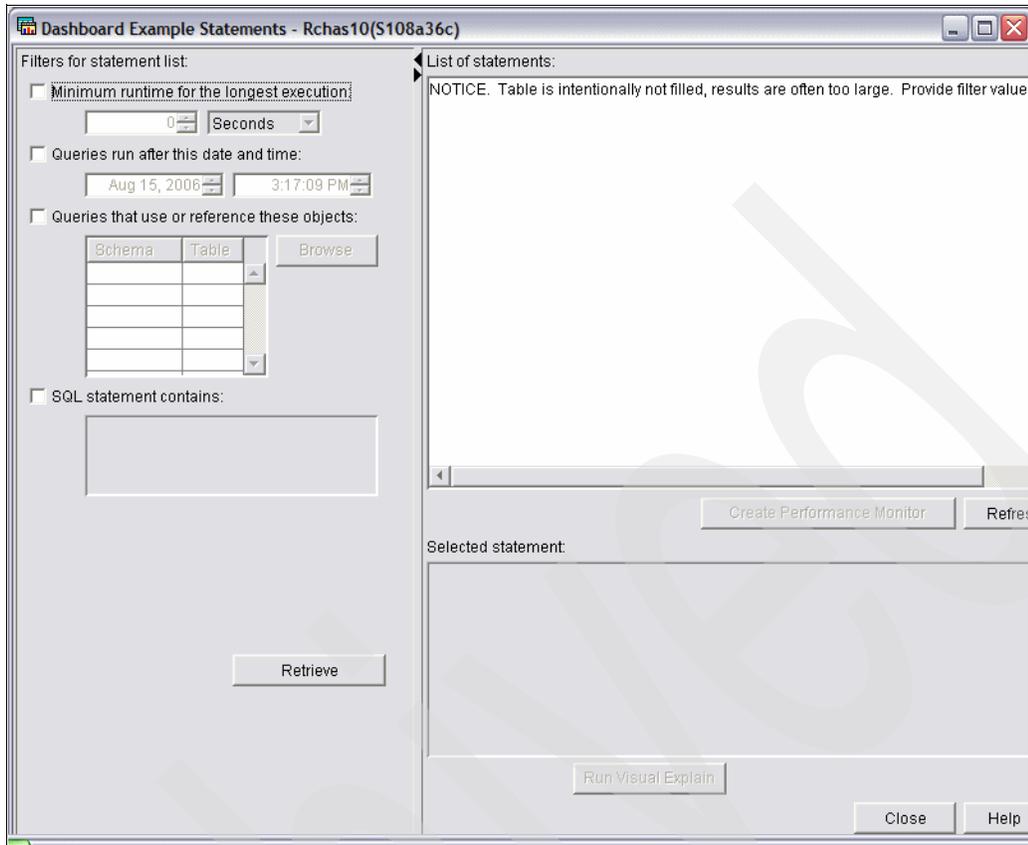


Figure 5-48 Interface to the collection's statements

Alternatively, simply leave the filter options blank to retrieve all data from the monitor table. Once you have made your choices, hit the **Retrieve** button and the List of Statements window

will be populated. In this example, Figure 5-49, we have requested to only show statements that reference the ORDERS table.

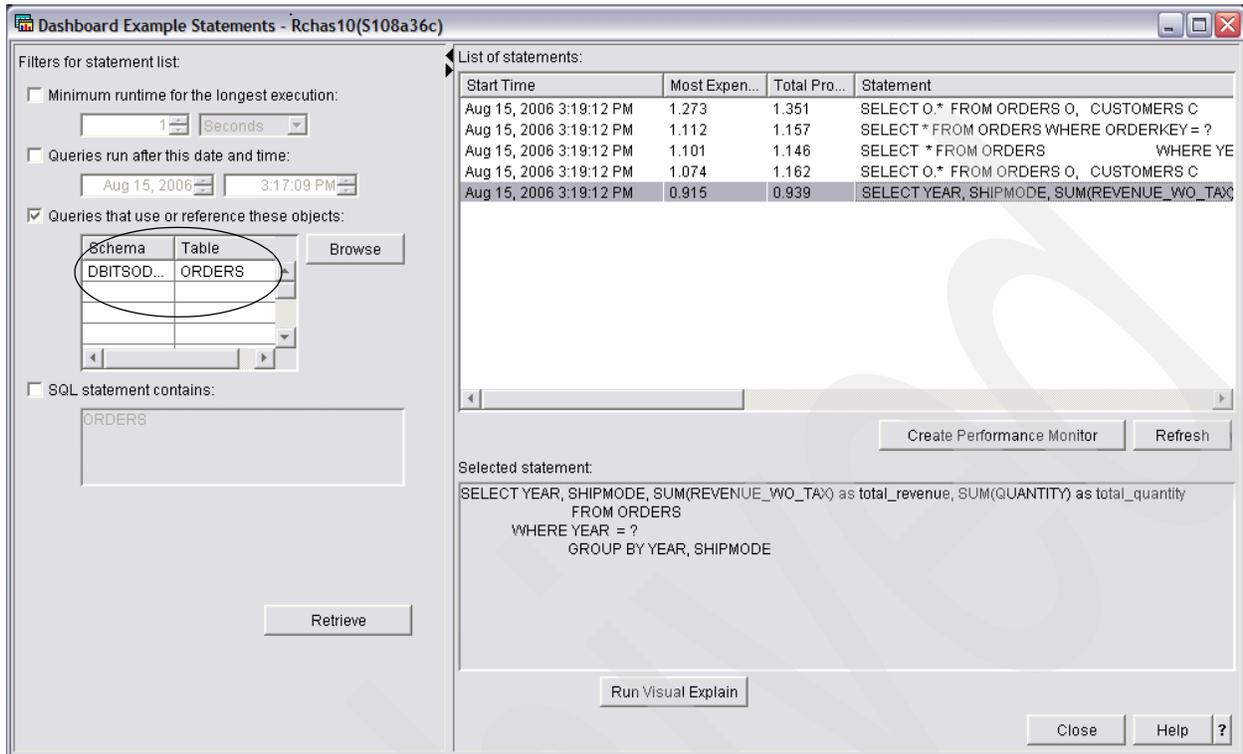


Figure 5-49 Statements filtered by user -specified criteria

Clicking any statement in the upper right pane will display the entire statement in the lower right pane.

From the List of Statements pane, you have the option to create a new SQL Performance Monitor from one of more of the statements listed. Select a single statement or hold down the shift key to select multiple statements, and then click the **Create Performance Monitor** button. Enter a new name and library for the performance monitor as shown in Figure 5-50.

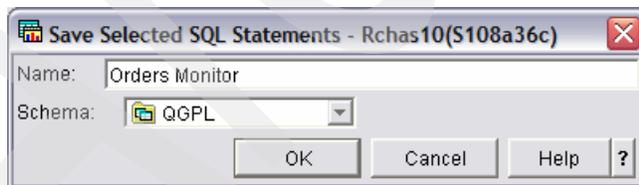


Figure 5-50 Creating a new performance monitor based on statements from original monitor

Information pertaining to the selected statements will be copied to a new performance monitor and the new monitor will automatically appear as an imported monitor in the list of SQL Performance Monitors available from the main iSeries Navigator window.

Tip: Use this option to create a subsetting performance monitor containing just the query or queries you are investigating. You can then send the smaller monitor file to your service representative rather than a possibly very large collection.

5.4.2 Launching Visual Explain

From the lower left pane, you can click Run Visual Explain to bring up the Visual Explain diagram of the query, which is shown in Figure 5-51.

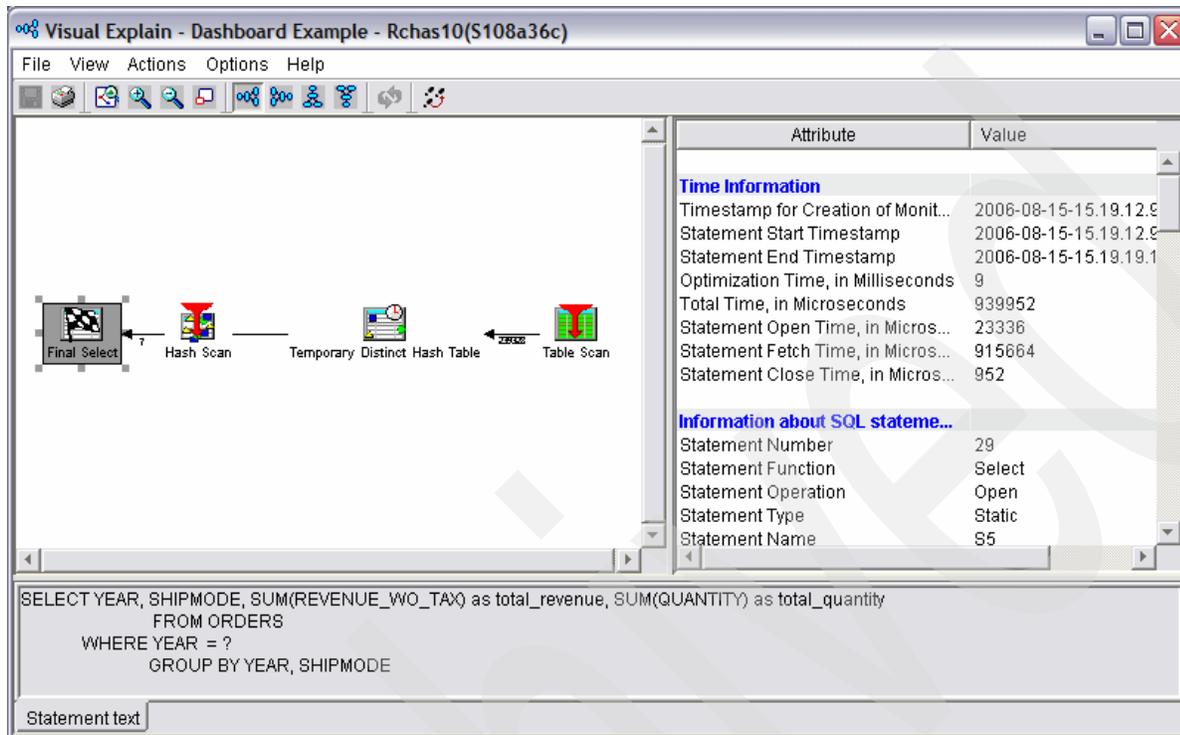


Figure 5-51 Visual Explain launched

You can now use the Visual Explain interface to further investigate your query. Refer to Chapter 8, “Analyzing database performance data with Visual Explain” on page 275 for a full discussion of the Visual Explain toolset.

5.5 Compare monitors

Another new addition to the V5R4 SQL Performance Monitor toolset is the Compare monitors feature. Using this feature, you can take two SQL Performance Monitors that have been collected against the same or a similar set of statements, and easily compare them on a statement by statement basis, based on several key factors. To access the Compare feature,

right click one or both of the monitors you wish to compare and select the Compare option, as shown in Figure 5-52.

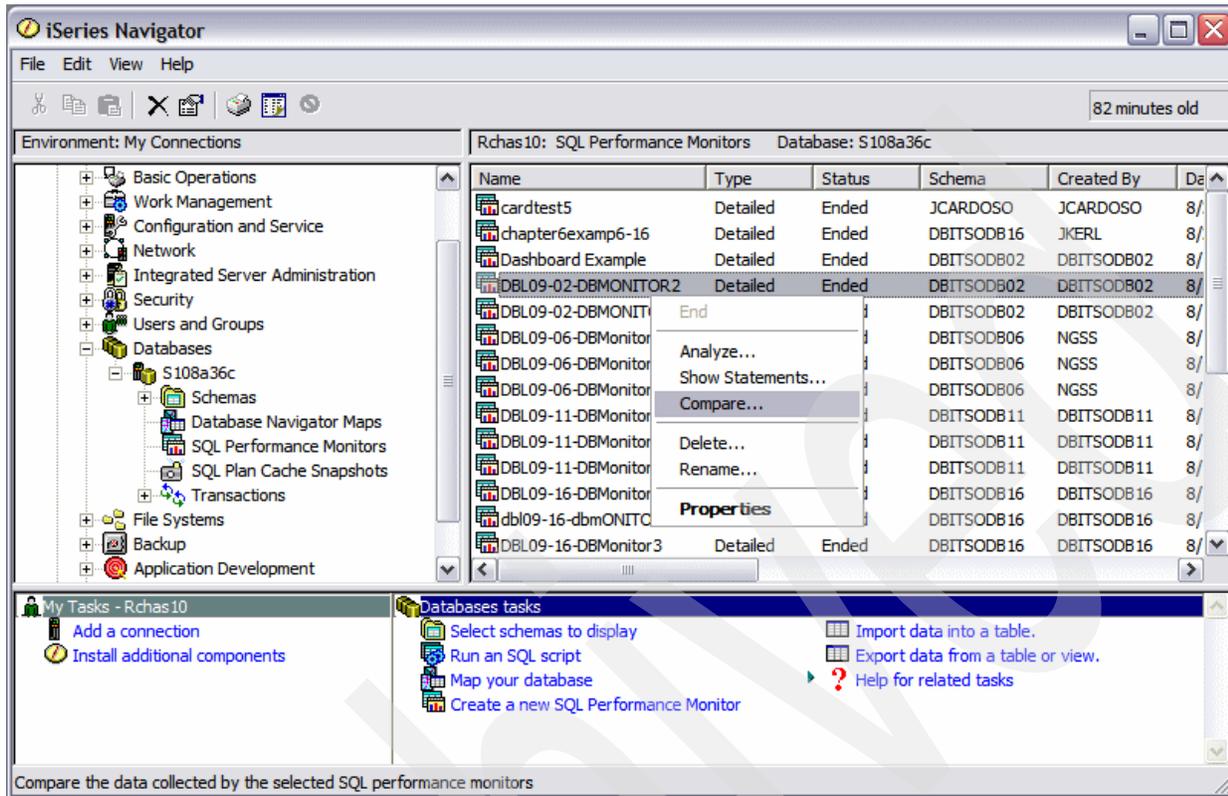


Figure 5-52 Selecting the compare option

You will be presented with a window where you can fill in the name of the second monitor file, if you had selected two initially, then both file names will be filled in as shown in Figure 5-53.

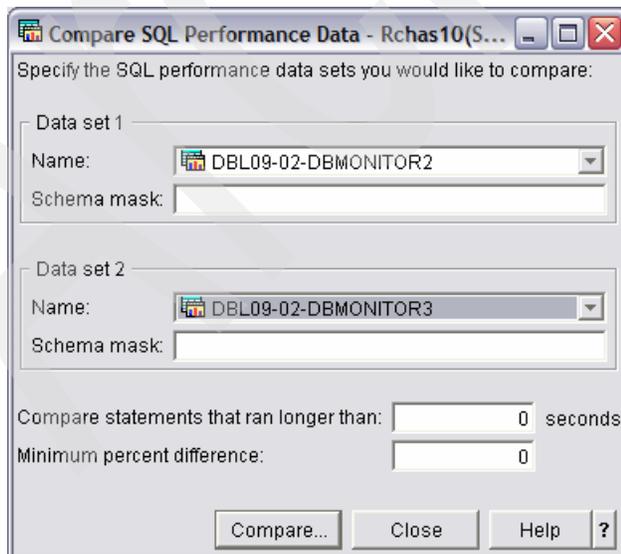


Figure 5-53 Enter names of SQL Performance Monitors to be compared.

You can set some additional criteria from this window as well:

- ▶ **Statement runtime filter:** By setting a value for the minimum runtime that should be considered for the comparison you can eliminate data for small queries which may not be of particular interest.
- ▶ **Minimum percent difference:** By setting a percentage here, the compare engine will only return information for queries where the percentage of difference for the particular comparison item exceeds the value entered.
- ▶ **Schema mask:** This convenient feature allows you to compare statements between two monitors where the SQL statements may be against tables in different schemas. For example, if you had one monitor over tables in TESTLIB and another monitor over tables in PRODLIB, you could compare them by setting TESTLIB as the schema mask for the first collection and PRODLIB as the schema mask for the second collection.

After choosing any of these settings, select the **Compare** button to have the tool begin the comparison. After the processing of the two monitor files has finished, you are presented with the initial window showing the statements that were found in each collection:



Figure 5-54 Matched statements

At the top of the window, you have the names of the first and second collections, on the left and right sides of the window, respectively. The number of unmatched statements is listed here as well. The above example shows that there was one statement from the first monitor file that was not found in the second monitor, but that there were no statements found in the second file that were not found in the first file. From this window you can all select the **Show Statements** button to enter the Show Statements interface that we looked at in the previous section.

Click the desired statement to expand the list of comparison data. Each comparison metric has three columns of information for the statement selected: the value from the first collection, the value from the second collection, and a percentage indicating the percent difference between the two (Figure 5-55).

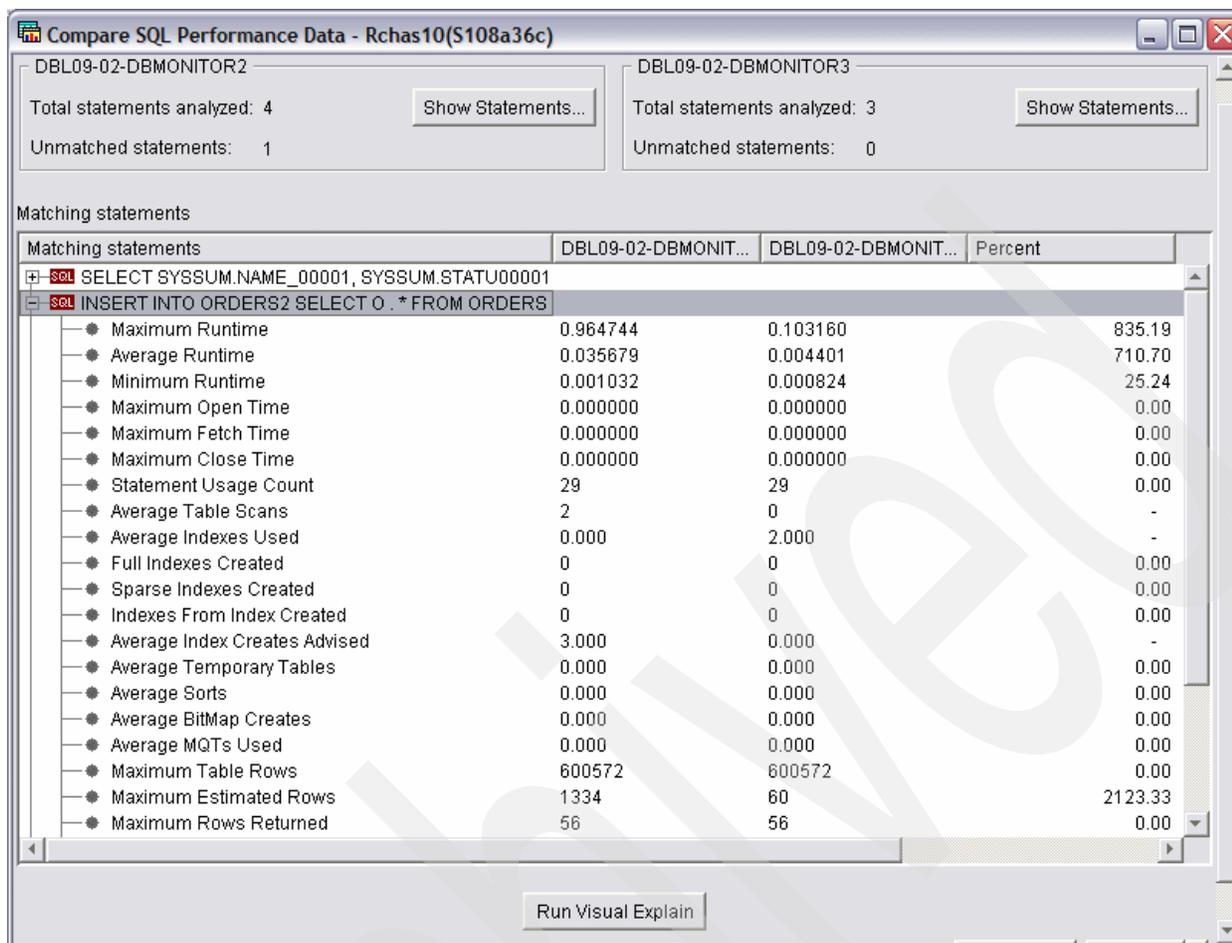


Figure 5-55 Comparison data expanded

From this window, you can get a quick overview of how the statement performed in one collection compared to the other. In the above example, you can quickly see the dramatic difference of over 700% for the runtime of the statement in question. Further review of the data shows that there was an average of two table scans used for the statement in the first collection and no table scans for the second, giving a big clue as to why the performance difference exists.

As seen throughout the new SQL performance monitor toolset, you can click Visual Explain to get the implementation in pictorial form and do further analysis. What makes the option unique here is that when you click the Visual Explain button at the bottom of the window, both

Visual Explain diagrams will be presented, stacked one on top of the other for easy comparison, as shown in Figure 5-56.

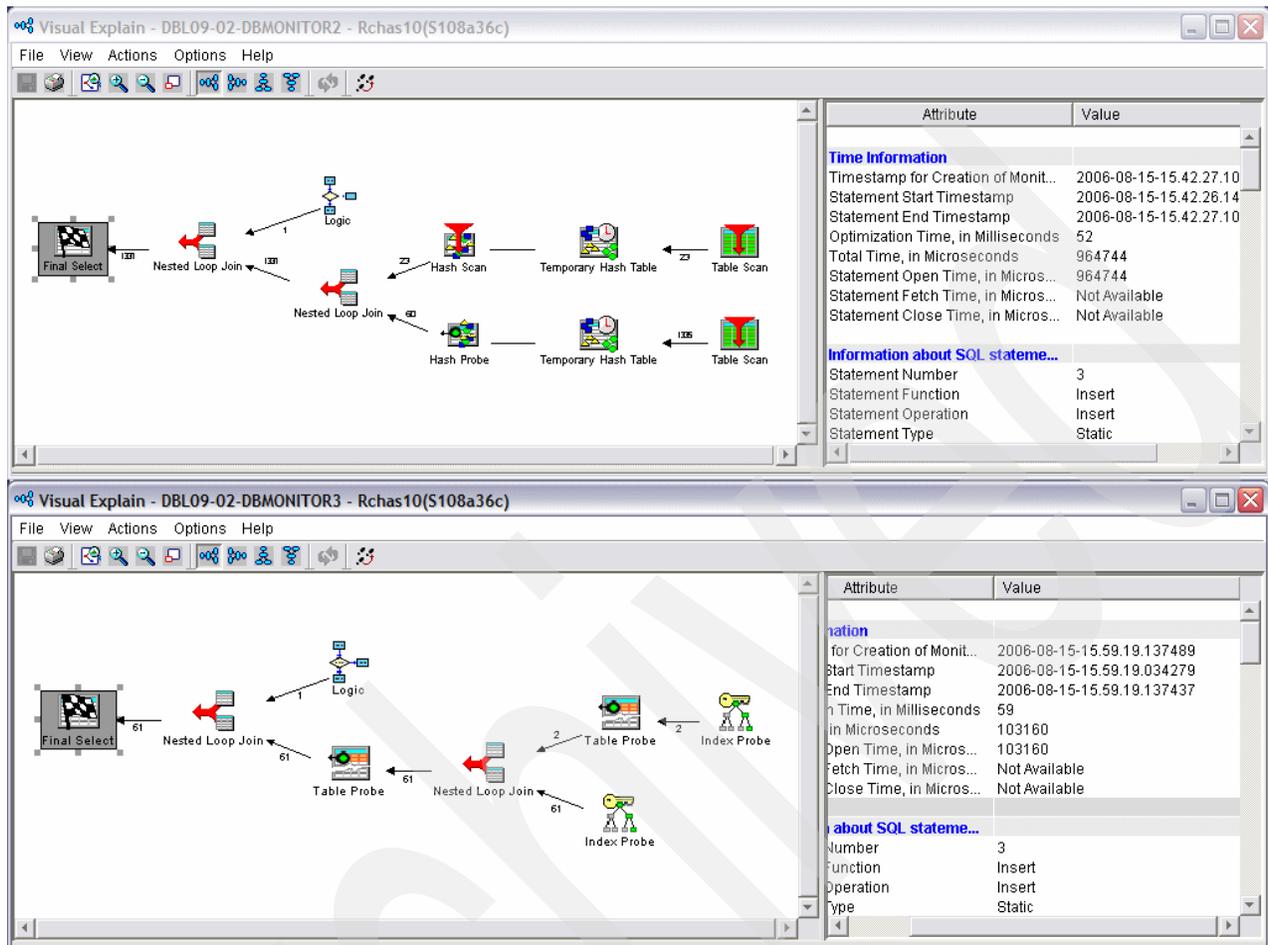


Figure 5-56 Double Visual Explain diagrams for easy comparison

With the comparison tool, you can start with two SQL Performance monitors, and in a few steps, find the statement representing the largest difference between the two collections and be presented with the Visual Explain diagrams showing the two implementations of that particular statement.

Note: It is important to note that since the Compare feature is a statement by statement comparison, it only works well if the monitors being compared have several of the same statements.

5.6 Case study

Now let us take a look at a practical example of using the new tools to investigate a query performance issue.

Tip: Before proceeding with the case study, review Chapter 8, “Analyzing database performance data with Visual Explain” on page 275 if you have not done so already.

5.6.1 A poor performing SQL statement

In this example, we have an SQL application which joins rows from a customer table and an order table. The application takes about a minute to read order information for 50 customers. The SQL statement being used in the application to retrieve the order information is shown in Example 5-1.

Example 5-1 Join operation from Customer table and Order table

```
SELECT O . *  
FROM ORDERS O , CUSTOMERS C  
WHERE O . CUSTKEY = C . CUSTKEY  
AND C . CUSTOMER = ?
```

We want to tune this query and improve its performance.

As the first step to any SQL performance analysis, a Detailed SQL Performance Monitor should be collected and analyzed using the Analysis Overview dashboard. So, we have done this and are now going to right-click the collection in the SQL Performance Monitor pane of the main iSeries Navigator window and select **Analyze** for our collection. Figure 5-57 shows the overview information in the dashboard for both the “Overview” folder and the “What implementations did the optimizer use?” folder.

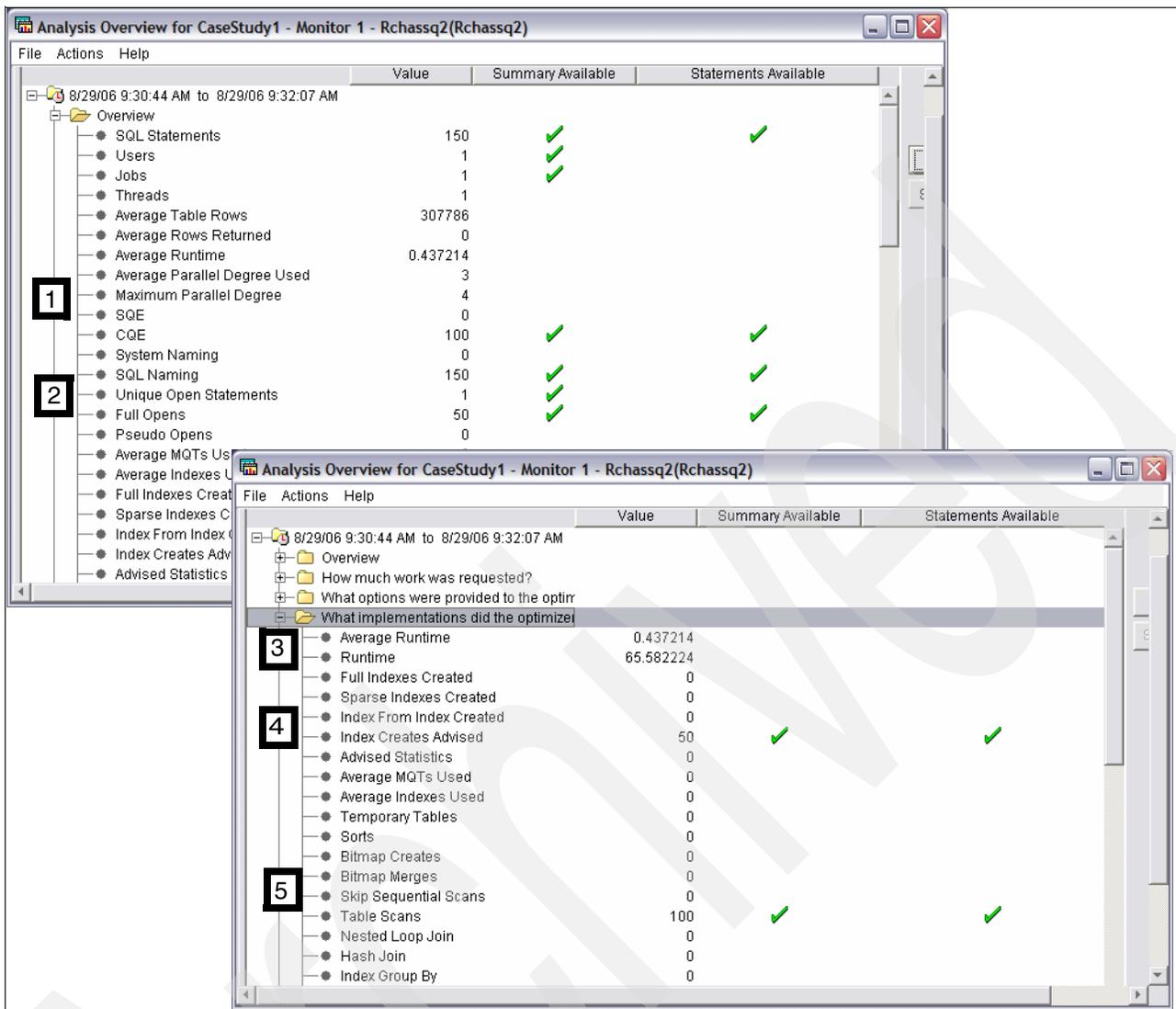


Figure 5-57 Analysis overview dashboard

A quick perusal of the dashboard overview and implementation information reveals several key focus areas:

1. The query is being optimized by CQE rather than by SQE.
2. The number of full opens is high, 50, compared to the number of unique open statements, just one.
3. Overall runtime of 65.6 seconds.
4. Indexes are being advised.
5. Tables scans are occurring.

Any one of the above items could be focused on to potentially improve performance. Let us explore them.

5.6.2 Why are table scans being done?

First we will investigate why table scans are occurring for this query. This could definitely be an area of improvement since we know we are not selecting most of the records in the tables, and in general table scans are best used when selecting a good portion of the table. Let's take a look at the Table Scan Statements report to see why a table scan is being done, Figure 5-58.

Archived

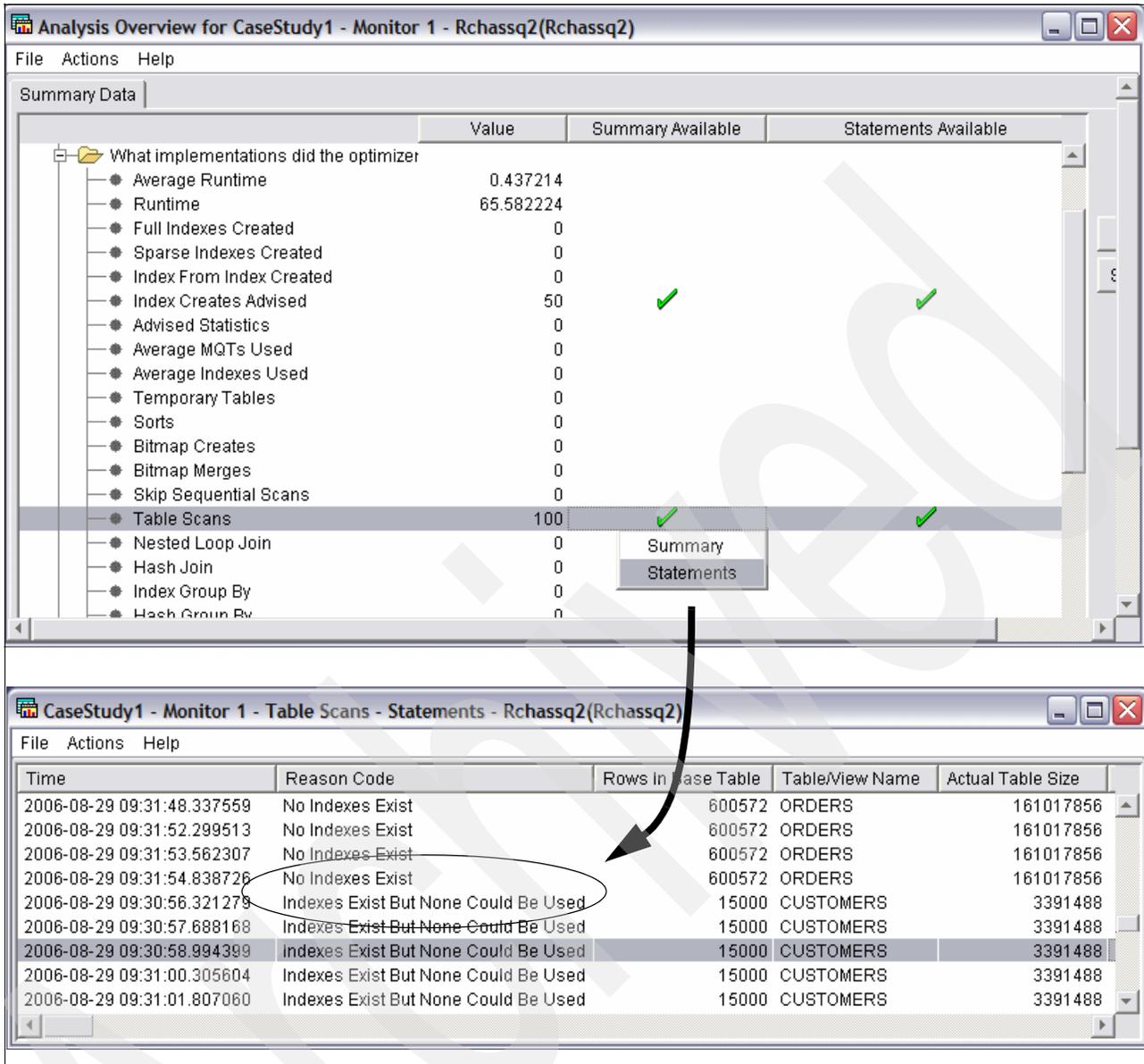


Figure 5-58 Table scan statement report

The report shows that table scans are done because no indexes exist over the ORDERS table and no suitable index exists over the CUSTOMERS table.

Recall that there is index advised information being returned for this query. Therefore, next we run the Index Creates Advised Statements report in order to see what the optimizer is recommending, Figure 5-59.

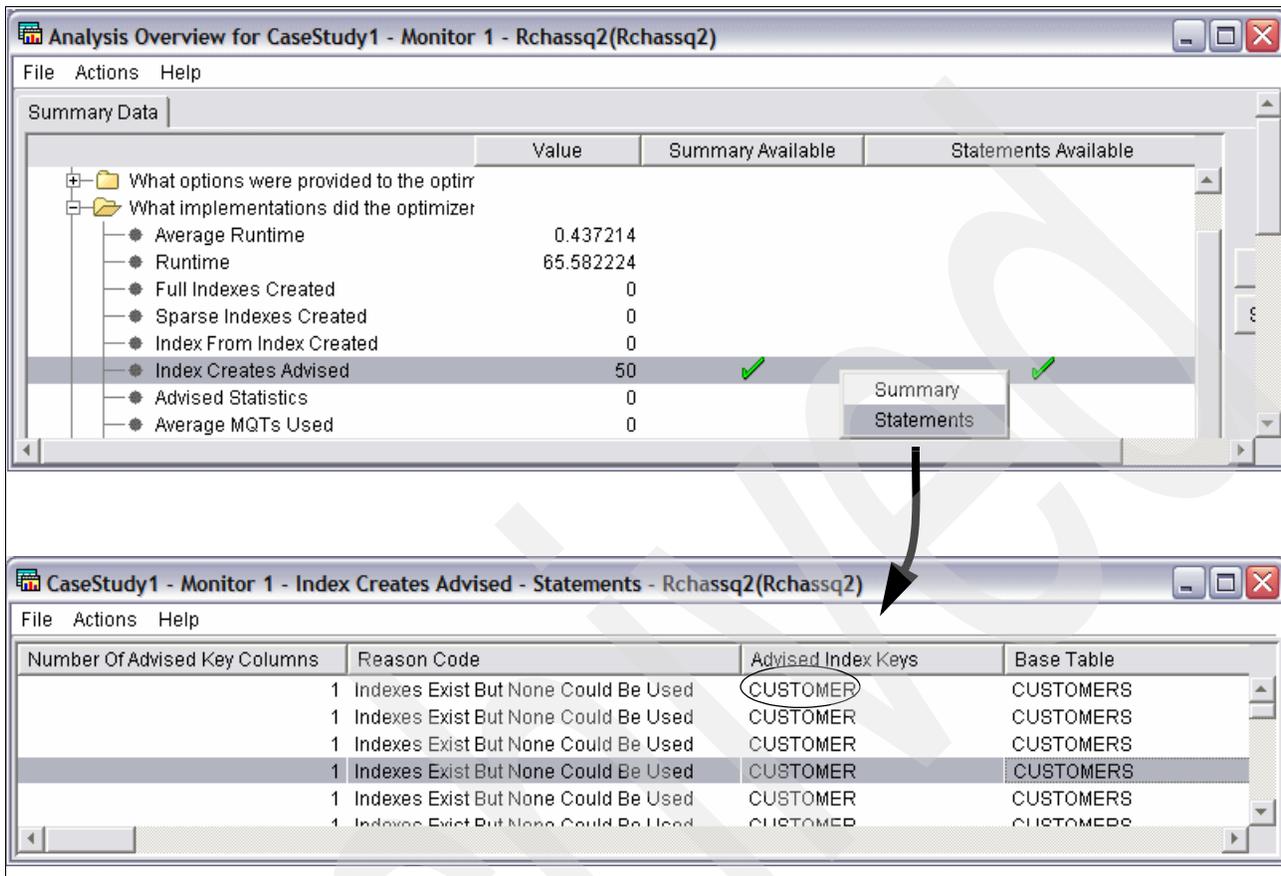


Figure 5-59 Index advised statement report

The optimizer is recommending an index over the CUSTOMERS table. But looking back to 3.6, “Index Advisor” on page 63, we know that CQE only advises indexes on local selection criteria. Its good indexing strategy to have indexes on the join criteria as well.

Refer back to the SQL statement in Example 5-1 on page 157. The perfect indexes for this query will cover both the local selection criteria and the join criteria. Therefore, we will want to create an index on CUSTOMERS keyed by CUSTOMER and CUSTKEY to cover the local selection and the join selection, and we will want an index on ORDERS keyed by CUSTKEY to cover its join selection.

Before we decide to create the indexes required for this SQL Statement let’s see the Visual Explain graph of this SQL statement. We can right-click one of the rows in the report, and because the resulting menu is context sensitive, we have the option Visual Explain the

statement as shown in the upper window of Figure 5-61. The Visual Explain graph for this SQL statement is illustrated in Figure 5-60.

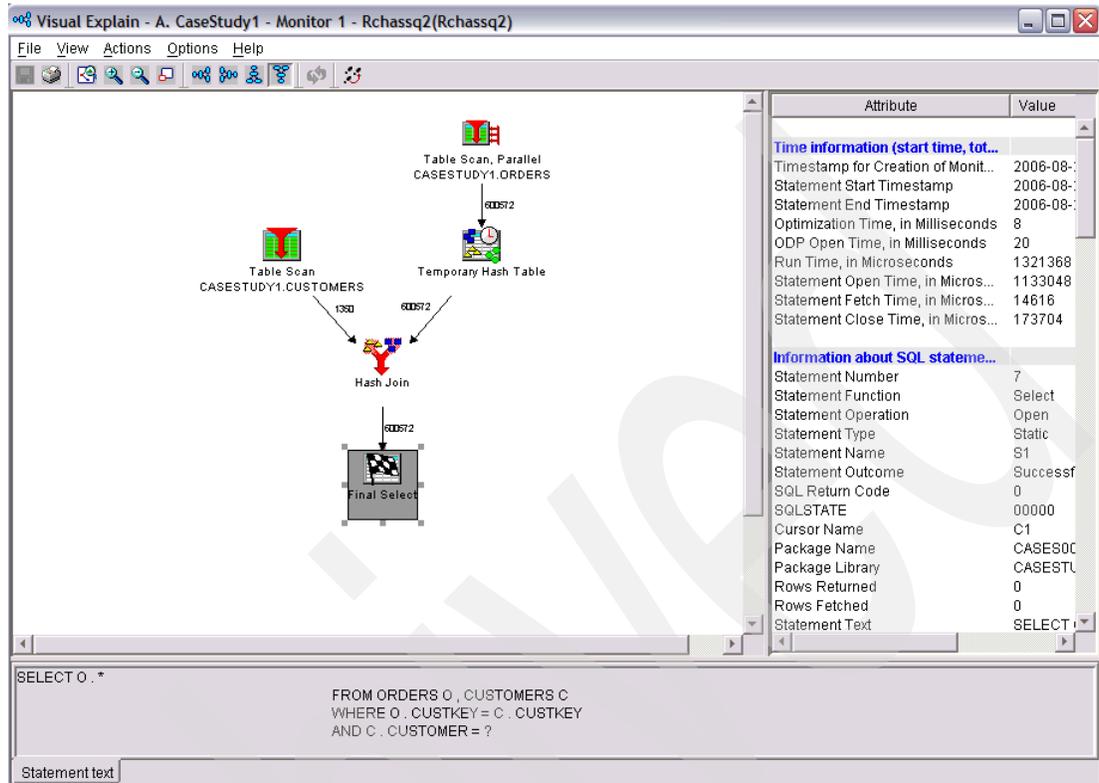


Figure 5-60 Visual Explain diagram without indexes

Now, let's proceed to create the required indexes for the SQL Statement. We can right-click one of the rows in the report, and because the resulting menu is context sensitive, we have the option to create the advised index right from the menu, Figure 5-61.

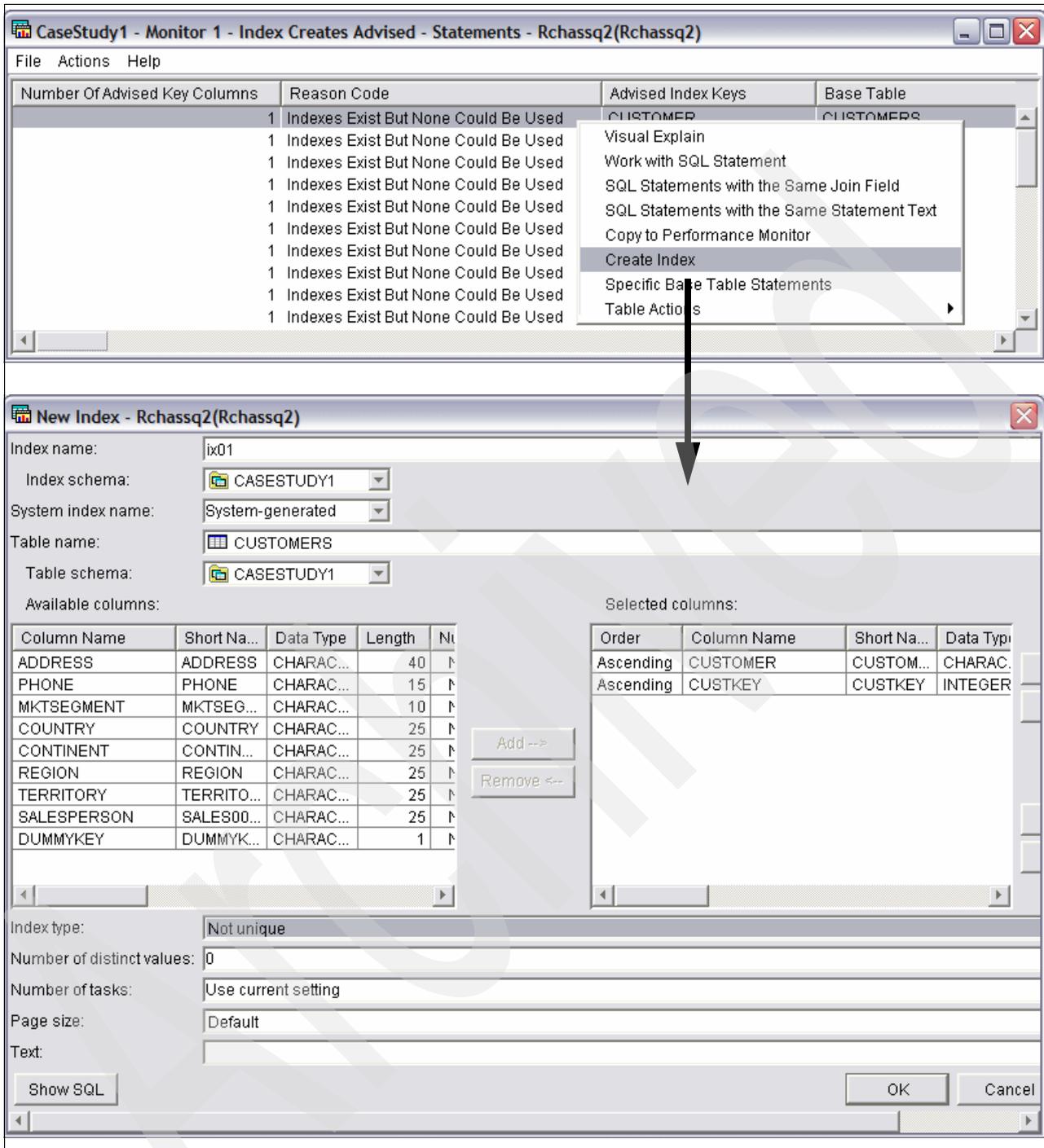


Figure 5-61 Creating an index right from the menu

The index create dialog is pre-populated with the advised index column, in this case, CUSTOMER. We selected CUSTKEY from the left pane and then clicked **Add** → to add it to the column list. We give the index a name, and the system creates the index when we click **OK**.

Remember, we also want to create an index over ORDERS to match its join criteria. We'll do that from the SQL Scripts window before testing the new indexes with Visual Explain. We

right-click the statement in any of the reports in which it appears and select **Work with SQL Statements** to bring it up in an SQL Scripts window.

In the SQL Scripts window that appears, already populated with the query, we will first create our second index, on ORDERS, as shown in Figure 5-62. We also have to replace the parameter marker in the local selection, represented by the question mark, with a real value.

Tip: If we do not know a real value to replace the parameter marker with, several of the Statement reports will give us this information. For example, in the “What types of SQL statements were requested?” folder, we can run the “Select Statements” statements report and check the “Variable Values” column.

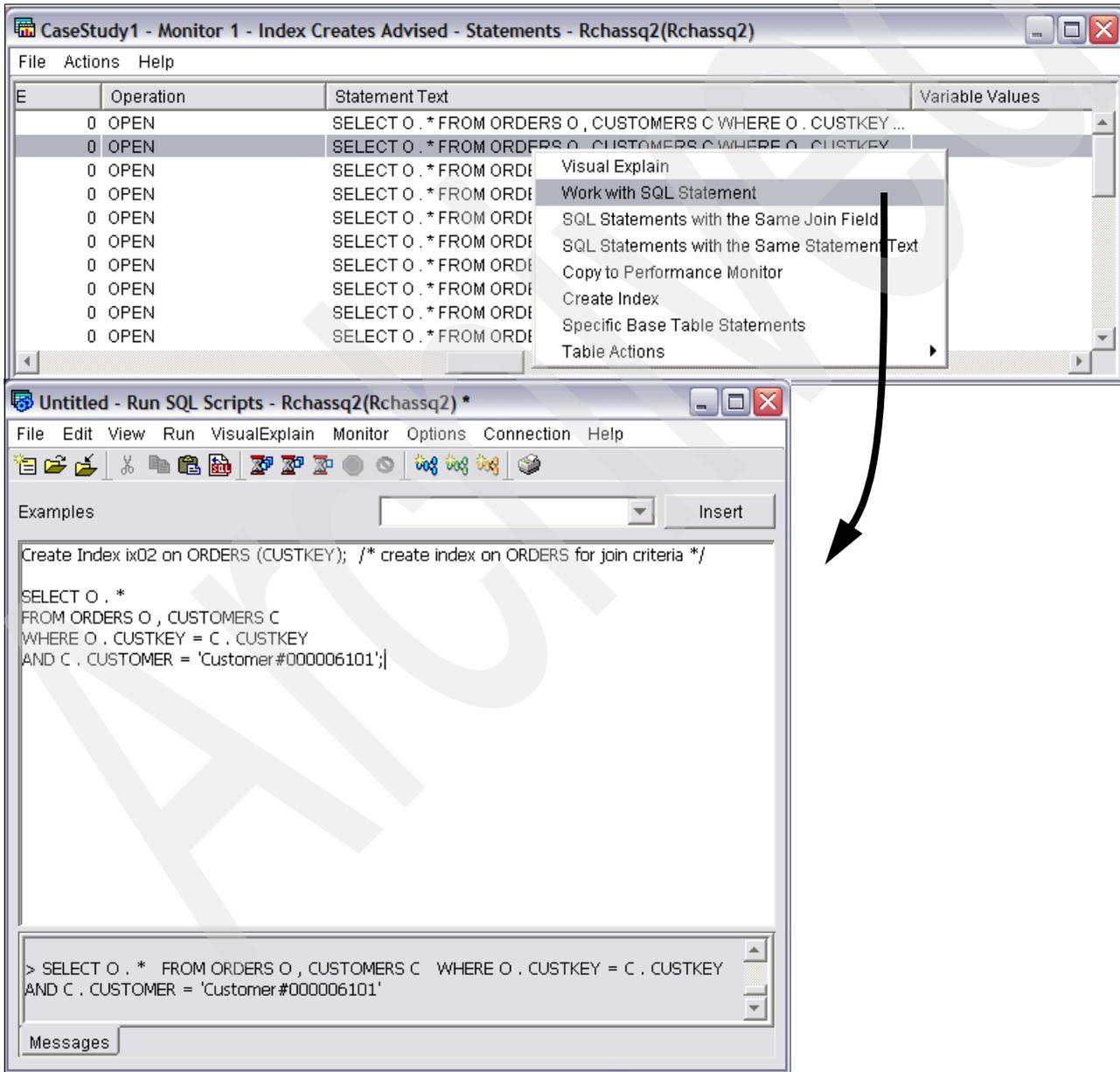


Figure 5-62 SQL Script to create an index and test the query

After the index is successfully created, we click the Visual Explain icon to get the implementation diagram shown in Figure 5-63. The diagram confirms that the indexes are now being used.

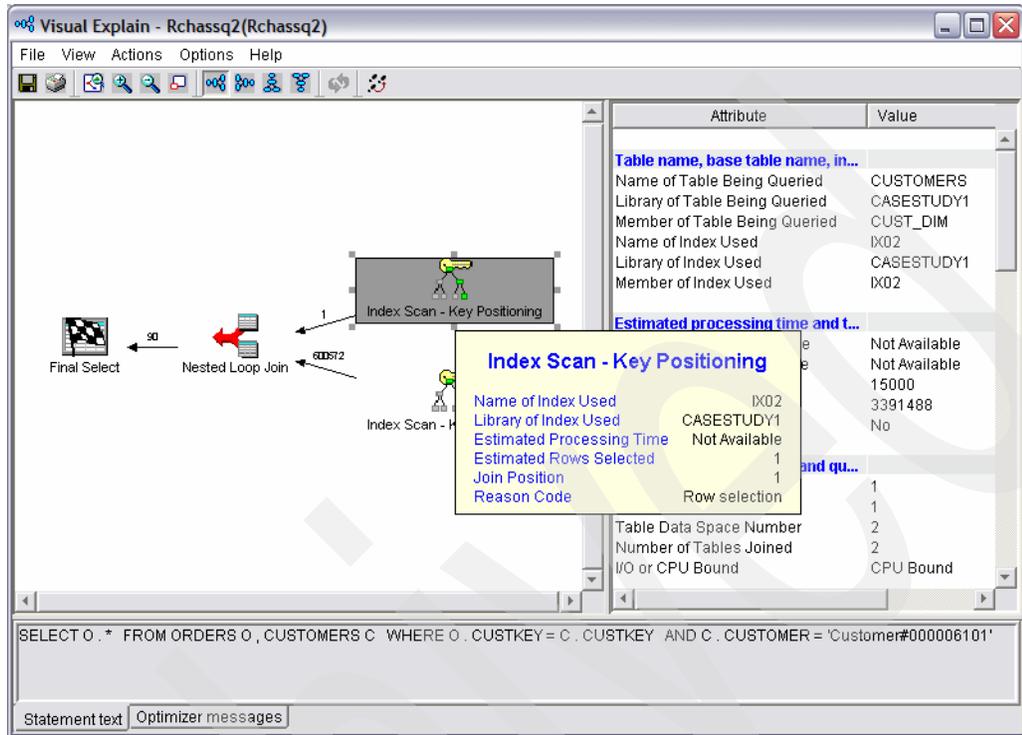


Figure 5-63 VE diagram showing the new indexes were used

The diagram shows that an index scan has been done over each file. By flying over the Index Scan icon with the pointer, we confirm that the indexes now used are those we just created.

Now that we have confirmed the indexes are being used, it is time to test our application to see if there is an improvement in run time. Therefore, we collect another SQL Performance Monitor and check the focus areas in the overview dashboard, Figure 5-64.

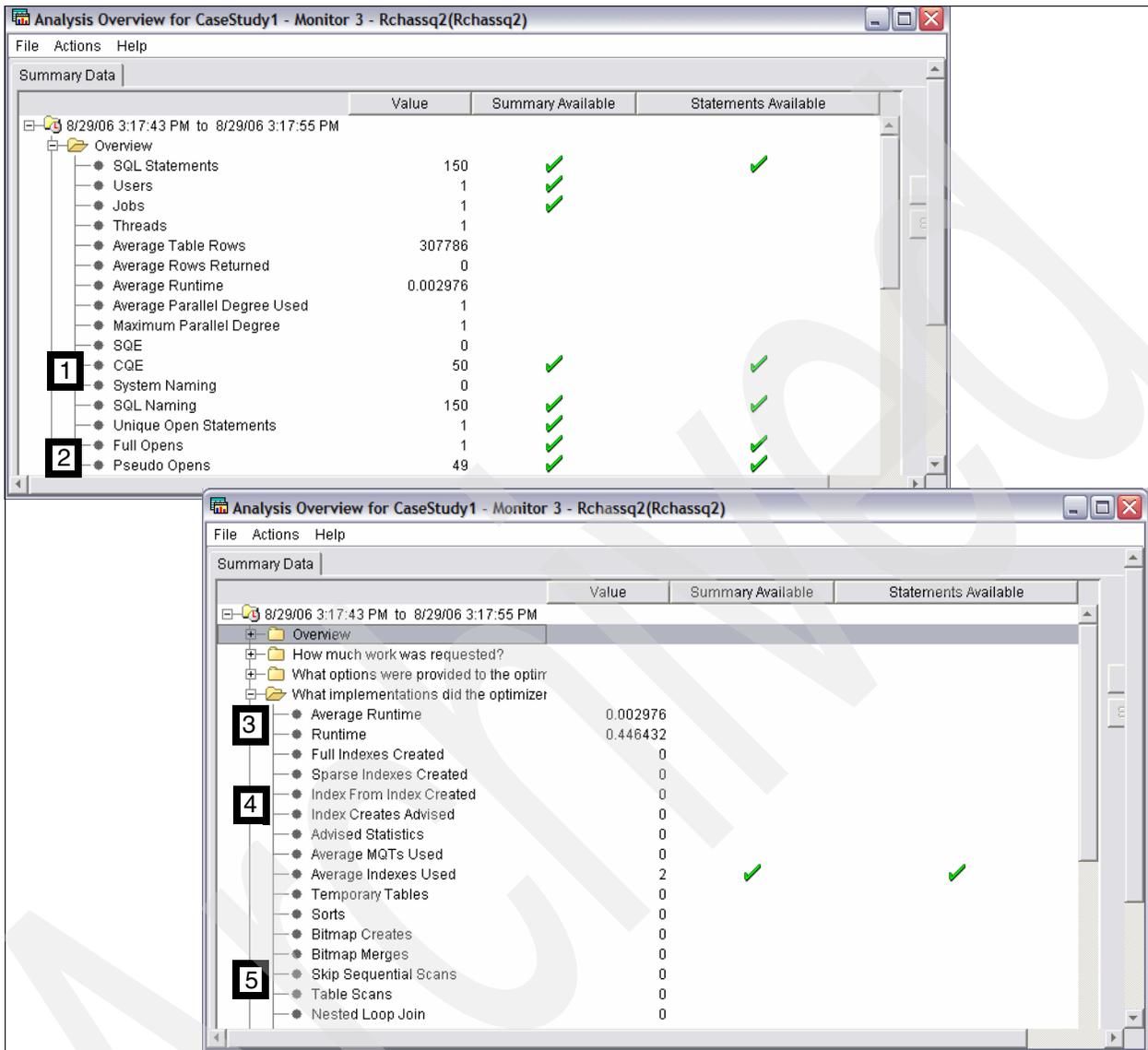


Figure 5-64 Overview analysis after indexes created

Note the following considerations:

- ▶ The query is still being optimized by CQE. In the next section, we will investigate that further.
- ▶ There is only one full open and 49 pseudo opens, indicating the open data path is now reusable. The reason why we now only have one full open is found by analyzing the two Visual Explain graphs shown in Figure 5-60 and Figure 5-63. The execution of the SQL Statement without indexes involves a hash join, and a hash table is created based on the particular host variable value for that particular execution, that hash table cannot be reused and therefore the cursor gets hard closed each time. On the other hand, this does not happen in the execution of the SQL statement with indexes.
- ▶ We also note the total runtime of 0.45 seconds, that is a better than a 100 times improvement over the original runtime.

- ▶ No indexes are being advised.
- ▶ Tables scans no longer occurring.

5.6.3 Why is CQE being used?

Since we know that SQE is the strategic direction for SQL on i5/OS, let us next try to determine why this query is being routed CQE. Although the runtime improvement already achieved by creating the indexes is impressive, perhaps we can improve it even further by ensuring that it is processed by the SQL Query Engine rather than the Classic Query Engine.

As a review, here are the SQE restrictions at V5R4. A query with any of the following attributes will be routed to CQE:

- ▶ Sort sequences
- ▶ UDTFs, UPPER & LOWER functions
- ▶ Logical file references
- ▶ Select/omit logical files
- ▶ Non-SQL interface
- ▶ Read triggers

To determine why our query is being routed to CQE, we will run the CQE Summary report as shown in Figure 5-65.

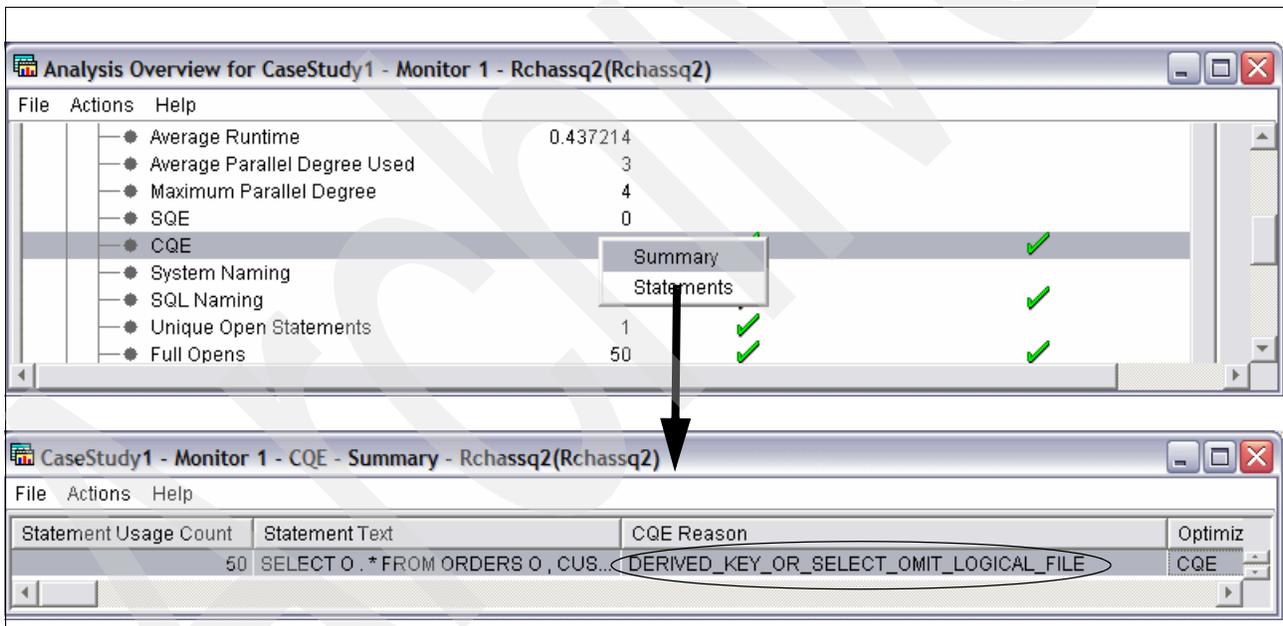


Figure 5-65 CQE summary report

The reason CQE is being used is due to DERIVED_KEY_OR_SELECT_OMIT_LOGICAL_FILE. A review of the logical files over the CUSTOMERS and ORDERS tables confirms that select/omit logicals exist. However, after doing some research, we learn that these select/omit logicals are from an old application and are no longer needed. Therefore, from another session, we delete the select/omit logical files. We can immediately test this change to see if we are now routing to SQE. We right-click the

record in the CQE report and select **Work with SQL Statement**. This brings up a SQL Scripts session, Figure 5-66, already populated with our statement.

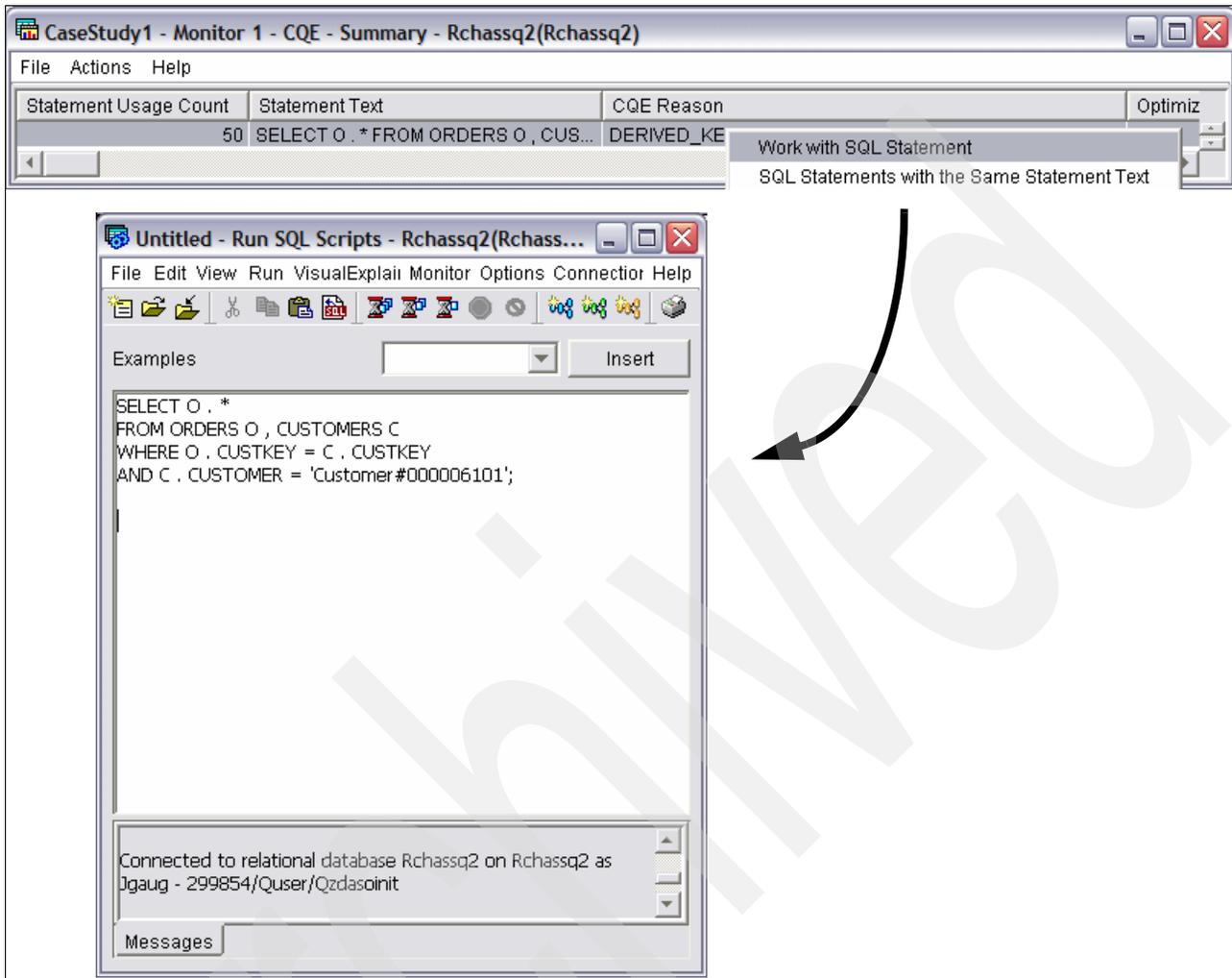


Figure 5-66 Work with the statement

From here, we can run Visual Explain and check the implementation to see if the query will now be routed through SQE. As in the last case, we must replace the parameter marker with a real value, as we have done in Figure 5-66. We then run Visual Explain.

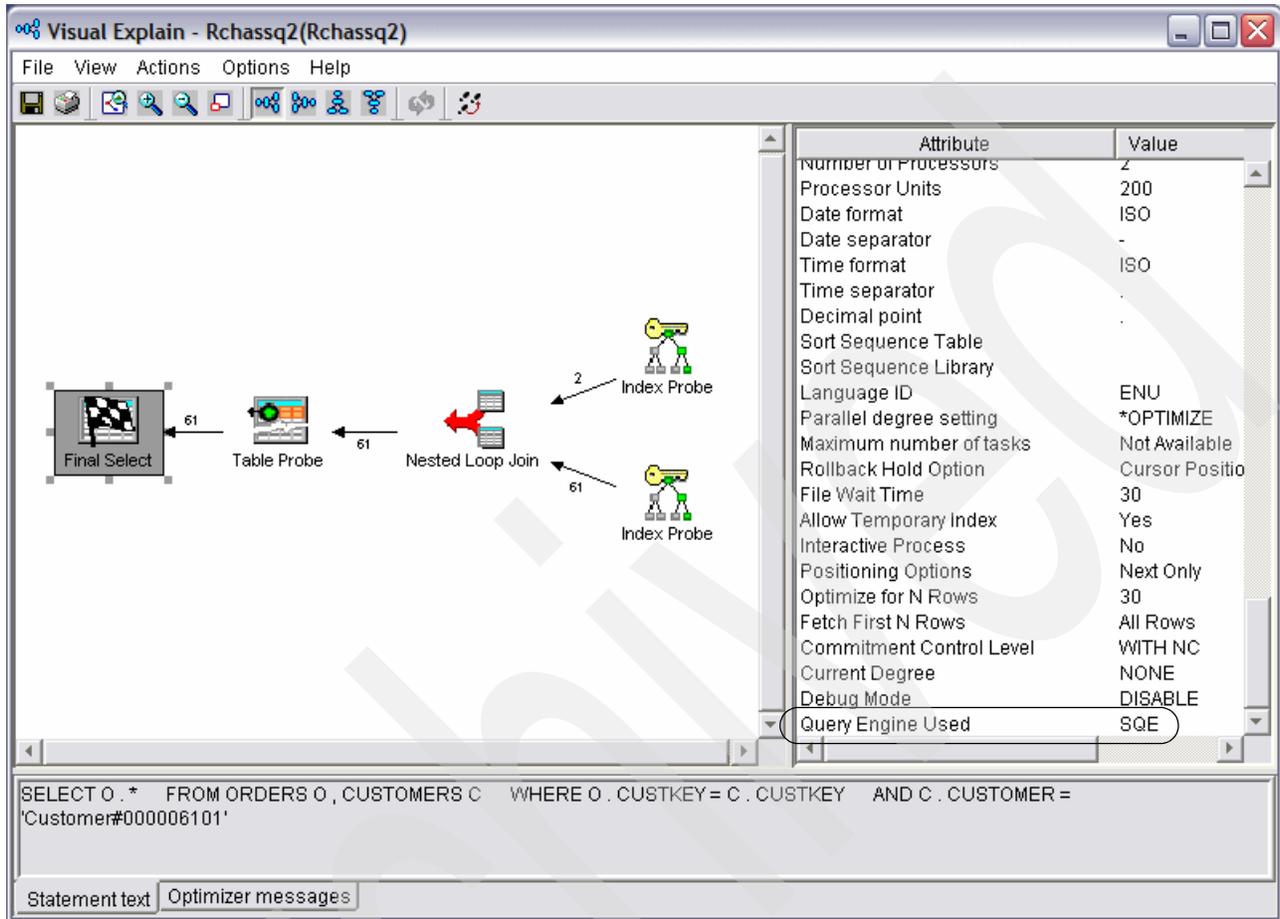


Figure 5-67 Visual Explain confirming SQE now used

We click the final select icon and then check the right pane under “Environment information for SQL”. There, we see that the query is now using SQE, Figure 5-67. So, let’s re-run the application now that we’ve eliminated the select/omit logical files and see how the SQE runtime compares to the CQE run time.

We again collect a SQL Performance monitor and we will check the focus areas in the dashboard (Figure 5-68).

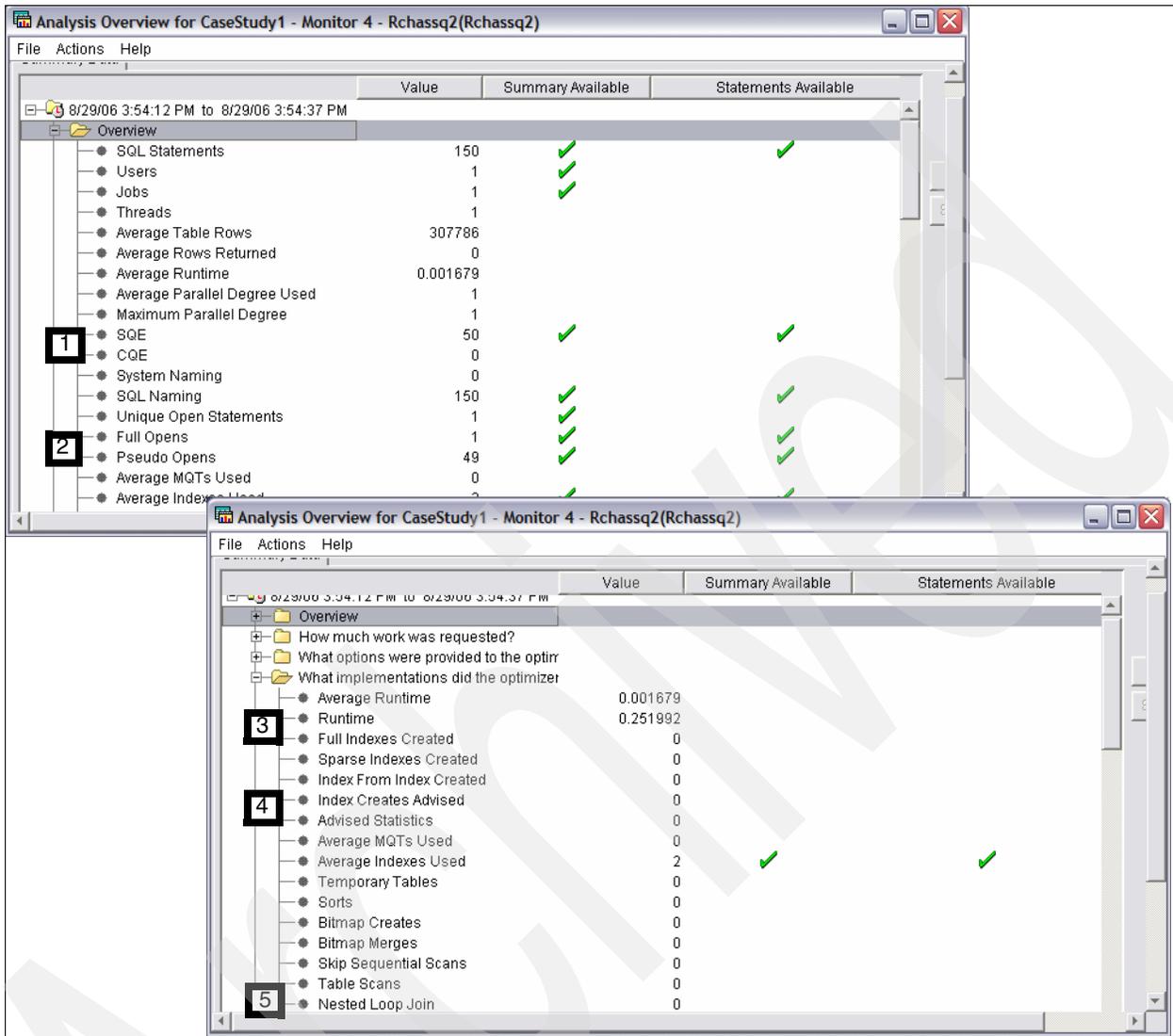


Figure 5-68 Analysis overview after eliminating select/omit logical files

The analysis confirms that:

- ▶ SQE is now being used.
- ▶ The ODP is still reusable as we only have one full open.
- ▶ Total runtime is again improved, now to only 0.25 seconds.
- ▶ There are still no indexes advised.
- ▶ There are still no table scans.

As a final confirmation of our work, let us run the original SQL Performance Monitor and the latest monitor through the comparison tool.

5.6.4 Comparison

For the comparison, we will select the two monitors from the SQL Performance Monitors representing the initial data collected and the final data collected, and then click **Compare**.

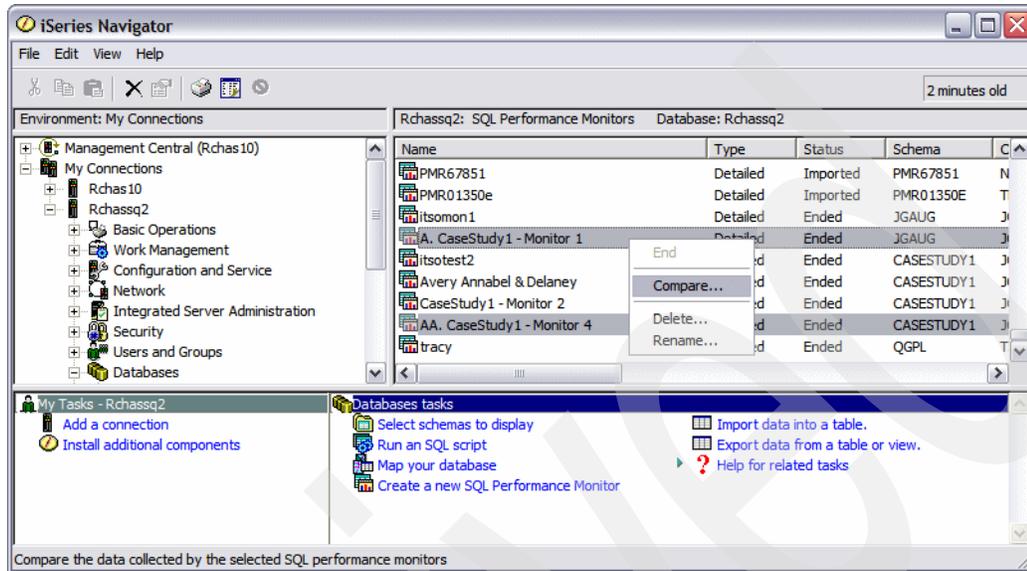


Figure 5-69 Select the two monitors to compare

From the comparison window, we can very quickly confirm the improvements.

Matching statements	A. CaseStudy1 - Monitor 1	AA. CaseStudy1 - Mon...	Percent
SQL SELECT O.* FROM ORDERS O, CUSTOMERS C WHI			
● Maximum Runtime	1.896104	0.234656	708.03
● Average Runtime	1.311644	0.005039	25929.84
● Minimum Runtime	1.257032	0.000304	413397.36
● Maximum Open Time	1.700120	0.234240	625.80
● Maximum Fetch Time	0.015328	0.000304	4942.10
● Maximum Close Time	0.239208	0.000128	186781.25
● Statement Usage Count	50	50	0.00
● Average Table Scans	2	0	-
● Average Indexes Used	0.000	2.000	-
● Full Indexes Created	0	0	0.00
● Sparse Indexes Created	0	0	0.00
● Indexes From Index Created	0	0	0.00
● Average Index Creates Advised	1.000	0.000	-
● Average Temporary Tables	0.000	0.000	0.00
● Average Sorts	0.000	0.000	0.00
● Average BitMap Creates	0.000	0.000	0.00
● Average MQTs Used	0.000	0.000	0.00
● Maximum Table Rows	600572	600572	0.00
● Maximum Estimated Rows	600572	60	1000853.33
● Maximum Rows Returned	0	0	0.00
● Average Table Rows	307786.000	307786.000	0.00
● Average Estimated Rows	300961.000	30.500	988657.37
● Average Rows Returned	0.000	0.000	0.00
● Optimizer Information	CQE	SQE	-
● Average Optimizer Time Outs	0.000	0.000	0.00
● Average Governor Time Outs	0.000	0.000	0.00

Figure 5-70 Side by side comparison

From this comparison, note the following:

- ▶ Average runtime reduced from 1.312 seconds to 0.005 seconds.
- ▶ Average table scans reduced from two to zero and average indexes used increased from zero to two.
- ▶ Average index creates advised decreased from one to zero.
- ▶ Average estimated rows reduced from about 300000 to 30.
- ▶ Statement went from CQE to SQE.

This completes our investigation. We have taken the application from a one minute run time to a quarter of a second with the addition of some indexes and by ensuring that it is routed to SQE. All this with the help of the new, easy-to-use SQL Performance Monitor tools.

Custom Database Monitor Analysis

In the previous chapters, we explained how the Detailed Database Monitor dumps all the performance data into one table. We also explained how the new iSeries Navigator graphical “dashboard” interface, *the Analyze Overview dashboard* discussed in Chapter 5, “Analyzing SQL performance data using iSeries Navigator” on page 117, can be used to identify and understand possible SQL performance problems. There are cases where you may require information that may not be provided by the graphical tools and that is when it is important to be able to understand how to query the performance data.

In this chapter, we explain the different records and columns in the table and how to directly query the Detailed Database Monitor data. This is useful because you can make your own queries that reduce the number of columns of information returned or specifically target in on an area that may require maneuvering through many levels in the graphical interface.

This chapter guides you in:

- ▶ Understanding the contents of the records of the Detailed Performance monitor table.
- ▶ Knowing which records to select when you have a question about a specific functions and features being used a database query.
- ▶ Determining what are the custom queries to use to locate an SQL query problem and some possible alternatives to use to resolve them.
- ▶ Providing a method of delving deeper into a particular SQL query problem uncovered by a high level analysis or the use of the iSeries Navigator GUI interface.

6.1 The Database Monitor record types

The Detailed Database Monitor collects different data and stores records in a single table in the order of occurrence. Within the Database Monitor table, each record contains a record type column. The Database Monitor uses the QQRID column to describe the type of information gathered in the particular record. 6.1.1, “Database Monitor record types” on page 174 shows the Database Monitor record types that are most often used for performance analysis. For a complete listing of all of the various records types and formats go to the V5R4 iSeries Information Center on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseres/v5r4/index.jsp>

On this Web page, perform the following steps:

1. Click **Printable PDFs and manuals** item on the iSeries Information Center menu.
2. Wait until the Printable PDFs and manuals window appears with the listing of all iSeries V5R4 manuals
3. Key performance and query optimization into the search field and press the **Search** button.
4. When the response is displayed, click the manual title *DB2 UDB for iSeries Database Performance and Query Optimization*.

Note: In the sections that follow, we identify the *columns* for each record type. The term “column” is also known as *field*.

6.1.1 Database Monitor record types

In this section, we list the Database Monitor record types that are most often used for performance analysis, as well as other record types. We also list the Global Database Monitor data columns and other columns that identify the tables or indexes used.

Record types most often used (QQRID value)

The following record types are most often used:

- ▶ 1000 Record: “The 1000 Record: SQL statement summary” on page 177)
- ▶ 3000 Record: “The 3000 Record: Arrival sequence (table scan)” on page 179
- ▶ 3001 Record: “The 3001 Record: using an existing index” on page 181
- ▶ 3002 Record: “The 3002 Record: temporary index created” on page 182
- ▶ 3003 Record: “The 3003 record: query sort” on page 183
- ▶ 3004 Record: “The 3004 record: temporary file” on page 184
- ▶ 3006 Record: “The 3006 record: access plan rebuild” on page 185
- ▶ 3007 Record: “The 3007 record: index evaluation” on page 186
- ▶ 3010 Record: “The 3010 record: host variables” on page 187
- ▶ 3014 Record: “The 3014 record: general query optimization information” on page 188
- ▶ 3015 Record: “The 3015 record: SQE statistics advised” on page 189
- ▶ 3020 Record: “The 3020 record: index advised (SQE)” on page 190

Database Monitor record types: other record types

The following record types are in the Database Monitor table but are not used in all the records:

- ▶ 3008 Record: Sub query processing
- ▶ 3018 Record: “The 3018 record: STRDBMON/ENDDDBMON” on page 189
- ▶ 3019 Record: “The 3019 record: rows retrieved detail” on page 189

- ▶ 3021 Record: Bitmap created
- ▶ 3022 Record: Bitmap merge
- ▶ 3023 Record: Temporal hash table created
- ▶ 3025 Record: DISTINCT processing
- ▶ 3026 Record: Set Processing (V5R4)
- ▶ 3027 Record: Subquery merge
- ▶ 3028 Record: Grouping
- ▶ 3030 Record: Query step processing
- ▶ 3031 Record: Recursive common table expression (V5R4)
- ▶ 5002 Record: Internal use only - SQE VE

Record types 3000 to 3008, 3014, and 3021 to 3028 occur during a full open and can be referred to as *optimization records*. Optimization records are much like debug messages. These records are necessary to determine the access plan for any given query in the Database Monitor data.

Global Database Monitor data columns

The following data columns are common to all record types:

- ▶ QQRID (record identifier): This column identifies the type of record.
- ▶ QQUCNT (unique number given for each query within a job): QQUCNT links together all Database Monitor records associated with all instances of a unique query within a job. The QQUCNT value assigned at full open time stays constant for all subsequent instances of that query. Non open data path (non-ODP) SQL operations (prepare, describe, commit) have QQUCNT = 0 and cannot be linked to a query. However, the QQ1000 column in the prepare or describe 1000 record contains the prepared SQL text.
This data column is not set for optimization records.
- ▶ Unique Query/Request Identifier - surrogate key
- ▶ QQJOB (job name)
- ▶ QQUSER (job user name)
- ▶ QVC102: This column refers to the CURRENT job user name.
- ▶ QQJNUM (job number): The job number is useful when multiple jobs are collected in one DB Monitor file.
- ▶ QQTIME (time at which the record was created): The time record can be useful when trying to determine which queries were running in a given time period.
- ▶ QQJFLD (join field): This column contains information that uniquely identifies a job and includes job name, job user name, and job number.
- ▶ QQ19: (thread identifier): This column might be useful for multi-threaded applications.

Figure 6-1 shows some of the Global Database Monitor columns.

	QQRID Record Type	QQUCNT Unique Counter	QQI5 Refresh Counter	QQJFLD Join Field	QQJOB Job Name	QQUSER User Name	QQJNUM Job Number	QQC21	QQ1000 Text Field
1	3018	0	-	AS27 QZDASOINITQUSER 022087	QZDASOINIT	QUSER	022087	-	
2	1000	0	0	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086	PD	- Lab 1 Task 3 exe select * from cust_c
3	1000	0	0	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086	DM	- Lab 1 Task 3 exe select * from cust_c
4	3000	3	-	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086		
5	3014	3	1	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086		
6	1000	3	0	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086	OP	- Lab 1 Task 3 exe select * from cust_c
7	1000	0	0	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086	DE	- Lab 1 Task 3 exe select * from cust_c
8	3019	3	0	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086	-	
9	1000	3	0	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086	FE	- Lab 1 Task 3 exe select * from cust_c
10	1000	3	0	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086	CL	CLOSE CRSR0002
11	1000	0	0	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086	PR	
12	1000	0	0	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086	PR	
13	1000	0	0	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086	PR	
14	1000	0	0	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086	PR	
15	1000	0	0	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086	PD	
16	1000	0	0	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086	DM	
17	3003	4	0	AS27 QZDASOINITQUSER 022086	QZDASOINIT	QUSER	022086		

Figure 6-1 Global Database Monitor columns

Tip: If you are going to use the Run SQL Script in iSeries Navigator, change the Java Database Connectivity (JDBC) setup to force translation for CCSID 65535, because the QQJFLD has been defined as FOR BIT DATA. Otherwise this column is shown in hexadecimal.

Reset the default value to use Visual Explain, otherwise it will fail.

Other columns that identify tables or indexes used

The following columns can also identify the tables or indexes that are used:

- ▶ **QQTLN:** Library of the table queried
- ▶ **QQTFN:** Name of the table queried
- ▶ **QQPTLN:** Base library
- ▶ **QQPTFN:** Base table
- ▶ **QQILNM:** Library of the index used
- ▶ **QQIFNM:** Name of the index used; *N when it is a temporary index
- ▶ **QVQTBL:** Queried table long name
- ▶ **QVQLIB:** Queried library long name
- ▶ **QVPTBL:** Base table long name
- ▶ **QVPLIB:** Base table library long name
- ▶ **QVINAM:** Index used long name
- ▶ **QVILIB:** Index library long name

How the data is organized in the Database Monitor table

The first occurrence of a unique query within the job always results in a full open. A *unique* query is one that requires a new ODP. SQL has determined that there is no existing ODP that can be used.

The presence of optimization records indicates a full open for an Open, Select into, Update, Delete, or Insert operation. Optimization records are immediately followed by SQL summary records (QQRID=1000) for that operation.

Subsequent occurrences of this query within the same job either run in reusable ODP or nonreusable ODP mode. Nonreusable mode is indicated by the presence of optimization records each time a particular query is run (full open). Reusable ODP mode is indicated by 3010 and 1000 records each time the given query is run (no optimization records or full open).

Linking query instances in the Database Monitor data

The data in the Database Monitor file is arranged chronologically. This organization can make it difficult to find all instances of a unique query. Use the QQJNUM, QQUSER, QQJOB, QQUCNT and QQI5 columns to view specific query instances. This is because QQUCNT and QQI5 columns link together all row types for a specific database request. The column QQUCNT is the surrogate key used to identify each unique query within a job. Be aware of the following situations:

- ▶ QQUCNT is assigned during a full open and is constant for all subsequent instances of a query.
- ▶ Non-ODP SQL operations (prepare, describe, commit) have QQUCNT = 0 and, therefore, cannot be linked to a query. If this is the case, you can use the QQ1000 column, which in the Prepare or Describe operation, contains the prepared SQL text.
- ▶ Non-ODP 1000 records (commit, prepare, and so on) have QQI5 = 0.

A full open occurs when there is an SQL update, insert, delete, or open operation and the QQI5 column is 0.

6.1.2 The 1000 Record: SQL statement summary

The 1000 record is the basic record type for any SQL query analysis. One record exists for each SQL operation (open, update, close, commit, and so on). The following columns are those that are most commonly used:

- ▶ **QQ1000:** Prepared text of SQL statement literals in the original SQL text might be replaced by parameter markers in prepared text if SQL was able to convert them during preparation (desired). For the original SQL text, use literal values from the matching 3010 record in the place of parameter markers or obtain the text from the step mode file using the QQSTIM time stamp from this record.
- ▶ **QQC21:** This column indicates the type of SQL operation (OP, FE, CL, UP, IN, DL, and so on). A value of FE indicates a Fetch summary record and not the actual number of fetch operations.

The ODP-related operation types are OP, IN, UP, DL, SI and SK.

- ▶ **QQI2:** Number of rows updated, inserted, or deleted
- ▶ **QQI3:** Number of rows fetched (only on FE rows)

This column indicates the actual number of rows fetched, and not the number of fetch attempts.

- ▶ **QQI6:** Elapsed time for this operation in microseconds

The time to fetch all rows might not be included in with Open and Select operations; you must look at the time on Fetch operation rows.

Access plan information

The following columns indicate information about the access plan:

- ▶ **QQC103** and **QQC104**: Package or program name and library
- ▶ **QVC18**: Dynamic SQL statement type
A value of E represents “extended dynamic”, a value of S represents “system wide” statement cache, and a value of L represents a prepared statement.
- ▶ **QQC22** and **QVC22**: Access plan rebuild code and subcode
This subcode is useful for IBM debug purposes.
- ▶ **QVC24**: Access plan save status
A value of *Ax* means that the access plan could not be saved. Values of *Bx* or a *blank* mean that the access plan was saved successfully.

ODP information

The following columns provide information about the ODP:

- ▶ **QQI5**: Query instance counter, where a value of 0 means that a full open occurred
- ▶ **QQC15**: Hard close reason code (for an HC operation type)
- ▶ **QVC12**: Pseudo open indicator
- ▶ **QVC13**: Pseudo close indicator
- ▶ **QQC181** and **QQC182**: Cursor and statement name

Example 6-1 shows a query with some of the most commonly used columns in the 1000 record.

Example 6-1 Common columns in the 1000 record

```
SELECT qqcnt AS "QQCNT Unique Counter"  
,qqc21 AS "QQC21 Statement Operation"  
,qgetim - qqstim AS "Elapsed Time"  
,qq1000 AS "QQ1000 Text"  
,qqi3 AS "QQI3 Fetched Rows"  
,qqc16 AS "QQC16 Data Conv."  
,qvc11 AS "QVC11 ALWCPYDTA"  
,QVC41 AS "QVC41 Cmt Control Lv1"  
,rrn(a)  
FROM mydbmon a  
WHERE qqrid = 1000  
ORDER BY rrn(a)  
OPTIMIZE FOR ALL ROWS;
```

Figure 6-2 shows the result of the query in Example 6-1. The first row shows that the Open operation took 5.917 milliseconds and only 0.561 milliseconds to fetch 6 rows. The query ran with ALWCPYDTA *OPTIMIZE and with commitment control set to *NONE. Also, there were no conversion problems when the query was run.

QQUCN...	QQC21 ...	Elapsed Time	QQ1000 Text	QQI3 ...	QQC16...	QVC11...	QVC41...
5OP		0.005917	SELECT * FROM QUSRBRM.QA1AAL FOR READ ONLY	00	0	UR	
5HC		0.000000	HARD CLOSE 1 CURSORS	00			
5FE		0.000561		6N	0	UR	
5CL		0.000561	CLOSE SQLCURSOR000000003	00	0	UR	
0PD		0.002050	SELECT * FROM QUSRBRM.QA1A1RMT WHERE R1...	00	0	UR	
6OP		0.006659	SELECT * FROM QUSRBRM.QA1A1RMT WHERE R1...	00	0	UR	
6HC		0.000000	HARD CLOSE 1 CURSORS	00			
6FE		0.000561		13	0	UR	
6CL		0.000561	CLOSE SQLCURSOR000000003	00	0	UR	
0PD		0.002366	SELECT * FROM QUSRBRM.QA1ASP WHERE SYNA...	00	0	UR	
7OP		0.008507	SELECT * FROM QUSRBRM.QA1ASP WHERE SYNA...	00	0	UR	
7HC		0.000000	HARD CLOSE 1 CURSORS	00			
7FE		0.000606		13	0	UR	
7CL		0.000606	CLOSE SQLCURSOR000000003	00	0	UR	
0PD		0.002272	SELECT * FROM QUSRBRM.QA1AAL WHERE ALSY...	00	0	UR	
8OP		0.008418	SELECT * FROM QUSRBRM.QA1AAL WHERE ALSY...	00	0	UR	
8HC		0.000000	HARD CLOSE 1 CURSORS	00			
8FE		0.000569		13	0	UR	
8CL		0.000569	CLOSE SQLCURSOR000000003	00	0	UR	
0PD		0.001912	SELECT * FROM QUSRBRM.QA1AAL	00	0	UR	
9OP		0.005894	SELECT * FROM QUSRBRM.QA1AAL	00	0	UR	
9HC		0.000000	HARD CLOSE 1 CURSORS	00			
9CL		0.000567	CLOSE SQLCURSOR000000003	00	0	UR	

Figure 6-2 Common columns in the 1000 record

6.1.3 The 30XX Records: Query Optimization Row Type

The 30XX record types are referred to as Optimization rows because:

- ▶ They contain information to construct the access plan for an SQL request
- ▶ The Presence of Optimization rows indicate that a Full Open occurred
- ▶ Data Access Method columns contain detail on the optimization process

QQEPT	Optimizer's estimated processing time
QQREST	Estimated number of rows selected
QQAJN	Estimated number of rows joined
QVPARD	Parallel degree requested
QVPARU	Parallel degree used

6.1.4 The 3000 Record: Arrival sequence (table scan)

The 3000 record points out queries in which the entire table is scanned. A table scan is generally acceptable in cases where a large portion of the table will be selected or the table has a small number of rows.

Table information

Table information is provided by the following columns:

- ▶ **QVQTBL** and **QVQLIB**: Table name and schema name respectively
- ▶ **QQPTFN** and **QQPTLN**: Short system table name and library name respectively
- ▶ **QQTOTR**: Number of rows in table

Query optimization details

The following columns provide details about query optimization:

QQRCD: Reason code, why table scan chosen

- ▶ T1 - No Indexes exist.
- ▶ T2 - Indexes exist, but none can be used.
- ▶ T3 - Optimizer choose table scans over available indexes

The query shown in Example 6-2 illustrates some of the most commonly used columns in the 3000 record.

Example 6-2 Common columns in the 3000 record

```
WITH xx AS (SELECT * FROM mydbmon WHERE qqrid = 3000),
yy AS
(SELECT qq1000 AS qqsttx
,qqjfld ,qqcnt ,qqc21 as qqop ,qqi4 as qqtt FROM mydbmon
WHERE qqrid = 1000
AND (qvc1c = 'Y' OR (qqc21 IN('DL', 'UP')
AND qqc181 <= ' ') OR qqc21 IN ('IN', 'IC', 'SK', 'SI') OR qqc21 LIKE '0%'))

SELECT qqop as "QOC21 Operation" ,qqtt as "QQI4 Total time (ms)" ,qqptln as
"QQPTLN Library" ,qqptfn as "QQPTFN Table" ,qqtotr as "QOTOTR Rows" ,qqr cod as
"QQRCD Reason" ,qqsttx as "QQ1000 Statement"
FROM xx a
LEFT JOIN yy b ON a.qqjfld = b.qqjfld AND a.qqcnt = b.qqcnt
ORDER BY a.qqjfld
DESC;
```

Figure 6-3 shows the result of the query in Example 6-2. The information from the 3000 record is joined with information from the 1000 record, to determine which queries did a table scan. Therefore, in a case where an index is not recommended, we can still look at the selection in the SQL text to see if a good index can be created.

QOC21 ...	QQI4 Tot...	QQPTLN Li...	QQPTFN T...	QQTOTR ...	QQRCD ...	QQIDXA Index Advis...	QQI2 Primary Ke...	QQIDXD Suggested Keys	QQ100
OP	12	QSYS2	SYSIXADV	327	T3	Y		1 TABLE_SCHEMA	select *1
OP	25	QSYS2	SYSIXADV	453	T3	N		0	
OP	6	DBITSODATA	ORDERS	600572	T3	Y		2 YEAR, MONTH	select i.y
OP	11	DBITSODATA	ORDERS	600572	T3	Y		2 MONTH, YEAR	select i.y
OP	8	DBITSODATA	ORDERS	600572	T3	Y		2 QUARTER, YEAR	select ye
OP	9	DBITSODATA	ORDERS	600572	T3	Y		2 YEAR, QUANTITY	select ye
OP	8	DBITSODATA	ORDERS	600572	T3	Y		2 YEAR, QUANTITY	select ye
OP	12	DBITSODATA	ORDERS	600572	T3	N		1 ORDERKEY	select di
OP	6	DBITSODATA	CUSTOMERS	15000	T1	N		1 MKTSEGMENT	select di
OP	7	DBITSODATA	ORDERS	600572	T1	N		1 SHIPMODE	select di
OP	58	DBITSODATA	ORDERS	600572	T1	N		1 RETURNFLAG	select di
OP	15	DBITSODATA	ORDERS	600572	T3	Y		3 RETURNFLAG, SUPPKEY, SHIPDATE	-- Query
OP	19	DBITSODATA	ORDERS	600572	T3	Y		3 RETURNFLAG, SUPPKEY, SHIPDATE	-- Query
OP	13	DBITSODATA	ORDERS	600572	T1	Y		2 SUPPKEY, SHIPDATE	-- Query
OP	13	DBITSODATA	ORDERS	600572	T3	Y		4 SHIPMODE, RETURNFLAG, SUPPK...	-- Query
OP	11	DBITSODATA	ORDERS	600572	T3	Y		3 SUPPKEY, RETURNFLAG, SHIPDATE	-- Query
OP	20	DBITSODATA	ORDERS	600572	T3	Y		3 RETURNFLAG, SUPPKEY, SHIPDATE	-- Query
OP	11	DBITSODATA	ORDERS	600572	T3	Y		2 SUPPKEY, SHIPDATE	-- Query
OP	25	DBITSODATA	ORDERS	600572	T3	Y		3 SHIPMODE, RETURNFLAG, SUPPKEY	-- Query
OP	11	DBITSODATA	ORDERS	600572	T3	Y		2 SUPPKEY, RETURNFLAG	-- Query
OP	9	DBITSODATA	ORDERS	600572	T3	Y		2 RETURNFLAG, SUPPKEY	-- Query
OP	7	DBITSODATA	ORDERS	600572	T3	Y		1 SUPPKEY	-- Query
OP	22	DBITSODATA	PARTS	20000	T3	Y		1 SIZE	select *

Figure 6-3 Common columns in the 3000 record

6.1.5 The 3001 Record: using an existing index

The 3001 record shows the index that will be used to access the table and why it was chosen. If the index was chosen for a join operation, additional information is given to help determine how the table “fits” in the join. The order of the 3001 records indicates the join order chosen by the optimizer.

Index and table information

The following columns provide index and table information:

- ▶ **QVINAM** and **QVILIB**: name of the chosen index and library
- ▶ **QVQTBL** and **QVQLIB**: name of the associated table and library

Query optimization details

The following columns indicate details about query optimization:

- ▶ **QRCOD**: Reason the index was selected
 - I1 Selection only
 - I2 Ordering or grouping
 - I3 Selection and ordering or grouping
 - I4 Nested loop join
 - I5 Record selection using bitmap
- ▶ **QQIDXA**: Index advised (Y or N)
- ▶ **QQI2**: Number of primary (key positioning) keys in QQIDXD column
- ▶ **QQIDXD**: Suggested keys for index (selection only)
- ▶ **QVC14**: Index only access indicator (Y or N)
- ▶ **QQIA**: Index page size

Note: A combination of the 3000 and 3001 records for a table indicates bitmap processing.

Example 6-3 shows a query with some of the most commonly used columns in the 3001 record.

Example 6-3 Common columns in the 3001 record

```
SELECT qqilnm as "QQILNM Library" ,qqifnm as "QQILFNM Table" ,qqtotr as "QQTOTR
Rows" ,qrcod as "QRCOD Reason" ,qqidxa as "QQIDXA Index Advised" ,qqi2 as
"QQI2 Primary Keys" ,qqidxd as "QQIDXD Suggested keys"
,qvc14 as "QCV41 Index Only"
,qqia as "QQIA Index page size"
FROM mydbmon a
WHERE qqrid = 3001
order by qqidxa desc;
```

Figure 6-4 shows the result of the query in Example 6-3. It shows the following information:

- ▶ The first four entries for IX_ORDERS were used to satisfy the join in the query (*reason I4*). However, the optimizer is advising the creation of an index with one primary key on the column YEAR.
- ▶ QA1ANET1 was used for ordering and grouping (*reason I2*). It also showed that this query was implemented using index only access (*QVC14 = 'Y'*).

- ▶ There are two entries, *NONE/MTI(Node_6, which refers to use of SQE temporary indexes.
- ▶ Some indexes are used for row selection (*reason II*).
- ▶ When DB2 is able to bypass accessing the table data, QVC14 is set to 'Y' to indicate that Index Access method is being used.

QQILNM Li...	QQIFNM Ta...	QQTOT...	QQR COD Reason	QQIDX A Index Advised	QQI2 Primary Keys	QQIDX D Suggest...	QVC14 Index Only	QQIA Index page size
DBITSODATA	IX_ORDERS	600572	14	Y		1 YEAR	N	65536
DBITSODATA	IX_ORDERS	600572	14	Y		1 YEAR	N	65536
DBITSODATA	IX_ORDERS	600572	14	Y		1 YEAR	N	65536
DBITSODATA	IX_ORDERS	600572	14	Y		1 YEAR	N	65536
QUSRBRM	QA1ANET2	2991	12	N		0	Y	8192
DBITSODATA	IX_ORDERS	600572	11	N		0	N	65536
DBITSODATA	IX_CUST2	15000	12	N		0	N	65536
DBITSODATA	IX_ORDERS	600572	11	N		0	N	65536
*NONE	MTI(Node_6	600572	11	N		0	N	65536
DBITSODATA	IX_ORDERS	600572	12	N		0	N	65536
DBITSODATA	IX_ORDERS	600572	12	N		0	N	65536
DBITSODATA	IX_ORDERS	600572	12	N		0	N	65536
DBITSODATA	IX_ORDERS	600572	11	N		0	N	65536
QUSRBRM	QA1ANET2	2991	12	N		0	Y	8192
QUSRBRM	QA1ANET2	2991	12	N		0	Y	8192
QUSRBRM	QA1ANET2	2991	12	N		0	Y	8192
DBITSODATA	IX_ORDERS	600572	11	N		0	N	65536
DBITSODATA	IX_CUST2	15000	12	N		0	N	65536
DBITSODATA	IX_ORDERS	600572	11	N		0	N	65536
*NONE	MTI(Node_6	600572	11	N		0	N	65536
DBITSODATA	IX_ORDERS	600572	12	N		0	N	65536
DBITSODATA	IX_ORDERS	600572	12	N		0	N	65536
DBITSODATA	IX ORDERS	600572	12	N		0	N	65536

Figure 6-4 Common columns from the 3001 record

6.1.6 The 3002 Record: temporary index created

The 3002 record shows instances in which the database optimizer decided that existing indexes are too costly or do not have the right key order for join, group by, or order by clauses. For this reason, the optimizer might decide to create a temporary index (Classic Query Engine (CQE)).

Index and table information

The following columns provide information about the index and table:

- ▶ **QQPTFN** and **QQPTLN**: Table name for which the index is built
- ▶ **OVINAM** and **QVILIB**: Temporary index name and library
- ▶ **QQR COD**: Reason why the index build was done
- ▶ **QQTOTR**: Number of rows in the table
- ▶ **QQRIDX**: Number of entries in the temporary index
- ▶ **QQ1000**: Name of the columns used for the index keys
 - Column names are the “short” column names.
 - QQ1000L can also be use.

Example 6-4 shows a query with some of the most commonly used columns in the 3002 record.

Example 6-4 Common columns in the 3002 record

```
SELECT qqcnt, qqc16 as "QQC16 Index Reused"
,qqptfn as "QQPTFN Table Name"
,qqptln as "QQPTLN Schema"
,qqtotr as "QQTOTR Rows"
,qqridx as "QQRIDX Entries in Temp Idx"
,qq1000 as "QQ1000 Key columns"
FROM mydbmon
WHERE QQRID = 3002
ORDER BY 6
Desc;
```

Figure 6-5 shows the result of the query in Example 6-4. It shows the following information:

- ▶ In the data sampled, we found the largest indexes, 600,572 entries, that were created for table ORDERS using SHIPDATE as the key field.
- ▶ The next largest index was for the table CUSTOMERS using MKTSEGMENT for 15,000 entries three times.
- ▶ There was an index built for the table SUPPLIERS using SUPPKEY for only 1,000 entries thirteen times.

QQUCNT	QQC16 Index Reused	QQPTFN Table Name	QQPTLN Schema	QQTOTR Rows	QQRIDX Entries in Temp Idx	QQ1000 Key columns
21	N	ORDERS	DBITSODATA	600572	600572	SHIPDATE ASC
96	N	ORDERS	DBITSODATA	600572	600572	SHIPDATE ASC
3	N	ORDERS	DBITSODATA	600572	600572	SHIPMODE ASC
3	N	ORDERS	DBITSODATA	600572	600572	SHIPDATE ASC
1	N	CUSTOMERS	DBITSODATA	15000	15000	MKTSEGMENT ASC
2	Y	CUSTOMERS	DBITSODATA	15000	15000	MKTSEGMENT ASC
3	N	CUSTOMERS	DBITSODATA	15000	15000	MKTSEGMENT ASC
174	N	DATES	DBITSODATA	1450	1450	YEAR ASC, MONTH ASC, DATEKEY ASC
176	N	DATES	DBITSODATA	1450	1450	YEAR ASC, MONTH ASC, DATEKEY ASC
37	N	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
38	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
39	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
40	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
112	N	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
113	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
114	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
115	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
174	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
176	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
20	N	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
21	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
22	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC

Figure 6-5 Common columns in the 3002 record

6.1.7 The 3003 record: query sort

The 3003 record shows that the database optimizer has decided to put selected rows into a temporary space and sort them. This is either cheaper than alternative indexing methods or an operation is forcing the optimizer to do so. An example is a UNION or an ORDER BY from the columns of several tables.

The following columns are the most commonly used:

- ▶ **QQSTIM:** Time stamp for the start of the refill and sort

- ▶ **QQETIM:** Time stamp for the end of the refill and sort
- ▶ **QQR COD:** Reason why a sort technique was chosen
 - **F1** - Query contains grouping columns (GROUP BY).
 - **F2** - Query contains ordering columns (ORDER BY).
 - **F3** - Grouping and ordering columns not compatible.
 - **F4** - DISTINCT was specified for the query.
 - **F5** - UNION was specified for the query.
 - **F6** - Query had to be implemented using sort.
 - **F7** - Query optimizer chose to use a sort rather than an index to order the results of query.
 - **F8** - Perform specified row selection to minimize I/O wait time.
 - **FC** - The query contains grouping fields and there is a read trigger.
- ▶ **QQI7:** Reason subcode for Union.
 - **51** - Query contains *UNION* and *ORDER BY*.
 - **52** - Query contains *UNION ALL*.
- ▶ **QQRSS:** The number of rows in a sort space

Keep in mind that sorting might increase the open time and cost since sorting is often performed at open time. If the number of rows sorted is small, then adding the right index might improve performance. Indexes can still be used to select or join records before the sort occurs. This does not indicate that the ODP is nonreusable.

The 1000 SQL summary record for the open might have a high elapsed time (QQI6 or QQI4). Sort buffers are refilled and sorted at open time, even in reusable ODP mode. However, high elapsed times might indicate a large answer set. In this case, the sort outperforms index usage (the situation in most cases).

6.1.8 The 3004 record: temporary file

The 3004 record shows that the database optimizer is forced to store intermediate results and rows in a temporary file because of the nature of the query. Examples are group by columns from more than one file or materializing view results.

The following columns are the most commonly used:

- ▶ **QQSTIM:** Time stamp for the start of a fill temporary results table
- ▶ **QQETIM:** Time stamp for the end of a fill temporary results table
- ▶ **QQTMPR:** Number of rows in a temporary table
- ▶ **QQR COD:** Reason for building temporary index

Example 6-5 shows a query with some of the most commonly used columns in the 3004 record.

Example 6-5 Common columns in the 3004 record

```

WITH xx AS
(SELECT * FROM mydbmon WHERE qqrid = 3004),
yy AS
(SELECT qq1000 AS qqsttx, qqjfld, qqcnt FROM mydbmon WHERE qqrid = 1000 AND
(qvc1c = 'Y' OR (qqc21 IN ('DL', 'UP'))
AND qqc181 <= ' ') OR qqc21 IN ('IN', 'IC', 'SK', 'SI') OR qqc21 LIKE '0%'))

```

```

SELECT qqstim as "Start timestamp" ,qgetim as "End timestamp",
DECIMAL((DAY(qgetim-qqstim)*24*3600)+
(HOUR(qgetim-qqstim)*3600)+(MINUTE(qgetim-qqstim)*60)+
(SECOND(qgetim-qqstim))+MICROSECOND(qgetim-qqstim)*.000001),18,3) AS "Temp.
population time" ,qrcod as "Reason" ,qqtmpr as "Rows in temp" ,qqstx as "SQL
text"
FROM xx a LEFT JOIN yy b ON a.qqjfld = b.qqjfld AND a.qqcunt = b.qqcunt
ORDER BY "Temp. population time" DESC;

```

Figure 6-6 shows the result of the query in Example 6-5. The information from the 3004 record is joined with information from the 1000 record so we know which queries were implemented using a temporary result file. The reason in the example is that the query contains a join condition that requires a temporary table (reason FA).

Start timestamp	End timestamp	Temp. population time	Reason	Rows in temp	SQL text
2005-10-01 16:16:42.722480	2005-10-01 16:17:44.837064	62.114	FA	1183	-- VLDB08select c.continent, c.c...
2005-10-01 16:19:42.244368	2005-10-01 16:20:44.038904	61.794	FA	1183	-- VLDB08select c.continent, c.c...
2005-10-01 16:19:28.438336	2005-10-01 16:19:28.881320	0.442	FA	15551	-- VLDB17select t.year, t.month, t...

Figure 6-6 Common columns in the 3004 record

6.1.9 The 3006 record: access plan rebuild

The 3006 record is *not* present on every full open. It is generated only when an access plan previously existed and has to be rebuilt. Also, the 3006 record is not generated when SQE switches between cached access plans (up to three) in the SQE plan cache for an SQL statement.

The following columns are the most commonly used:

- ▶ **QRCOD**: Rebuild reason code
- ▶ **QCC21**: Rebuild reason code subtype (IBM debug purposes)
- ▶ **QVC22**: Previous rebuild code
- ▶ **QVC23**: Previous rebuild code subtype
- ▶ **QCC11**: Plan required optimization

Y Plan had to be reoptimized and rebuilt

N QAQQINI Reoptimize option prevent access plan rebuild

The 1000 row contains an indicator regarding whether the rebuilt access plan can be saved (QVC24).

Example 6-6 shows a query with some of the most commonly used columns in the 3006 record.

Example 6-6 Common columns in the 3006 record

```

WITH xx AS
(SELECT * FROM qqpl.qzg0000478 WHERE qqrid = 3006),
yy AS
(SELECT qq1000 AS qqstx, qqstim, qgetim, qqjfld, qqcunt FROM qqpl.qzg0000478
WHERE qqrid = 1000 AND qqc21 <> 'MT'
AND (qvc1c = 'Y' OR (qqc21 IN('DL', 'UP')
AND qqc181 <= ' ') OR qqc21 IN ('IN', 'IC', 'SK', 'SI') OR qqc21 LIKE '0%'))

```

```

SELECT  qqttime AS "Time", qqtim1 AS "Last Access Plan Rebuilt", qqcl1 AS
"Optimization req.",
qqrcod AS "AP Rebuild Reason", HEX(qqc21) AS "Rebuild Subcode", HEX(qvc22) AS
"Original Rebuild Reason", HEX(qvc23) AS "Original Rebuild Subcode",
varchar(qqsttx,20000) AS "Statement Text"
FROM xx a LEFT JOIN yy b ON a.qqjfld = b.qqjfld AND a.qqucnt = b.qqucnt WHERE
(a.qqttime BETWEEN b.qqstim AND b.qqetim OR b.qqstim IS NULL) ORDER BY "Time";

```

Figure 6-7 show the result of the query in Example 6-6. The information from the 3006 record is joined with information from the 1000 record, so we know which queries had access plans rebuilt. The example shows three reasons:

- ▶ The access plan was rebuilt because of system programming changes (reason A7).
- ▶ The storage pool changed, or the DEGREE parameter of CHGQRYA command changed (reason AB).
- ▶ This is the first run of the query after a Prepare operation. That is, it is the first run with real actual parameter marker values.

Time	Last Access Plan Rebuilt	Optimization req.	AP Rebuild Reason	Rebuild Subcode	Original Rebuild Reason	Original Rebuild Subcode	Statement Text
2005-09-25 22:26:46...	2005-09-25 22:26:46.825712	Y	A7	0806	0000	0000	update item_fact i set EXPANDER...
2005-09-25 22:32:49...	2005-09-25 22:26:46.675132	Y	AB	0009	0000	0000	select c.continent, c.country, c.region...
2005-09-26 00:25:45...	2005-09-25 22:26:46.926264	Y	AB	0009	0000	0000	SELECT AL1.CUSTOMER, AL5.MON...
2005-09-26 00:25:48...	2005-09-26 00:25:47.970400	Y	A7	0806	0000	0000	SELECT AL1.CUSTOMER, AL5.MON...
2005-10-01 16:16:25...	2005-10-01 16:16:25.140992	Y	B3	0802	0000	0000	-- VLDB07select c.continent, c.countr...
2005-10-01 16:16:39...	2005-10-01 16:16:39.039512	Y	A7	0806	0000	0000	-- VLDB08select c.continent, c.countr...

Figure 6-7 Common columns in the 3006 record

6.1.10 The 3007 record: index evaluation

The 3007 record shows all indexes that are evaluated for a given table, including which (if any) were selected for use in this query and which were not and why. Reason codes are listed next to each index. A reason code of 0 indicates that the index was selected.

Documentation classifies this record as the “Optimizer time out” even though a 3007 row will be generated when the optimizer does not time-out. In CQE indexes are evaluated in order from newest to oldest, in the same order as shown by the DSPDBR CL command, excluding the views. To ensure that an index is evaluated, you can delete and recreate it. This way it becomes first in the list. On the other hand SQE orders the indexes by selectivity(most selective), then costs them and selects the cheapest one.

The following columns are the most commonly used:

- ▶ **QVQTBL**: Table name
- ▶ **QVQLIB**: Table library name
- ▶ **QQC11**: Optimizer timed out (Y or N)
- ▶ **QQ1000**: Contains library qualified index names, each with a reason code
 - Reason code 0 indicates that an index was selected.
 - Other codes are displayed in the second level text of CPI432C and CPI432D messages.
 - Documentation and iSeries Navigator reports classify this as the “Optimizer timed out”; the 3007 row will be generated even when the optimizer does not time out.

Example 6-7 shows a query with some of the most commonly used columns in the 3007 record.

Example 6-7 Common columns in the 3007 record

```
WITH xx AS
(SELECT * FROM mydbmon WHERE qqrid = 3007),

yy AS
SELECT qq1000 AS qqsttx, qqjfld, qqcnt
FROM mydbmon
WHERE qqrid = 1000
AND (qvc1c = 'Y' OR (qqc21 IN('DL', 'UP')
AND qqc181 <= ' ') OR qqc21 IN ('IN', 'IC', 'SK', 'SI') OR qqc21 LIKE '0%'))

SELECT qq1000,
varchar(qqsttx,20000) AS "Statement Text"
FROM xx a LEFT JOIN yy b
ON a.qqjfld = b.qqjfld AND a.qqcnt = b.qqcnt
```

Figure 6-8 shows the result of the query in Example 6-7. This is an example with information from the qq1000 column in the 3007 record. The information is joined with information from the 1000 record so we know the index evaluation for a given query. The first row in the example shows that four indexes were evaluated and the last one (CUST_00003) was used to implement the query.

QQ1000	Statement Text
STAR100G/CUST_00005 4, STAR100G/CUST_00002 4, STAR100G/CUST_00001 4, STAR100G/CUST_00003 0	SELECT AL1.YEAR,AL1.QUARTI
STAR100G/ITEM_EV13 17, STAR100G/ITEM_00014 4, STAR100G/ITEM_00012 4, STAR100G/ITEM_00007 4, STAR100G/ITEM_00004 4, STAR100G/ITEM...	SELECT AL1.YEAR,AL1.QUARTI
STAR100G/TIME_00003 4, STAR100G/TIME_00001 4, STAR100G/TIME_00002 0	SELECT AL1.YEAR,AL1.QUARTI
STAR100G/CUST_00005 4, STAR100G/CUST_00002 4, STAR100G/CUST_00001 4, STAR100G/CUST_00003 0	SELECT AL1.YEAR,AL1.QUARTI
STAR100G/ITEM_EV13 17, STAR100G/ITEM_00014 4, STAR100G/ITEM_00012 4, STAR100G/ITEM_00007 4, STAR100G/ITEM_00004 4, STAR100G/ITEM...	SELECT AL1.YEAR,AL1.QUARTI
STAR100G/TIME_00003 4, STAR100G/TIME_00001 4, STAR100G/TIME_00002 0	SELECT AL1.YEAR,AL1.QUARTI
STAR100G/ITEM_00014 6, STAR100G/ITEM_00012 6, STAR100G/ITEM_00011 6, STAR100G/ITEM_00010 6, STAR100G/ITEM_00009 17, STAR100G/ITE...	SELECT AL1.CUSTOMER,AL5.I
STAR100G/TIME_00003 0, STAR100G/TIME_00002 4, STAR100G/TIME_00001 4	SELECT AL1.CUSTOMER,AL5.I
STAR100G/CUST_00005 6, STAR100G/CUST_00004 4, STAR100G/CUST_00003 6, STAR100G/CUST_00002 6, STAR100G/CUST_00001 0	SELECT AL1.CUSTOMER,AL5.I

Figure 6-8 Common columns from the 3007 record

6.1.11 The 3010 record: host variables

The 3010 record shows substitution values for host variables or parameter markers in the query text (refer to the QQ1000 column in the 1000 record). This record appears just prior to each instance of an open, update, delete, or insert with subselect. This record is not displayed for insert with values. Data might not match exactly for updates with parameter markers in the SET clause.

- ▶ **QQ1000:** This column contains substitution values for host variables or parameter markers. The values (separated by commas) correspond left to right, with host variables and parameter markers. All values are displayed as a character, with no special indication of type. A floating point value is displayed as *F.
- ▶ **QQUCNT** and **QQI5:** These columns must be used to determine to which exact query the substitution values belong. The 3010 row type is not generated for the INSERT with Values statement.

Example 6-8 Common columns from the 3010 record

```

WITH aa AS
(SELECT * FROM mydbmon WHERE qqrid = 3010),
bb AS
(SELECT * FROM mydbmon WHERE qqrid = 1000 AND qqc21 IN
('OP','SI','SK','IN','UP','DL'))

SELECT a.qqrid, a.qqjnum, a.qqcunt, a.qqi5, a.qq1000, b.qq1000
FROM aa a LEFT JOIN bb b ON a.qqjnum = b.qqjnum AND a.qqcunt = b.qqcunt;

```

The results of the query shown in Example 6-8 are displayed in Figure 6-9. The query shows the Host variables and the associated SQL statement for each instance of the SQL statement by job.

QQRID	QQJNUM	QQUCNT	QQI5	QQ1000	QQ1000
3010	855517	1	0	A	DELETE FROM QUSRBRM.QA1ANET2 WHERE RMTSYS NOT IN (SELECT R1MSYS FROM ...)
3010	855517	6	0	ASLPAR2	SELECT * FROM QUSRBRM.QA1A1RMT WHERE R1MSYS = ? FOR READ ONLY
3010	855517	7	0	*SYSPCY	SELECT * FROM QUSRBRM.QA1ASP WHERE SYNAME = ? FOR READ ONLY
3010	855517	8	0	ASLPAR2, NETMR	SELECT * FROM QUSRBRM.QA1AAL WHERE ALSYNM = ? AND ALNETID = ? FOR REAC
3010	855517	10	0	ASLPAR2, NETMR, *MEDINV	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNM = ?, ALNETID = ?, ALTCPNAME =
3010	855517	11	0	AS400A7	SELECT * FROM QUSRBRM.QA1A1RMT WHERE R1MSYS = ? FOR READ ONLY
3010	855517	12	0	*SYSPCY	SELECT * FROM QUSRBRM.QA1ASP WHERE SYNAME = ? FOR READ ONLY
3010	855517	13	0	AS400A7, NETMR	SELECT * FROM QUSRBRM.QA1AAL WHERE ALSYNM = ? AND ALNETID = ? FOR REAC
3010	855517	15	0	AS400A7, NETMR, *MEDINV	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNM = ?, ALNETID = ?, ALTCPNAME =
3010	855517	16	0	AS400BW	SELECT * FROM QUSRBRM.QA1A1RMT WHERE R1MSYS = ? FOR READ ONLY
3010	855517	17	0	*SYSPCY	SELECT * FROM QUSRBRM.QA1ASP WHERE SYNAME = ? FOR READ ONLY
3010	855517	18	0	AS400BW, NETMR	SELECT * FROM QUSRBRM.QA1AAL WHERE ALSYNM = ? AND ALNETID = ? FOR REAC
3010	855517	20	0	AS400BW, NETMR, *MEDINV	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNM = ?, ALNETID = ?, ALTCPNAME =
3010	855517	21	0	AS400DB	SELECT * FROM QUSRBRM.QA1A1RMT WHERE R1MSYS = ? FOR READ ONLY
3010	855517	22	0	*SYSPCY	SELECT * FROM QUSRBRM.QA1ASP WHERE SYNAME = ? FOR READ ONLY
3010	855517	23	0	AS400DB, NETMR	SELECT * FROM QUSRBRM.QA1AAL WHERE ALSYNM = ? AND ALNETID = ? FOR REAC
3010	855517	25	0	AS400DB, NETMR, *MEDINV	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNM = ?, ALNETID = ?, ALTCPNAME =
3010	855517	26	0	AS400P1	SELECT * FROM QUSRBRM.QA1A1RMT WHERE R1MSYS = ? FOR READ ONLY
3010	855517	27	0	*SYSPCY	SELECT * FROM QUSRBRM.QA1ASP WHERE SYNAME = ? FOR READ ONLY
3010	855517	28	0	AS400P1, NETMR	SELECT * FROM QUSRBRM.QA1AAL WHERE ALSYNM = ? AND ALNETID = ? FOR REAC
3010	855517	30	0	AS400P1, NETMR, *MEDINV	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNM = ?, ALNETID = ?, ALTCPNAME =
3010	855517	31	0	AS400SA	SELECT * FROM QUSRBRM.QA1A1RMT WHERE R1MSYS = ? FOR READ ONLY
3010	855517	32	0	*SYSPCY	SELECT * FROM QUSRBRM.QA1ASP WHERE SYNAME = ? FOR READ ONLY

Figure 6-9 Common columns from the 3010 record

6.1.12 The 3014 record: general query optimization information

The 3014 record is displayed with full open optimization records. In most cases, one 3014 record is displayed per full open. You might see multiple 3014 records if the query consists of multiple separately run queries, for example, a subquery with grouping functions or views that need results materialized for use in the outer query. Values in this column help to identify the type of query that this record represents and the amount of time it took to open the cursor for this query.

This record also has summary information for the query or subquery. In most cases, there is one 3014 row per full open. Subselects and materialized views can cause multiple 3014 rows. It contains values for most of the settings that impact the query optimizer.

- ▶ **QQC102:** This column contains a library for the QAQQINI file if one is used. A value of *N indicates that no QAQQINI file was used.
 - Many of the QAQQINI settings are found in the 3014 row; a couple are found in the 1000 row type.
- ▶ **QVP154:** Memory pool size
- ▶ **QVP155:** Memory pool ID

- ▶ **QQC16:** This column contains a Y when SQE is used to process the SQL statement (CQE = N).
- ▶ **QVC43:** Column will contain the reason why SQE was not used

6.1.13 The 3015 record: SQE statistics advised

The 3015 record is generated by SQE when it determines that a column statistic needs to be collected or refreshed.

The following columns are commonly used:

- ▶ **QVQTBL:** Table name
- ▶ **QVQLIB:** Table library name
- ▶ **QQC11:** Statistics request type
 - N** No statistic existed for the column
 - S** Column statistic was stale
- ▶ **QQ1000:** Name of the column identified in Statistic Advice

Remember: Column statistics are created in the background automatically by default for all Statistics Advised. Use QQUCNT to analyze the SQL request to determine if an index is better than a column statistic.

6.1.14 The 3018 record: STRDBMON/ENDDBMON

The 3018 record shows when the detailed performance monitor is started or ended by issuing the Control Language Commands STRDBMON/ENDDBMON.

The following columns are the most commonly used:

- ▶ **QQC11:** Type of job monitored
 - **C** - Current
 - **J** - Job name
 - **A** - All
- ▶ **QQC12:** Command type
 - **S** - STRDBMON
 - **E** - ENDDBMON
- ▶ **QQC301:** Monitored job information
 - ***** - Current job
 - Job number/User/Job name
 - ***ALL** - All jobs
- ▶ **QQ1000L:** STRDBMON command texts

6.1.15 The 3019 record: rows retrieved detail

The 3019 record shows a summary of the Fetch or Retrieve operations performed by DB2. For non-SQL interfaces, such as OPNQRYF, it is the only way to determine number of rows returned and the amount of time to retrieve those rows. This record can also be used to analyze SQL requests.

The following columns are the most commonly used:

- ▶ **QQI1**: CPU time to return all rows, in milliseconds
- ▶ **QQI2**: Clock time to return all rows, in milliseconds
- ▶ **QQI3**: Number of synchronous database reads
- ▶ **QQI4**: Number of synchronous database writes
- ▶ **QQI5**: Number of asynchronous database reads
- ▶ **QQI6**: Number of asynchronous database writes
- ▶ **QQI7**: Number of rows returned
- ▶ **QQI8**: Number of calls to retrieve rows returned

6.1.16 The 3020 record: index advised (SQE)

The query optimizer analyzes the row selection in the query and determines, based on default values, if creation of a permanent index improves performance. If the optimizer determines that a permanent index might be beneficial, it returns the key columns necessary to create the suggested index. For more information about this advisor, refer to the *DB2 UDB for iSeries Database Performance and Query Optimization* manual searching on Query optimizer index advisor. You can find instructions on the steps to view this manual 6.1, “The Database Monitor record types” on page 174.

The following columns are commonly used:

- ▶ **QQIDX**: Index advised (Y/N)
- ▶ **QQIDX**: Columns for the index advised (also **QQ1000L** for long names)
- ▶ **QQI1**: Number of indexes advised
- ▶ **QRCOD**: Reason code
 - **I1** - Row selection
 - **I2** - Ordering/Grouping
 - **I3** - Row selection and Ordering/Grouping
 - **I4** - Nested loop join
 - **I5** - Row selection using bitmap processing

The following query in Example 6-9 shows the SQL you can use to analyze which indexes are advised and the frequency of the advice.

Example 6-9 Record 3020 Index advised analysis

```
SELECT count(*) AS "No. Times Advised",
       qvplib AS "QVPLIB Schema",
       qvptbl AS "QVPTBL Table Name",
       cast(substr(qq1000l, 1, 100) AS CHAR(100))
       AS Keys_Advised
FROM mydbmon
WHERE qqrid = 3020
AND qqjnum = '855513'
GROUP BY qvplib, qvptbl, cast(substr(qq1000l, 1, 100) AS CHAR(100))
ORDER BY 1 Desc, 2, 3, 4;
```

The report in Figure 6-10 is the result of selecting and analyzing the 3020 records. The report shows a list of the most frequent advised indexes, the advised keys and the table on which they are advised. At the top of the list, an index composed of table columns, CUSTOMER and CUSTKEY was advised for table CUSTOMER, 28 times.

The report was organized by the number of times advised in descending sequence by table and advised keys.

No. Times Advised	QVPLIB Schema	QVPTBL Table Name	KEYS ADVISED
28	DBITSODATA	CUSTOMERS	CUSTOMER, CUSTKEY
24	DBITSODATA	ORDERS	RETURNFLAG, CUSTKEY
22	DBITSODATA	SUPPLIERS	SUPPKEY
18	DBITSODATA	PARTS	PARTKEY
15	DBITSODATA	DATES	YEAR, DATEKEY
12	DBITSODATA	ORDERS	SHIPDATE
12	DBITSODATA	ORDERS	YEAR, QUANTITY
10	DBITSODATA	DATES	DATEKEY
10	DBITSODATA	SUPPLIERS	SUPPLIER, SUPPKEY
9	DBITSODATA	DATES	YEAR, MONTH, DATEKEY
9	DBITSODATA	ORDERS	RETURNFLAG, QUANTITY
8	DBITSODATA	ORDERS	RETURNFLAG, SUPPKEY
7	DBITSODATA	DATES	YEAR, MONTH

Figure 6-10 Index Advised analysis

6.1.17 The 3030 record: materialized query table

Enhancements have been added to the Database Monitor data to indicate usage of materialized query tables (MQT).

Note: The query optimizer support for recognizing and using MQTs is available with V5R3 i5/OS PTF SI17164 and DB group PTF SF99503 level 4.

Although an MQT can contain almost any query, the optimizer only supports a limited set of query functions when matching MQTs to user-specified queries. The user-specified query and the MQT must both use SQE. The optimizer only uses one MQT per query.

- ▶ **1000/3006 record:** QQC22 has a new reason code of B5, which indicates that the access plan needed to be rebuilt because the MQT was no longer eligible to be used. The reasons might be that:
 - The MQT no longer exists
 - A new MQT was found
 - The enable or disable query optimization changed
 - Time since the last REFRESH TABLE exceeds the MATERIALIZED_QUERY_TABLE_REFRESH_AGE QAQQINI option
 - Other QAQQINI options no longer match
- ▶ **3014 record:** This record shows the new QAQQINI file values:
 - Logical field name QQMQTR, physical field name QQI7, contains the MATERIALIZED_QUERY_TABLE_REFRESH_AGE duration. If the QAQQINI parameter value is set to *ANY, the timestamp duration will be 99999999999999.
 - Logical field name QQMQTU, physical field name QVC42. The first byte of QVC24 contains the MATERIALIZED_QUERY_TABLE_USAGE. Supported values are:
 - N** *NONE, no materialized query tables are used in query optimization and implementation.
 - A** User-maintained refresh-deferred query tables might be used.
 - U** Only user-maintained materialized query tables might be used.
- ▶ **3000, 3001, 3002 records:** New columns have been added to the 3000, 3001, and 3002 records to indicate that a table or tables were replaced with an MQT. The logical field

name QQMQT, physical field name QQC13, is either Y or N, indicating that this is an MQT, which replaced a table or tables in the query.

- ▶ **3030 record:** The new 3030 record contains information about the MQTs that are examined. This record is only written if MQTs are enabled and MQTs exist over the tables specified in the query.

For a complete and details description of MQTs, refer to the white paper named: *The creation and use of materialized query tables within IBM DB2 for i5/OS* written by Michael Cain available on the Web at:

http://www-03.ibm.com/servers/enable/site/education/abstracts/438a_abs.html

6.2 Introduction to query analysis

The Detailed Database Monitor table can be analyzed by using SQL. This is a time-consuming approach unless you have predefined queries. A lot of predefined reports exist via the new iSeries Navigator V5R4 GUI “dashboard” interface as explained in Chapter 5, “Analyzing SQL performance data using iSeries Navigator” on page 117, however, you can also write your own queries.

In this chapter, we present several queries to help you analyze the database performance data. You can run the queries under all SQL interfaces that access the iSeries server. The green-screen interface is intentionally not selected, because it can have a major negative performance impact on some server models with reduced interactive capacity. All queries in this chapter are run through the Run SQL Scripts window in iSeries Navigator.

To start the interface, in iSeries Navigator, select **Databases** → **your database**, right-click, and select **Run SQL Scripts** (as shown in Figure 6-11).

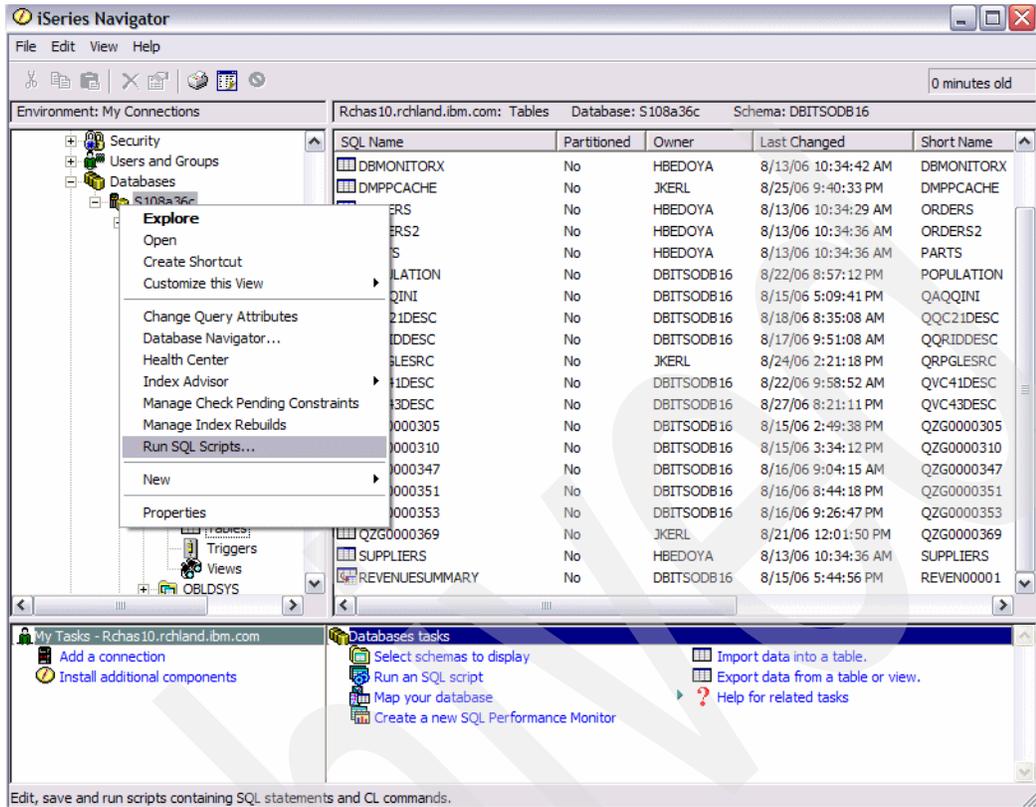


Figure 6-11 Starting the Run SQL Scripts option

6.3 Tips for analyzing the Database Monitor files

In the following sections, we present different tips to make it easier to analyze the performance data using custom-made queries. The idea is to let you copy the different SQL requests, so you can use them in your own analysis.

6.3.1 Using an SQL ALIAS for the Database Monitor table

By creating an SQL alias for the Detailed Database Monitor table that you are analyzing, you can use the same names for the analysis. When you analyze the next Detailed Database Monitor table, you use the SQL DROP ALIAS statement and then create an SQL ALIAS statement with the same name over the other table.

If you want to use DBMONLIB.MYDBMON, then use the following SQL CREATE ALIAS statement:

```
CREATE ALIAS DBMONLIB.MYDBMON FOR ibmfr.LAURA1608;
```

Before you analyze the next Database Monitor data, be sure to enter the SQL DROP ALIAS statement:

```
DROP ALIAS DBMONLIB.MYDBMON
```

6.3.2 Using a subset of the Database Monitor table for faster analysis

The Database Monitor table often is large and contains information about many jobs. Therefore, running queries on the data can sometimes be slower than desired. You can try to reduce the time that the analysis queries take by collecting only the jobs that you want. However, sometimes this is not possible and, even if it is, batch jobs can generate a lot of Database Monitor output. Also, using interactive tools, such as Start SQL (STRSQL), can result in longer run times on server models.

If the response time is slow during the analysis, consider the following tips:

- ▶ Before you start the analysis, see how big the output table is for the collected Database Monitor.
- ▶ Create a smaller table from the main Database Monitor table with only the rows in which you are interested. You can use the following technique:

```
CREATE TABLE smaller-table AS (SELECT * FROM big-dbmon-table  
WHERE QQJNUM IN('job-number','job-number'...)) WITH DATA
```

- ▶ Another way to reduce the Database Monitor data is to include a time range in the SQL selection criteria, for example:

```
and qqtime > '2005-03-16-12.00.00.000000' and qqtime <  
'2005-03-16-12.05.00.000000'
```

This shows only five minutes of collection.

You can adjust these techniques as needed to speed up your analysis.

6.3.3 Using SQL views for the Database Monitor table

When using queries to analyze the Database Monitor table, you can make the queries more readable by using views. For example, we look at a query that shows table scans, which can be through a simple view, making it easier to see an overview of the query.

Example 6-10 shows the query before we create a view.

Example 6-10 Query before using an SQL view

```
WITH tablescans AS (  
  SELECT qqjfld,qqcnt,qqrest,qqtotr  
  FROM mydbmon  
  WHERE qqrid=3000)  
SELECT SUM(qqi6) "Total Time (Mics)", COUNT(*) "Times Run",  
  a.qqcnt, integer(avg(b.qqrest)) "Est Rows Selected",  
  integer(avg(b.qqtotr)) "Total Rows in Table", qq10001  
FROM MYDBMON a, tablescans b  
WHERE qqrid=1000 AND a.qqjfld=b.qqjfld AND  
  qqc21 IN ('OP','SI','UP','IN','DL')  
GROUP BY a.qqcnt, qq10001  
ORDER BY "Total Time" DESC;
```

Then we create an SQL view as shown in Example 6-11.

Example 6-11 Creating an SQL view

```
CREATE VIEW LASSE0410.TABLESCANS AS SELECT QQJFLD, QQUCNT, QQRCD, QQREST, QQTOTR  
FROM MYDBMON WHERE qqrid=3000;
```

The query runs with the new view, with the assumption that the schema is in the library list as shown in Example 6-12.

Example 6-12 Query after using the SQL view

```
SELECT SUM(qqi6) "Total Time (Mics)", COUNT(*) "Times Run",  
  a.qqcnt, integer(avg(b.qqrest)) "Est Rows Selected",  
  integer(avg(b.qqtotr)) "Total Rows in Table", qq10001  
1  
FROM mydbmon a, tablescans b  
WHERE qqrid=1000 AND a.qqjfld=b.qqjfld AND  
  qqc21 IN ('OP','SI','UP','IN','DL')  
GROUP BY a.qqcnt, qq10001  
ORDER BY "Total Time" DESC;
```

In the remainder of this chapter, we do not use views, but rather table expressions. When you analyze your own Database Monitor data collection, you might find situations where you can benefit from using SQL views.

6.3.4 Creating additional indexes over the Database Monitor table

In Chapter 2, "DB2 for i5/OS performance basics" on page 11, we cover the importance of indexes for SQL performance. The Database Monitor table tends to become quite large in size. Therefore, it is important to create indexes on the Database Monitor table over the common selection, grouping, and ordering clauses.

The following examples are some additional key combinations that a user can add to the automatic indexes created by using the iSeries Navigator V5R4 GUI “dashboard”:

- ▶ QQJFLD, QQUCNT, QQI5
- ▶ QQRID & QQ1000

Figure 6-12 shows the SQL statements to create the additional indexes described previously.

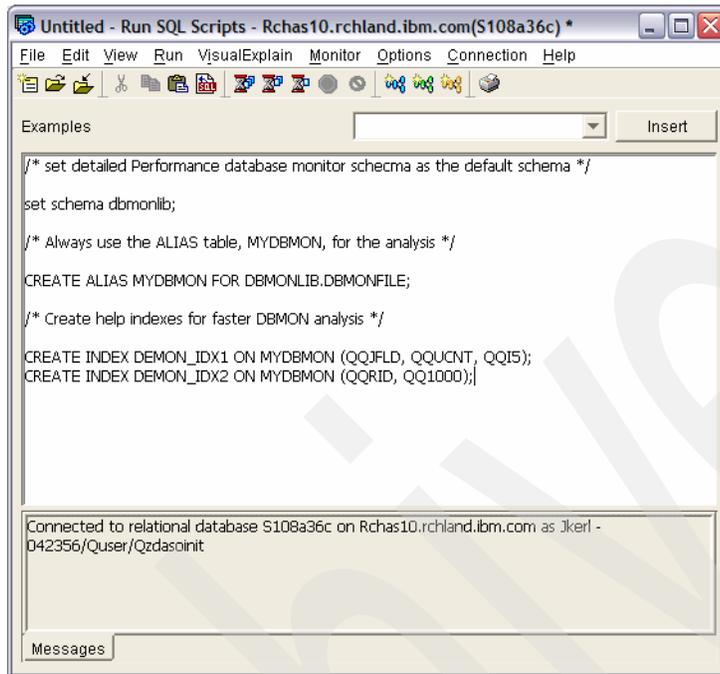


Figure 6-12 Creating additional indexes for faster analysis

You can try other combinations as necessary. Remember to combine the selection, grouping, and ordering clauses.

Note: When you use iSeries Navigator to start a detailed SQL Performance Monitor, as soon as you end the data collection or import the monitor data file and start the analysis via the new iSeries Navigator V5R4 GUI *dashboard*, it provides six indexes based on the following key columns:

- ▶ QRID (Asc), QQC21 (Asc)
- ▶ QC11 (Asc), QQRID (Asc)
- ▶ QJFLD (Asc), QQIS (Asc)
- ▶ QUCNT (Asc)
- ▶ QC12(Asc), QQRID (Asc), QQTIME (Asc)
- ▶ QRID (Asc), QQI5 (Asc)

6.4 Database Monitor query examples

In this section, we present a series of queries to help solve specific questions in the detection and resolution of SQL performance issues. Most of these queries have more elaborate equivalents in the SQL Performance Monitor predefined reports as indicated. However, it is still useful to be familiar with the Database Monitor table.

Before running the queries, we presume that an SQL SET SCHEMA and an SQL CREATE ALIAS are done as shown in Figure 6-12.

6.4.1 Finding SQL requests that are causing problems

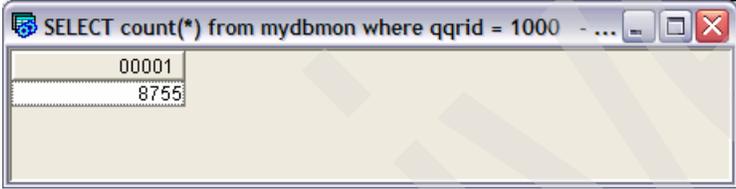
To find the SQL that is causing problems, expect to use more queries for the investigation, because it is not only the running time that matters.

First, take an overview over the data collected. To determine the number of SQL requests that were done during the collection, run the query shown in Example 6-13.

Example 6-13 Number of SQL requests in the Database Monitor table

```
SELECT count(*) FROM mydbmon  
WHERE qqrid=1000;
```

Figure 6-13 shows the result of the query in Example 6-13.



The screenshot shows a window titled "SELECT count(*) from mydbmon where qqrid = 1000". The window contains a table with two rows of data:

00001
8755

Figure 6-13 Number of SQL requests in the DBMON collection

The result is smaller than the number of rows in the Database Monitor table. The number of rows per SQL request can be from 1 to over 20, depending on the complexity of the SQL request. Each SQL request has one QQRID = 1000 record.

For an overview of the most time-consuming jobs running SQL, we use the query shown in Example 6-14.

Example 6-14 Time-consuming jobs

```
SELECT SUM(qqi6) "Total Time (Mics)", COUNT(*) "Total SQL Requests",  
qqjnum, qquser, qqjob FROM mydbmon  
WHERE qqrid=1000  
GROUP BY qqjob,qquser,qqjnum ORDER BY 1 DESC;
```

Figure 6-14 shows the output of the query in Example 6-14.

Total Time (Mics)	Total SQL Requests	QQJNUM	QQJOB	QQUSER
103592568	986	855513	QZDASOINIT	QUSER
1388248	135	855561	QZDASOINIT	QUSER
767416	74	855569	QZDASOINIT	QUSER
593432	147	855544	QBRMSYNC	QBRMS
498960	147	855528	QBRMSYNC	QBRMS
490496	147	855541	QBRMSYNC	QBRMS
489952	147	855554	QBRMSYNC	QBRMS
488600	147	855546	QBRMSYNC	QBRMS
488184	147	855535	QBRMSYNC	QBRMS
488072	147	855525	QBRMSYNC	QBRMS
485984	147	855531	QBRMSYNC	QBRMS
483688	147	855543	QBRMSYNC	QBRMS
483376	147	855559	QBRMSYNC	QBRMS
480056	147	855540	QBRMSYNC	QBRMS

Figure 6-14 Time-consuming jobs

The total time is in microseconds. Therefore, you must divide the time by 1000000 to see it in seconds.

To find SQL requests that might cause the problems, look at different information. For example, you should know whether the Database Monitor collection is from a sample period of the day or from a period when response problems were observed. You can also determine which SQL requests in a specific job or jobs take the most processing time.

6.4.2 Total time spent in SQL

During the analysis of monitor data, you can see the percentage of time that is spent in DB2. To begin, you find the start time and end time to have a duration of the Database Monitor data collection as shown in Example 6-15.

Example 6-15 Duration of the Database Monitor data collection

```
SELECT MIN(qqtime) "Start Time", MAX(qqtime) "End Time",
MAX((DAY(qqtime)*24*3600)+(HOUR(qqtime)*3600)+(MINUTE(qqtime)*60)+
(SECOND(qqtime))+(MICROSECOND(qqtime)*.000001)) -
MIN((DAY(qqtime)*24*3600)+(HOUR(qqtime)*3600)+(MINUTE(qqtime)*60)+
(SECOND(qqtime))+(MICROSECOND(qqtime)*.000001)) AS "Duration"
FROM mydbmon WHERE qqrid<>3018
```

Figure 6-15 shows the result of Example 6-15.

Start Time	End Time	Duration
2006-01-30 10:01:54.733263	2006-01-30 10:52:55.504214	3060.770951

Figure 6-15 Duration of Database Monitor data collection

To get the duration of the Database Monitor data collection for the job, select the job number as shown in Example 6-16. For qqjnum = '999999', substitute the job number.

Example 6-16 Duration of the Database Monitor collection for one job

```
SELECT MIN(qqtime) "Start Time", MAX(qqtime) "End Time",  
MAX(qqtime) - MIN(qqtime) "Duration (Sec)"  
FROM mydbmon WHERE qqrid<>3018 and qqjnum = '999999';
```

You can also find the total number of seconds spent by using the SQL statement shown in Example 6-17.

Example 6-17 Time spent in SQL

```
SELECT SUM(qqi6)/1000000 "Total Time (Sec)" FROM mydbmon  
WHERE qqrid=1000;
```

If stored procedures are used, then they count as double, because both the stored procedure and the SQL requests in the stored procedure generate records in the Database Monitor table. A good approximation is to exclude the stored procedure from the total time used in the SQL shown Example 6-18.

Example 6-18 Time spent in SQL excluding stored procedures

```
SELECT SUM(qqi6)/1000000 "Total Time (Sec)" FROM mydbmon  
WHERE qqrid=1000 AND qqc21 <> 'CA' ;
```

Figure 6-16 shows the output for Example 6-18.



Figure 6-16 Time spent in SQL

The time spent in SQL might not seem so relevant for the total run, but when the focus is on individual jobs, or applications, then it is relevant. This means that a selection of the job should be added to the query.

6.4.3 Individual SQL elapsed time

To find the SQL requests that count for the most run time, use the query shown in Example 6-19.

Example 6-19 SQL request sorted on the total run time

```
SELECT SUM(qqi6) "Total Time" , COUNT(*) "Nbr Times Run", qq10001 "SQL Request"  
FROM MYDBMON  
WHERE qqjnum = '999999' AND qqrid=1000 AND qqcnt<>0  
GROUP BY qq10001 ORDER BY 1 DESC;
```

The SQL in Example 6-19 eliminates reporting all none ODP operations by using qqcnt <>0 in the local selection. Also note that this report does not include Open, Fetch, and Close run times. The result shown in Figure 6-17 comes from our test collection.

Total Time (Mics)	Nbr Times Run	SQL Request
37806976	188	
14193232	1	update mcain.items_xset expander = ?where returnflag = ?and year = ?and month = ?
13020152	1	update mcain.items_xset expander = ?where returnflag = ?
12268904	1	delete from mcain.items_x where returnflag = ?
8711680	1	insert into mcain.items_x(select * from star1g.item_fact where returnflag = ?)
6033256	3	update mcain.items_xset expander = ?where custkey = ?
2609232	1	delete from mcain.items_xwhere custkey in (?, ?, ?)
1997440	1	delete from mcain.items_xwhere custkey = ?
1071752	2	select distinct suppkey from orders where returnflag = upper(?)order by suppkey
1005288	1	insert into mcain.items_x(select * from star1g.item_fact where custkey < ?)
639672	5	-- Query 101 SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2.ORDERKEY, AL3.PART, AL4.SU...
240688	1	SELECT AL2.YEAR, AL2.MONTHNAME, AL2.WEEK, AL1.CUSTOMER, AL3.PART, AL4.QUANTITY, AL4.ORDERK...
191160	2	select *from orderswhere shipdate in (?,...
110320	3	select distinct returnflagfrom ordersorder by returnflag
97152	4	SELECT AL4.SUPPLIER, AL5.YEAR, AL5.MONTHNAME, SUM(AL2.QUANTITY)FROM orders AL2, suppliers AL4, ...
95216	4	-- Query 104 SELECT AL2.YEAR, AL2.MONTHNAME, AL2.WEEK, AL1.CUSTOMER, AL3.PART, AL4.QUANTITY,...
79992	3	-- Query 102 SELECT AL5.YEAR, AL5.MONTH, AL1.SALESPERSON, SUM(AL2.REVENUE_WO_TAX) FROM
64648	3	select tyear, tquarter, tmonth, sum(i.quantity)from orders i, dates twhere i.shipdate = t.datekeyand tyear = ?and t.qu...
60704	2	SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2.ORDERKEY, AL3.PART, AL4.SUPPLIER, AL3.MFGR, AL2...
58720	2	select tyear, tquarter, sum(i.quantity)from orders i, dates t, customers cwhere i.shipdate = t.datekeyand i.custkey...
54232	2	-- Query 105SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2.ORDERKEY, AL3.PART, AL4.SUPPLIER, AL3...

Figure 6-17 SQL requests sorted by Total Time

The top item in report in Figure 6-17 has the largest total time and number of times run, but the problem is that the SQL Request column is blank. This means that the SQL may be only found in the Open, Fetch or Close monitor records.

In order to check, execute the following SQL request shown in Example 6-20.

Example 6-20 SQL request for Open, Fetch and Close high run times

```

WITH OpenFetchClose AS
  (SELECT qqjfld, sum(qqi6) AS qqi6_total
   FROM MYDBMON
   WHERE qqrid = 1000 AND qqc21 IN ('SI', 'OP', 'FE', 'CL', 'IN', 'UP', 'DL',
   'HC')
   GROUP BY qqjfld)
SELECT sum(x.QQI6_Total) AS "Total Time (Mics)", COUNT(*) AS "No. Times Run",
z.qq10001
FROM OpenFetchClose x, MYDBMON z
WHERE x.qqjfld = z.qqjfld AND z.qqrid = 1000
AND z.qqc21 IN ('SI', 'OP', 'IN', 'UP', 'DL') AND qqi5 = 0
GROUP BY qq10001
ORDER BY 1 DESC;

```

Total Time (Mics)	No. Times Run	SQL Statement
18599320	3	select count(*) from mcaim.items_x where returnflag = ?
14193232	1	update mcaim.items_x set expander = ? where returnflag = ? and year = ? and month = ?
13020152	1	update mcaim.items_x set expander = ? where returnflag = ?
12268904	1	delete from mcaim.items_x where returnflag = ?
8711680	1	insert into mcaim.items_x (select * from star1 g.item_fact where returnflag = ?)
6033256	2	update mcaim.items_x set expander = ? where custkey = ?
5529368	1	select distinct returnflag from mcaim.items_x
4859472	306	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNN = ?, ALNETID = ?, ALTCPNAME = ?, ALSNANAME = ?, ...
2941616	4	select i.year, i.month, upper(c.customer), sum(quantity) from orders i, customers c where i.custkey = c.custke...
2896472	51	DELETE FROM QUSRBRM.QA1ANET2 WHERE RMTSYS NOT IN (SELECT R1MSYS FROM QUSRBRM.QA1A1R...
2609232	1	delete from mcaim.items_x where custkey in (?, ?, ?)
2355896	306	SELECT * FROM QUSRBRM.QA1ASP WHERE SYNAME = ? FOR READ ONLY
2128792	306	SELECT * FROM QUSRBRM.QA1AAL WHERE ALSYNN = ? AND ALNETID = ? FOR READ ONLY
2099136	2	-- Query 3d SELECT custkey, sum(quantity) as total_quantity FROM Orders WHERE supkey = ? AND returnflag = ? A...
2072016	2	-- Query 2d SELECT custkey, sum(quantity) as total_quantity FROM Orders WHERE supkey = ? AND returnflag = ? G...
2003520	306	SELECT * FROM QUSRBRM.QA1A1RMT WHERE R1MSYS = ? FOR READ ONLY
1997440	1	delete from mcaim.items_x where custkey = ?
1959616	2	-- Query 1d SELECT custkey, sum(quantity) as total_quantity FROM Orders WHERE supkey = ? GROUP BY custkey...
1709168	306	SELECT * FROM QUSRBRM.QA1AAL
1368976	1	select * from mcaim.items_x where custkey = ?
1150888	51	INSERT INTO QUSRBRM.QA1A2NET SELECT DISTINCT RMTSYS AS N2RMT, RMTRNI AS N2RNI, OBJ AS N2...
1122680	2	select distinct supkey from orders where returnflag = upper(?) order by supkey
1005288	1	insert into mcaim.items_x (select * from star1 g.item_fact where custkey < ?)

Figure 6-18 Open, Fetch and Close high run time report

In the report in Figure 6-18, the top item shows the SQL statement with the highest run time.

A large number of the same SQL request can give a high total run time. Therefore, it is also relevant to look at the SQL requests with the longest average run time as shown in the query in Example 6-21.

Example 6-21 SQL requests sorted by average runtime

```

SELECT SUM(qqi6) "Total Time (Mics)", COUNT(*) "Nbr Times Run",
SUM(qqi6)/COUNT(*) "Average Run Time", qq10001 "SQL Request"
FROM MYDBMON
WHERE qqrid=1000 AND qqcnt>0
GROUP BY qq10001 ORDER BY 3 DESC;

```

The execution of the query in Example 6-21 produces the result shown in Figure 6-19.

Total Time (Mic...	Nbr Times Run	Average Run Time	SQL Request
14193232	1	14193232	update mcain.items_xset expander = ?where returnflag = ?and year = ?and month = ?
13020152	1	13020152	update mcain.items_xset expander = ?where returnflag = ?
12268904	1	12268904	delete from mcain.items_xwhere returnflag = ?
8711680	1	8711680	insert into mcain.items_x(select * from star1g.item_fact where returnflag = ?)
2609232	1	2609232	delete from mcain.items_xwhere custkey in (?, ?, ?)
6033256	3	2011085	update mcain.items_xset expander = ?where custkey = ?
1997440	1	1997440	delete from mcain.items_xwhere custkey = ?
1005288	1	1005288	insert into mcain.items_x(select * from star1g.item_fact where custkey < ?)
1071752	2	535876	select distinct suppkkey from orders where returnflag = upper(?)order by suppkkey
2406888	1	2406888	SELECT AL2.YEAR, AL2.MONTHNAME, AL2.WEEK, AL1.CUSTOMER, AL3.PART, AL4.QUANT...
639672	5	127934	-- Query 101 SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2.ORDERKEY, AL3.P...
357528	3	119176	select *from orderswhere shipdate in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
86032	1	86032	select * from orders where returnflag = ?
129664	2	64832	select distinct mktsegment from customers order...
57464	1	57464	select distinct shipmode from orders order by shipmode
2896472	51	56793	DELETE FROM QUSRBRM.QA1ANET2 WHERE RMTSYS NOT IN (SELECT R1MSYS FROM QUSRBR...
44176	1	44176	select * from orders where custkey < ?
168552	4	42138	select distinct returnflagfrom ordersorder by returnflag
78736	2	39368	select suppkkey from orders where returnflag = ? and year = ?
38744	1	38744	select custkey from orders where year = ? and month = ? and returnflag = ?
35984	1	35984	select * from mcain.items_xwhere custkey = ?
38805200	1197	32418	
60704	2	30352	SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2.ORDERKEY, AL3.PART, AL4.SUPPLIE...

Figure 6-19 SQL requests sorted on Average Runtime

Important: Notice that the queries in Example 6-19 and Example 6-21 use QQ1000L field whose length is a 2MB CLOB in the GROUP BY. If the monitor table contains SQL statements that are less than 1000 characters, then you should use the QQ1000 column for performance reasons. When using QQ1000, the results of the SUM and COUNT functions may not be accurate especially if the SQL statements are longer than 1000 characters.

6.4.4 Analyzing SQL operation types

To get an overview of the different SQL operation types that are run during the performance data collection, use the query shown in Example 6-22.

Example 6-22 SQL operation types

```
SELECT SUM(qqi6) "Total Time (Mics)", COUNT(*) "Nbr of Requests",
qqc21 "Operation Type" FROM MYDBMON
WHERE qqrid=1000
GROUP BY qqc21 ORDER BY 1 DESC;
```

Figure 6-20 shows the output of the query in Example 6-22.

Total Time (Mics)	Nbr of Requests	Operation Type
38805200	1197	FE
38106112	311	UP
20770560	109	DL
13254696	1503	OP
10867856	53	IN
5553096	1665	PD
914920	1504	CL
747184	12	CA
448296	310	PR
354224	7	SI
192528	2	CN
153976	1	CT
16768	26	XA
2920	7	SS
1448	4	CM
512	1	EI
0	2037	HC
0	6	JR

Figure 6-20 SQL operation types

This example gives you an idea of the SQL operations that run the most and the amount of time that they account for. In our example, there were 1197 FETCH operations.

6.4.5 Full open analysis

The first time (or times) that an open occurs for a specific statement in a job, a full open operation is required. A *full open* creates an ODP that is then used to fetch, update, delete, or insert rows. An ODP might be cached at close time, so that if the SQL statement is run again during the job, the ODP is reused. Such an open is called a *pseudo-open* and is much less expensive than a full open.

A normal SQL application has many fetches, inserts, updates, and deletes. A desirable situation is that most of the operations share the ODP so that a full open does not have to be done over and over again.

Reusable ODPs

Reusable ODP usually happens after the second execution of an SQL statement within the connection or job, if the statement is reusable. Because the reusable statements are significantly faster than the nonreusable ones, you can find the statements that are not reusing the ODP. The QQUCNT value is assigned at full open time and stays constant for all subsequent reusable instances of that particular query.

Nonreusable ODPs are indicated by the presence of optimization records each time a particular query is run (full open). Reusable ODPs are indicated by 3010 and 1000 records each time the given query is run (no optimization records or full open). To understand why an ODP is not reusable, look at the hard close reason.

To find the number of SQL requests affected by full opens, you use the query shown in Example 6-23.

Example 6-23 SQL requests affected by Full Opens

```
SELECT SUM(qqi6) "Total Time (Mics)" , COUNT(*) "Nbr Full Opens", qq1000
FROM mydbmon
WHERE qqjnum = '999999' AND qqrid=1000 AND qqi5=0
AND qqc21 IN ('OP','SI', 'DL', 'IN', 'UP')
GROUP BY qq1000 ORDER BY 1 DESC;
```

Figure 6-21 shows the results of the query in Example 6-23.

Total Time (Mics)	Nbr Full Opens	QQ1000
14193232	1	update mcain.items_xset expander = ?where returnflag = ?and year = ?and month = ?
13020152	1	update mcain.items_xset expander = ?where returnflag = ?
12268904	1	delete from mcain.items_x where returnflag = ?
8711680	1	insert into mcain.items_x(select * from star1g.item_fact where returnflag = ?)
4038792	2	update mcain.items_xset expander = ?where custkey = ?
2609232	1	delete from mcain.items_xwhere custkey in (?, ?, ?)
1997440	1	delete from mcain.items_xwhere custkey = ?
1071752	2	select distinct suppkey from orders where returnflag = upper(?)order by suppkey
1005288	1	insert into mcain.items_x(select * from star1g.item_fact where custkey < ?)
839672	5	- Query 101 SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2.ORDERKEY, AL3.PART,
240688	1	SELECT AL2.YEAR, AL2.MONTHNAME, AL2.WEEK, AL1.CUSTOMER, AL3.PART, AL4.QUANTITY, AL4.
191160	2	select *from orderswhere shipdate in (?, ?)
110320	3	select distinct returnflagfrom ordersorder by returnflag
97152	4	SELECT AL4.SUPPLIER, AL5.YEAR, AL5.MONTHNAME, SUM(AL2.QUANTITY)FROM orders AL2, suppli
95216	4	- Query 104 SELECT AL2.YEAR, AL2.MONTHNAME, AL2.WEEK, AL1.CUSTOMER, AL3.PART, AL4.Q
79992	3	- Query 102 SELECT AL5.YEAR, AL5.MONTH, AL1.SALESPERSON, SUM(AL2.REVENUE_WO_TAX) FI
64648	3	select tyear, tquarter, tmonth, sum(i.quantity)from orders i, dates twhere i.shipdate = t.datekeyand tyear =
60704	2	SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2.ORDERKEY, AL3.PART, AL4.SUPPLIER, AL3.MI
58720	2	select tyear, tquarter, sum(i.quantity)from orders i, dates t, customers cwhere i.shipdate = t.datekeyand

Figure 6-21 SQL requests affected by full opens

To analyze the full opens, you must next copy and paste the SQL request or part of the SQL request into a query that we use for analysis. From the previous example, copy the following SQL statement:

```
"SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2.ORDERKEY, AL3.PART,..."
```

Copy the selected job number and SQL statement into the query shown in Example 6-24. Be sure to substitute the job number for the 9's in a.qqjnum = '999999' and SQL for the X's that follow the LIKE predicate.

Example 6-24 SQL requests affected by Full Opens in a single job

```
SELECT a.qqjnum, a.qqrid, a.qqcnt, a.qqi5, a.qqc21, a.qqc15 "HC Reason Code",
a.qqc22 "Rebuild Reason Code" ,
a.qqc181 "Cursor Name", a.qqc182 "Stmt Name" , a.qq1000
FROM MYDBMON a
WHERE a.qqjnum = '999999'
AND a.qqrid = 1000
AND ((a.qqcnt IN (SELECT b.qqcnt FROM MYDBMON b
WHERE b.qqjnum = '999999'
AND b.qqrid = 1000
AND b.qqc21 = 'OP'
AND b.qq1000 LIKE 'XXXXXXXXXXXXXXXXX%'))
OR (qqc21 IN ('DI', 'ST', 'CM', 'RO'))))
ORDER BY a.qqtime;
```

Figure 6-22 shows the results for Example 6-24.

QQJNUM	QQRID	QQUCNT	QQI5	HC Reas...	Rebuild R...	Cursor Name	Stmt Name	QQ1000
855513	1000	67	0	OP	NA	CRSR0069	STMT0069	SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, ,
855513	1000	67	0	HC	NA	CRSR0069	STMT0069	HARD CLOSE 1 CURSORS
855513	1000	67	0	FE	NA	CRSR0069	STMT0069	
855513	1000	67	0	CL	NA	CRSR0069		CLOSE CRSR0069
855513	1000	142	0	OP	NA	CRSR0069	STMT0069	SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, ,
855513	1000	142	0	FE	NA	CRSR0069	STMT0069	
855513	1000	142	0	CL	NA	CRSR0069		CLOSE CRSR0069
855513	1000	142	0	HC	NA	CRSR0069	STMT0069	HARD CLOSE 1 CURSORS

Figure 6-22 Analysis of SQL requests affected by Full Opens in a single job

Look at the values in the QQC21 column for ODP analysis, which shows one of the following abbreviations:

- HC** Hard Close
- DI** Disconnect
- ST** Set Transaction
- CM** Commit
- RO** Rollback

Identify the reasons why a Hard Close is being done by looking in the QQC15 column, some of the reason codes are:

- 2** Exclusive Lock
- 4** Host Variable Reuse Restriction
- 6** Cursor Restriction (after first execution)
- 7** Cursor Hard Close Requested (proprietary attribute)
- 9** Cursor Threshold
- C** DRDA® AS Cursor Closed
- D** DRDA AR Not WITH HOLD
- J** File Override Change
- K** Program Invocation Change
- L** File Open Options Change
- O** Library List Change
- P** Exit Processing (End Connection)
- Q** SetSession User

Example 6-25 shows another query to find the number of full opens and pseudo-opens for the SQL request.

Example 6-25 Number of full opens and pseudo-opens

```
SELECT SUM(qqi6) "Total Time (Mics) " ,
SUM(CASE WHEN qvc12 = 'N' THEN 1 ELSE 0 END) "Full Opens",
SUM(CASE WHEN qvc12 = 'Y' THEN 1 ELSE 0 END) "Pseudo Opens",
QQ10001
FROM MYDBMON
WHERE qqrid=1000 and qqjnum = '999999'
AND qqc21 IN ('OP', 'SI', 'DL', 'IN', 'UP')
GROUP BY qq10001
ORDER BY 1 DESC;
```

Figure 6-23 shows the results of the query in Example 6-25.

Total Time (Mics)	Full Opens	Pseudo Opens	QQ1000L
14193232	1	0	update mcaim.items_xset expander = ?where returnflag = ?and year = ?and month = ?
13020152	1	0	update mcaim.items_xset expander = ?where returnflag = ?
12268904	1	0	delete from mcaim.items_x where returnflag = ?
8711680	1	0	insert into mcaim.items_x(select * from star1.g.item_fact where returnflag = ?)
6033256	2	1	update mcaim.items_xset expander = ?where custkey = ?
4859472	306	0	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNM = ?, ALNETID = ?, ALTCPNAME = ?, ALSNANAME = ?
2896472	51	0	DELETE FROM QUSRBRM.QA1ANET2 WHERE RMTSYS NOT IN (SELECT R1MSYS FROM QUSRBRM.QA1A
2609232	1	0	delete from mcaim.items_xwhere custkey in (?, ?, ?)
2108976	306	0	SELECT * FROM QUSRBRM.QA1ASP WHERE SYNAME = ? FOR READ ONLY
1997440	1	0	delete from mcaim.items_xwhere custkey = ?
1908168	306	0	SELECT * FROM QUSRBRM.QA1AAL WHERE ALSYNM = ? AND ALNETID = ? FOR READ ONLY
1780000	306	0	SELECT * FROM QUSRBRM.QA1A1RMT WHERE R1MSYS = ? FOR READ ONLY
1518072	306	0	SELECT * FROM QUSRBRM.QA1AAL
1150888	51	0	INSERT INTO QUSRBRM.QA1A2NET SELECT DISTINCT RMTSYS AS N2RMT , RMTRNI AS N2RNI , OBJ AS N
1071752	2	0	select distinct suppkkey from orders where returnflag = upper(?)order by suppkkey
1005288	1	0	insert into mcaim.items_x(select * from star1.g.item_fact where custkey < ?)
997152	51	0	DELETE FROM QUSRBRM.QA1A2NET WITH NC
639672	5	0	-- Query 101 SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2.ORDERKEY, AL3.PART,
277600	2	0	select * from mcaim.items_x where returnflag = ?

Figure 6-23 Number of full opens and pseudo-opens

The total time in Figure 6-23 includes the full opens and pseudo-opens for each SQL request. To look at the time for full opens and pseudo-opens separately, you must add QVC12 to the GROUP BY clause from the previous query, as shown in Example 6-26.

Example 6-26 Number of full opens and pseudo-opens looked separately

```
SELECT SUM(qqi6) "Total Time (Mics) " ,
SUM(CASE WHEN qvc12 = 'N' THEN 1 ELSE 0 END) "Full Opens",
SUM(CASE WHEN qvc12 = 'Y' THEN 1 ELSE 0 END) "Pseudo Opens",
QQ1000L
FROM MYDBMON
WHERE qqrid=1000 and qqjnum = 'XXXXXX'
AND qqc21 IN ('OP', 'SI', 'DL', 'IN', 'UP')
GROUP BY qvc12,qq1000l
ORDER BY 1 DESC;
```

Figure 6-24 shows the results of the query in Example 6-26.

Total Time (Mics)	Full Opens	Pseudo Opens	QQ1000L
14193232	1	0	update mcain.items_xset expander = ?where returnflag = ?and year = ?and month = ?
13020152	1	0	update mcain.items_xset expander = ?where returnflag = ?
12268904	1	0	delete from mcain.items_x where returnflag = ?
8711680	1	0	insert into mcain.items_x(select * from star1g.item_fact where returnflag = ?)
4859472	306	0	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNM = ?, ALNETID = ?, ALTCPNAME = ?, ALSNANAME = ..
4038792	2	0	update mcain.items_xset expander = ?where custkey = ?
2896472	51	0	DELETE FROM QUSRBRM.QA1ANET2 WHERE RMTSYS NOT IN (SELECT R1MSYS FROM QUSRBRM.QA1...
2609232	1	0	delete from mcain.items_xwhere custkey in (?, ?, ?)
2108976	306	0	SELECT * FROM QUSRBRM.QA1ASP WHERE SYNAME = ? FOR READ ONLY
1997440	1	0	delete from mcain.items_xwhere custkey = ?
1994464	0	1	update mcain.items_xset expander = ?where custkey = ?
1908168	306	0	SELECT * FROM QUSRBRM.QA1AAL WHERE ALSYNM = ? AND ALNETID = ? FOR READ ONLY
1780000	306	0	SELECT * FROM QUSRBRM.QA1A1RMT WHERE R1MSYS = ? FOR READ ONLY
1518072	306	0	SELECT * FROM QUSRBRM.QA1AAL
1150888	51	0	INSERT INTO QUSRBRM.QA1A2NET SELECT DISTINCT RMTSYS AS N2RMT , RMTRNI AS N2RNI , OBJ AS ..
1071752	2	0	select distinct suppkkey from orders where returnflag = upper(?)order by suppkkey
1005288	1	0	insert into mcain.items_x(select * from star1g.item_fact where custkey < ?)
997152	51	0	DELETE FROM QUSRBRM.QA1A2NET WITH NC
639672	5	0	-- Query 101 SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2.ORDERKEY, AL3.PART, ...
357528	3	0	select *from orderswhere shipdate in (?, ?)
262600	51	0	SELECT * FROM QUSRBRM.QA1AAL FOR READ ONLY
240688	1	0	SELECT AL2.YEAR, AL2.MONTHNAME, AL2.WEEK, AL1.CUSTOMER, AL3.PART, AL4.QUANTITY, AL...
168552	4	0	select distinct returnflagfrom ordersorder by returnflag

Figure 6-24 Full Opens and Pseudo-opens shown separately

To analyze the full opens, copy and paste the SQL request or part of the SQL request into a query that used for analysis. From the previous query and result, we analyze the first SQL request which is the most expensive. The first SQL request is similar to the third request. Therefore, we use the LIKE predicate with two wild cards to ensure that we only retrieve information for the first SQL request. Example 6-27 shows the query.

Example 6-27 Looking at the number of full opens and pseudo-opens separately

```

SELECT qqc181 "Cursor Name",qqi6 "Exec Time (Mics) ", qqcnt, qqi5, qqc21,
qqc15 "HC Reason", qq10001 , qqc22 "Rebuild Reason Code",
qqc182 "Stmt Name"
FROM MYDBMON
WHERE qqjnum='999999' AND
(qqrid = 1000 and qqc21 in ('OP') AND
UCASE(qq10001) LIKE 'SELECT SUM(QUANTITY)%SHIPMODE =%')
OR (qqc21 IN ('HC', 'DI', 'ST', 'CM', 'RO') )
ORDER BY qqtime;

```

Figure 6-25 shows the result of the query in Example 6-27.

Cursor Name	Exec Time (Mics)	QQUCNT	QQI5	QQC21	HC Reason	QQ1000L	Rebuild Reason Code	Stmnt Nar
SQLCURSOR000000003	0	29	0	HC	6	HARD CLOSE	1 CURSORS	SQLSTAT
SQL CURSOR	0	30	0	HC	6	HARD CLOSE	1 CURSORS	SQLSTAT
SQLCURSOR000000003	0	31	0	HC	6	HARD CLOSE	1 CURSORS	SQLSTAT
SQLCURSOR000000003	0	32	0	HC	6	HARD CLOSE	1 CURSORS	SQLSTAT
CRSR0040	0	38	0	HC	6	HARD CLOSE	1 CURSORS	STMT004I
SQLCURSOR000000003	0	33	0	HC	6	HARD CLOSE	1 CURSORS	SQLSTAT
SQLCURSOR000000003	0	34	0	HC	6	HARD CLOSE	1 CURSORS	SQLSTAT
SQL CURSOR	0	35	0	HC	6	HARD CLOSE	1 CURSORS	SQLSTAT
CRSR0039	0	37	0	HC	6	HARD CLOSE	1 CURSORS	STMT003I
CRSR0038	0	36	0	HC	6	HARD CLOSE	1 CURSORS	STMT003I
CRSR0037	0	35	0	HC	6	HARD CLOSE	1 CURSORS	STMT003I
CRSR0036	0	34	0	HC	6	HARD CLOSE	1 CURSORS	STMT003I
CRSR0035	0	33	0	HC	6	HARD CLOSE	1 CURSORS	STMT003I
CRSR0034	0	32	0	HC	6	HARD CLOSE	1 CURSORS	STMT003I
CRSR0033	0	31	0	HC	6	HARD CLOSE	1 CURSORS	STMT003I
CRSR0032	0	30	0	HC	6	HARD CLOSE	1 CURSORS	STMT003I
CRSR0031	0	29	0	HC	6	HARD CLOSE	1 CURSORS	STMT003I
CRSR0030	0	28	0	HC	6	HARD CLOSE	1 CURSORS	STMT003I
CRSR0029	0	27	0	HC	6	HARD CLOSE	1 CURSORS	STMT002I

Figure 6-25 Analysis of the Full Opens

From the previous example, we can see that a Full Open took place the first time. The indication that it was a Full Open is the code in QQC21 and that QQI5 is 0. The Full Open took 1.5 seconds. The query did not go into reusable mode and the cursor was hard closed because of a cursor restriction (reason 6 in QQC15), this is a normal situation and it always happens on the first OPEN. The second Full Open took place again, but this time a hard close did not occur leaving the ODP to be reused. Subsequent executions reused the ODP. We can see this by looking at the QQUCNT and QQI5 fields. The number 8 in QQUCNT was assigned during the second Full Open and stayed constant for all subsequent instances of that query. QQI5 has the number assigned to each instance of the query. Notice that the execution time is minimum when the query entered into reusable mode.

For the complete list of statement types (QQC21) and the list of hard close reasons (QQC15), go to the V5R4 iSeries Information Center on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseres/v5r4/index.jsp>

Select Printable PDFs and manuals, when that list is displayed perform a search on performance. When the list performance of PDF manuals are displayed, select the Database Performance and Query Optimization manual and either download or open it to view.

When the manual is opened perform a search on QQC21 and choose from the results the reference to QQC21 Statement Operations.

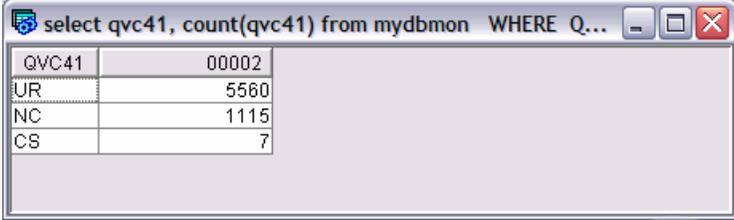
6.4.6 Isolation level used

You can see the number of statements that were run under each isolation level. This information provides you with a high level indication of the isolation level used. The higher the isolation level is, the higher the chance of contention is between users, which are seen as job locks and seizes. A high level of Repeatable Read or Read Stability use is likely to produce a high level of contention. Always use the lowest level isolation level that still satisfies the application design requirement as indicated in the query shown in Example 6-28.

Example 6-28 Isolation level summary

```
select qvc41, count(qvc41) from mydbmon  
WHERE QQRID = 1000  
GROUP BY qvc41 order by 2 desc;
```

Figure 6-26 shows the results of the query in Example 6-28.



QVC41	count(qvc41)
00002	
UR	5560
NC	1115
CS	7

Figure 6-26 Isolation level summary

The values can be translated using the following codes:

- RR** Repeating Read. In the SQL 1999 Core standard, Repeating Read is called *serializable*.
- RS** Read Stability. In the SQL 1999 Core standard, Read Stability is called *Repeatable Read*.
- CS** Cursor Stability. In the SQL 1999 Core standard, Cursor Stability is called *Read Committed*.
- CSKL** Cursor Stability KEEP LOCKS.
- UR** Uncommitted Read. In the SQL 1999 Core standard, Uncommitted Read is called *Read Uncommitted*.
- NC** No Commit.

As you can see from the previous example, most reads are done using Cursor Stability. This means that the isolation level is low, and therefore, the possibility for contention is low.

6.4.7 Table scan

A table scan operation is an efficient way to process all the rows in the table and verify that they satisfy the selection criteria specified in the query. Its efficiency is accomplished by bringing necessary data into main memory via a large I/O request and asynchronous prefetches.

The table scan is generally acceptable in cases where a large portion of the table is selected or the selected table contains a small number of records. To address cases where the entire table is scanned, but a relatively small number of rows is selected, building an index on the selection criteria is the best alternative and fully supported by the Database Monitor data.

Assuming that you have collected Detailed Database Monitor data, you can use the query shown in the Example 6-29 to see the statements that have resulted in table scan operations.

Example 6-29 Table scan operations

```
WITH tablescans AS (
  SELECT qqjfld,qqucnt,qqrest,qqtotr
  FROM MYDBMON
  WHERE qqrid=3000)
SELECT SUM(qqi6) "Total Time (Mics)", COUNT(*) "Times Run",
  a.qqucnt, integer(avg(b.qqrest)) "Est Rows Selected",
  integer(avg(b.qqtotr)) "Total Rows in Table", qq1000
FROM MYDBMON a, tablescans b
WHERE qqrid=1000 AND a.qqjfld=b.qqjfld AND
  qqc21 IN ('OP','SI','UP','IN','DL')
GROUP BY a.qqucnt, qq1000
ORDER BY "Total Time" DESC;
```

Figure 6-27 shows the results of the query in Example 6-29.

Total Time (Mics)	Times Run	QQUCNT	Est Rows Selected	Total Rows in Table	QQ1000
14193232	1	157	106821	12002430	update mcain.items_xset expander = ?where returnflag = ?and year = ?and
13020152	1	156	282056	12002430	update mcain.items_xset expander = ?where returnflag = ?
12268904	1	163	282048	12002100	delete from mcain.items_xwhere returnflag = ?
8711680	1	164	348070	12002430	insert into mcain.items_x(select * from star1 g.item_fact where returnflag = ?
5792944	102	1	1498	1499	DELETE FROM QUSRBRM.QA1ANET2 WHERE RMTSYS NOT IN (SELECT
3992096	2	159	1200242	12002430	update mcain.items_xset expander = ?where custkey = ?
2609232	1	162	292	12002430	delete from mcain.items_xwhere custkey in (?, ?, ?)
2280832	51	10	1	6	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNM = ?, ALNETID = ?,
2041160	1	158	1200242	12002430	update mcain.items_xset expander = ?where custkey = ?
1997440	1	161	97	12002430	delete from mcain.items_xwhere custkey = ?
1005288	1	182	7782	12002430	insert into mcain.items_x(select * from star1 g.item_fact where custkey < ?)
997152	51	2	16	16	DELETE FROM QUSRBRM.QA1A2NET WITH NC
722064	3	180	11667	12150	SELECT AL2.YEAR, AL2.MONTHNAME, AL2.WEEK, AL1.CUSTOMER, A
545040	1	75	60057	600572	select distinct suppkey from orders where returnflag = upper(?)order by sup
527472	51	15	1	6	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNM = ?, ALNETID = ?,
526712	1	150	60057	600572	select distinct suppkey from orders where returnflag = upper(?)order by sup
516240	51	30	1	6	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNM = ?, ALNETID = ?,
514704	51	20	1	6	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNM = ?, ALNETID = ?,
512896	51	35	1	6	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNM = ?, ALNETID = ?,
507328	51	25	1	6	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNM = ?, ALNETID = ?,
475256	1	174	20000	20000	-- Query 101 SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER
425240	51	7	0	1	SELECT * FROM QUSRBRM.QA1ASP WHERE SYNAME = ? FOR READ C
341240	51	27	0	1	SELECT * FROM QUSRBRM.QA1ASP WHERE SYNAME = ? FOR READ C

Figure 6-27 Table scan operations

In the previous query, notice the following columns:

- ▶ QQJFLD and QQUCNT

These columns are join fields that are required to uniquely identify an SQL statement in the Database Monitor file.

- ▶ QQC21

Since we are joining common table expression table scans back to the 1000 record in the Database Monitor file, we must ensure that we only join to 1000 records that can cause table scans to occur. We accomplish this by verifying that the QQC21 field operation is either open, select, update, delete, or insert. We also include the last three operations because they might have subselects or correlated subqueries.

► QQREST and QQTOTR

We included the QQREST (estimated rows selected) and QQTOTR (total rows in table) columns to give you an idea of the selectivity of the statement. A great difference between these two columns is a good indicator that index is a better alternative to a table scan operation.

► QQRID

A table scan operation is uniquely identified by the record ID (QQRID) value of 3000. We include it as selection criteria in the common table expression `tablescans`.

► QQI6

This column indicates a table scan operation with a cumulative elapsed time in microseconds for each individual query. Since we use it as a cost indicator, we ordered the output in descending order based on this value.

Note: Focus your efforts on optimizing statements that are displayed at the top. If a statement is run lots of times, then even a small performance improvement can improve the application performance. For other statements, consider the total number of rows in the table before taking further action. If this number is relatively small, your optimization efforts are best spent elsewhere.

6.4.8 Temporary index analysis

A *temporary index* is a temporary object that allows the optimizer to create and use a radix index for a specific query. The temporary index has all of the same attributes and benefits as a radix index that is created by a user through the CREATE INDEX SQL statement or Create Logical File (CRTL) CL command. The temporary index can be used to satisfy a variety of query requests, but it is only considered by the Classic Query Engine (CQE) when the query contains ordering, grouping, or joins.

The created indexes might only contain keys for rows that satisfy the query (such indexes are known as *sparse indexes* or *select/omit logical files*). In many cases, the index created might be perfectly normal and the most efficient way to perform the query.

Look at the query in Example 6-30, which list temporary index builds ordered by the entries in the index.

Example 6-30 Temporary index creation

```
SELECT qqcnt, qqc16 AS "QQC16 Index Reuse",
qqptfn AS "QQPTFN Table Name", qqptln AS "QQPTLN Schema",
qqtotr "QQTOTR Rows in Table", qqridx AS "QQRIDX Entries in Index",
qq1000 AS "Key Columns"
FROM MYDBMON
WHERE qqrid=3002
AND qqjnum = '999999'
ORDER BY 6 DESC;
```

Figure 6-28 shows the result of the query in Example 6-30.

QQUCNT	QQC16 Index Reuse	QQPTFN Table Name	QQPTLN Schema	QQTOTR Row in Table	QQRIDX Entries in Index	QQ1000 Key Columns
21	N	ORDERS	DBITSODATA	600572	600572	SHIPDATE ASC
96	N	ORDERS	DBITSODATA	600572	600572	SHIPDATE ASC
174	N	DATES	DBITSODATA	1450	1450	YEAR ASC, MONTH ASC, DATEKEY ...
176	N	DATES	DBITSODATA	1450	1450	YEAR ASC, MONTH ASC, DATEKEY ...
37	N	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
38	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
39	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
40	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
112	N	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
113	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
114	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
115	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
174	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC
176	Y	SUPPLIERS	DBITSODATA	1000	1000	SUPPKEY ASC

Figure 6-28 Temporary index creation

We explain some of the columns used in the previous query:

► QQUCNT

This column uniquely identifies a query for a given job. It includes QQJFLD to uniquely identify the query across many jobs.

► QQTFN and QQTLN

These columns indicate the short object names. Use QVQTBL and QVQLIB for long SQL table name and the long SQL schema name.

► QQRIDX

The number of entries in the temporary index.

Compare the total rows in the table with the entries in the temporary index to gauge whether the selection is built into a temporary index. If QQTOTR is greater than QQRIDX, then the selection is built into a temporary index.

► QQ1000

For this particular row type, the QQ1000 column contains key fields that are advised by the query optimizer to satisfy criteria for join, ordering, grouping, scroll able cursors, and so on. These are the reasons for the temporary index build. Therefore, use these keys in any permanent index that you are going to build.

In the list of key columns in column QQ1000, the optimizer lists what it considers the suggested primary and secondary key columns. *Primary key columns* are columns that should significantly reduce the number of keys selected based on the corresponding query selection. *Secondary key columns* are columns that might or might not significantly reduce the number of keys selected.

The optimizer can perform index scan-key positioning over any combination of the primary key columns, plus one additional secondary key column. Therefore it is important that the first secondary key column be the most selective secondary key column. The optimizer uses index scan-key selection with any of the remaining secondary key columns. While index scan-key selection is not as fast as index scan-key positioning, it can still reduce the number of keys selected. Therefore, be sure to include the secondary key columns that are fairly selective.

► QQRID

A row type of 3002 indicates a temporary index build so we have it in our selection criteria.

We modify the previous query to gather more information about the advised index and query optimizer reasons for building the temporary index. Example 6-31 shows the modified query.

Example 6-31 Reason for building a temporary index

```

WITH qq1000 AS (
  SELECT qqjfld, qqcnt, qq1000 FROM MYDBMON
  WHERE qqrid = 1000 AND qq21 IN ('OP','IN','UP','DL','SI'))
SELECT a.qqetim-a.qqstim "Index Build Time", a.qqrcod "Reason Code", a.qqtfn
"File", a.qqtlm "Library", a.qqtotr "Rows in Table", a.qqridx "Entries in Index",
SUBSTR(a.qq1000, 1, 100) "Key Fields", a.qqidxa "Index Advised", a.qqi2 "Nbr of
Primary Keys", SUBSTR(a.qqidxd, 1,100) "Keys Advised", a.qvc16 "Index from index",
b.qq1000
FROM MYDBMON a LEFT JOIN qq1000 b ON a.qqjfld = b.qqjfld AND a.qqcnt = b.qqcnt
WHERE a.qqrid=3002 ORDER BY "Index Build Time" DESC;

```

Figure 6-29 shows the output of the query in Example 6-31.

Index Build Time	Reason Code	File	Library	Rows in Table	Entries in Index	Key Fields
0.599111	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.088051	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.083495	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.082151	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.081717	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.081206	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.081094	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.079450	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.079108	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.078791	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.077822	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.077623	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.077294	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.077084	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.075771	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.075751	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.075376	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.075232	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.074913	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.074250	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.074019	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.073707	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,
0.073641	I2	OFFER ...	SCN2WCS	6316	6316	CATEN00001,

Figure 6-29 Reason for building temporary index

We joined our initial query back to record 1000 of the base Database Monitor table to obtain the SQL text for which the temporary index was built. The query uses the following columns:

- ▶ **QQ1000**
This column indicates that SQL statement that caused query optimizer to build a temporary index.
- ▶ **QQC21, QVC1C and QQC181**
When joining back to row type 1000, we care only about the operations that can cause query optimizer to advise an index. Therefore, we check for all the appropriate operation types contained in field QQC21. Additional criteria might be contained in the QVC1C field, SQL Statement Explainable, and QQC181, Cursor Name for the statement.

► QRCOD

This column indicates the reason code for an index build. You most commonly see the following reason codes:

- I2 Ordering/grouping
- I4 Nested loop join
- I3 Selection and ordering/grouping

For a detailed list of possible reason codes, search on Database Monitor qrcod 3002 in the V5R4 iSeries Information Center on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp>

► QI2

This column contains the number of suggested primary key columns that are listed in column QIDXD. These are the left-most suggested key columns. The remaining key columns are considered secondary key columns and are listed in order of expected selectivity based on the query. For example, assuming that QIDXK contains a value of 4 and QIDXD specifies seven key columns, then the first four key columns specified in QIDXK are the primary key columns. The remaining three key columns are the suggested secondary key columns.

► QVC16

This column indicates that a temporary index was built from an existing index, which is usually a short-running operation.

Note: Building permanent indexes to replace temporary indexes can provide great returns for a little time spent in analyzing Database Monitor data. Do not overuse this easy method for short running and non repetitive temporary index builds.

6.4.9 Index advised

The optimizer analyzes the row selection in the query and determines, based on default values, if the creation of a permanent index improves performance. If the optimizer determines that a permanent index might be beneficial, it returns the key columns necessary to create the suggested index. Advised indexes can be used to quickly tell if the optimizer recommends creating a specific permanent index to improve performance.

While creating an index that is advised typically improves performance, this is not a guarantee. After the index is created, much more accurate estimates of the actual costs are available. Based on this new information, the optimizer might decide that the cost of using the index is too high. Even if the optimizer does not use the index to implement the query, the new estimates available from the new index provide more detailed information to the optimizer that might produce better performance.

To look for the indexes advised by the optimizer, use the query shown in Example 6-32.

Example 6-32 Index advised

```
SELECT qqcnt, qvqtbl "Table Name", qvqlib "Schema",
       qqi2 "Nbr of Primary Keys",
       SUBSTR(qqidxd, 1,100) "Keys Advised"
FROM MYDBMON
WHERE qqrid IN (3000, 3001, 3002) and qqidxa='Y'
ORDER BY 5,2;
```

Figure 6-30 shows the results of the query in Example 6-32.

Figure 6-30 Index advised

This query uses the following columns:

- ▶ QQUCNT

This column uniquely identifies a query for a given job. Include QQJFLD to uniquely identify a query across many jobs.

- ▶ QVQTBL and QVQLIB

These columns indicate the long SQL table name and the long SQL schema name. Use QQTFN and QQTLN for the short object names.

- ▶ QQIDX

In the list of key columns contained in the QQIDX column, the optimizer has listed what it considers the suggested primary and secondary key columns. Primary key columns should significantly reduce the number of keys selected based on the corresponding query selection. Secondary key columns might or might not significantly reduce the number of keys selected.

The optimizer can perform index scan-key positioning over any combination of the primary key columns, plus one additional secondary key column. Therefore it is important that the first secondary key column is the most selective secondary key column. The optimizer uses index scan-key selection with any of the remaining secondary key columns. While index scan-key selection is not as fast as index scan-key positioning, it can reduce the number of keys selected. Therefore, include secondary key columns that are fairly selective.

- ▶ QQI2

This column contains the number of suggested primary key columns that are listed in the QQIDX column. These are the left-most suggested key columns. The remaining key columns are considered secondary key columns and are listed in order of expected selectivity based on the query. For example, assuming that QQIDXK contains a value of 4 and QQIDX specifies seven key columns, then the first 4 key columns specified in QQIDXK are the primary key columns. The remaining three key columns are the suggested secondary key columns.

► QQRID

Index advice is contained in three different row types and the query looks at all of them:

- 3000 - Table scan operation

We discussed the table scan operation previously as well as the need for indexes where a query is highly selective.

- 3001 - Index used

In cases where an existing index was used, the query optimizer might still recommend an index. A recommended index might be better than the selected index, but not always. Keep in mind that advised keys are for selection only and that you need to consider JOIN, ORDER BY, or GROUP BY clause criteria.

- 3002 - Temporary index created

For temporary indexes, we recommend that you use a different query altogether because we do not illustrate the QQ1000 column for row type 3002 in this query. In this case, QQ1000 includes keys used for a temporary index as well as their order (ascending or descending).

► QQIDXA

We specified Y in this column since we are interested only in the rows for which query optimizer has advised indexes.

Our initial query serves our need perfectly when we collect a very specific set of data (that is, a single job). However, if you perform a system-wide Database Monitor collection, you must use a query that is a bit more sophisticated.

We look at a query that enables us to perform costing by total run time required by the SQL statements that caused indexes to be advised. Example 6-33 shows this query.

Example 6-33 Costing of SQL statements where an index is advised

```
WITH qq1000 AS ( SELECT qqjfld, qqcnt, qq1000,
decimal(qq6/1000000,21,6) AS "Total Runtime (sec)"
FROM MYDBMON
WHERE qqrid = 1000 AND qq5 = 0 AND (qvc1c = 'Y' OR (qqc21 IN('DL', 'UP') AND
qqc181 <= ' ') OR
qqc21 IN ('IN', 'IC', 'SK', 'SI') OR qqc21 LIKE '0%'))
SELECT b."Total Runtime (sec)", a.qqtn "File Name",
a.qq1n "Library Name", a.qq2 "Nbr of Primary Keys",
substr(a.qqidxd,1,100) "Keys Advised", b.qq1000
FROM MYDBMON a LEFT JOIN qq1000 b ON
a.qqjfld = b.qqjfld AND a.qqcnt = b.qqcnt WHERE qqrid IN
(3000,3001,3002) AND qqidxa = 'Y' ORDER BY "Total Runtime (sec)" DESC;
```

Figure 6-31 shows the output of the query in Example 6-33.

Total Runtime (sec)	File Name	Library Name	Nbr of Primary Keys	Keys Advised	QQ1000
14.193232	ITEMS_X	MCAIN	3	MONTH, YEAR, RETURNFLAG	... update mcain.items_xset expander = ?where returnfl
13.020152	ITEMS_X	MCAIN	1	RETURNFLAG	... update mcain.items_xset expander = ?where returnfl
12.268904	ITEMS_X	MCAIN	1	RETURNFLAG	... delete from mcain.items_x where returnflag = ?
8.711680	ITEM_FACT	STAR1G	1	RETURNFLAG	... insert into mcain.items_x(select * from star1g.item_f
2.609232	ITEMS_X	MCAIN	1	CUSTKEY	... delete from mcain.items_xwhere custkey in (?, ?, ?)
2.041160	ITEMS_X	MCAIN	1	CUSTKEY	... update mcain.items_xset expander = ?where custkey
1.997632	ITEMS_X	MCAIN	1	CUSTKEY	... update mcain.items_xset expander = ?where custkey
1.997440	ITEMS_X	MCAIN	1	CUSTKEY	... delete from mcain.items_xwhere custkey = ?
0.240688	DATES	DBITSODATA	3	YEAR, WEEK, MONTHNAME	... SELECT AL2.YEAR, AL2.MONTHNAME, AL2.WEI
0.087288	QA1A1RMT	QUSRBRM	2	R1MACT, R1MSYS	... DELETE FROM QUSRBRM.QA1ANET2 WHERE RMT
0.086032	ORDERS	DBITSODATA	1	RETURNFLAG	... select * from orders where returnflag = ?
0.079248	QA1A1RMT	QUSRBRM	2	R1MACT, R1MSYS	... DELETE FROM QUSRBRM.QA1ANET2 WHERE RMT
0.075952	QA1A1RMT	QUSRBRM	2	R1MACT, R1MSYS	... DELETE FROM QUSRBRM.QA1ANET2 WHERE RMT
0.071696	QA1A1RMT	QUSRBRM	2	R1MACT, R1MSYS	... DELETE FROM QUSRBRM.QA1ANET2 WHERE RMT
0.068232	QA1A1RMT	QUSRBRM	2	R1MACT, R1MSYS	... DELETE FROM QUSRBRM.QA1ANET2 WHERE RMT
0.067976	QA1A1RMT	QUSRBRM	2	R1MACT, R1MSYS	... DELETE FROM QUSRBRM.QA1ANET2 WHERE RMT
0.067048	QA1A1RMT	QUSRBRM	2	R1MACT, R1MSYS	... DELETE FROM QUSRBRM.QA1ANET2 WHERE RMT
0.066728	QA1A1RMT	QUSRBRM	2	R1MACT, R1MSYS	... DELETE FROM QUSRBRM.QA1ANET2 WHERE RMT
0.065080	QA1A1RMT	QUSRBRM	2	R1MACT, R1MSYS	... DELETE FROM QUSRBRM.QA1ANET2 WHERE RMT
0.064560	QA1A1RMT	QUSRBRM	2	R1MACT, R1MSYS	... DELETE FROM QUSRBRM.QA1ANET2 WHERE RMT
0.064288	QA1A1RMT	QUSRBRM	2	R1MACT, R1MSYS	... DELETE FROM QUSRBRM.QA1ANET2 WHERE RMT
0.063752	QA1A1RMT	QUSRBRM	2	R1MACT, R1MSYS	... DELETE FROM QUSRBRM.QA1ANET2 WHERE RMT
0.062880	QA1A1RMT	QUSRBRM	2	R1MACT, R1MSYS	... DELETE FROM QUSRBRM.QA1ANET2 WHERE RMT

Figure 6-31 Costing of SQL statements where an index is advised

We joined our initial query back to record 1000 of the base Database Monitor table to obtain the SQL text and total statement run time. This query uses the following columns:

- QQ1000

This column indicates that an SQL statement caused query optimizer to advise an index.

- QQI6

This column indicates the total runtime required by the query, which is a good indicator of the index benefit for a table scan and temporary index build operations. For the existing index factors, such as join, grouping and ordering play into total runtime calculation, so this field might not be an accurate cost indicator.

- QQC21, QVC1C and QQC181

When joining back to row type 1000, we are concerned about only the operations that can cause the query optimizer to advise an index. Therefore, we check for all the appropriate operation types contained in column QQC21. Additional criteria might be contained in the QVC1C column, SQL Statement Explainable, and QQC181, Cursor Name for the statement.

The remainder of the statement is much like the initial query. One difference is that here we order by total runtime of the SQL statement, which provides us with a good costing indicator. This costing indicator helps us to focus on the worst-performing statements and build indexes intelligently.

You can see the number of different indexes that are advised and how many times they are advised. To see this information, run the query shown in Example 6-34.

Example 6-34 Listing of distinct index advised

```
SELECT distinct qqvtbl "Table", qqidxd "Key Fields",
       count(*) "Times advised"
FROM MYDBMON
WHERE qqrid IN (3000, 3001, 3002) and qqidxa='Y'
GROUP BY qqvtbl, qqidxd
ORDER BY qqvtbl, qqidxd;
```

Figure 6-32 shows the results of the query in Example 6-34.

QQVPTBL Table	QQIDX Key Fields	Times advised
ORDERS	SUPPKEY, RETURNFLAG, CUSTKEY	2
ORDERS	SUPPKEY, RETURNFLAG, SHIPDATE	3
ORDERS	SUPPKEY, SHIPDATE	6
ORDERS	YEAR	4
ORDERS	YEAR, MONTH	4
ORDERS	YEAR, PARTKEY	2
ORDERS	YEAR, QUANTITY	20
ORDERS	YEAR, RETURNFLAG	2
ORDERS	YEAR, RETURNFLAG, SUPPKEY	2
ORDERS	YEAR, SUPPKEY	2
PARTS	BRAND, MFG, PARTKEY	2
PARTS	PART, PARTKEY	4
PARTS	PARTKEY	4
PARTS	SIZE	6
QA1A1RMT	R1MACT, R1MSYS	51
QA1A1RMT	R1MSYS	306
SUPPLIERS	COUNTRY, SUPPKEY	2
SUPPLIERS	SUPPLIER, SUPPKEY	10
SYSIXADV	TABLE_SCHEMA	1

Figure 6-32 Different index advised

This is a good example where creation of the index on table QA1A1RMT with the key field, R1MSYS, is advised 306 times. There is also another index recommendation for the same table for key fields, R1MACT, R1MSYS, advised 51 times. Since the left-most key field in the two advised indexes are different, neither can be used in place of the other.

6.4.10 Access plan rebuilt

An access plan consists of one or more integrated steps (nodes) that are assembled to retrieve and massage data from DB2 tables to produce results that are desired by the information requestor. These steps might involve selecting, ordering, summarizing, and aggregating data elements from a single table or from related (joined) rows from multiple tables.

Each SQL query executes an access plan to retrieve the data that you requested. If the access plan does not exist already, the system builds one dynamically, adding overhead to the total time required to satisfy your request.

As a general rule, we want to avoid access plan rebuilds. That said, there are several perfectly valid reasons for access plan rebuilds, for example:

- ▶ Deleting or recreating the table to which the access plan refers
- ▶ Deleting an index that is used by the access plan
- ▶ Applying database PTFs
- ▶ Table size changing by 10%
- ▶ Creating a new statistic, automatically or manually
- ▶ Refreshing a statistic, automatically or manually
- ▶ Removing a stale statistic
- ▶ CQE rebuilding an access plan if there was a two-fold change in memory pool size
- ▶ SQL Query Engine (SQE) looking for a ten-fold change in memory pool size if memory pool is defined with a pool paging option of *CALC (also known as *expert cache*) (If paging is set to *FIXED, SQE behaves the same as CQE.)
- ▶ Specifying REOPTIMIZE_ACCESS_PLAN (*YES) in the QAQQINI table or in the SQL script
- ▶ Specifying REOPTIMIZE_ACCESS_PLAN (*FORCE) in the QAQQINI table or in the SQL script
- ▶ Changing the number of CPUs (whole or fractions using logical partition (LPAR)) that are available to a query results in rebuilding the access plan.
- ▶ Access plans are marked as invalid after an OS/400 release upgrade.
- ▶ The SQE Plan Cache is cleared when a system initial program load (IPL) is performed.

The SQE access plan rebuild activity takes place below the machine interface (MI). Therefore, compared to CQE, you should see much less performance degradation caused by lock contention on SQL packages, caches, and program objects.

At times, even though the optimizer rebuilt the access plan, the system fails to update the program object. The most common reason for this failure is that other jobs are using the same program and the optimizer cannot obtain the exclusive lock on the program object to save the rebuilt access plan. Another reason is that the job does not have proper authority to the program or the program is currently being saved. The query still runs, but access plan rebuilds continue to occur until the program is updated.

The rebuilt access plan might be saved in the existing access plan space within the program, SQL package, or cache. If a new access plan is greater than the existing one, new space is allocated and the plan is saved in that newly-allocated space. If the number of access plan rebuilds is high, some application redesign might be necessary.

We look at access plan rebuild information that is available in Database Monitor data by using the query shown in Example 6-35.

Example 6-35 Access plan rebuilds information

```
WITH rebuilds AS (
  SELECT qqjfld, qqcnt, qqrkod
  FROM MYDBMON
  WHERE qqrid=3006 )
SELECT a.qqcnt, b.qqrkod "Rebuild Reason",
       qvc24 "Plan Saved Status", qq1000
FROM MYDBMON a, rebuilds b
WHERE a.qqjfld=b.qqjfld AND a.qqrid=1000 AND
      a.qqc21 NOT IN ('MT','FE','CL','HC')
ORDER BY 4, 1;
```

Figure 6-33 shows the result of the query in Example 6-35.

QQUCNT	Rebuild Reason	Plan Saved Status	QQ1000
174	A4	B8	-- Query 101 SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2...
4	B3	B5	select * from customers where mktsegment = upper(?) order by customer
3	B3		select * from qsys2.sysixadv where table_schema = upper(?)
1	A9	B8	select distinct mktsegment from customers
3	A9	B8	select distinct shipmode from orders order by shipmode
75	B3		select distinct suppkay from orders where returnflag = upper(?)order by suppkay
11	B3		select i.year, i.month, upper(c.customer), sum(quantity)from orders i, custome...
11	B3		select i.year, i.month, upper(c.customer), sum(quantity)from orders i, custome...
194	B3		select i.year, i.month, upper(c.customer), sum(quantity)from orders i, custome...
9	B3		select i.year, upper(c.customer), sum(quantity)from orders i, customers cwher...
9	B3		select i.year, upper(c.customer), sum(quantity)from orders i, customers cwher...
192	B3		select i.year, upper(c.customer), sum(quantity)from orders i, customers cwher...
9	B3		SELECT *FROM ordersWHERE shipmode = upper(?)and linenumbr BETWEEN ...
172	B3		SELECT *FROM ordersWHERE shipmode = upper(?)and linenumbr BETWEEN ...
13	B3		SELECT DBXRDBN, DBXRDEV, DBXRMOD, DBXRMTN, DBXRLCL, DBXRNET, D...
24	B3		SELECT DBXRDBN, DBXRDEV, DBXRMOD, DBXRMTN, DBXRLCL, DBXRNET, D...
199	B3		SELECT DBXRDBN, DBXRDEV, DBXRMOD, DBXRMTN, DBXRLCL, DBXRNET, D...
12	B3		SELECT DBXRDBN, DBXRDEV, DBXRMOD, DBXRMTN, DBXRLCL, DBXRNET, D...
23	B3		SELECT DBXRDBN, DBXRDEV, DBXRMOD, DBXRMTN, DBXRLCL, DBXRNET, D...
198	B3		SELECT DBXRDBN, DBXRDEV, DBXRMOD, DBXRMTN, DBXRLCL, DBXRNET, D...

Figure 6-33 Access plan rebuilds information

This query uses the following columns:

► QQRID

Our common table expression rebuild contains only row type 3006, which has information specific to access plan rebuilds. Row type 3006 is *not* present on every full open. It is only generated when an access plan previously existed and now has to be rebuilt.

Row type 3006 is also not generated when SQE switches between cached access plans (up to three) in the SQE plan cache for an SQL statement.

► QQJFLD and QQUCNT

By now you know that QQJFLD and QQUCNT are join fields required to uniquely identify the SQL statement in the Database Monitor file.

► QQRKOD

This column provides the reason code for the access plan rebuild. There are over twenty possible reason codes. For a detailed description of the reason codes, search on Database Monitor qqrkod 3006 in the V5R4 iSeries Information Center. Click **Database monitor view 3006 - Access Plan Rebuild** to view all columns and scroll down to column QQRKOD on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseres/v5r4/index.jsp>

► QVC24

This column indicates the reason code for why the access plan was saved or not saved.

All A* codes mean that the access plan was *not* saved. AB means that a lock could not be obtained, A6-A9 means that not enough space was available, and AA means that a plan was saved by another job.

All B* codes mean that the access plan was saved, with a blank value or B3, B4, or B6 meaning that a plan was saved “in place”, and B1, B2, B5, B7, or B8 meaning that the plan was saved in a “new” space.

For a detailed description of each reason code, search on Database Monitor qvc24 1000 in the V5R4 iSeries Information Center. Click **Database monitor view 1000 - SQL Information** to view all columns and scroll down to column QVC24 on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp>

► QQ1000

This column indicates the SQL statement that caused query optimizer to rebuild the access plan.

► QQC21

In this query field, QQC21 is pulled from row type 1000 and represents the operation type for the SQL statement. We exclude continuation records, fetches, closes, and hard closes. The output is in ascending order based on the statement text and the unique statement counter. This query provides basic information about access plan rebuilds.

Let us rewrite the query to include more information related to access plan rebuilds as shown in Example 6-36.

Example 6-36 Extended access plan rebuild information

```
WITH rebuilds AS (  
  SELECT qqjfld, qqcnt, qqrcod, qqc21, qqc11, qqtim1 FROM MYDBMON  
  WHERE qqrid=3006 )  
  SELECT b.qqrcod "Rebuild Reason", hex(b.qqc21) "Reason Subcode (for IBM  
  debug)", a.qvc24 "Plan Saved Status", a.QQC103 "Package/Program Name",  
  a.QQC104 "Package/Program Library", a.qvc18 "Statement Type",  
  b.qqc11 "Plan Reoptimized", b.qqtim1 "Last Rebuilt", a.qq1000  
  FROM MYDBMON a, rebuilds b  
  WHERE a.qqjfld=b.qqjfld AND a.qqcnt = b.qqcnt and a.qqrid=1000  
  AND a.qqc21 NOT IN ('FE','CL','HC')  
  ORDER BY "Rebuild Reason", "Plan Saved Status";
```

Figure 6-34 shows the output of the query in Example 6-36.

Rebuild Reason	Reason Subcode (for IBM debug)	Plan Saved Status	Packa...	Package/...	Statement Ty
AB	0001				L
AB	0001	B8			L
A1	0001	B8			L
A1	0001	B8			L
A1	0001	B8			L
A1	0001	B8			L
A1	0001	B8			L
A4	0002				L
A4	0002	B8			L
A4	0002	B8			L
A4	0002	B8			L
A4	0001	B8			L
A4	0001	B8			L
A4	0001	B8			L
A5	0001	B8			L
A7	0842				L
A7	0842				L
A7	0842				L
A7	0842	B8			L
A7	0842	B8			L
A7	0001	B8			L
A7	0842	B8			L
A7	0004	B8			L

Figure 6-34 Extended access plan rebuild information

The query uses the following columns:

► QQC21

In this query, we pull the QQC21 column from row type 3006. In this row type, field QQC21 contains the access plan rebuild reason subcode. This subcode is in hex and should only be used when reporting a problem to IBM Support.

We still use QQC21 from row type 1000 to exclude undesired 1000 records, like we did in the initial query.

► QQC11

This column contains a simple character flag that indicates if an access plan required optimization. If the value is Y, then the plan was reoptimized. If the value is N, then the plan was not reoptimized. If this value is never Y, it is possible that QAQQINI file contains a REOPTIMIZE_ACCESS_PLAN setting that prevents the query optimizer from reoptimizing the access plan.

► QQTIM1

The value in this column indicates the time stamp of last access plan rebuild.

► QQC103

This column contains the name of the package or program that contains the SQL statement that caused query optimizer to rebuild the access plan.

► QQC104

This column indicates the name of the library that contains the program or package listed in the QQC103 column.

► QVC18

This column describes query statement type. If statement was done as part of dynamic processing, it is flagged with E for “extended dynamic”, with S for “system wide cache”, and L for “local prepared statement”.

6.4.11 Query sorting

The 3003 record from the Database Monitor table shows that the database optimizer has decided to place selected rows into a temporary space and sort them. The presence of a 3003 record does not necessarily indicate poor performance. The optimizer selected a query sort because it is either cheaper than the alternative indexed methods or it is forced to do so, for example when UNION is used or ORDER BY uses columns from more than one table.

Indexes can still be used to select or join rows before the sort occurs. The 3006 record does not indicate that the ODP is nonreusable.

Sort buffers are refilled and sorted at open time, even in reusable ODP mode.

Sorting might increase the open (OP) time or cost since sorting is often performed at open (OP) time. This means that it might take some time to return the first row in the result set to the user.

High elapsed times for a query sort might indicate a large answer set. In this case, the sort outperforms index usage (the situation in most cases). You should not attempt to build indexes for queries with large result sets, unless you are going to add selection criteria to the SQL statement's WHERE clause to further reduce the result set.

If the answer set is small, but the optimizer does not have the right indexes available to know that, creating indexes over selection columns can help by giving the optimizer statistics and an alternative method of accessing the data. This is possible only if the optimizer is not forced to use the sort (that is via a UNION or ORDER BY on columns from more than one table).

Look at which queries involve the use of a query sort. You can do a query sort by using the query shown in Example 6-37.

Example 6-37 Use of a query sort

```
WITH sorts AS (  
  SELECT qqjfld, qqcnt  
  FROM mydbmon  
  WHERE qqrid=3003 )  
SELECT SUM(qqi6) "Total Time" , COUNT(*) "Nbr Times Run", a.qqcnt, qq1000  
FROM mydbmon a, sorts b  
WHERE qqrid=1000 AND a.qqjfld=b.qqjfld and a.qqcnt=b.qqcnt AND  
  qqc21 IN ('OP','SI','UP','IN','DL')  
GROUP BY a.qqcnt, qq1000  
ORDER BY "Total Time" DESC;
```

Figure 6-35 shows the output of the query in Example 6-37.

Total Time	Nbr Times Run	QQUCNT	QQ1000
1150888	51	3	INSERT INTO QUSRBRM.QA1A2NET SELECT DISTINCT RMTSYS AS N2RMT , R...
545040	1	75	select distinct supkey from orders where returnflag = upper(?)order by supkey
526712	1	150	select distinct supkey from orders where returnflag = upper(?)order by supkey
475256	1	174	-- Query 101 SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2...
240688	1	180	SELECT AL2.YEAR, AL2.MONTHNAME, AL2.WEEK, AL1.CUSTOMER, AL3.PA...
115304	2	1	select distinct returnflagfrom ordersorder by returnflag
50312	1	55	-- Query 102 SELECT AL5.YEAR, AL5.MONTH, AL1.SALESPERSON, SUM(A...
46432	1	184	select distinct returnflagfrom ordersorder by returnflag
44920	1	54	-- Query 101 SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2...
44872	1	176	-- Query 101 SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2...
39888	1	170	SELECT AL4.SUPPLIER, AL5.YEAR, AL5.MONTHNAME, SUM(AL2.QUANTITY)FRO...
38512	1	173	-- Query 101 SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2...
37672	1	181	-- Query 104 SELECT AL2.YEAR, AL2.MONTHNAME, AL2.WEEK, AL1.CUST...
36112	1	129	-- Query 101 SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2...
35392	1	87	SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2.ORDERKEY, AL3.P...
32000	1	58	-- Query 105 SELECT AL5.YEAR, AL5.MONTH, AL5.MONTHNAME, AL3.PART...
29768	2	4	select distinct orderkeyfrom orderswhere custkey = ?order by orderkey
29544	1	2	SELECT AL5.YEAR, AL5.MONTH, AVG(AL2.DAYS_ORDER_TO_SHIP)FROM order...
29064	1	69	--- Query 10SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2.ORDER...
28976	1	53	-- Query 100 SELECT AL4.SUPPLIER, AL5.YEAR, AL5.MON...
27296	1	61	-- Query 108 SELECT AL2.YEAR, AL2.MONTH, AL1.CUSTOMER, SUM(AL2.R...
26296	1	68	SELECT AL5.YEAR, AL5.MONTH, AL1.CUSTOMER, SUM(AL2.QUANTITY) FROM o...
25320	1	63	-- Query 110 SELECT AL4.YEAR, AL4.MONTH, AL1.CUSTOMER, AL4.PARTK...
25312	1	142	SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOMER, AL2.ORDERKEY, AL3.P...

Figure 6-35 Use of a query sort

This query uses the following columns:

- ▶ **QQRID**
Our common table expression sort contains only row type 3003, which has information specific to SQL statements using query sorts.
- ▶ **QQJFLD and QQUCNT**
The QQJFLD and QQUCNT are join fields required to uniquely identify the SQL statement in the Database Monitor file.
- ▶ **QQC21**
Since we are joining common table expression sorts back to the 1000 record in the Database Monitor file, we must ensure that we only join to 1000 records that can cause query sorts to occur. This is accomplished by verifying that QQC21 field operation is either open, select, update, delete, or insert. We need to include last three operations because they might have subselects or correlated subqueries.
- ▶ **QQI6**
This column indicates a table scan operation cumulative elapse time, in microseconds for each individual query. Since we use it as a cost indicator, we have ordered the output in descending order based on this value.

The query that we have outlined so far is insufficient in helping us to decide if building an index or modifying the SQL statement is desired. Therefore, we revise the query (refer to Example 6-38) to include data that is necessary to make the decision if any action is possible.

Example 6-38 Including data showing possible action

```
WITH sorts AS (SELECT qqjfld, qqcnt FROM mydbmon WHERE qqrid=3003 ),
summation AS (SELECT SUM(qqi6) "Total Time" , COUNT(*) "Nbr Times Run", a.qqjfld,
a.qqcnt, qq1000 FROM mydbmon a, sorts b WHERE qqrid=1000 AND a.qqjfld=b.qqjfld
AND a.qqcnt = b.qqcnt and qqc21 IN ('OP','SI','UP','IN','DL') GROUP BY a.qqjfld,
```

```

a.qqcunt, qq1000), fetches AS (SELECT a.qqjfld, a.qqcunt, integer(avg(a.qqi3))
"Rows Fetched" FROM mydbmon a, summation b WHERE qqrid=1000 AND
a.qqjfld=b.qqjfld and a.qqcunt = b.qqcunt AND
qqc21 = 'FE' GROUP BY a.qqjfld, a.qqcunt) SELECT b."Total Time",
b."Nbr Times Run",
a.qqrcod "Reason Code", a.qqi7 "Reason subcode for Union",
a.qqrss "Number of rows sorted", c."Rows Fetched",
a.qqi1 "Size of Sort Space", a.qqi2 "Pool Size",
a.qqi3 "Pool ID", a.qvbndy "I/O or CPU bound", a.qqcunt, b.qq1000 FROM
summation b LEFT OUTER JOIN fetches c ON b.qqjfld = c.qqjfld AND
b.qqcunt = c.qqcunt INNER JOIN mydbmon a
ON b.qqjfld = a.qqjfld AND b.qqcunt = a.qqcunt WHERE a.qqrid = 3003
ORDER BY b."Total Time" DESC;

```

Figure 6-36 shows the output from the query in Example 6-38.

Total Time	Nbr Times Run	Reason Code	Reason subcode for Union	Number of rows sorted	Rows Fetched	Size of Sort Space	Pool Size
545040	1 F7		0	16271	1000	4299264	10734936064
526712	1 F7		0	16271	1000	4299264	10734936064
475256	1 F7		0	1	0	0	10734936064
240688	1 F7		0	3	237	0	10734936064
58232	1 F7		0	10	3	0	10734936064
57072	1 F7		0	10	3	0	10734936064
50312	1 F7		0	760	6	0	10734936064
46432	1 F7		0	10	3	0	10734936064
44920	1 F7		0	0	0	0	10734936064
44872	1 F7		0	1	1	0	10734936064
39888	1 F7		0	0	8	0	10734936064
38512	1 F7		0	0	0	0	10734936064
37672	1 F7		0	3	2	0	10734936064
36112	1 F7		0	0	0	0	10734936064
35392	1 F7		0	0	0	0	10734936064
33264	1 F7		0	16	-	0	10734936064
32000	1 F7		0	20444	0	0	10734936064
29544	1 F7		0	11	3	0	10734936064
29064	1 F7		0	0	0	0	10734936064
28976	1 F7		0	0	4	0	10734936064
28272	1 F7		0	16	-	0	10734936064
27888	1 F7		0	16	-	0	10734936064
27824	1 F7		0	16	-	0	10734936064

Figure 6-36 Include data showing action possible

This query uses the following columns:

- ▶ QQRCD

This column indicates the reason for choosing the query sort technique. The value in this column helps to identify whether the sort required of the query optimizer determined that the cost of the sort is better than any other implementation (such as an index).

If you can change the SQL statement itself, any reason code is available for optimization efforts. Or perhaps you cannot change the SQL statement (that is to optimize the third-party ERP application) and can only build indexes and change other environmental factors to help performance (that is, increase the pool size). In this case, focus your optimization efforts on query sorts with reason code F7 (optimizer chose sort rather than index due to performance reasons) and F8 (optimizer chose sort to minimize I/O wait time).

For a detailed description of each reason code, search on Database Monitor qqrcod 3003 in the V5R4 iSeries Information Center. Click **Database monitor view 3003 - Query sort** to view the columns and execute a **Find** to search for QQRCD on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp>

► QQI7

This column indicates the reason subcode for the UNION clause. If the query sort reason code lists F5 (UNION was specified for query), this column contains one of two subcodes. A value of 51 means that there is also an ORDER BY in the statement. A value of 52 means that the query specifies UNION ALL rather than simply UNION.

► QQRSS

This column tells us the number of rows that are contained in the sort space. You can use this value, along with the reason code, to determine if the indexed approach is possible and possibly cheaper (for a small result set). Compare the value of QQRSS with the value of QQI3 from the corresponding 1000 FE record for this open to determine the number of rows that were fetched from the sort space.

If the number of rows in sort space is large, but the actual number of rows fetched is small, consider adding OPTIMIZE FOR *n* ROWS to the query to help the optimizer make a better decision.

Building a more perfect index for the selection criteria might also help the optimizer make a better decision and use an index for the implementation method rather than a query sort.

► QQI3 from row type 1000 operation id FE (fetch)

This column tells us the number of rows that were fetched from the sort space to satisfy a user request. As described in the QQRSS column description, the value in this column is used to gauge whether more information is required by the query optimizer to make better costing decisions.

► QQI1

This column indicates the size of the sort space.

► QQI2

This column indicates the pool size.

► QQI3 from row type 3003

This column indicates the pool ID.

► QVBNDY

This column contains a flag that indicates whether the query sort is CPU or I/O bound.

We have taken the base query and modified it to include more information about the query sort implementation. This additional information helps you make more intelligent decisions when deciding to optimize SQL statements using query sorts as the implementation method. The most valuable new columns indicate a reason code and the number of actual rows fetched for the query.

Changing your SQL statement or adding the OPTIMIZE FOR *x* ROWS syntax is most likely to help alleviate issues that pertain to long query sort times. For highly selective queries where sort space is disproportionately larger than actual rows fetched, building a more perfect index might help the optimizer.

6.4.12 SQE advised statistics analysis

With the introduction of SQE to OS/400 V5R2, the collection of statistics was removed from the optimizer and is now handled by a separate component called the *Statistics Manager*. The Statistics Manager has two major functions:

- ▶ Create and maintain column statistics
- ▶ Answer questions that the optimizer asks when finding the best way to implement a given query

These answers can be derived from table header information, existing indexes, or single-column statistics. Single-column statistics provide estimates of column cardinality, most frequent values, and value ranges.

These values might have been previously available through an index, but statistics have the advantage of being precalculated and are stored with the table for faster access. Column statistics stored with a table do not dramatically increase the size of the table object. Statistics per column average only 8 to 12 KB in size. If none of these sources are available to provide a statistical answer, then the Statistics Manager bases the answer on default values (filter factors).

By default, this information is collected automatically by the system. You can manually control the collection of statistics by manipulating the QDBFSTCCOL system value or by using the iSeries Navigator graphical user interface (GUI). However, unlike indexes, statistics are not maintained immediately as data in the tables changes.

There are cases where the optimizer advises the creation of statistics. The query shown in Example 6-39 lists the optimizer advised statistics.

Example 6-39 Query optimizer advised statistics

```
SELECT  qqcnt,  qvqtbl "Table",  qvqlib "Schema",
        qqcl1 "Reason Stat Advised",
        SUBSTR(qq1000,1,100) "Column name"
FROM  mydbmon
WHERE  qqrid=3015
ORDER BY 2,5;
```

Figure 6-37 shows the output of the query in Example 6-39.

QQUCNT	QVQTBL Table	QVQLIB Schema	Reason Stat.Advised	Column name
16	CUSTOMERS	DBITSODATA	N	CONTINENT
91	CUSTOMERS	DBITSODATA	N	CONTINENT
16	CUSTOMERS	DBITSODATA	N	COUNTRY
91	CUSTOMERS	DBITSODATA	N	COUNTRY
3	CUSTOMERS	DBITSODATA	N	MKTSEGMENT
78	CUSTOMERS	DBITSODATA	N	MKTSEGMENT
186	CUSTOMERS	DBITSODATA	N	MKTSEGMENT
3	CUSTOMERS	DBITSODATA	N	MKTSEGMENT
55	CUSTOMERS	DBITSODATA	N	SALESPERSON
130	CUSTOMERS	DBITSODATA	N	SALESPERSON
177	CUSTOMERS	DBITSODATA	N	SALESPERSON
14	DATES	DBITSODATA	N	DATEKEY
89	DATES	DBITSODATA	N	DATEKEY
197	DATES	DBITSODATA	N	DATEKEY
14	DATES	DBITSODATA	N	MONTH
89	DATES	DBITSODATA	N	MONTH
197	DATES	DBITSODATA	N	MONTH
18	DATES	DBITSODATA	N	MONTHNAME
93	DATES	DBITSODATA	N	MONTHNAME
167	DATES	DBITSODATA	N	MONTHNAME

Figure 6-37 Query optimizer advised statistics

This query uses the following columns:

- ▶ **QQRID**
Our selection criteria selects only 3015 rows, which contain information exclusive to SQE advised statistics.
- ▶ **QQJFLD and QQUCNT**
QQJFLD and QQUCNT are join fields that are required to uniquely identify the SQL statement in the Database Monitor file.
- ▶ **QVQTBL and QVQLIB**
These columns refer to the long SQL table name and the long SQL schema name. Use QQTFN and QQTLN for short object names.
- ▶ **QQC11**
This column indicates the reason that the statistic was advised. This can happen only for two reasons, where N indicates a new statistic and S indicates a stale statistic. A statistic can become stale for several reasons; one of the most common reasons is that a base physical table's number of rows has changed by 15 percent.
- ▶ **QQ1000**
Column QQ1000 for row type 3015 contains the name of the column for which a statistic is advised. There might be multiple recommendations for a single SQL query, with each row containing a different column name in the QQ1000 column.

Keep in mind that column statistics are created in the background automatically by default for all advised statistics. Therefore, in general, no manual action is required on your end to build these statistics. The only exception is if the automatic statistics collection is turned off.

Although statistics provide a powerful mechanism for optimizing queries, do not underestimate and disregard the importance of implementing a sound indexing strategy. Well-defined indexes enable SQE to consistently provide efficient query optimization and performance. Statistics cannot be used to access a table or sort the data during query execution.

A good indexing strategy is both beneficial for providing statistics and mandatory for efficient and fast query execution. Therefore, you should replace indexes with statistics only if the indexes were created for the sole purpose of providing statistical information to the query optimizer. In cases where an index provides fast and efficient access to a set of rows in a table, DB2 Universal Database for iSeries continues to rely on its indexing technology to provide statistics information and a data access method.

Since indexes are the premier way to improve query optimizer intelligence and data access implementation choices, we look at the query in Example 6-40, which correlates SQE advised statistics with any query optimizer recommended indexes.

Example 6-40 Correlation of SQE statistics and optimizer recommended indexes

```
WITH
  advisedIndexes AS (
    SELECT qqjfld, qqcnt, qqi2 "Nbr of Primary Keys",
           SUBSTR(qqidxd, 1,100) "Keys Advised" FROM mydbmon
    WHERE qqrid IN (3000, 3001, 3002) and qqidxa='Y')
SELECT a.qqcnt, a.qqtn "Table", a.qqtn "Schema",
       CASE a.qqc11
         WHEN 'N' THEN 'No Statistic Exists'
         WHEN 'S' THEN 'Stale Statistic Exists'
         ELSE 'Unknown'
       END AS "Reason Stat Advised", SUBSTR(a.qq1000,1,100) "Column name",
       a.qqi2 "Statistics Importance", a.qvc1000 "Statistics Identifier",
       b."Nbr of Primary Keys", b."Keys Advised"
FROM mydbmon a LEFT OUTER JOIN advisedIndexes b
  on a.qqjfld = b.qqjfld and a.qqcnt = b.qqcnt
WHERE qqrid=3015 ORDER BY a.qqcnt DESC;
```

Figure 6-38 shows the output of the query in Example 6-40.

QQUCNT	Table	Schema	Reason Stat Advised	Column name	Statistics Importance	Statistics Identifier
197	DATES	DBITSODATA	No Statistic Exists	MONTH	0	
197	DATES	DBITSODATA	No Statistic Exists	YEAR	0	
197	DATES	DBITSODATA	No Statistic Exists	QUARTER	0	
197	DATES	DBITSODATA	No Statistic Exists	DATEKEY	0	
197	ORDERS	DBITSODATA	No Statistic Exists	SHIPDATE	0	
193	ORDERS	DBITSODATA	No Statistic Exists	MONTH	0	
190	ORDERS	DBITSODATA	No Statistic Exists	QUARTER	0	
189	ORDERS	DBITSODATA	No Statistic Exists	QUANTITY	0	
189	ORDERS	DBITSODATA	No Statistic Exists	YEAR	0	
189	ORDERS	DBITSODATA	No Statistic Exists	YEAR	0	
188	ORDERS	DBITSODATA	No Statistic Exists	QUANTITY	0	
188	ORDERS	DBITSODATA	No Statistic Exists	YEAR	0	
188	ORDERS	DBITSODATA	No Statistic Exists	YEAR	0	
187	ORDERS	DBITSODATA	No Statistic Exists	ORDERKEY	0	
187	ORDERS	DBITSODATA	No Statistic Exists	ORDERKEY	0	
187	ORDERS	DBITSODATA	No Statistic Exists	ORDERKEY	0	
186	CUSTOMER...	DBITSODATA	No Statistic Exists	MKTSEGMENT	0	
185	ORDERS	DBITSODATA	No Statistic Exists	SHIPMODE	0	
184	ORDERS	DBITSODATA	No Statistic Exists	RETURNFLAG	0	
177	CUSTOMER...	DBITSODATA	No Statistic Exists	SALESPERSON	0	
177	CUSTOMER...	DBITSODATA	No Statistic Exists	SALESPERSON	0	
171	ORDERS	DBITSODATA	No Statistic Exists	ORDERDATE	0	
171	ORDERS	DBITSODATA	No Statistic Exists	LINENUMBER	0	

Figure 6-38 Correlation of SQE statistic and optimizer recommended indexes

This query uses the following columns:

- ▶ QQIDXA

This column contains a flag of Y or N that indicates whether index was advised. We use this information to filter out queries for which no index was recommended by the query optimizer (QQIDXA = 'Y').

- ▶ QQIDXD

This column indicates columns for which an index was advised. This field lists primary keys first followed by zero or more secondary keys. Secondary selection keys are less likely to have a significant positive impact on a query's performance.

- ▶ QQI2

This column indicates the number of primary keys contained in the QQIDXD field. For the most benefit, build an index over the primary key fields.

- ▶ QQRID

We only focus on row types 3000, 3001, and 3002, which contain query optimizer index suggestions.

- ▶ QQI2

This column indicates the importance of a statistic.

- ▶ QVC1000

This column contains the statistics identifier.

This query attempts to correlate the SQE advised statistics to the query optimizer index suggestions. The idea is that we should attempt to build indexes for cases where an index can be used by the query optimizer for the data access method.

If an index is solely used for statistical purposes, the advantage should be given to SQE statistics due to their low maintenance overhead. An exception to this recommendation is if statistics must be current at all times. The only way to accomplish this is by having an index set with the *IMMED maintenance attribute.

6.4.13 Fetched and Retrieved detail rows

Specifying *DETAIL in the TYPE parameter of the STRDBMON command indicates that detail rows, as well as summary rows, must be collected for fetch operations. The same is true for detailed SQL Performance Monitor in iSeries Navigator.

The 3019 row can be used to indicate the amount of time SQL and non-SQL queries are spending in fetching and retrieving data. Non-SQL queries do not generate record 1000 rows and therefore no SQL statement. An example of such non-SQL queries are OPNQRYF, Query/400 and QQQQry API. For non-SQL queries, the only way to determine the number of rows that are returned and the total time elapsed to return those rows is to collect detail rows. While the detail row contains valuable information, it creates a slight performance degradation for each block of rows returned. Therefore, you must closely monitor its use. You can use a detailed row for SQL analysis since the information it contains is also valuable in the SQL environment.

A large number of physical I/O operations can indicate that a larger pool is necessary or that SETOBJACC might be used to bring some of the data into main memory beforehand.

The query in Example 6-41 shows the most time-consuming SQL statements.

Example 6-41 Most time -consuming SQL statements

```
WITH retrieved AS (  
  SELECT  qqjfld, qqi3, qqi5  
  FROM mydbmon  
  WHERE   qqrid=3019)  
SELECT  SUM(qqi6) "QI6 Total Time" , COUNT(*) "Nbr Times Run",  
        SUM(b.qqi3) "QI3 Sync DB Reads", SUM(b.qqi5) "QI5 ASync DB Reads",  
qq1000,  
        qqc11, qqc21  
FROM mydbmon a, retrieved b  
WHERE   a.qqjfld=b.qqjfld AND qqrid=1000 AND qqcnt<>0  
GROUP BY qq1000, qqc11, qqc21  
ORDER BY 1 DESC;
```

Figure 6-39 shows the output of the query in Example 6-41.

QQI6 Total Time	Nbr Times Run	QQI3 Sync DB Rea...	QQI5 ASync DB Reads	QQ1000	QQC11	QQC21
61885840	1286	21	27139		S	FE
14193232	1	1	23	update mcaim.items_xset expander = ?where returnflag = ?and year = ?an...	U	UP
14017448	7	6	148	update mcaim.items_xset expander = ?where custkey = ?	U	UP
13020152	1	3	71	update mcaim.items_xset expander = ?where returnflag = ?	U	UP
12268904	1	1	120	delete from mcaim.items_x where returnflag = ?	D	DL
8711680	1	0	0	insert into mcaim.items_x(select * from star1g.item_fact where returnflag = ...	I	IN
4878608	308	0	0	UPDATE QUSRBRM.QA1AAL SET ALRGRP = ?, ALSYNM = ?, ALNETID = ...	U	UP
2896472	51	0	0	DELETE FROM QUSRBRM.QA1ANET2 WHERE RMTSYS NOT IN (SELEC...	D	DL
2609232	1	2	77	delete from mcaim.items_xwhere custkey in (?, ?, ?)	D	DL
2108976	306	0	0	SELECT * FROM QUSRBRM.QA1ASP WHERE SYNAME = ? FOR READ ...	S	OP
1997440	1	1	81	delete from mcaim.items_xwhere custkey = ?	D	DL
1908168	306	0	0	SELECT * FROM QUSRBRM.QA1AAL WHERE ALSYNM = ? AND ALNE...	S	OP
1780000	306	0	0	SELECT * FROM QUSRBRM.QA1A1RMT WHERE R1MSYS = ? FOR RE...	S	OP
1598464	3	0	0	select distinct suppkey from orders where returnflag = upper(?)order by s...	S	OP
1195912	8	0	0	-- Query 101 SELECT AL5.YEAR, AL5.MONTHNAME, AL1.CUSTOME...	S	OP
1150888	51	0	0	INSERT INTO QUSRBRM.QA1A2NET SELECT DISTINCT RMTSYS AS N2...	I	IN
1005288	1	7	1092	insert into mcaim.items_x(select * from star1g.item_fact where custkey < ?)	I	IN
997152	51	0	0	DELETE FROM QUSRBRM.QA1A2NET WITH NC	D	DL
562224	970	0	0	CLOSE SQLCURSOR000000003	S	CL
395288	4	1	0	select *from orderswhere shipdate in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?...	S	OP
262600	51	0	0	SELECT * FROM QUSRBRM.QA1AAL FOR READ ONLY	S	OP
240688	1	0	0	SELECT AL2.YEAR, AL2.MONTHNAME, AL2.WEEK, AL1.CUSTOMER,...	S	OP
175368	5	0	0	select distinct returnflagfrom ordersorder by returnflag	S	OP

Figure 6-39 Most time consuming SQL statements

This query uses the following columns:

- ▶ **QQRID**
In the common table expression retrieved, we select only 3019 rows, getting a subset of data with detailed row information.
- ▶ **QQJFLD**
This column indicates the join column (unique per job).
- ▶ **QQI3**
This column indicates the number of synchronous database reads. We present a cumulative count for each SQL statement.
- ▶ **QQI5**
This column indicates the number of asynchronous database reads. We present a cumulative count for each SQL statement.
- ▶ **QQI6 from row type 1000**
This column indicates the cumulative elapse time in microseconds for each individual query. Since we use it as a cost indicator, we ordered the output in descending order based on this value.
- ▶ **QQ1000**
This column contains the SQL statement.
- ▶ **QQUCNT**
This column contains the unique statement identifier. We use this value to exclude non-unique statement identifiers.
- ▶ **QQC21**
This column contains the SQL request operation identifier. We use this value to exclude continuation records from our analysis. Continuation records are used to display statement text for statements that cannot fit into the single QQ1000 field.

This query gives us detailed information about the amount of reads that the longest running SQL statements have performed.

Row 3019 contains other interesting statistics. We view them by running the query shown in Example 6-42.

Example 6-42 Row 3019 statistics

```

WITH retrieved AS (
  SELECT qqjfld,qqi1,qqi2,qqi3,qqi5, qqi4,qqi6,qqi7,qqi8
  FROM mydbmon
  WHERE qqrid=3019)
SELECT SUM(a.qqi6) "Total Time" , COUNT(*) "Nbr Times Run",
       SUM(b.qqi1) "CPU time in milliseconds", SUM(b.qqi2) "Clock Time in
milliseconds",      SUM(b.qqi3) "Sync DB Reads", SUM(b.qqi5) "Async DB Reads",
       SUM(b.qqi4) "Sync DB Writes", SUM(b.qqi6) "Async DB Writes",
       SUM(b.qqi7) "Number of rows returned",
       SUM(b.qqi8) "Nbr of calls to get rows", qq1000
FROM mydbmon a, retrieved b
WHERE a.qqjfld=b.qqjfld AND qqrid=1000 AND qqcnt<>0
GROUP BY qq1000
ORDER BY "Total Time" DESC;

```

Figure 6-40 shows the output of the query in Example 6-42 (we show two windows).

Total Time	Nbr Times Run	CPU time in millisecond...	Clock Time in milliseconds	Sync DB Reads	Async DB Reads
61885840	1286	15816	39585	21	27139
14193232	1	5178	9605	1	23
14017448	7	5546	9977	6	148
13020152	1	5010	8777	3	71
12268904	1	4773	8414	1	120
8711680	1	1543	1984	0	0
4878608	308	10	14	0	0
2896472	51	93	125	0	0
2609232	1	1447	2515	2	77
2108976	306	8	16	0	0
1997440	1	1114	1965	1	81
1908168	306	4	16	0	0
1780000	306	3	19	0	0
1598464	3	664	1011	0	0
1195912	8	14	22	0	0
1150888	51	3	5	0	0
1005288	1	112	782	7	1092
997152	51	6	12	0	0
562224	970	16	55	0	0
395288	4	9	20	1	0
262600	51	1	4	0	0
240688	1	267	420	0	0
175368	5	0	0	0	0

Figure 6-40 Row 3019 statistics - left part of results

To show all of the columns, the left part of the results is shown in Figure 6-40 and the right part is shown in Figure 6-41.

Sync DB Writes	Async DB Writes	Number of rows returned	Nbr of calls to get rows	QQ1000
1	37	46673	1662	update mcain.items_xset expander = ?wf
0	0	372405	372405	update mcain.items_xset expander = ?wf
0	0	493	493	update mcain.items_xset expander = ?wf
0	0	325669	325669	update mcain.items_xset expander = ?wf
0	0	325667	325667	delete from mcain.items_x where returnfla
0	94	325668	1545	insert into mcain.items_x(select * from st
0	0	616	616	UPDATE QUSRBRM.QA1AAL SET ALRGRF
0	0	51	51	DELETE FROM QUSRBRM.QA1ANET2 W
0	0	331	331	delete from mcain.items_xwhere custkey
0	0	306	306	SELECT * FROM QUSRBRM.QA1ASP WF
0	0	1	1	delete from mcain.items_xwhere custkey
0	0	306	306	SELECT * FROM QUSRBRM.QA1AAL WH
0	0	306	306	SELECT * FROM QUSRBRM.QA1A1RMT
0	0	40	18	select distinct suppkkey from orders where
1	2	1	5	-- Query 101 SELECT AL5.YEAR, AL
0	0	816	102	INSERT INTO QUSRBRM.QA1A2NET SEL
0	3	7782	38	insert into mcain.items_x(select * from st
0	0	867	867	DELETE FROM QUSRBRM.QA1A2NET W
0	0	1225	1276	CLOSE SQLCURSOR000000003
0	0	336	3	select *from orderswhere shipdate in (?,?
0	0	306	357	SELECT * FROM QUSRBRM.QA1AAL FO
0	0	237	1	SELECT AL2.YEAR, AL2.MONTHNAM
0	0	12	4	select distinct returnflafrom ordersorder

Figure 6-41 Row 3019 statistics - right part of results

This query uses the following columns:

Note: In the following columns, we present cumulative values for each SQL statement that generated the 3019 record. A statement is deemed unique as long as it can fit in the single QQ1000 record.

- ▶ QQI1
This column indicates the CPU time in milliseconds.
- ▶ QQI2
This column indicates the clock time in milliseconds.
- ▶ QQI4
This column indicates the number of synchronous database writes.
- ▶ QQI6
This column indicates the number of asynchronous database writes.
- ▶ QQI7
This column indicates the number of rows returned.
- ▶ QQI8
This column indicates the number of calls to retrieve rows returned.

6.4.14 Materialized query tables

Materialized query tables (MQTs), also referred to as *automatic summary tables* or *materialized views*, can provide performance enhancements for queries. This enhancement is done by precomputing and storing results of a query in the MQT. The database engine can

use these results instead of recomputing them for a user-specified query. The query optimizer looks for any applicable MQTs and can choose to implement the query using a given MQT provided this is a faster implementation choice. For more details on this topic refer to 3.7.2, “MQT (Materialized Query Tables) Evaluator” on page 70. You can also find more information in the white paper *The creation and use of materialized query tables within IBM DB2 for i5/OS*, which is available from the DB2 for i5/OS Web site at:

<http://www.ibm.com/series/db2>

Click the Key Benefits tab to view additional options on the tab. Click **Articles & White Papers** on to view the list of available white papers.

As a SQL performance analyst it is interesting to determine whether an MQT is being used. Example 6-43 shows a query that you can run to view the reasons why an MQT is not being used. You can also check whether MQTs are replacing the existing table names in the query. You can check the QQC13 column of either the 3000, 3001, or 3002 record of the SQL Performance Monitor data. The 3000 record is for use when a table scan is done. The 3001 record is for use when an index is used, and the 3002 record is for when an index is created.

To look at the MQTs on your system and see if they are being used, run the following Display File Description (DSPFD) command and SQL Performance Monitor data to check. First you must determine which tables are MQTs. Create a file with the following command:

```
SELECT TABLE_NAME, TABLE_SCHEMA from QSYS2.SYSTABLES
WHERE TABLE_TYPE = 'M'
```

Then to see which MQTs are being used or are not in queries, run the query shown in Example 6-43 against the file created from the DSPFD command, using SQL performance data.

Example 6-43 Query to find MQTs

```
WITH MQTTABLES as (
    SELECT TABLE_NAME, TABLE_SCHEMA from QSYS2.SYSTABLES
    WHERE TABLE_TYPE = 'M')
SELECT count(*) as Times_Used,
    a.TABLE_NAME as MQT_file,
    a.TABLE_SCHEMA as MQT_library,
    b.qqc13 as Used_in_Query
FROM MQTTABLES a left outer join mydbmon b
    ON a.TABLE_NAME = b.qqtfn and a.TABLE_SCHEMA = b.qqtln
GROUP BY a.TABLE_NAME, a.TABLE_SCHEMA, b.qqc13
ORDER BY b.qqc13
```

Next, to determine why MQTs are not being used, run the query shown in Example 6-44.

Example 6-44 Reason why MQTs are not used

```
SELECT substr(a.qq1000,1,100) AS MQT_RC,b.qq1000
FROM mydbmon a, mydbmon b
WHERE a.qqrid = 3030
    AND b.qqrid = 1000
    AND a.qqjfld = b.qqjfld
    AND a.qq1000 <> '0'
    AND b.qqc11 in ('S','U','I','D')
    AND b.qqc21 NOT IN ('CL', 'HC', 'FE')
GROUP BY B.QQ1000, A.QQ1000
```

Archived

SQE Plan Cache and SQE Plan Cache Snapshots

In this chapter we discuss SQE Plan Cache and SQE Plan Cache Snapshots. We provide you with the understanding of what an SQE Plan Cache is and show you what are the possible functions you can perform on SQE Plan Cache. We also show you how to create a snapshot out of the SQE Plan Cache for further SQL optimization analysis. You can create the snapshot either through the iSeries Navigator GUI interface, through calling a Stored Procedure or even through an IBM Exit Program so that the plan cache is dumped at each IPL.

7.1 SQE Plan Cache and SQE Plan Cache Snapshot

Plan Cache and Plan Cache Snapshot are newly introduced in V5R4. In this chapter, we discuss them both further, providing you with a better understanding of what Plan Cache is, what a Plan Cache Snapshot is and how they can integrate with existing iSeries Navigator tools or even new tools in V5R4, to assist you in optimizing your queries.

7.2 SQE Plan Cache

The SQE Plan Cache is a repository that contains the access plans for queries that were optimized by SQE. The purpose of the SQE Plan Cache is to:

- ▶ Facilitate the reuse of a query access plan when the same query is re-executed
- ▶ Store runtime information for subsequent use in future query optimizations

For a more detailed description of What is the SQE Plan Cache, refer to 2.2.9, “SQE Plan Cache”.

7.2.1 Viewing the properties of the SQE Plan Cache

The Plan Cache contains a wealth of information about the SQE queries being run through the database. Its contents are viewable through the iSeries Navigator GUI interface.

Note: SQE Plan Cache is named SQL Plan Cache in the iSeries Navigator GUI interfaces.

This SQE Plan Cache interface provides a window into the database query operations on the system. The interface to the SQE Plan Cache resides under the **iSeries Navigator** → **system name** → **Database** as shown in Figure 7-1.

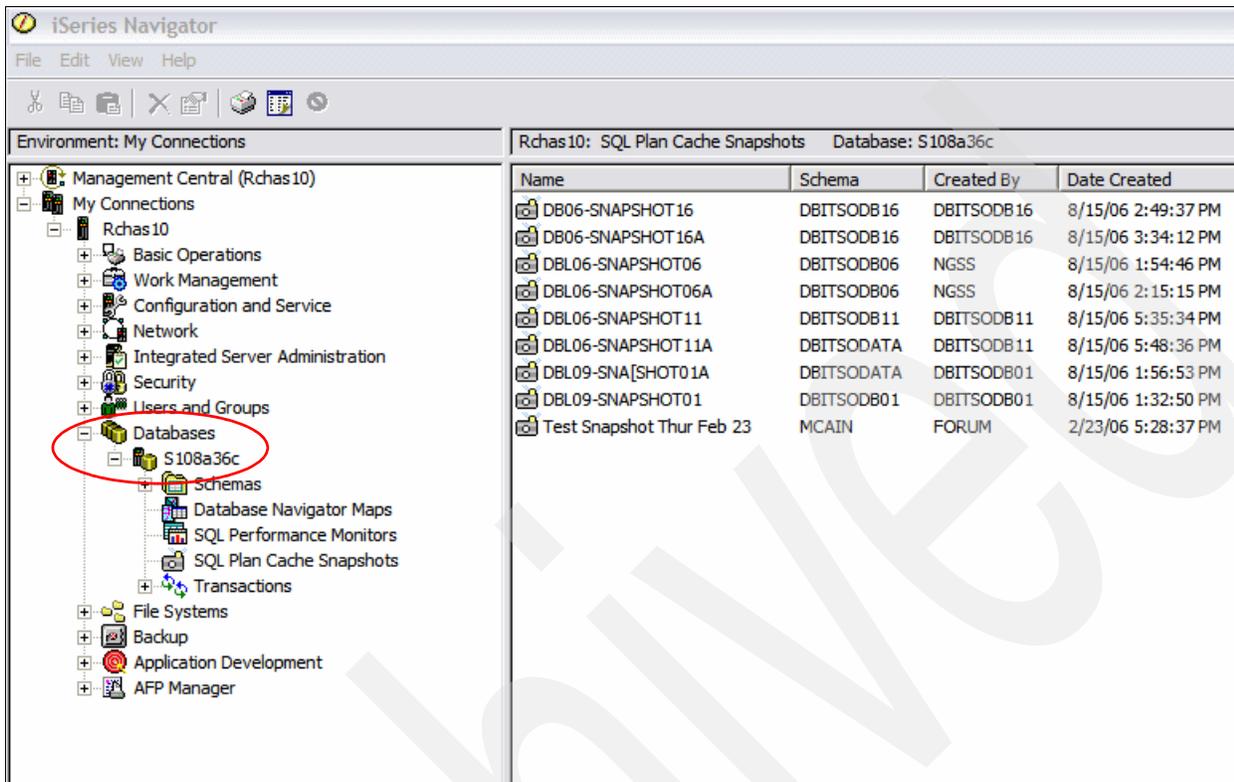


Figure 7-1 View SQE Plan Cache

As shown in Figure 7-2, select **iSeries Navigator** → **system name** → **Database** → **SQL Plan Cache Snapshot** → **SQL Plan Cache** → **Properties** to see the Plan Cache Properties window.

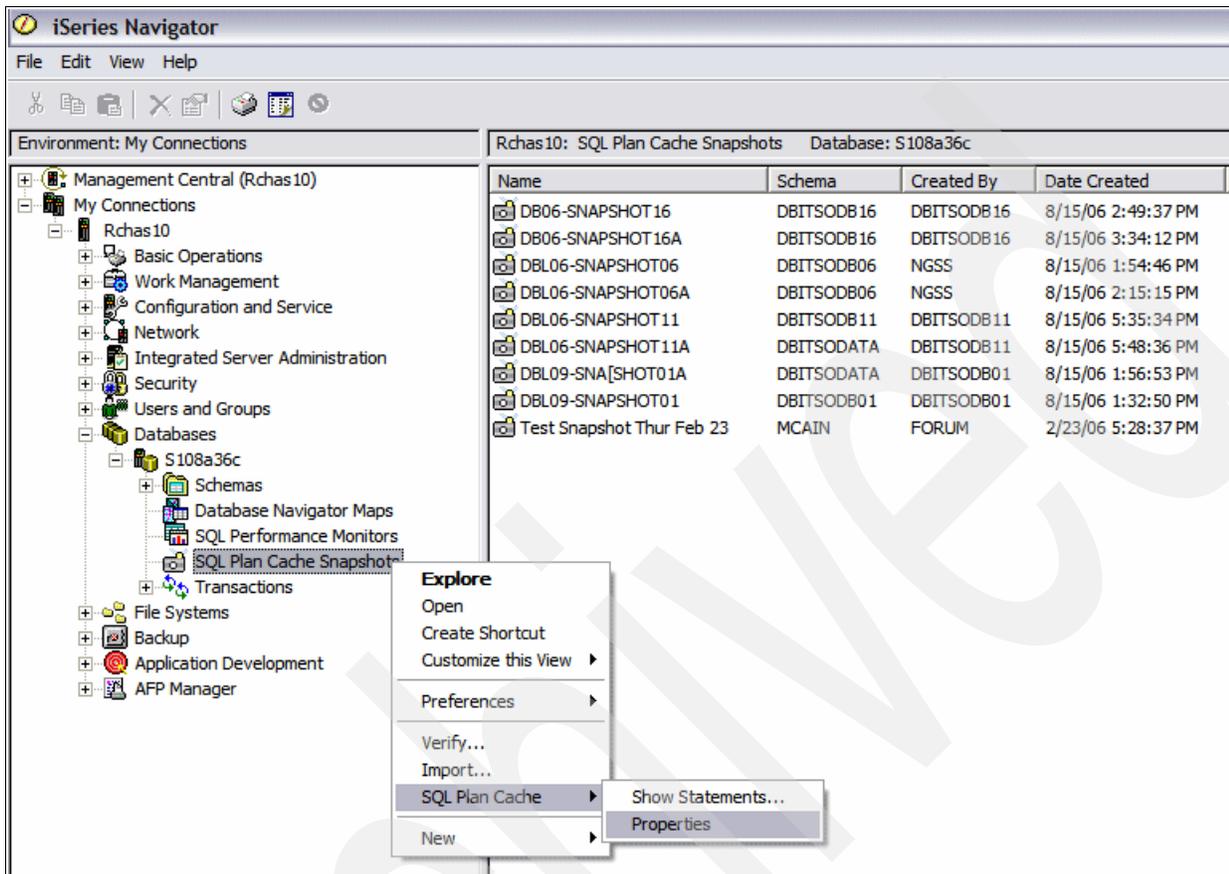
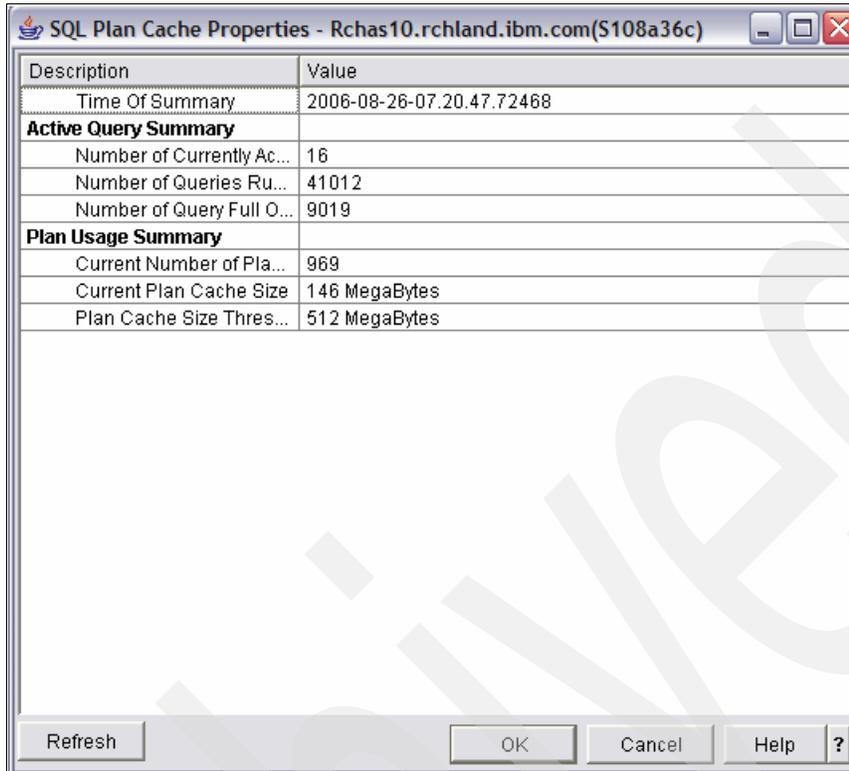


Figure 7-2 Path to SQE Plan Cache Properties

As shown in Figure 7-3, the Plan Cache Properties window shows high level information about the cache, including for example, cache size, number of plans, number of full open and pseudo-opens that have occurred. This information can be used to view overall database

activity. If tracked over time, it provides trends to help you better understand the database utilization peaks and valleys throughout the day and week.



Description	Value
Time Of Summary	2006-08-26-07.20.47.72468
Active Query Summary	
Number of Currently Ac...	16
Number of Queries Ru...	41012
Number of Query Full O...	9019
Plan Usage Summary	
Current Number of Pla...	969
Current Plan Cache Size	146 MegaBytes
Plan Cache Size Thres...	512 MegaBytes

Refresh OK Cancel Help ?

Figure 7-3 SQE Plan Cache Properties

7.2.2 Viewing the content of the SQE Plan Cache

As shown in Figure 7-4, by right-clicking the SQE Plan Cache icon, a series of options are shown which allow different views of current SQE Plan Cache of the database.

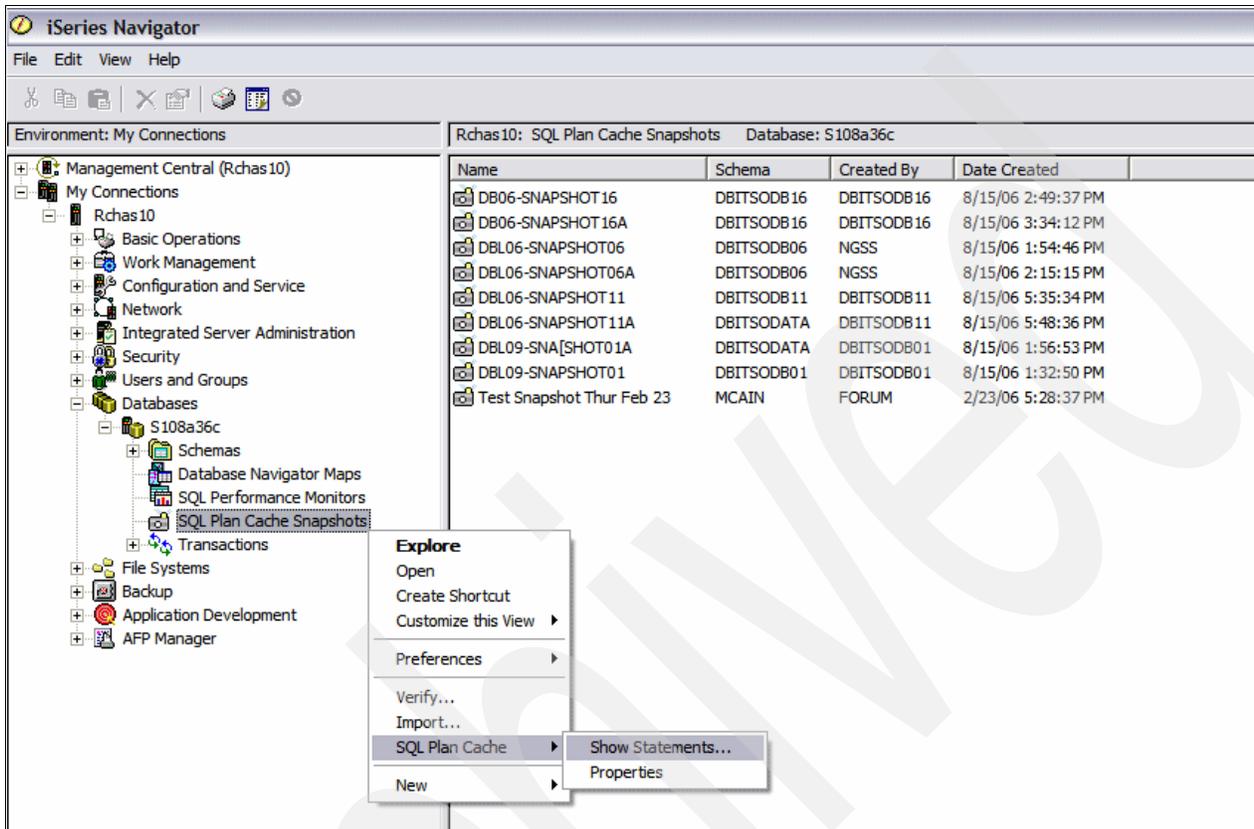


Figure 7-4 Path to SQE Plan Cache Show Statements

From Figure 7-4, select **SQL Plan Cache** → **Show Statements** to bring up a window (shown in Figure 7-5) with filtering capability. This window provides a direct view of the current SQE Plan Cache on the system.

As shown in Figure 7-5, note that the **Retrieve** action needs to be performed (pushed) to fill the display. The information shows the SQL query text, the last time the query was run, the most expensive single instance run of the query, total processing time consumed by the query, total number of times the query has been run and information about the user and job that first created the plan entry. It also shows how many times (if any) that the database

engine was able to reuse the results of a prior run of the query to avoid rerunning the entire query.

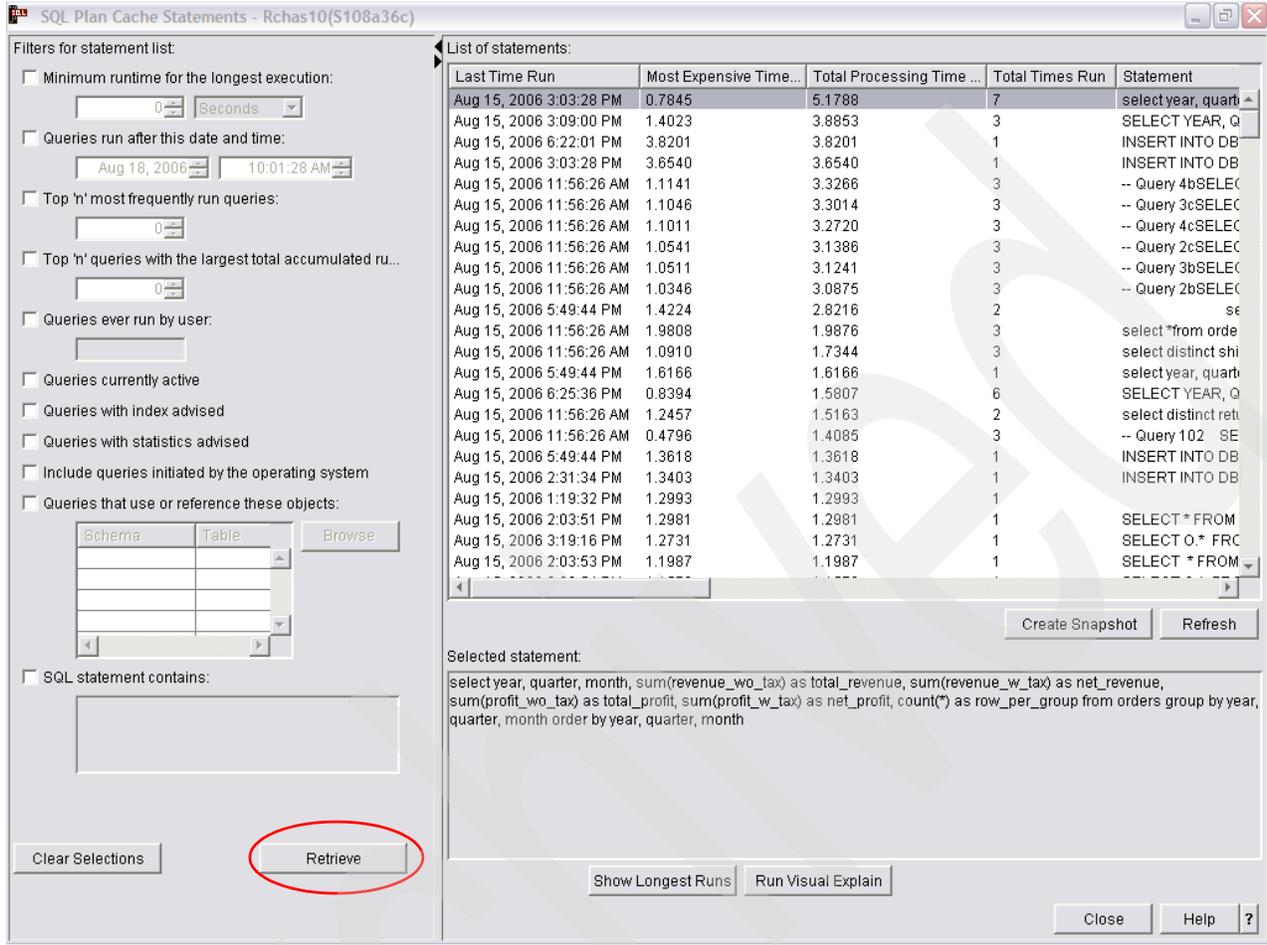


Figure 7-5 SQE Plan Cache Show Statements

7.2.3 Using the filter options

The window shown in Figure 7-6 provides filtering options which allow the user to more quickly isolate specific criteria of interest. No filters are required to be specified (the default), although adding filtering can shorten the time it takes to show the results. The list of queries that is returned is ordered by default so that those consuming the most processing time are shown at the top. You can reorder the results by clicking on the column heading for which you want the list ordered. Repeated clicking toggles the order from ascending to descending.

The filtering options provide a way to focus in on a particular area of interest:

- ▶ **Minimum runtime for the longest execution:** Filter to those queries with at least one long individual query instance runtime
- ▶ **Queries run after this date and time:** Filters to those queries that have been run recently
- ▶ **Top “n” most frequently run queries:** Finds those queries run most often.
- ▶ **Top “n” queries with the largest total accumulated runtime:** Shows the top resource consumers. This equates to the first n entries shown by default when no filtering is given. Specifying a value for n improves the performance of getting the first window of entries, though the total entries displayed is limited to n.

- ▶ **Queries ever run by user:** Provides a way to see the list of queries a particular user has run. Note that if this filter is specified, the user and job name information shown in the resulting entries still reflect the originator of the cached entry, which is not necessarily the same as the user specified on the filter.
- ▶ **Queries currently active:** Shows the list of cached entries associated with queries that are still running or are in pseudo-close mode. As with the user filtering, the user and job name information shown in the resulting entries still reflects the originator of the cached entry, which is not necessarily the same as the user currently running the query (there may be multiple users running the query).

Note: Current SQL for a job (right-click the Database icon) is an alternative for viewing a particular job's active query.

- ▶ **Queries with index advised:** Limits the list to those queries where an index was advised by the optimizer to improve performance.
- ▶ **Queries with statistics advised:** Limits the list to those queries where a statistic not yet gathered might have been useful to the optimizer if it was collected. The optimizer automatically gathers these statistics in the background, so this option is normally not that interesting unless, for whatever reason, you want to control the statistics gathering yourself.
- ▶ **Include queries initiated by the operating system:** includes into the list the “hidden” queries initiated by the database itself behind the scenes to process a request. By default the list only includes user initiated queries.
- ▶ **Queries that use or reference these objects:** Provides a way to limit the entries to those that referenced or use the tables or indexes specified.
- ▶ **SQL statement contains:** Provides a wildcard search capability on the SQL text itself. It is useful for finding particular types of queries. For example, queries with a FETCH FIRST clause can be found by specifying “fetch”. The search is case insensitive for ease of use. For example, the string “FETCH” finds the same entries as the search string “fetch”.

As shown in Figure 7-6, multiple filter options can be specified. Note that in a multi-filter case, the candidate entries for each filter are computed independently and only those entries that are present in all the candidate lists are shown. So, for example, if you specified options Top “n” most frequently run queries and Queries ever run by user, you are shown those most-run entries in the cache that happen to have been run at some point by the specified user. You are not necessarily be shown the most frequently run queries run by the user (unless those queries also happen to be the most frequently run queries in the entire cache).

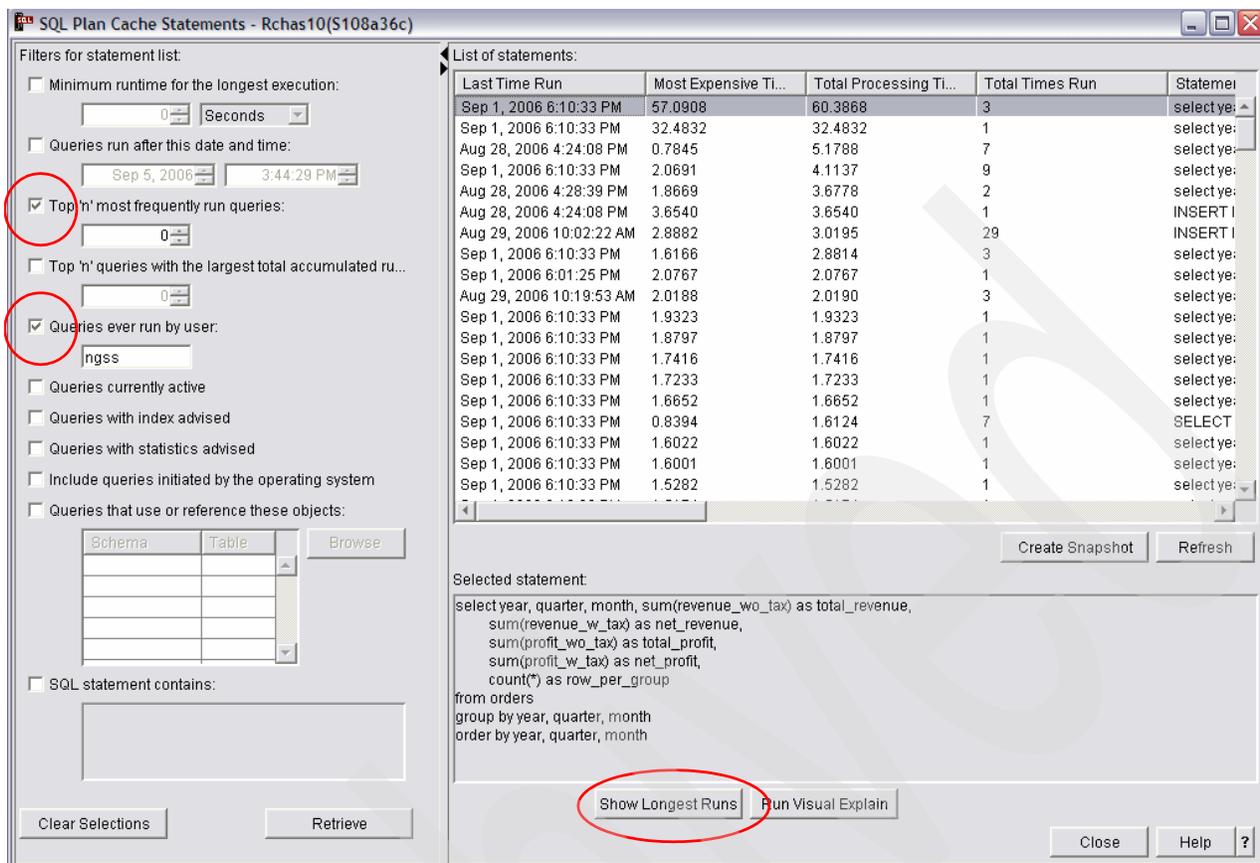


Figure 7-6 Filter with Top “n” most frequently run queries and Queries ever run by user options

From Figure 7-6, when an individual entry is chosen, more detailed information about that entry can be seen. Select **Show Longest Runs** shows details of up to ten of the longest running instances of that query. The result is shown in Figure 7-7.

Time Run	Processing Time ...	Records ...	User Name	Job Name	Job User	Job Num...	CPU Tim...	Synchron...	Synchron...
Aug 15, 2...	0.7845	36	NGSS	QZDASOI...	QUSER	040559	0.5867	0	0
Aug 15, 20...	0.7358	36	NGSS	QZDASOI...	QUSER	040559	0.5577	0	0
Aug 15, 20...	0.7343	36	NGSS	QZDASOI...	QUSER	040559	0.5558	0	0
Aug 15, 20...	0.7338	36	NGSS	QZDASOI...	QUSER	040559	0.5552	0	0
Aug 15, 20...	0.7320	36	NGSS	QZDASOI...	QUSER	040559	0.5556	0	0
Aug 15, 20...	0.7304	36	NGSS	QZDASOI...	QUSER	040559	0.5548	0	0
Aug 15, 20...	0.7276	36	NGSS	QZDASOI...	QUSER	040559	0.5540	0	0

Figure 7-7 Show Longest Runs

7.2.4 Finding and Visual Explaining a query from the SQE Plan Cache

SQE Plan Cache can be integrated with Visual Explain, allowing you to visualize what the optimizer is performing and further assist you in making decisions in query optimization.

As shown in Figure 7-8, **Run Visual Explain** can be performed against the chosen query to show the details of the query plan.

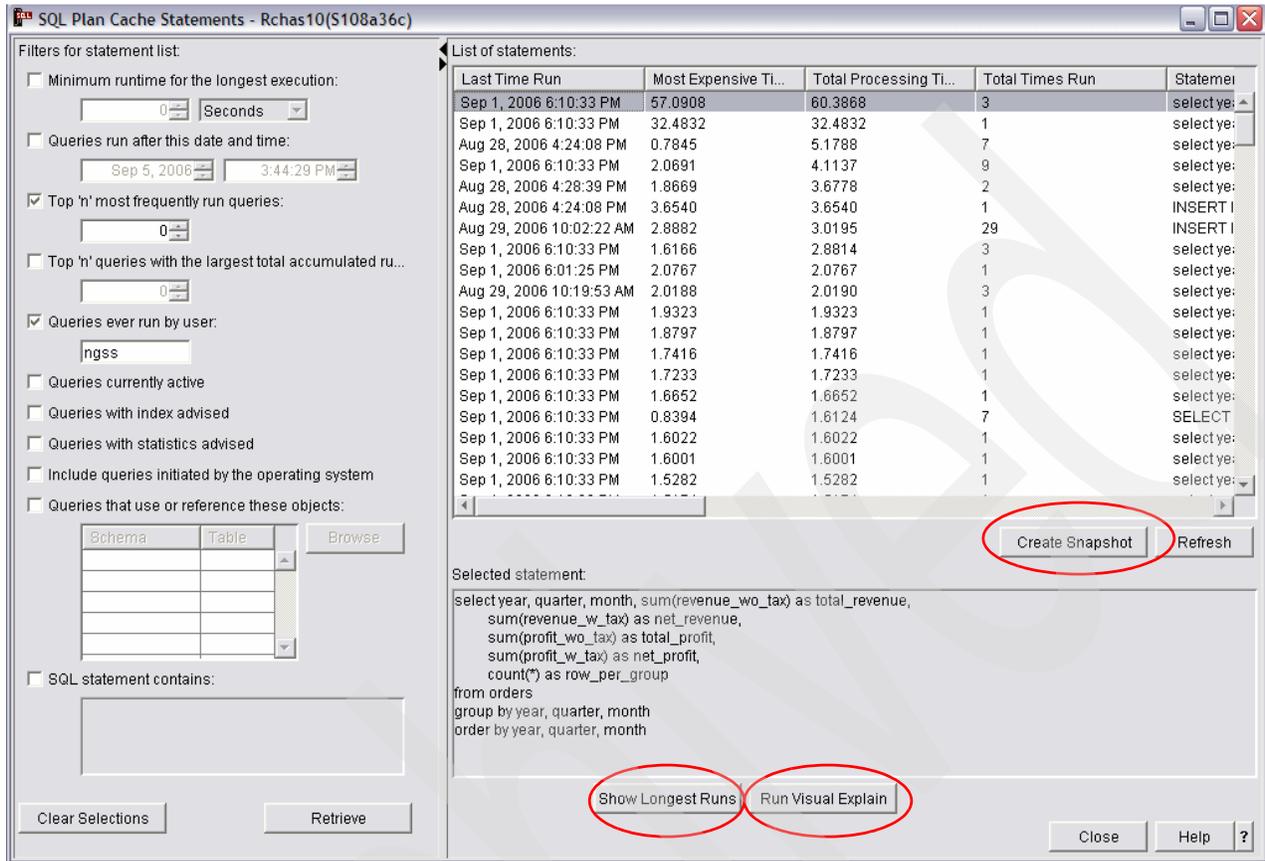


Figure 7-8 Run Visual Explain on SQL Plan Cache Statement window

This leads you to the Visual Explain window as shown in Figure 7-9. The information presented can be used in multiple ways to help with performance tuning. For example, Visual Explain of key queries can be utilized to show advice for creating an index to improve those queries.

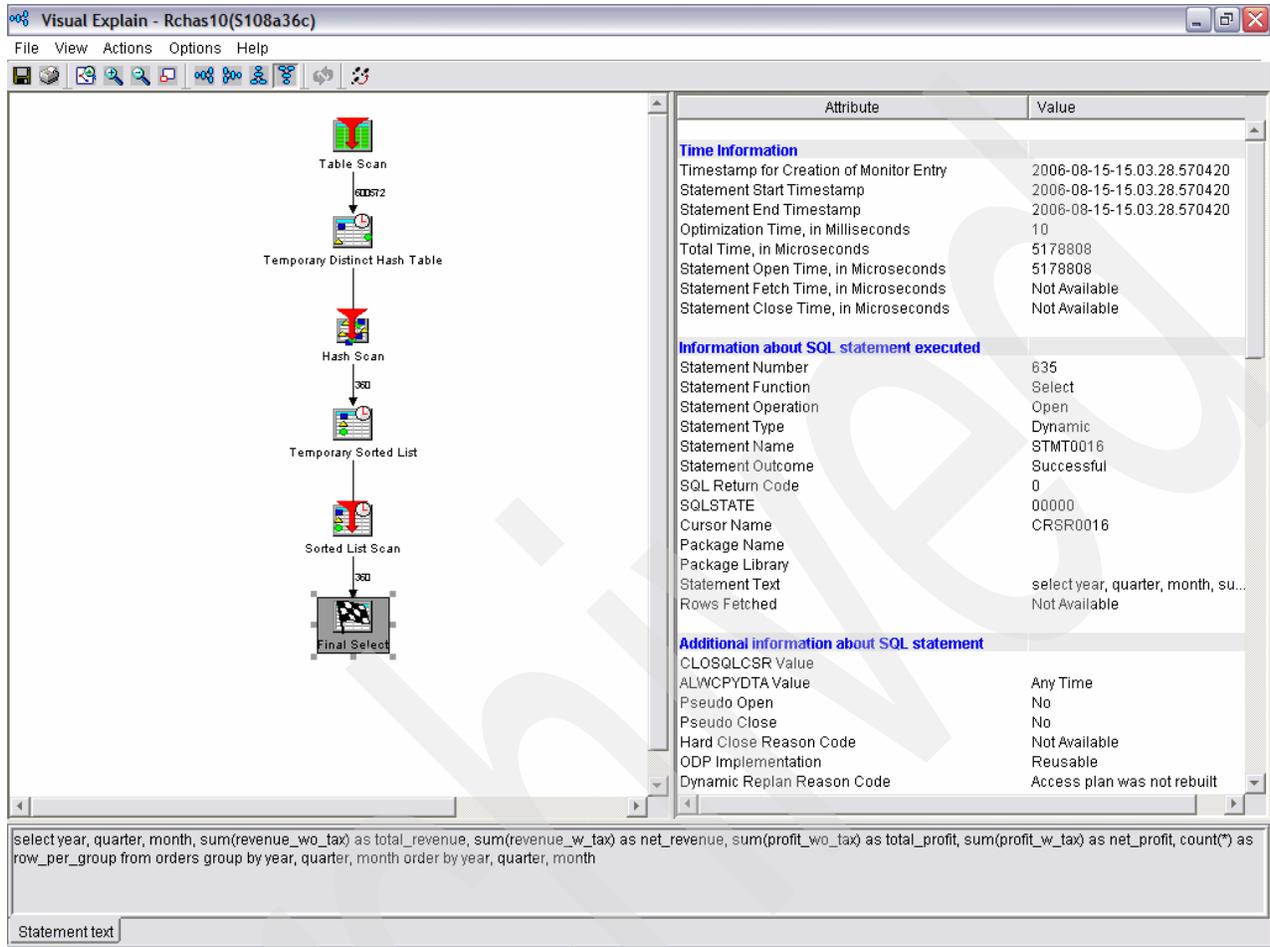


Figure 7-9 Visual Explain window

Alternatively, from the Figure 7-8, the **Show Longest Run** option gives you information for determining if the query is being run during a heavy utilization period and can potentially be rescheduled to a more opportune time.

Note: The user and job name information given for each entry is the user and job that initially caused the creation of the cached entry (the user where full optimization took place).

Finally, if one or more entries are highlighted, you can generate a snapshot (database performance monitor file) for those selected entries by selecting the **Create Snapshot** button (Figure 7-8).

Note: For more information about Visual Explain, refer to Chapter 8, “Analyzing database performance data with Visual Explain” on page 275.

7.3 SQE Plan Cache Snapshot

Clicking the SQE Plan Cache folder shows a list of any snapshots gathered so far. A snapshot is a database monitor file generated from the SQE Plan Cache and can be treated very much the same as the SQL Performance Monitors list. The same analysis capability exists for snapshots as exists for traditional SQL performance monitors.

The SQE Plan Cache is cleared each time you IPL the system. This means that once you IPL the system you lose all the contents of the SQE Plan Cache. For this reason it is good to have a way of storing or having a copy of the contents of the SQE Plan Cache in a separate place. SQE Plan Cache snapshots are images of the contents of the SQE Plan Cache.

There are three ways to create an SQE Plan Cache Snapshot. There are explained in sections:

- ▶ “Creating an SQE Plan Cache Snapshot using iSeries Navigator”
- ▶ “Creating an SQE Plan Cache Snapshot using Stored Procedure”
- ▶ “Creating an SQE Plan Cache Snapshot using an Exit Program”

Note: SQE Plan Cache Snapshot is named SQL Plan Cache Snapshot in the iSeries Navigator GUI interfaces.

7.3.1 Creating an SQE Plan Cache Snapshot using iSeries Navigator

You need to add your schema into the schema list before proceeding to create any SQE Plan Cache snapshot.

As shown in Figure 7-10, to add your additional schema into the Schema List, select **iSeries Navigator** → **system name** → **Database** → **database name** and right-click **Schemas** and select **Select Schemas to Display**.

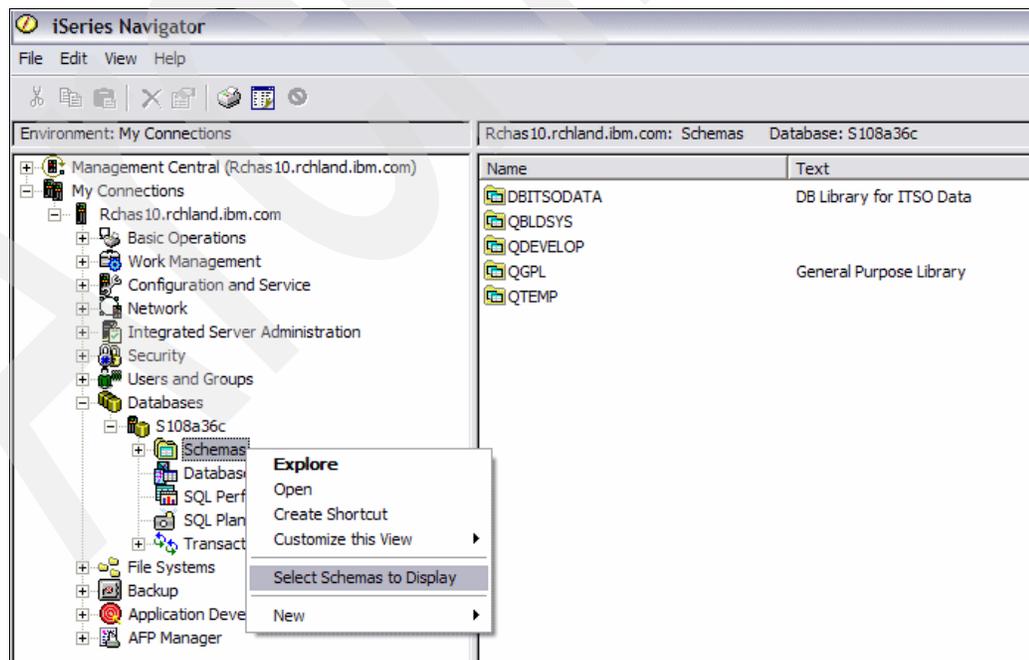


Figure 7-10 Path to adding your Schema to the Schema List

As shown in Figure 7-11, you can use the **Search for Schemas** option and the **Search** button to browse for your schema. Once found, you just have to highlight your schema name and use the **Add** button to add it into the Schema List. Alternatively, select the **Enter Schema Names** option, enter your schema name and use the **Add** button to add it into the Schema List.

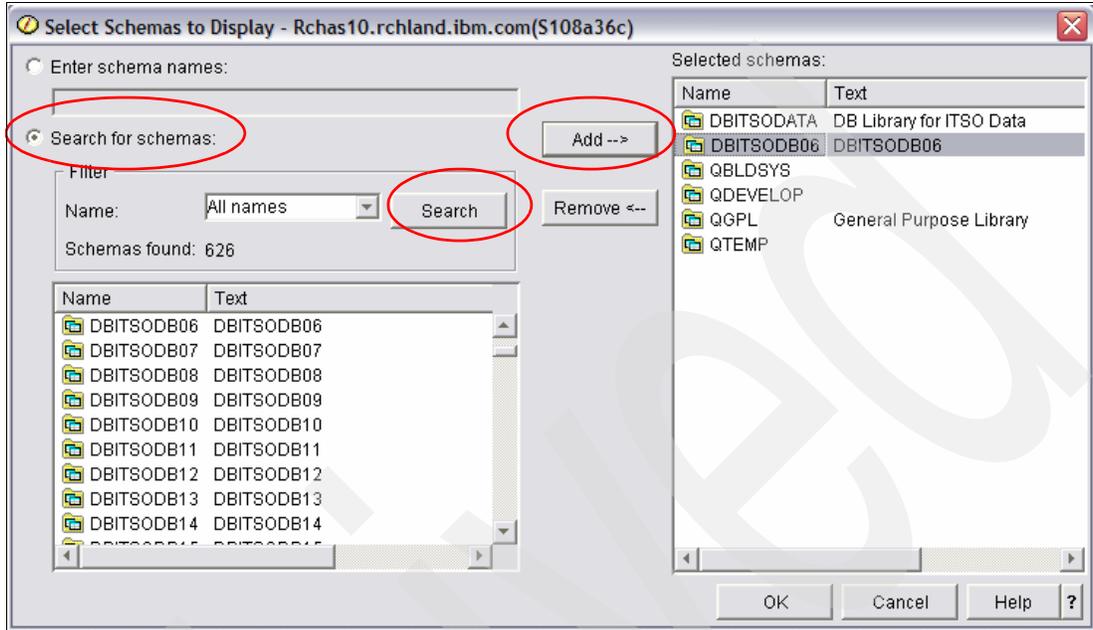


Figure 7-11 Select schema to display

As shown in Figure 7-12, you see your schema name appears in the right pane of the SQL Plan Cache Snapshot. Select **Database**, right-click **SQL Plan Cache Snapshot** and select **New** → **Snapshot** to create a new snapshot of the SQE Plan Cache. Unlike the snapshot

option under Show Statements, it allows you to create a snapshot without having to first view the queries.

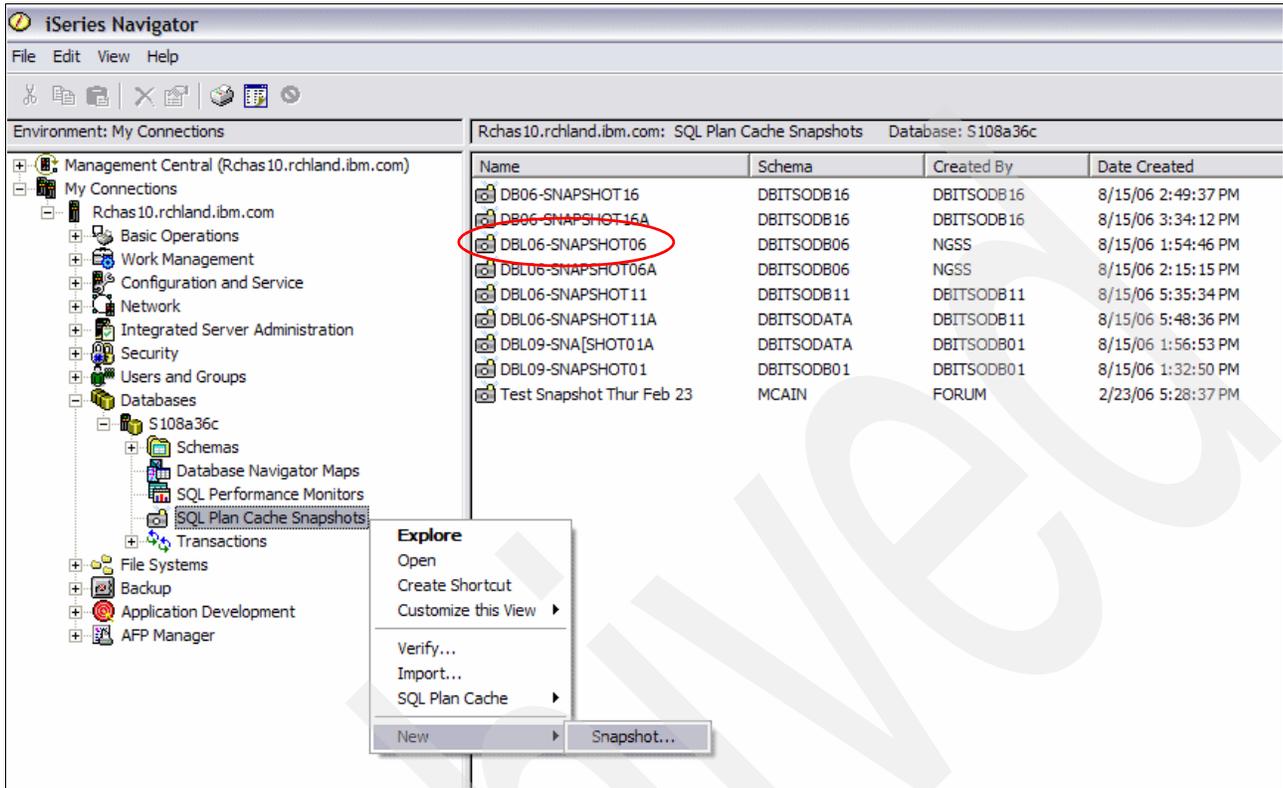


Figure 7-12 Create SQE Plan Cache Snapshot

As shown in Figure 7-13, a new Snapshot of the SQE Plan Cache window appears. You provide a snapshot name and select your schema from the drop down list. You can select **Include all plan cache entries** or **Include plan cache entries that meet the following criteria** to further narrow down the selection like choosing suboptions: **Queries with index**

advised. Click **OK** and wait a moment until you see your SQE Plan Cache Snapshot object in the right pane of iSeries Navigator.

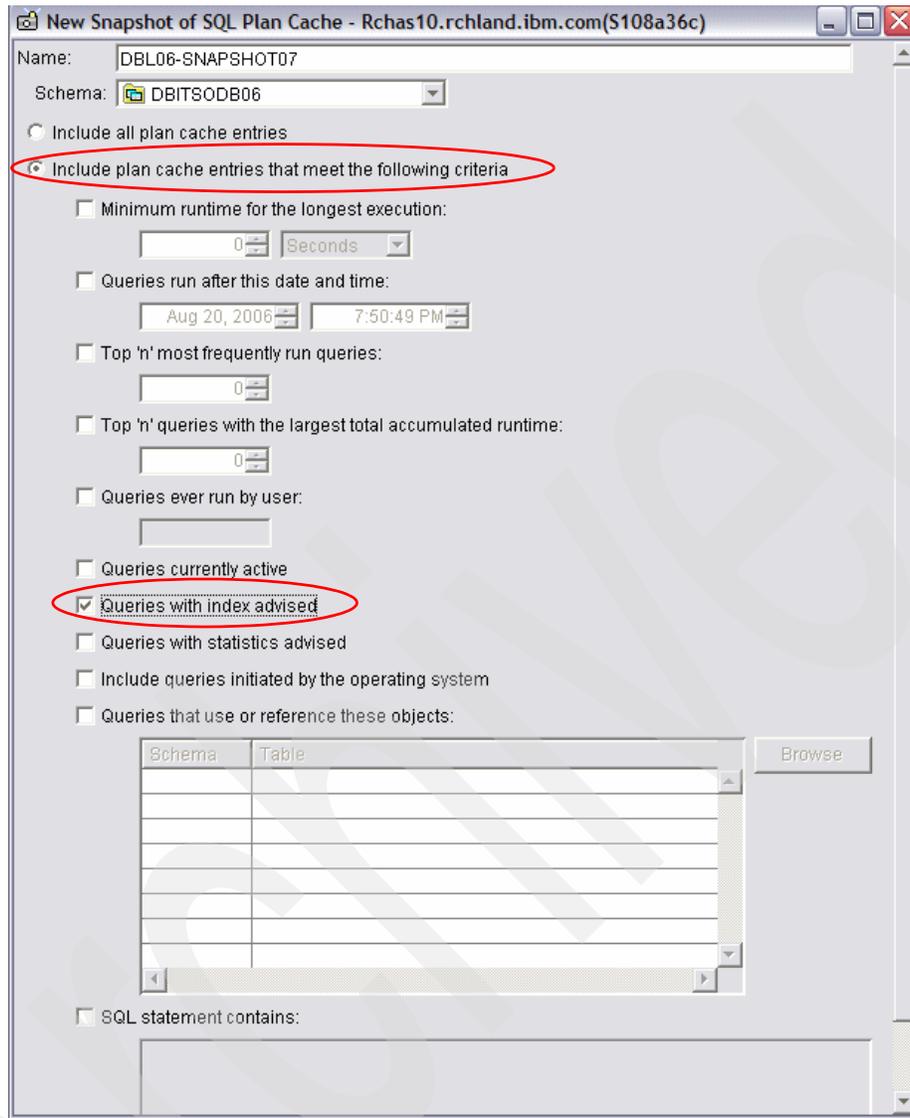


Figure 7-13 Suboptions in creating SQE Plan Cache Snapshot

There are a total of eleven suboptions that you can choose in the filtering criteria for SQE Plan Cache Snapshot creation:

1. **Minimum runtime for the longest execution:** Select this to include queries that exceed a certain amount of time. Select a number and then a unit of time.
2. **Queries run after a specific date and time:** Select this to include queries run at a specified date and time. Select a date and time.
3. **Top “n” most frequently run queries:** Select this to include queries that are most frequently run. Specify the number of most frequently run queries.
4. **Top “n” queries with the largest total accumulated runtime:** Select this to include queries with the largest total accumulated runtime. Run time is defined as the cost for I/O and CPU. Specify the number of longest-running queries.

5. **Queries ever run by a specific user:** Select this to include queries run by a certain user. Specify a user ID. You can specify the entire ID or use a wildcard. For example, “QZDAS*” returns all user IDs where the name starts with “QZDAS”.
6. **Queries currently active:** Select this to include queries that are currently running.
7. **Queries with index advised:** Select this to include queries that have an index advised.
8. **Queries with statistic advised:** Select this to include queries that have statistics advised.
9. **Include queries initiated by the operating system:** Select this to include queries that were initiated by the operating system.
10. **Queries that use or reference specific tables:** Select this to include queries that use or reference certain objects. Click **Browse** to select objects to include.
11. **SQL statements that contain specific syntax:** Select this to include only those queries that contain a specific type of SQL statement. For example, specify SELECT if you only want to include queries that are using SELECT.

Note: These selection criteria only apply to SQL statements that are served by SQE. For SQL statements served by CQE, you need to start SQL Performance Monitor.

7.3.2 Creating an SQE Plan Cache Snapshot using Stored Procedure

The SQE plan cache is an actively changing cache. Therefore, it is important to realize that it contains timely information. If information over long periods of time is of interest, consider implementing a method of performing periodic snapshots of the SQE plan cache to capture trends and heavy usage periods.

The stored procedure, QSYS2.DUMP_PLAN_CACHE, provides the simplest way to create a snapshot from the SQE plan cache. The DUMP_PLAN_CACHE API takes two parameters, library name and file name, for identifying the resulting snapshot. If the file does not exist, it is

created. For example, to dump the SQE plan cache to a database performance monitor file (snapshot) in schema DBITSODB06, run this SQL script as shown in Figure 7-14:

```
CALL QSYS2.DUMP_PLAN_CACHE('DBITSODB06','SNAPSHOT1');
```

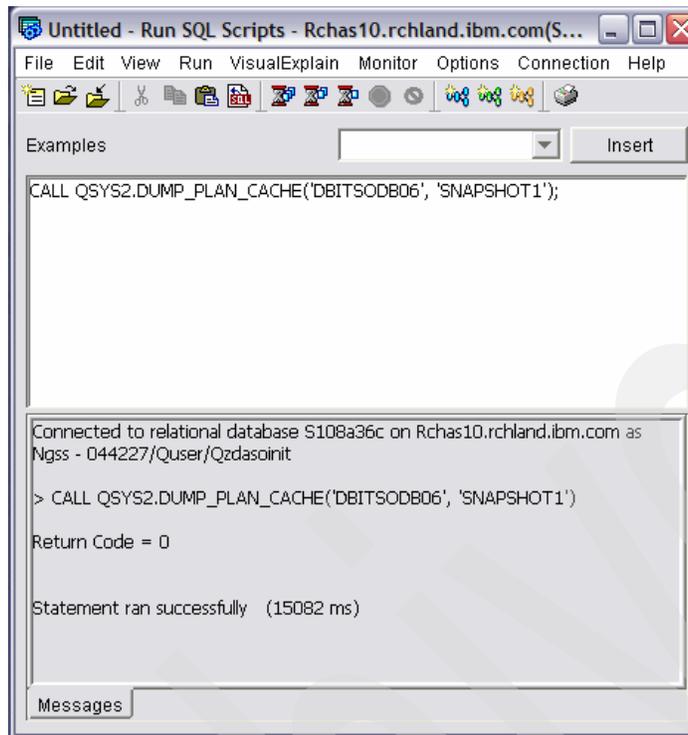


Figure 7-14 Run Dump Plan Cache stored procedure

As shown in Figure 7-15, now go to DBITSODB06 schema and select **Table**. You should find the SNAPSHOT1 table you have just dumped.

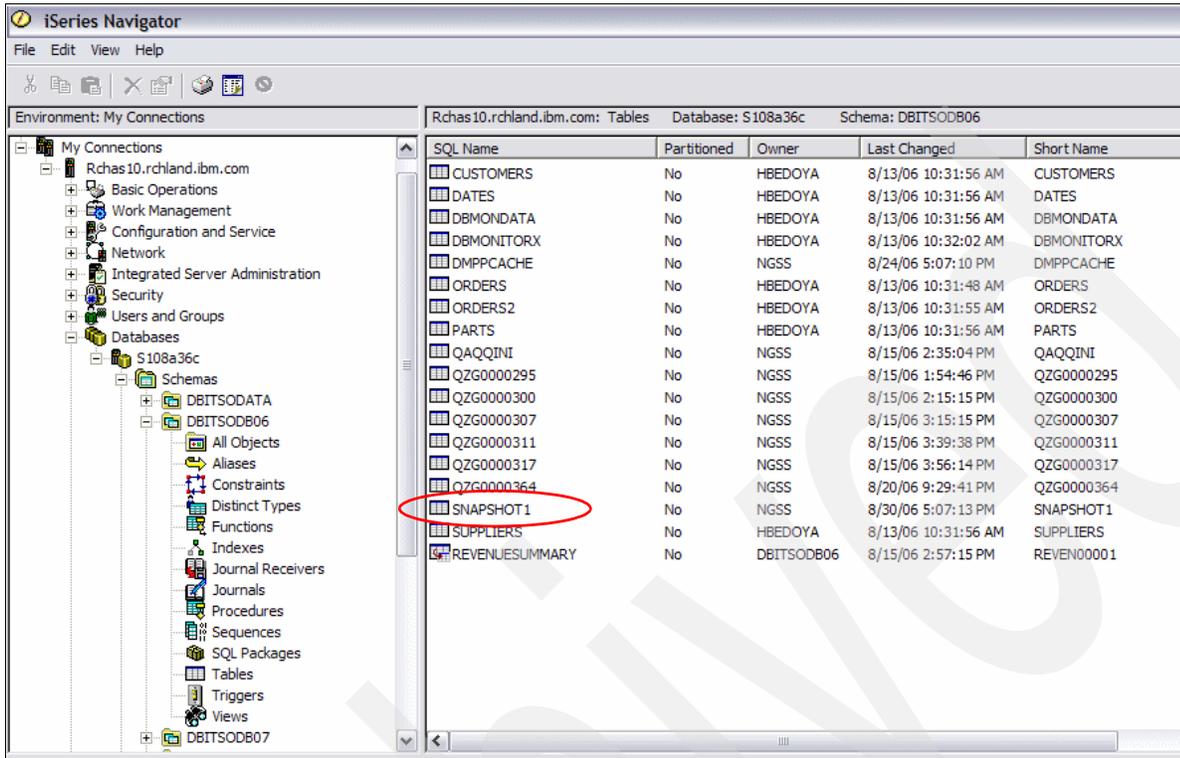


Figure 7-15 Locate the SNAPSHOT1 table in DBITSODB06 schema

As shown in Figure 7-16, you have to import the snapshot into the iSeries Navigator prior to performing any performance analysis. To do so, right-click **SQL Plan Cache Snapshot** and select **Import**.

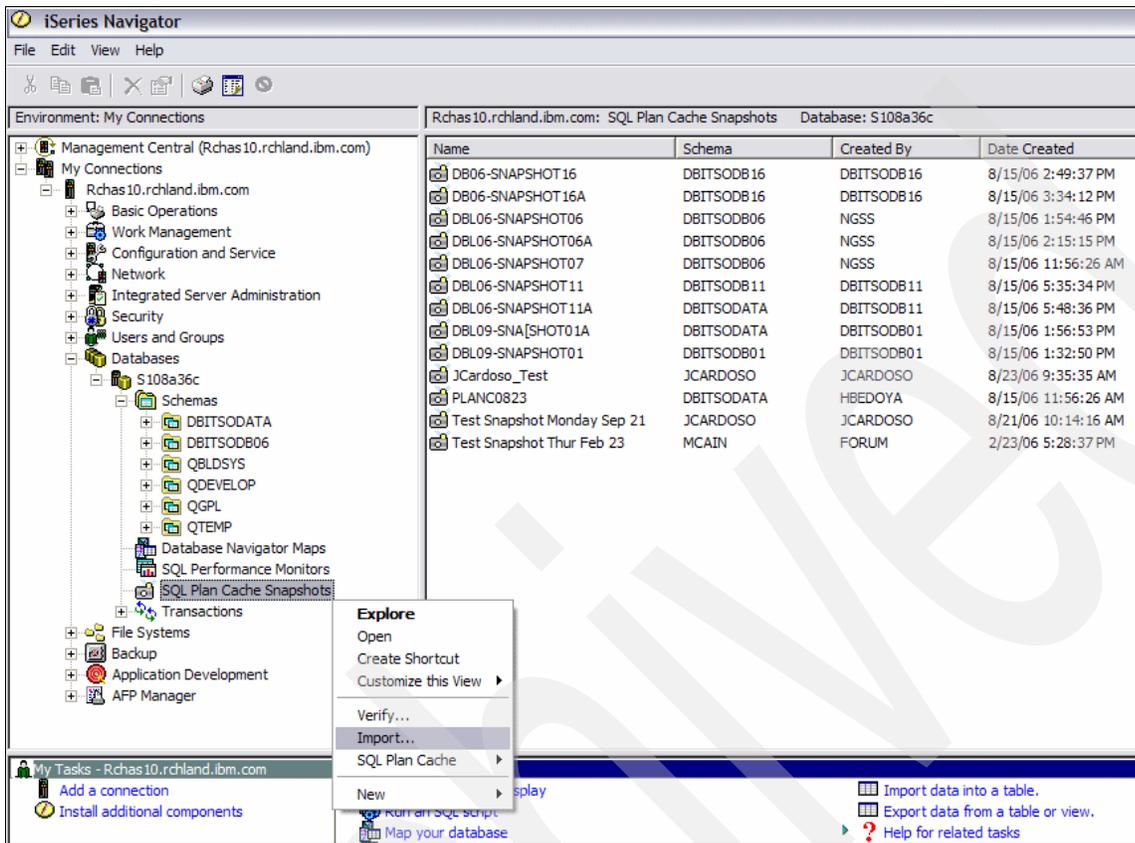


Figure 7-16 Import the Plan Cache dumped

As shown in Figure 7-17, provide a meaningful description for the **Name** field and select **DBITSODB06** for the Schema field. Then select **SNAPSHOT1** for the Table field and click **OK**.

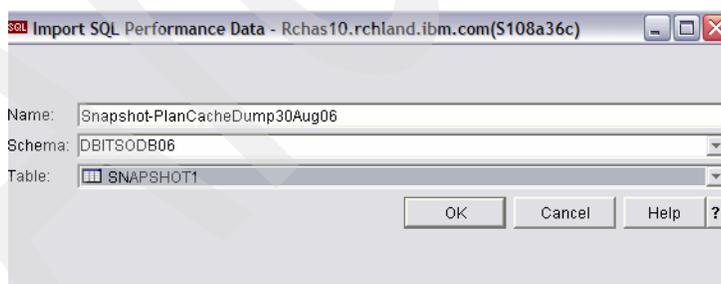


Figure 7-17 Select DBITSODB06 schema and SNAPSHOT1 table

In Figure 7-18, you should find the SQE Plan Cache Snapshot created for SNAPSHOT1 which you dumped using DUMP_PLAN_CACHE stored procedure. You can perform further action on this snapshot such as **Analyze**, **Show Statements** and so on.

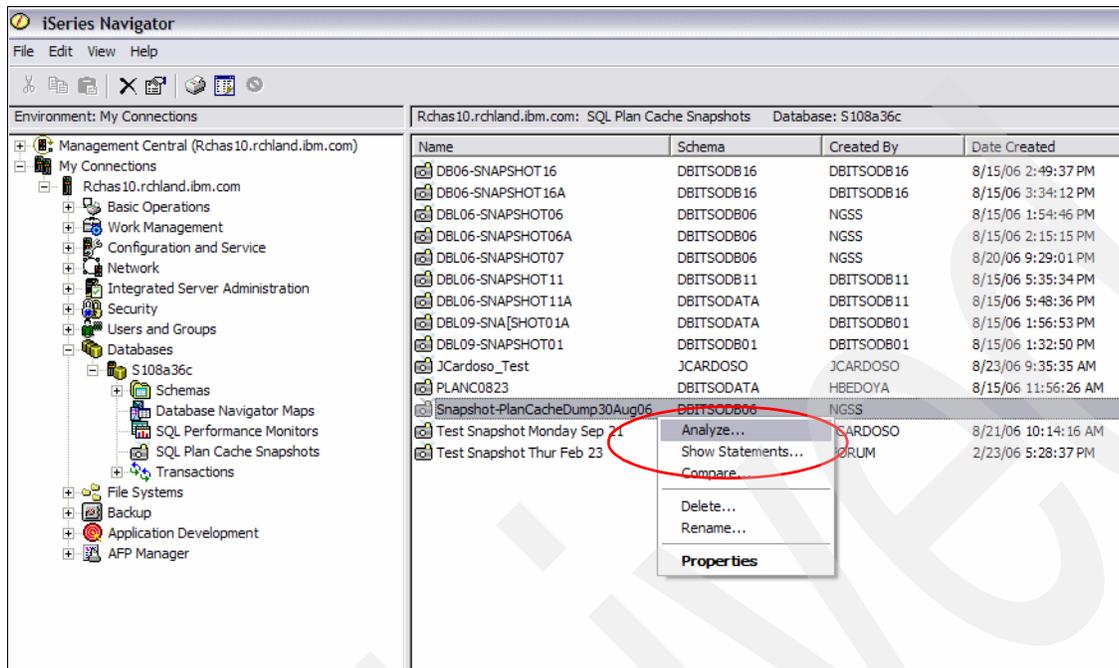


Figure 7-18 Perform further work on SNAPSHOT1 SQE Plan Cache Snapshot

7.3.3 Creating an SQE Plan Cache Snapshot using an Exit Program

You can make use of exit point QIBM_QWC_PWRDWNSYS to call QSYS2.DUMP_PLAN_CACHE each time PWRDWNSYS and ENDSBS commands are issued. The program is called before the system actually powers down. You can automate capturing the plan cache snapshot prior to each IPL. Alternatively, you can use the DUMP_PLAN_CACHE API in conjunction with job scheduling (for example), to programmatically perform periodic snapshots capture.

In this topic, we provide a sample RPG program as shown in Example 7-1 that calls the QSYS2.DUMP_PLAN_CACHE with two variables: the library name (DBITSODB06) and the table name (DMPPCACHE) to hold the plan cache dump.

The program sends a message to QSYSOPR message queue once the plan cache is dumped successfully:

“Plan Cache Dump successful!”

Alternatively, if the dump is not successful, the program also sends a message to QSYSOPR message queue:

“Plan Cache Dump Failed!”

Example 7-1 RPG program to call QSYS2.DUMP_PLAN_CACHE

```

***** Beginning of data *****
H
D*
D** Variables for system operator message
D*
D  OKMsg      S          50A  INZ('Plan Cache Dump successful!')
D  ErrorMsg   S          50A  INZ('Plan Cache Dump Failed!')
D  SQLCODE2   S          2A
D  SQLCODEC   S          5A
D  MsgQueue   S          10A  INZ('QSYSOPR')
D*
D  SCHEMA_NAME S          10A  INZ('DBITSODB06')
D  TABLE_NAME S          10A  INZ('DMPPCACHE')
D*
C*****
C* this RPG program will called by PWRWDNSYS exit point SED AS
C* to the SQL statements IBM_QWC_PWRWDNSYS format PWRD0100
C*****
C*
C* EXECUTE SQL CALL TO
C* CALL QSYS2/DUMP_PLAN_CACHE('--SCHEMA OR LIB NAME--',
C*                               '--TABLE NAME--')
C*
C/EXEC SQL
C+ CALL QSYS2/DUMP_PLAN_CACHE(:SCHEMA_NAME, :TABLE_NAME)
C/END-EXEC
C*
C*
C** CHECK FOR 00 IN POSITIONS 1-2 OF SQLSTATE CODE
C*
C* Convert SQLCODE from numeric to character
C      MOVE      SQLCODE      SQLCODEC
C* Select first 2 characters to see if 00
C      SUBST     SQLCODEC:2    SQLCODE2
C      SQLCODE2  IFEQ         '00'
C* LET SYSTEM OPERATOR KNOW PLAN CACHE DUMP OK
C*
C
C  OKMsg      DSPLY      MsgQueue
C*
C      ELSE
C* LET SYSTEM OPERATOR KNOW PLAN CACHE DUMP FAILED
C  ErrorMsg   DSPLY      MsgQueue
C*
C*
C      END
C      SETON                                     LR
***** End of data *****

```

Issue the following command to add the PRG program into QIBM_QWC_PWRDWNSYS Exit program:

```
ADDEXITPGM EXITPNT(QIBM_QWC_PWRDWNSYS) FORMAT(PWRD0100) PGMNBR(*LOW)
PGM(DBITSODB06/DUMPPCACHE)
```

The plan cache would be dumped to DBITSODB06/DUMPPCACHE at each IPL. You can proceed to import the plan cache in and carry out further performance analysis.

Note: You can enhance the RPG program to dynamically name the plan cache dump (using a sequence object, time stamp, and so on.) so as not to overwrite the existing one with the same name (in this case DUMPPCACHE)

7.3.4 Analyzing an SQE Plan Cache Snapshot

In this section, we illustrate the tool used for analyzing an SQE snapshot. To do this, perform the following steps:

1. Right-click your newly-created snapshot name and select **Properties** as shown in Figure 7-19.

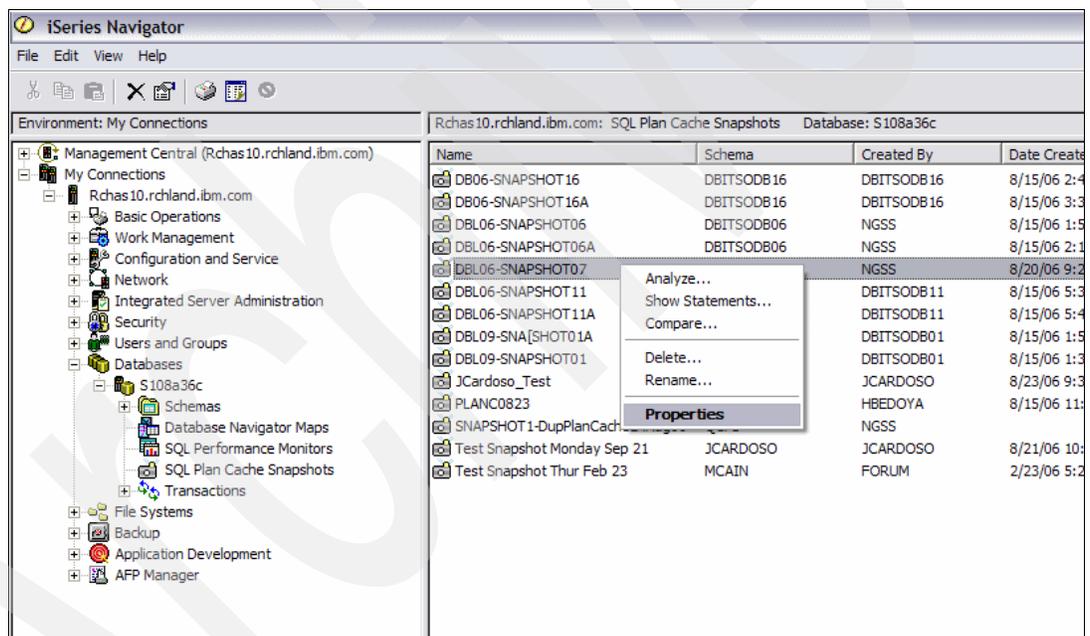


Figure 7-19 Path to Snapshot Properties

The information of a snapshot is stored in a table (QZGXXXXXXX). This table can be:

- Saved and restored
- FTPed to another system
- Queried by SQL

An SQE Plan Cache Snapshot is the materialization of the SQE plan information residing in the plan cache. The format of the data is similar to detailed database monitor data (by design) but not identical. Figure 7-20 shows the properties of a snapshot.

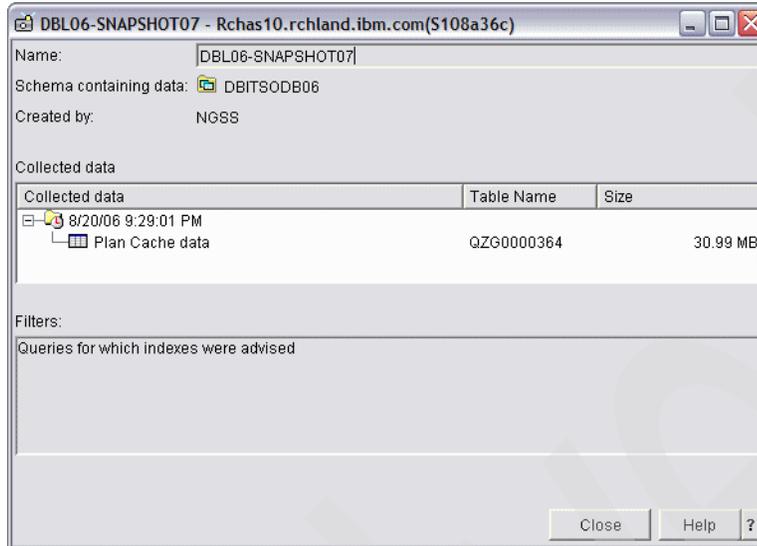


Figure 7-20 SQE Plan Cache Snapshot Properties

- Once you have created the SQE Plan Cache Snapshot, start to analyze the performance information contained in the snapshot. As shown in Figure 7-21, right-click your snapshot name and select **Analyze**.

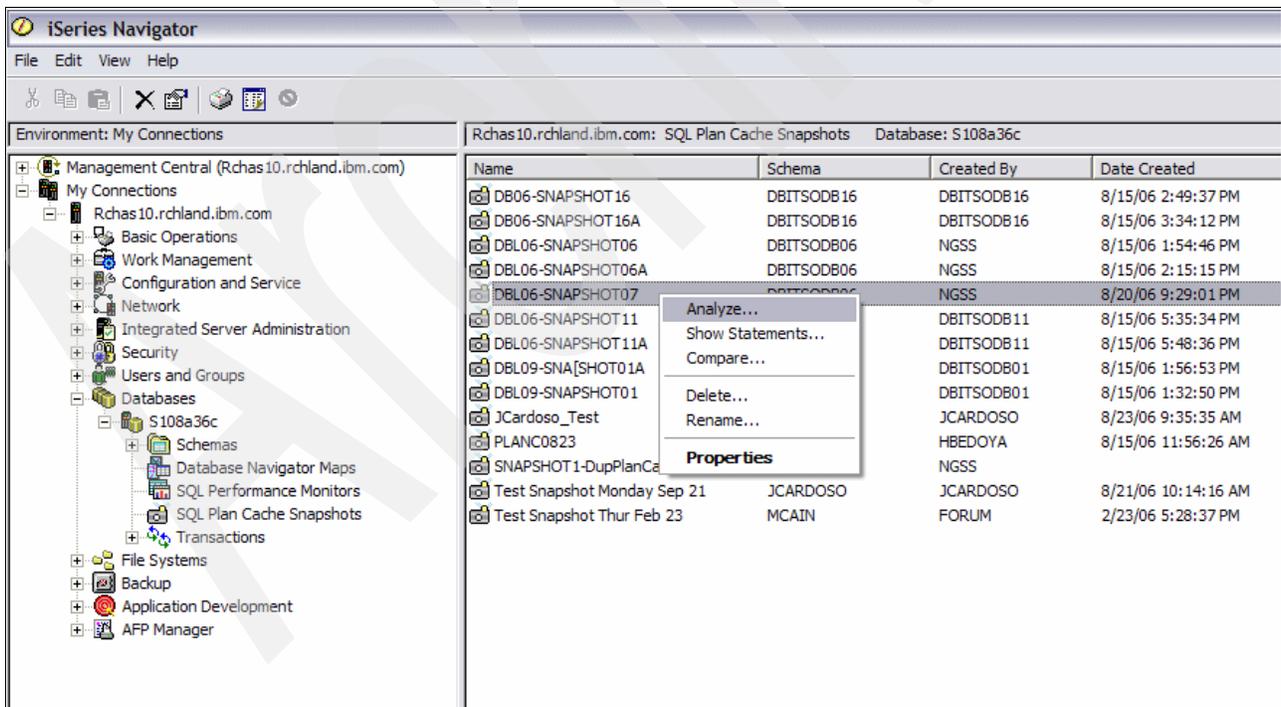


Figure 7-21 Path to Analyze your snapshot

3. A pop-up information window asking you to be patient appears. After waiting a while, the **Analysis Overview Dashboard** window appears as shown in Figure 7-22.

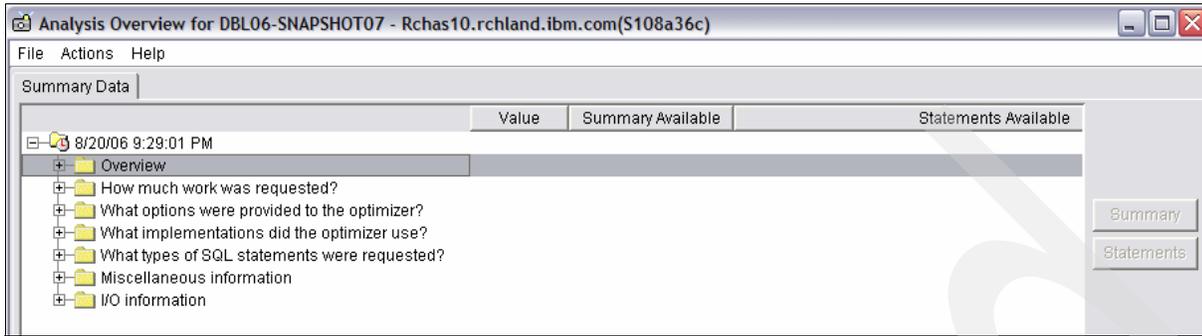


Figure 7-22 The Analysis Overview Dashboard

Note: The Dashboard shown in Figure 7-22 is the same dashboard used and displayed by the detailed SQL Database Monitor, described in Chapter 5, “Analyzing SQL performance data using iSeries Navigator” on page 117.

- As shown in Figure 7-23, expand the **Overview** folder if it is not already expanded. This allows you to see if Summary Analysis and Statements are available for each corresponding options.

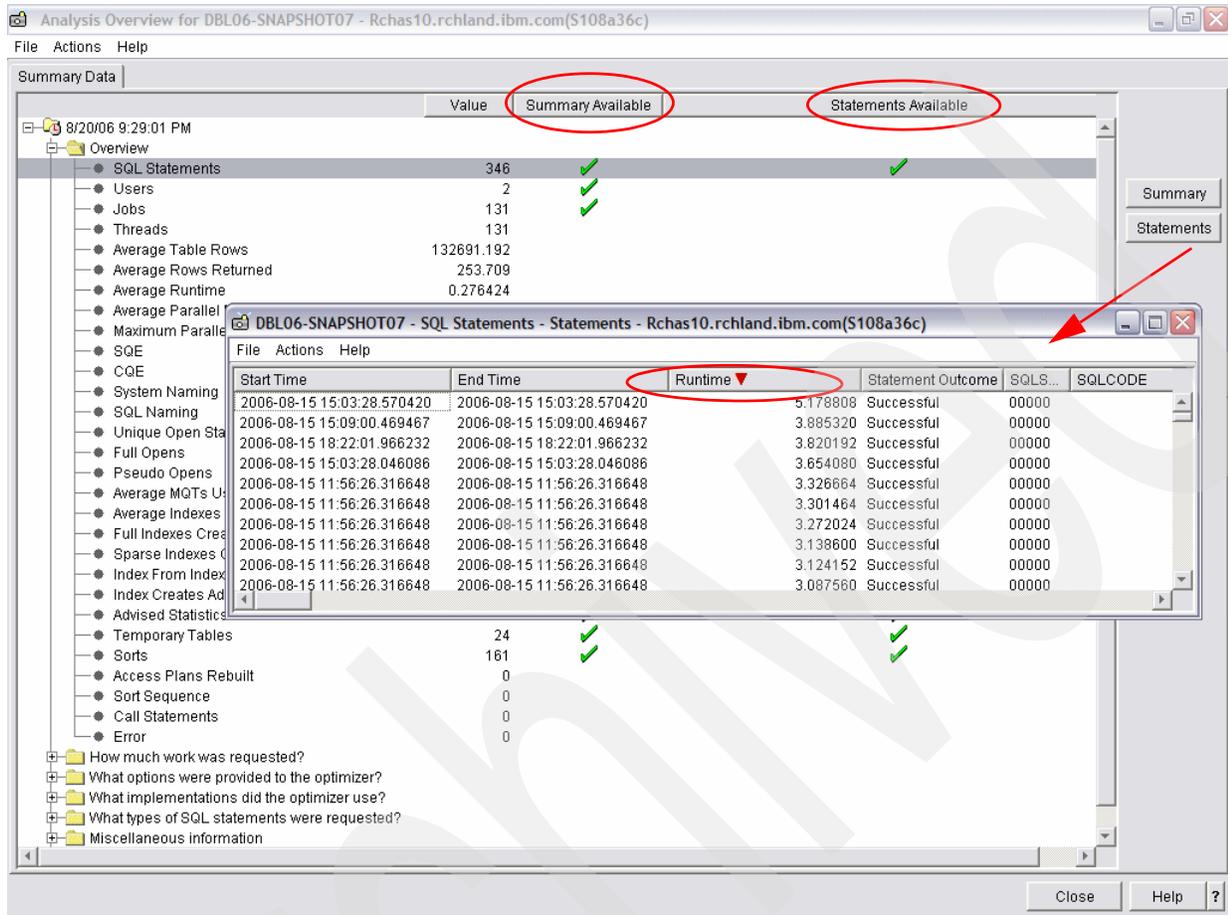


Figure 7-23 SQL Statements and Statements options of the Dashboard

- As shown in Figure 7-23, highlight SQL Statements and click the **Statements** button. This leads you to a new window that contains various information of all SQL statements in the snapshot. You can scroll to the right to view more information. Notice the Runtime column allows you to sort either in ascending or descending order. By default, the SQL statement with the longest runtime appears at the topmost line in the window. The same feature is available for all the listed columns.

Note: You can specify ascending or descending sorting order of any column by clicking the column heading once or twice. For example, if you click the column heading named Result Rows to specify descending order, you can see the statement with the “largest result at set” at the topmost line.

- As shown in Figure 7-24, select the topmost line which has the longest run time and select **Action** → **Visual Explain** which leads you to the graphical representation of the access path of the selected statement.

Start	Visual Explain	Runtime	Statement Outcome	SQLSTATE	SQLCODE	Operation	Statement Text
2006-08-15 11:56:26.316648	Work with SQL Statement	5.178808	Successful	00000	0	OPEN	select year, quarter, month, sum(revenue_wo
2006-08-15 11:56:26.316648	SQL Statements with the Same Join Field	3.895320	Successful	00000	0	OPEN	SELECT YEAR, QUARTER, MONTH, SUM(RE
2006-08-15 11:56:26.316648	SQL Statements with the Same Statement Text	3.820192	Successful	00000	0	INSERT	INSERT INTO DBITSODB11.REVEN00001 SE
2006-08-15 11:56:26.316648	Copy to Performance Monitor	3.654080	Successful	00000	0	INSERT	INSERT INTO DBITSODB06.REVEN00001 SE
2006-08-15 11:56:26.316648	Specific XID Statements	3.326664	Successful	00000	0	OPEN	-- Query 4bSELECT*FROMOrdersWHEREsup
2006-08-15 11:56:26.316648	SQL Statements in the SQL Statement	3.301464	Successful	00000	0	OPEN	-- Query 3cSELECT*FROMOrders i,suppliers :
2006-08-15 11:56:26.316648		3.272024	Successful	00000	0	OPEN	-- Query 4cSELECT*FROMOrders i,suppliers :
2006-08-15 11:56:26.316648		3.138600	Successful	00000	0	OPEN	-- Query 2cSELECT*FROMOrders i,suppliers :
2006-08-15 11:56:26.316648		3.124152	Successful	00000	0	OPEN	-- Query 3bSELECT*FROMOrdersWHEREsup
2006-08-15 11:56:26.316648		3.087560	Successful	00000	0	OPEN	-- Query 2bSELECT*FROMOrdersWHEREsup
2006-08-15 17:49:44.659257		2.821664	Successful	00000	0	OPEN	select year, quarter, month, su

Figure 7-24 Visual Explain from the SQL statements

You can now explore Visual Explain information to identify why this statement has the longest run time.

- Refer to Figure 7-23 again. From the Dashboard window select the **Overview** folder and click **Statements**. Now refer to Figure 7-24, highlight the topmost line, select **Actions** → **Work with SQL Statement**. The **Run SQL Scripts** window in Figure 7-25 appears showing you the involved SQL statements. This helps you to execute the statement or modify it to test for a possible performance improvement.

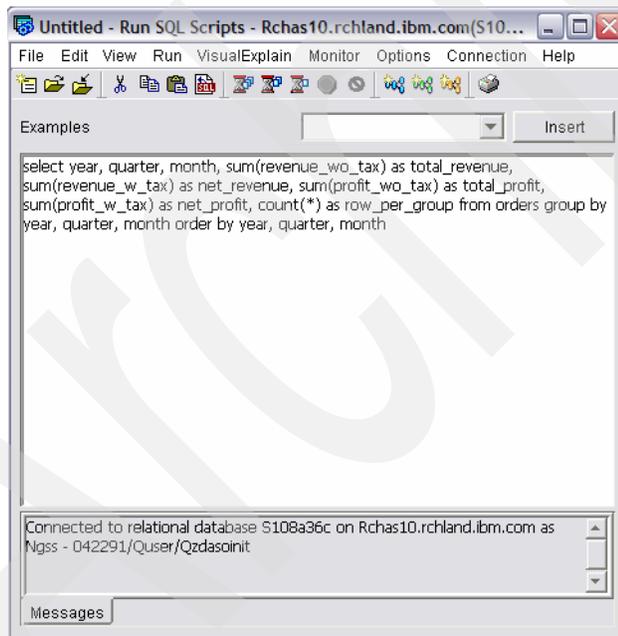


Figure 7-25 The Run SQL Scripts window showing the involved SQL statements

- As shown in Figure 7-26, explore other options in the Dashboard. Expanding the **Miscellaneous Information** folder leads you to Maximum Run Time, Open Time, Fetch Time, Close Time, Other Time information such as insert operation (all in seconds), the maximum number of rows contained in a table being accessed (Maximum Table Rows), and the largest result set size (Maximum Rows Returned) of the statement in the snapshot.

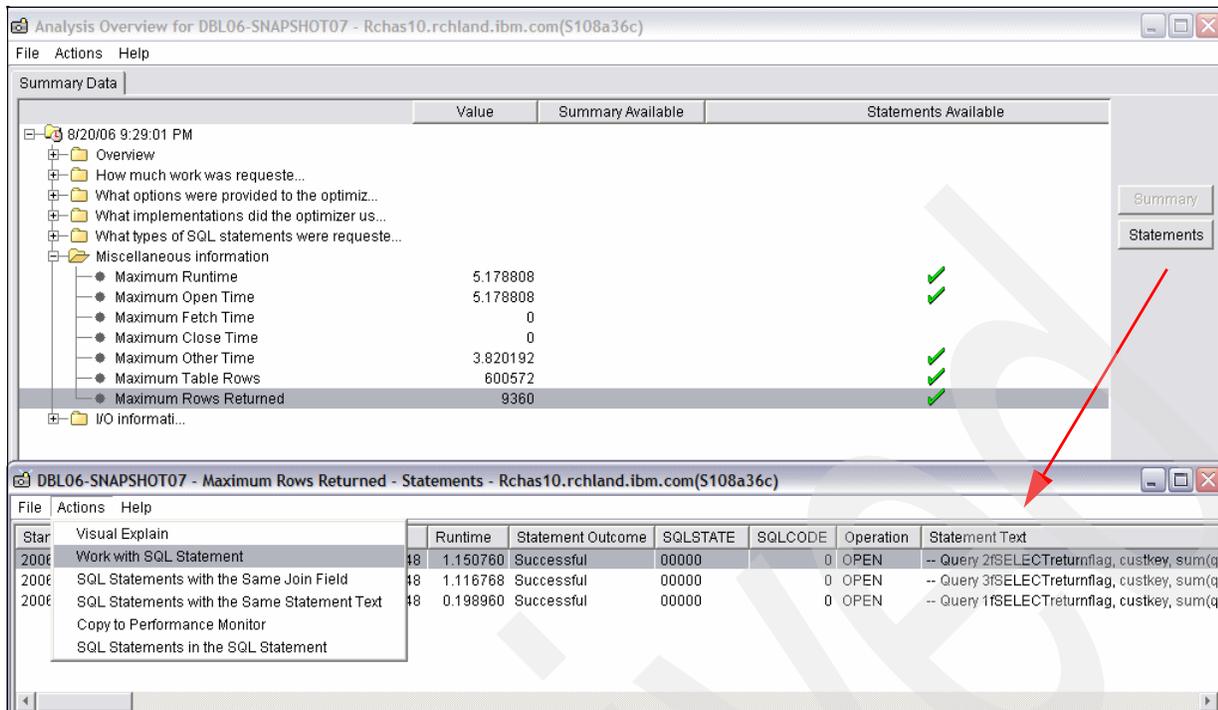


Figure 7-26 Work with Statements of Maximum Rows Returned

9. As shown in Figure 7-26, select **Maximum Rows Returned** → **Statements** which will show you a new window. Highlight the topmost line of the Maximum Rows Returned and select **Actions** → **Work with Statements** which will lead you to the Run SQL Scripts window. You can also right-click the highlighted line select from the pop-up menu that offers the same options.
10. As shown in Figure 7-27, you see the SQL statement in the Run SQL Script window. You can now choose to work with the selected statement in many ways such as invoking Visual

Explain to see the access plan of the statement, running the statement to see the result set, or modifying the statement to test for ways that can improve its performance.

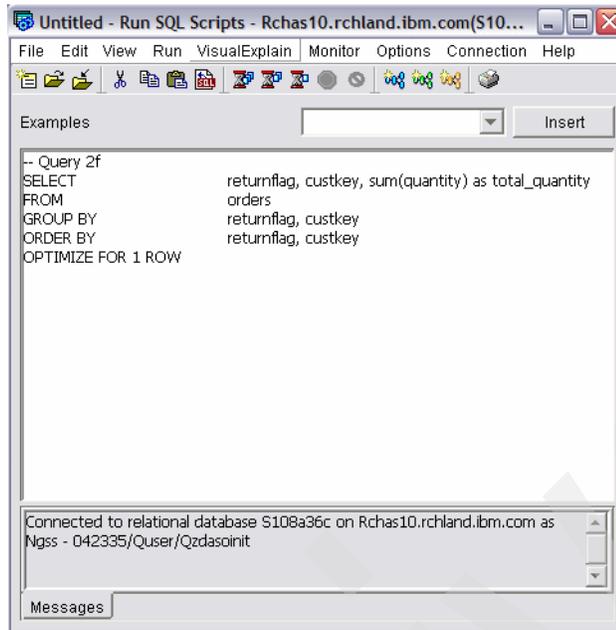


Figure 7-27 SQL Statements in the Run SQL Scripts window

7.3.5 Working with SQL statements from an SQE Plan Cache Snapshot

The Show Statements option from SQL Plan Cache Snapshot lists the SQL statements for which a snapshot contains the plan cache details, and further allows a Visual Explain graph to be produced. The SQL statements window is also coupled with filtering capabilities, allowing you to retrieve and review SQL queries based on criteria of your interest as explained in the following steps.

Note: This Show Statements option activated from SQL Plan Cache Snapshot is equivalent to Show Statements option activated from SQL Plan Cache and SQL Performance Monitors.

1. To access the SQL statements window, right-click your snapshot name and select **Show Statements** as shown in Figure 7-28.

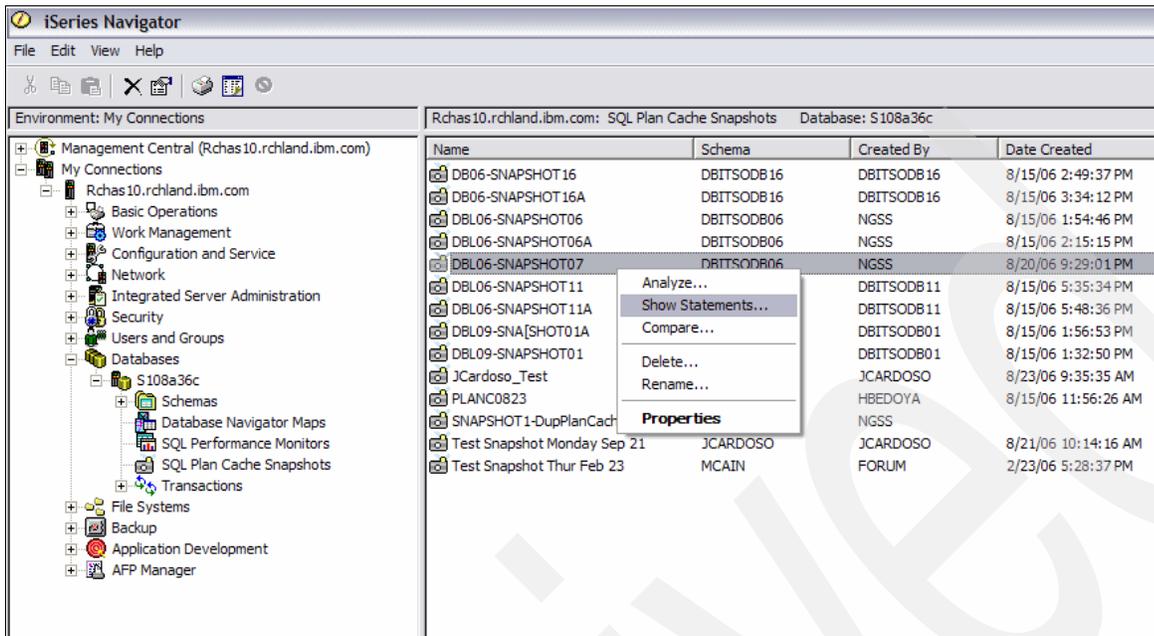


Figure 7-28 Path to snapshot object Show Statements

Note: The Show Statements in V5R4 is the equivalent of List Explainable Statements in V5R3.

There are four filtering criteria that you can use before clicking the Retrieve button to get the SQL statements contained in the SQE Plan Cache Snapshot:

- Minimum runtime for the longest execution
- Queries run after a specific date and time
- Queries that use or reference specific tables
- SQL statements that contain a specified text pattern

- Assuming you are interested in Select statements, use the SQL statement contains filter and type SELECT in the text box, then click **Retrieve**. Refer to Figure 7-29 for the illustration.

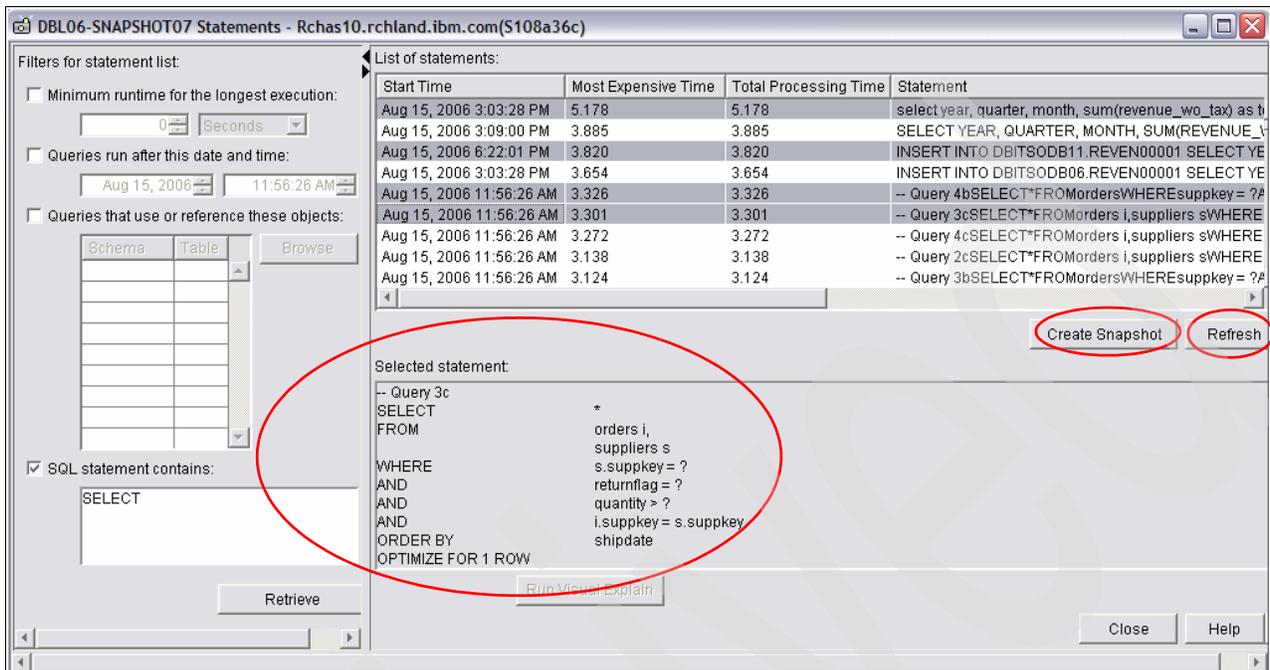


Figure 7-29 Using SQE Plan Cache Snapshot Filtering option to list SQL statements

Note: The text box of the SQL statement contains is *NOT* case sensitive. Typing SELECT or select yields the same result.

- On the right pane, there are two options you can perform:
 - Creating another snapshot for some selected statements
 - Invoking Visual Explain to see the access plan of the selected statement

Clicking any line in the list of statements shows you the full SQL statement in the grey text box on the right pane.

Note: You can select multiple statements by holding down either the Shift key (for a range of multiple lines) or Ctrl key (for selecting multiple individual lines) while clicking the lines.

You can now perform further SQL statement performance analysis using any one of the options, either with the new snapshot you created or with the Visual Explain tool.

7.3.6 An example of finding table scans in a SQE Plan Cache Snapshot

iSeries Navigator provides predefined report to help you in analyzing the performance data in the SQE Plan Cache Snapshot. Assuming you are interested in table scan operations that run many times, use the report to check if there is any possibility of tuning the statement that causes the table scans.

To do so, perform the following steps:

1. From the SQE Plan Cache Snapshot Dashboard window in Figure 7-30, select **Actions** → **Analysis Queries**.

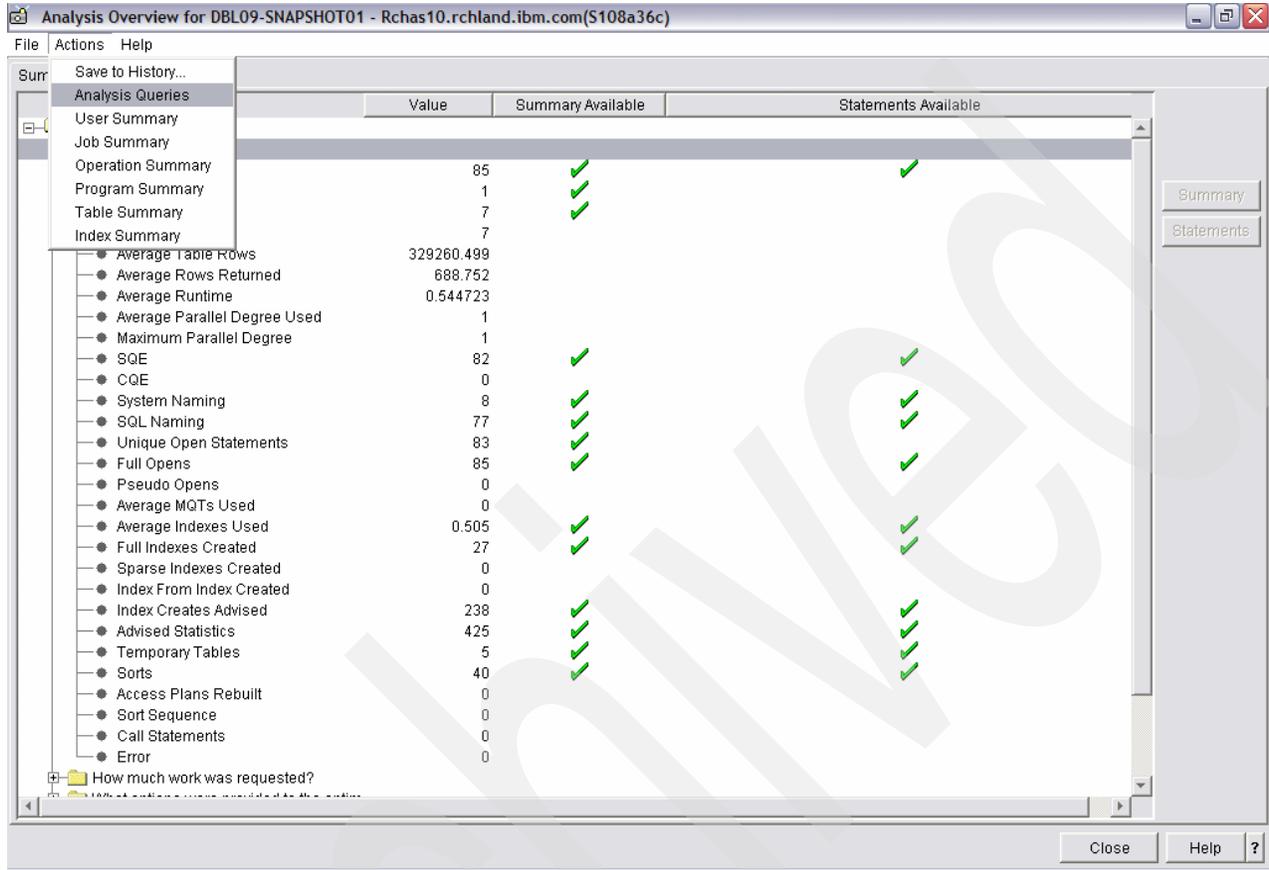


Figure 7-30 The Dashboard

2. As shown in Figure 7-31, now locate and right-click the line **Table Scan Summary** and select **View Results**.

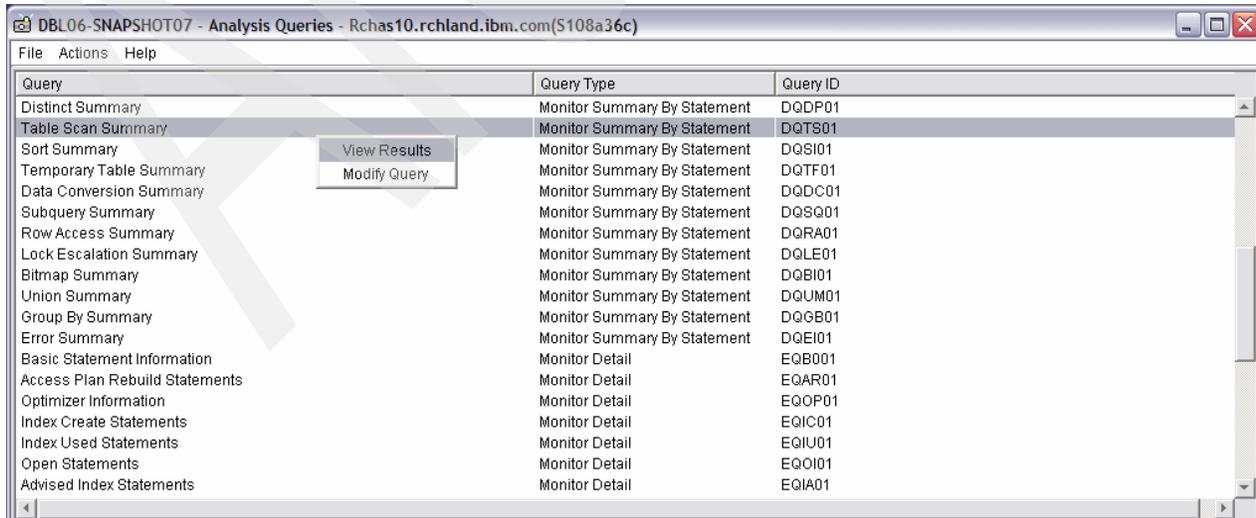


Figure 7-31 View Results on Table Scan Summary

- As shown in Figure 7-32, the result window appears. Scroll to the right to locate the column named Full Opens. Click twice on the column bar to sort the lines in descending order. Highlight the topmost line for easy viewing and scroll further to the column named Index Creates Advised.

Full Opens	Pseudo O...	Average MQTs Us...	Average Indexes Used	Full Indexes Created	Sparse Indexes Created	Index From Index Created	Index Creates Advised
18		0.000	1.000	0	0	0	18
4		0.000		0	0	0	4
4		0.000		0	0	0	8
4		0.000		0	0	0	4
4		0.000		0	0	0	12
4		0.500		0	0	0	6
4		0.000		0	0	0	8
4		0.000		0	0	0	4
3		0.000	2.000	0	0	0	10
2		0.500		0	0	0	3
2		0.000	3.000	0	0	0	9
2		0.000		0	0	0	6
1		0.000		0	0	0	4
1		0.000		0	0	0	2
1		0.000		0	0	0	1
1		0.000		0	0	0	2
1		0.000		0	0	0	1
1		0.000		0	0	0	1
1		0.000		0	0	0	6
1		0.000		0	0	0	1
1		0.000		0	0	0	2
1		0.000		0	0	0	3

Figure 7-32 Lookup Full Opens and Index Creates Advised

This is how you identify the table scan operation with the highest number of Full Open operations (which is an expensive operation in terms of performance). If you see a high number of Index Creates Advised, then this is an opportunity for you to explore that particular line (representing an SQL statement) to see its details for more performance consideration.

- To see the SQL statement, right-click the highlighted line and select **Work with Statements**. This leads you to the Run SQL scripts window with the corresponding statement as shown in Figure 7-33.

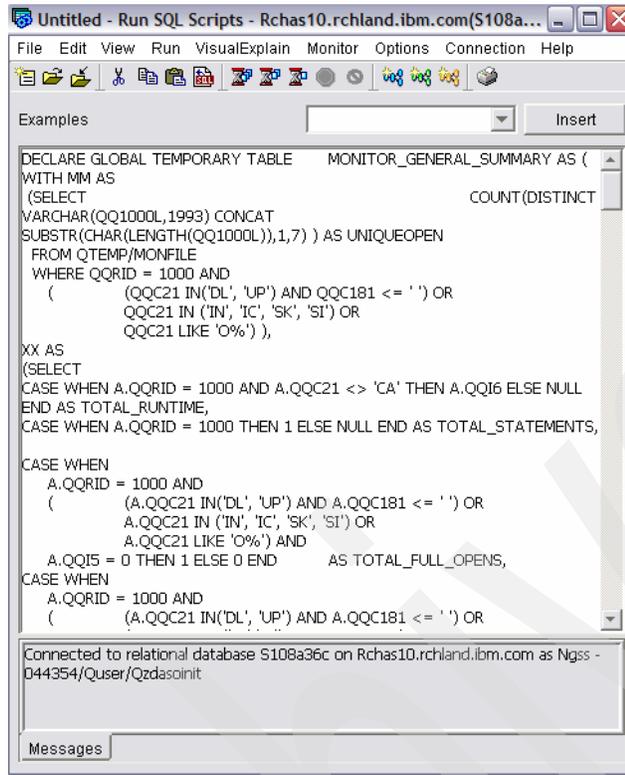


Figure 7-33 Viewing the SQL statements in the Run SQL Scripts window

7.3.7 Comparing SQE Plan Cache Snapshots

The Compare function of SQE Plan Cache Snapshots allows you to compare two sets of SQE Plan Cache Snapshots which contain the query performance monitor data. This function is beneficial for comparing the query performance before and after a change of environment variables such as:

- ▶ Application code changes
- ▶ Operating system changes
- ▶ PTF installation
- ▶ Changes in system value settings or application level settings
- ▶ Introduction of new workload
- ▶ Changes in system hardware resource
- ▶ Changes in infrastructure

With this comparison function, you can review if the environment changes affect your query performance. Also by comparing one snapshots captured before performing query optimization and one captured after the optimization exercise, you can verify if the optimization performed satisfactorily.

In this section, we will compare two SQE Plan Cache Snapshots, one before an MQT table was created and one after the MQT table was created with the following steps.

Note: Take into account the following considerations:

- ▶ If you have created an MQT in your query optimization process, when you use the filtering options to narrow down your selection in the SQL Plan Cache creation stage, you should not filter by using the option “Queries that use or reference these objects” (for example the base table name in your schema). The optimizer refers to the original base table before your MQT creation and it uses the MQT after your MQT creation. With this, you will not find any matching SQL statements for further comparison and analysis.
- ▶ For more information about Materialized Query Table (MQT), refer to Chapter 3, “Overview of tools to analyze database performance” on page 33.

1. As shown in Figure 7-34, to compare two SQE Plan Cache Snapshots, right-click the **SQE Plan Cache name** and select **Compare**.

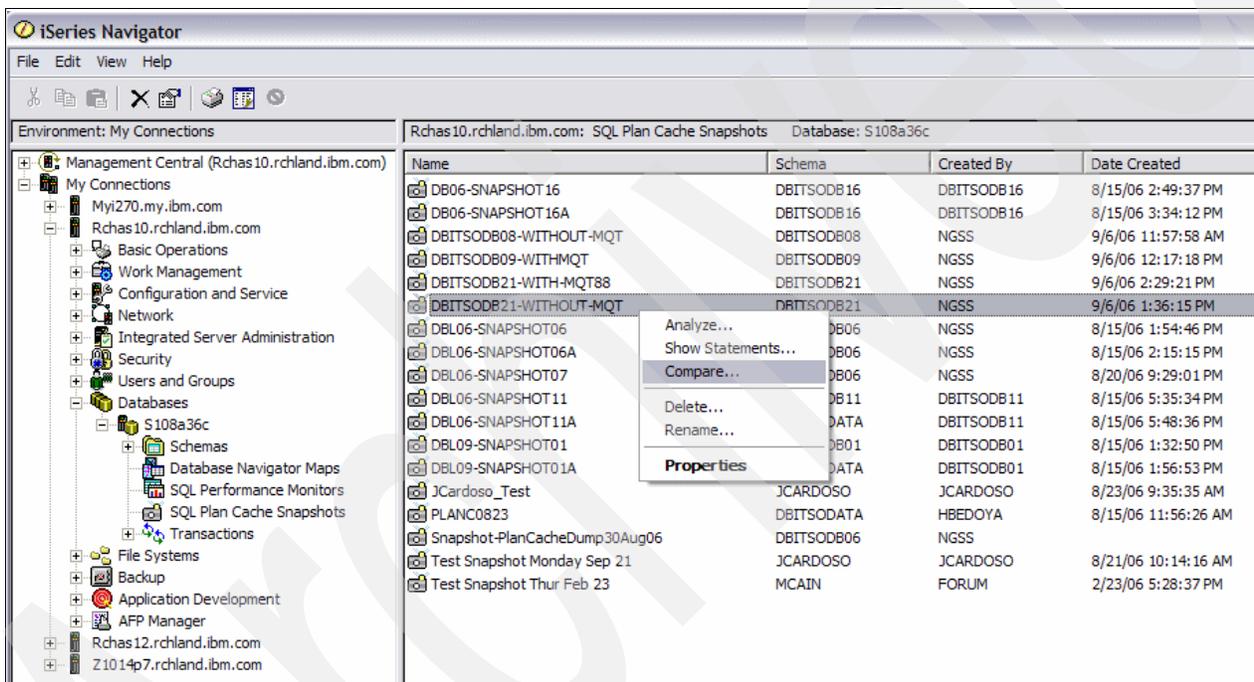


Figure 7-34 Path to compare SQE Plan Cache Snapshots

2. As shown in Figure 7-35, fill the following fields:

- **Data set 1:** Use Data set 1 to specify information about the first monitor. You can select a monitor name from the Name field. Then select any names that you want the compare to ignore.
- **Data set 2:** Use Data set 2 to specify information about the second monitor. You can select a monitor name from the Name field. Then select any names that you want the compare to ignore.
- **Compare statements that ran longer than:** Specifies the minimum runtime for statements to be compared.
- **Minimum percent difference:** The minimum difference in key attributes of the two statements being compared that determines if the statements are considered equal or not.

Note: This is an example of Schema Mask usage: you have an application running in a test schema and have it optimized. Now you move it to the production schema and you want to compare how it executes there. The statements in the compare are identical except that the statements in the test schema use “TEST” and the statements in the production schema use “PROD”. You can use the schema mask to ignore “TEST” in the first monitor and to ignore “PROD” in the second monitor so that the statements in the two monitors appear identical.

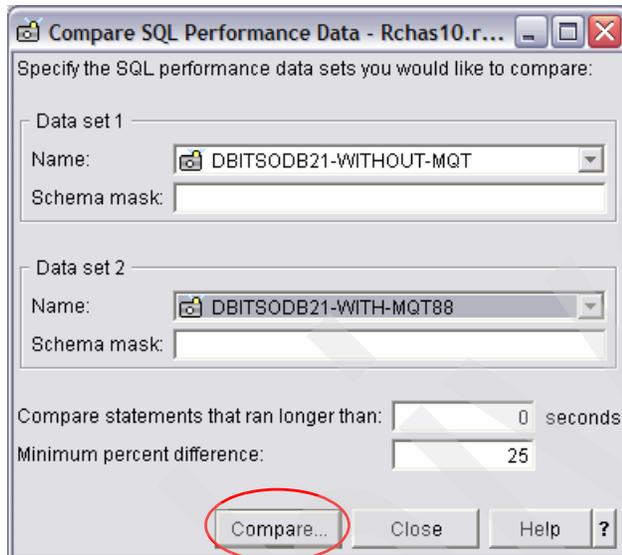


Figure 7-35 Specify two sets of snapshot

3. Once you have selected data set 1 and data set 2, click **Compare**.

You see the window in Figure 7-36 with the SQL statements compared between the two snapshots. You can expand the “+” sign to explore further. You can also use the Show

Statement function on the snapshot set or Run Visual Explain on a selected SQL statement.

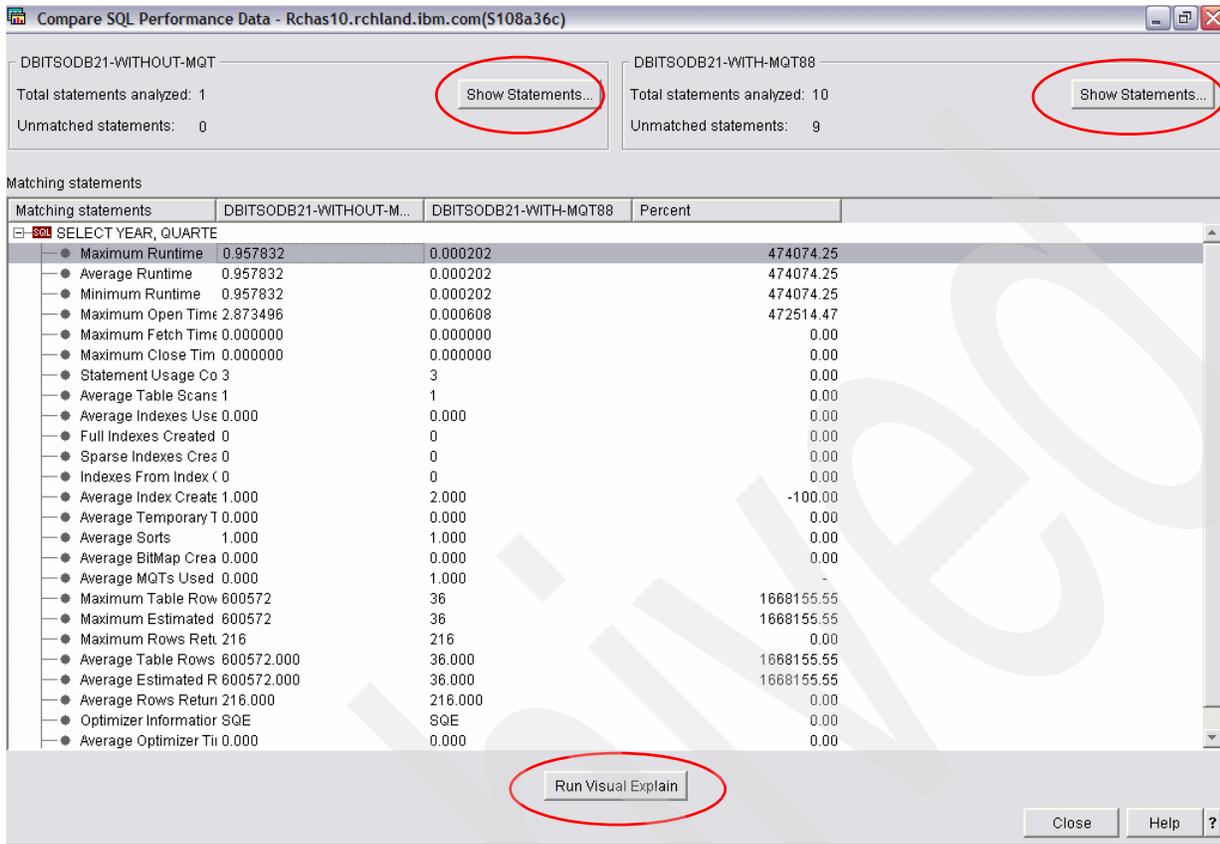


Figure 7-36 Comparison of two snapshots

4. Select the line that you are interested in, for example the Maximum Runtime line and click **Run Visual Explain** to view the graphical representation of the optimizer implementation

of a query request. In Figure 7-37, you see the two sets of Visual Explain display, one for the first snapshot and the other for the second snapshot in comparison.

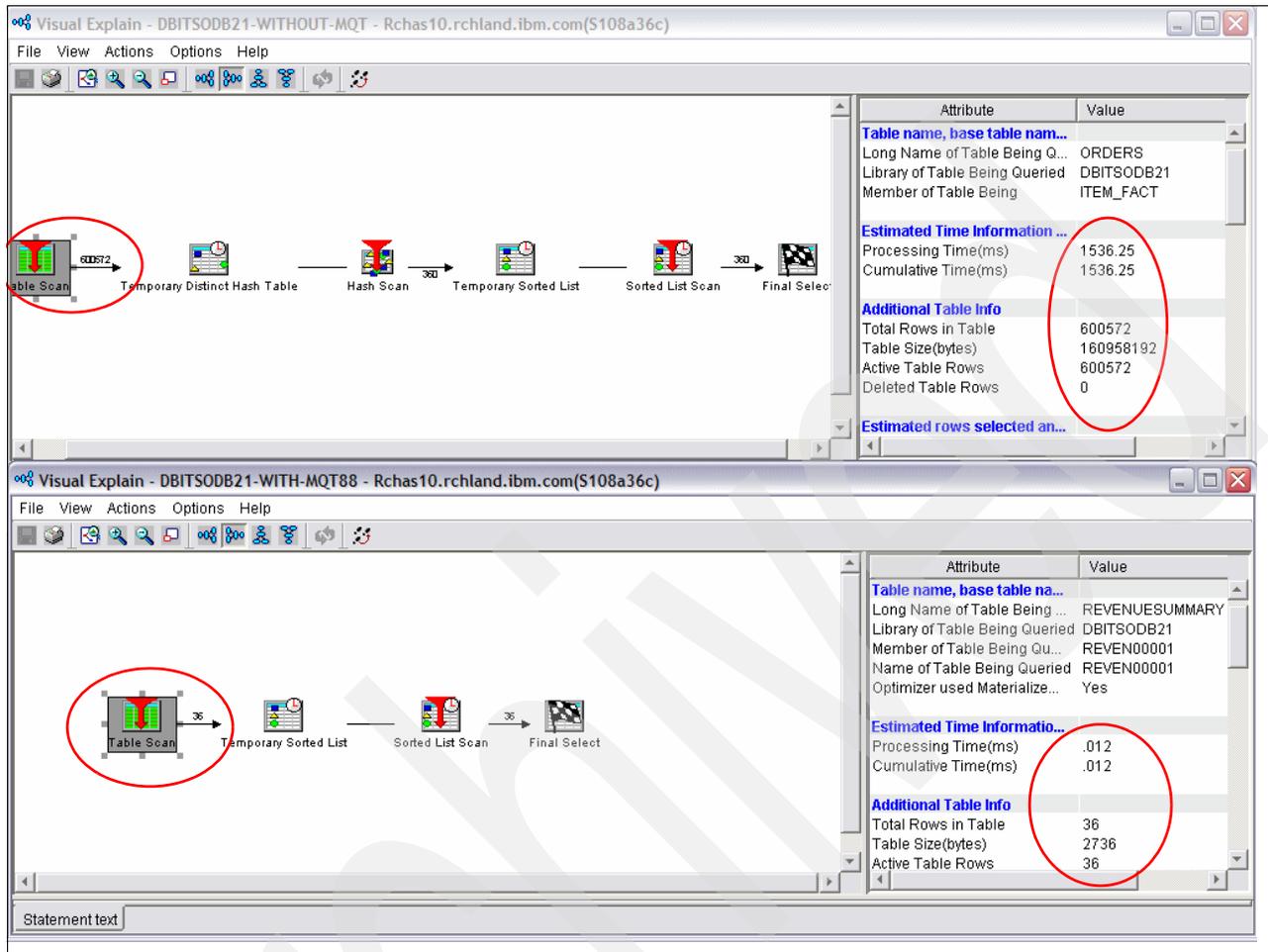


Figure 7-37 Visual Explain for comparing two snapshots

You can now compare both snapshots, implementation methods graphically by viewing the left hand pane, and statistically on the right hand pane.

In this example, you can see that the top visual explain picture shows that a table scan with total 600,572 rows resulting in total estimated processing time of 1536.25ms and bottom visual explain picture shows that a table scan with total 36 rows resulting in total estimated processing time of 0.12ms. By analyzing both graphical details on the left pane and statistical details on the right pane for both snapshots, you can easily make a comparison if the optimization you performed is satisfactory.

Note: Refer to Chapter 8, “Analyzing database performance data with Visual Explain” on page 275 for more information about using Visual Explain as a tool for query performance analysis and optimization.

Archived

Analyzing database performance data with Visual Explain

The launch of Visual Explain with iSeries Navigator Version 4, Release 5, Modification 0 (V4R5M0) in DB2 Universal Database for iSeries was of great interest to database administrators working in an iSeries server environment. This feature has been described as a quantum leap forward in database tuning for query optimization. Visual Explain provides an easy-to-understand graphical interface that represents the optimizer implementation of the query.

For the first time, you can see, in graphic detail, how the optimizer has implemented the query. You can even see all of the facts and figures that the optimizer used to make its decisions. Best of all, the information is presented in one place, in color, with easy-to-follow displays. There is no more jumping between multiple windows, trying to determine what is happening. Even better, if you currently have iSeries Navigator, you already have Visual Explain.

With all of this in mind, is such a richly featured product complicated to use? As long as you are familiar with database tuning, you will enjoy using Visual Explain and want to learn more.

This chapter answers the following questions:

- ▶ Where do I find Visual Explain?
- ▶ How do I use it?
- ▶ What can it be used for?
- ▶ Will it tune my Structured Query Language (SQL) queries?
- ▶ How can we integrate Visual Explain with SQE Plan Cache, SQE Plan Cache Snapshot, The Detailed Database Monitors and Current SQL for a Job?

8.1 What is Visual Explain

Visual Explain provides a graphical representation of the optimizer implementation of a query request. The query request is broken down into individual components with icons that represent each unique component. Visual Explain also includes information about the database objects that are considered and chosen by the query optimizer. Visual Explain's detailed representation of the query implementation makes it easier to understand where the greatest cost is being incurred.

Visual Explain shows the job run environment details and the levels of database parallelism that were used to process the query. It also shows the access plan in diagram form, which allows you to zoom to any part of the diagram for further details.

If query performance is an issue, Visual Explain provides information that can help you to determine whether you need to:

- ▶ Rewrite or alter the SQL statement
- ▶ Change the query attributes or environment settings
- ▶ Create new indexes

Best of all, you do not have to run the query to find this information. Visual Explain has a modeling option that allows you to explain the query without running it. That means that you can try any of the changes suggested and see how they are likely to work, before you decide whether to implement them. Furthermore, in V5R4 you have the option Explain While Running which provides more accurate visual explanation.

You can also use Visual Explain to:

- ▶ View the statistics that were used at the time of optimization
- ▶ Determine whether an index was used to access a table
If an index was not used, Visual Explain can help you determine which columns might benefit from being indexed.
- ▶ View the effects of performing various tuning techniques by comparing the before and after versions of the query graph
- ▶ Obtain information about each operation in the query graph, including the total estimated cost and number of rows retrieved
- ▶ View the debug messages issued by the query optimizer during the query execution

Visual Explain is an advanced tool to assist you with the task of enhancing query performance, although it does not actually do this task for you. You still need to understand the process of query optimization and the different access plans that you can implement.

Visual Explain is a perfect match with the SQL Plan Cache, SQE Plan Cache snapshot and Detailed Database Monitors.

8.2 Finding Visual Explain

Visual Explain is a component of iSeries Navigator and is available under the Databases icon. To locate the Databases icon, you must establish a session on your selected iSeries server using the iSeries Navigator icon.

From the SQL Script Center, you can access Visual Explain directly, either from the menu or from the toolbar as explained in 8.2.1, "The SQL Script Center" on page 278.

Visual Explain is based on *detailed* optimizer information. As shown in Figure 8-1, the optimizer sends detailed feedback information to SQE Plan Cache and to Detailed Database Monitor. In V5R4 you can also drill down into Visual Explain from the following interfaces:

- ▶ SQE Plan Cache
- ▶ SQE Plan Cache Snapshots which is based on SQE Plan Cache information
- ▶ Detailed Database Monitor

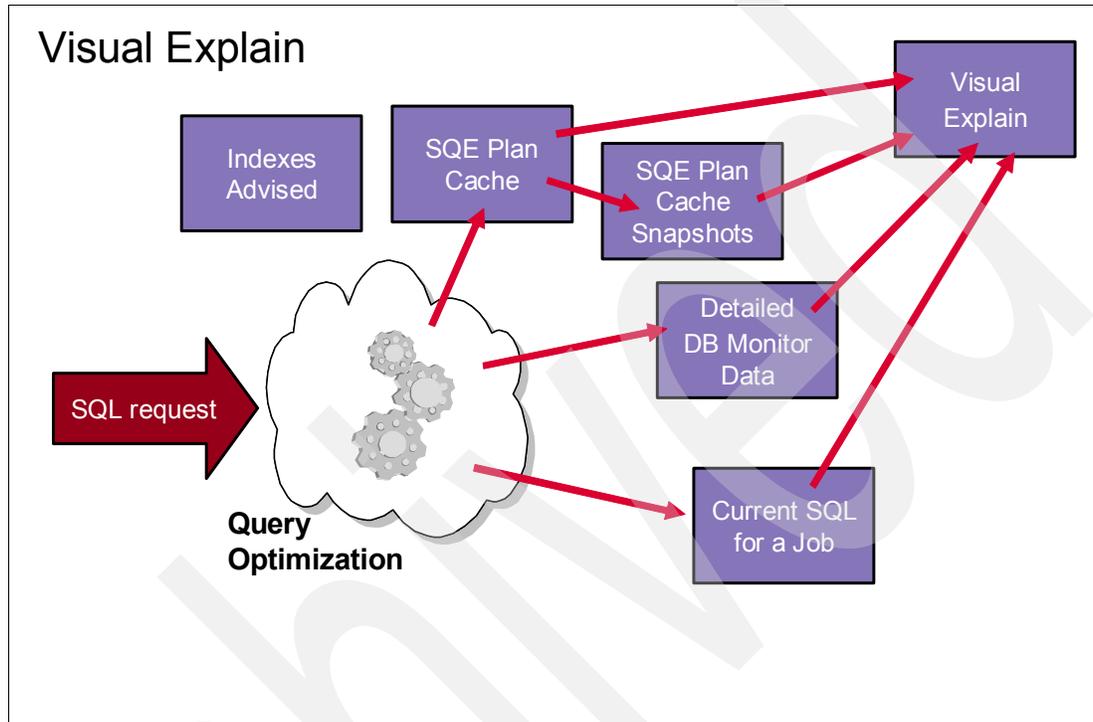


Figure 8-1 Feedback Mechanisms integrated with Visual Explain

Note: Visual Explain supports both SQE and CQE as described below:

- ▶ SQE
 - SQE Plan Cache
 - SQE Plan Cache Snapshot
 - Detailed Database Monitor
 - Current SQL for a Job
 - Run SQL Script
- ▶ CQE
 - Detailed Database Monitor
 - Run SQL Script

Also from the Current SQL for a Job window we can drill down into Visual Explain by clicking the **Visual Explain** button as shown in Figure 8-2.

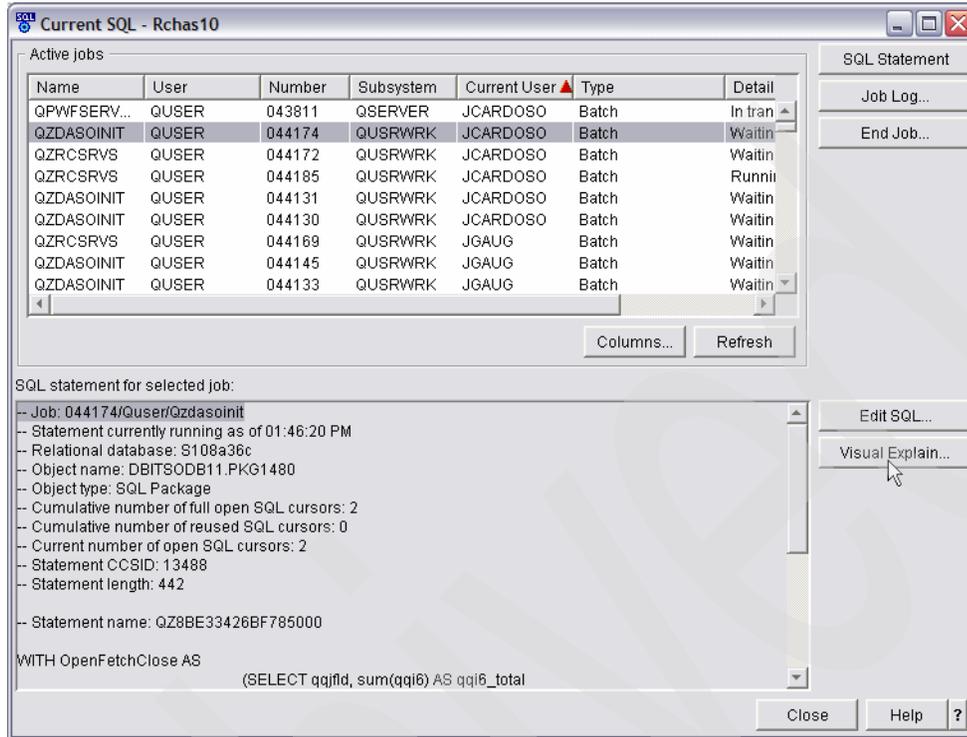


Figure 8-2 Drilling down into Visual Explain from Current SQL for a Job window

Another way to access Visual Explain is through the SQL Performance Monitors. SQL Performance Monitor is used to create Database Monitor data and to analyze the monitor data with predefined reports.

Visual Explain works with the monitor data that is collected by SQL Performance Monitor on that system or by the Start Database Monitor (STRDBMON) command. Visual Explain can also analyze Database Monitor data that is collected on other systems after data is restored on the iSeries server.

For more information about the SQL Performance Monitor, refer to Chapter 5, “Analyzing SQL performance data using iSeries Navigator” on page 117.

The Run SQL Script window (SQL Script Center) provides a direct route to Visual Explain. The window is used to enter, validate, and execute SQL commands and scripts and to provide an interface with i5/OS through the use of CL commands.

8.2.1 The SQL Script Center

To access the SQL Scripts Center, in iSeries Navigator, expand **Databases**. Then select any database, right-click, and select **Run SQL Scripts**. The Run SQL Scripts window (shown in Figure 8-3) opens with the toolbar. Reading from left to right, there are icons to create, open, and save SQL scripts, followed by icons to cut, copy, paste, and insert generated SQL (from V5R1) statements within scripts.

The hour glass icons indicate to run the statements in the Run SQL Scripts window. From left to right, they are run all of the statements (Run All), run all of the statements from the cursor

to the end (Run From Selected), or run the single statement identified by the cursor position (Run Selected).

To the right of the hour glasses is a Stop button, which is the color red when a run is in progress.

There are three Visual Explain icons in the colors blue, green and orange. The left Visual Explain icon (blue-Explain Only) helps to explain the SQL statement. The middle Visual Explain icon (green-Run And Explain) enables you to run and explain the SQL statement. The right Visual Explain icon (orange-Explain While Running) enables you to *explain* while *running* the SQL statement.



Figure 8-3 Toolbar from Run SQL Scripts

These three options are also available from the Visual Explain menu (as shown in Figure 8-4). You can choose either option to start Visual Explain.



Figure 8-4 SQL Scripts Center Visual Explain options

The final icon in the toolbar is the Print icon.

You can use SQL Performance Monitors to record SQL statements that are explainable by Visual Explain. We recommend that you obtain access via the SQL Performance Monitors icon, because it provides the full list of monitors.

8.2.2 Explain Only

The Visual Explain Only option (Ctrl + E or the blue toolbar icon) submits the query request to the optimizer and provides a visual explanation of the SQL statement and the access plan that will be used when executing the statement. In addition, it provides a detailed analysis of the results through a series of attributes and values associated with each of the icons. It does not actually run the SQL statement.

To optimize an SQL statement, the optimizer validates the statement, gathers statistics about the SQL statement, and creates an access plan. When you choose the Visual Explain Only option, the optimizer processes the query statement internally with the query time limit set to zero. Therefore, it proceeds through the full validation, optimization, and creation of an access plan and then reports the results in a graphical display.

Note: When you choose Visual Explain Only, Visual Explain might not be able to explain such complex queries as hash join, temp join results, and so on. In this case, you must choose *Explain While Running* or *Run and Explain* for the SQL statements to see the graphical representation.

8.2.3 Run and Explain

The Run and Explain option (Ctrl + U or the green toolbar icon) also submits the query request to the optimizer. It provides a visual explanation of the SQL statement and the access plan used when executing the statement. It provides a detailed analysis of the results through a series of attributes and values associated with each of the icons.

However, it does not set the query time limit to zero and, therefore, continues with the execution of the query. This leads to the display of a results window in addition to the Visual Explain graphics.

Note: Consider the following:

- ▶ Visual Explain might show a representation that is different from the job or environment where the actual statement was run since it might be explained in an environment that has different work management settings.
- ▶ If the query is implemented with multiple steps (that is, joined into a temporary file, with grouping performed over it), the Visual Explain Only option cannot provide a valid explanation of the SQL statement. In this case, you must use the *Explain While Running* or *Run and Explain* options.

8.2.4 Explain While Running

Visual Explain was also enhanced in V5R4. One of the enhancements is the *Explain While Running* tool. Before V5R4 long-running queries could be visual explained by using Visual Explain Only option. Sometimes Explain Only did not provide an accurate graph. In V5R4, Explain While Running option provides a better and more accurate graph to visual explain long-running queries. There are two ways to access Explain While Running from Run SQL Script interface:

- ▶ Click the **Explain While Running** button (as shown in Figure 8-5).

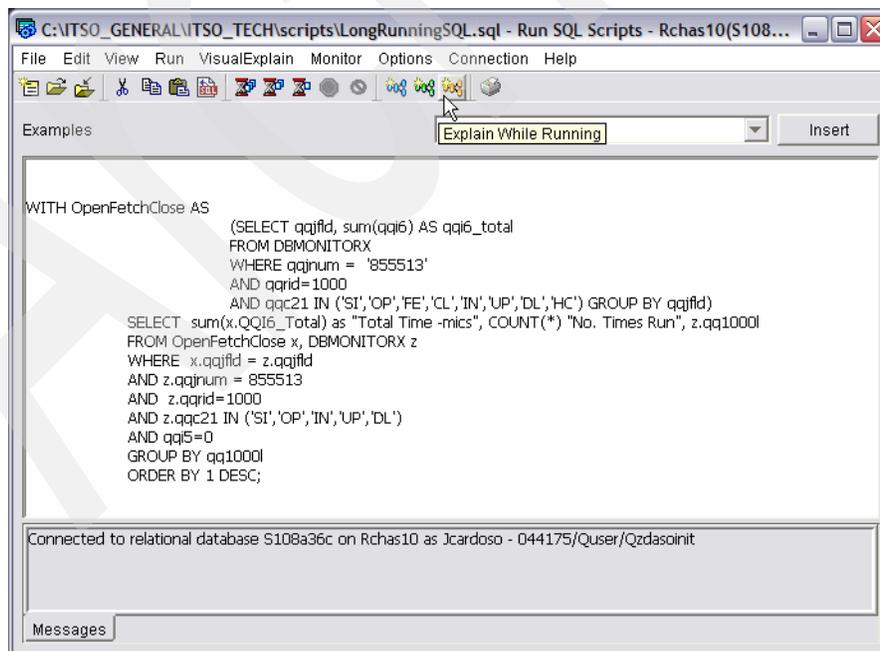


Figure 8-5 Explain While Running button in Run SQL Scripts window

- Select **Visual Explain** → **Explain While Running** menu items (as shown in Figure 8-6).

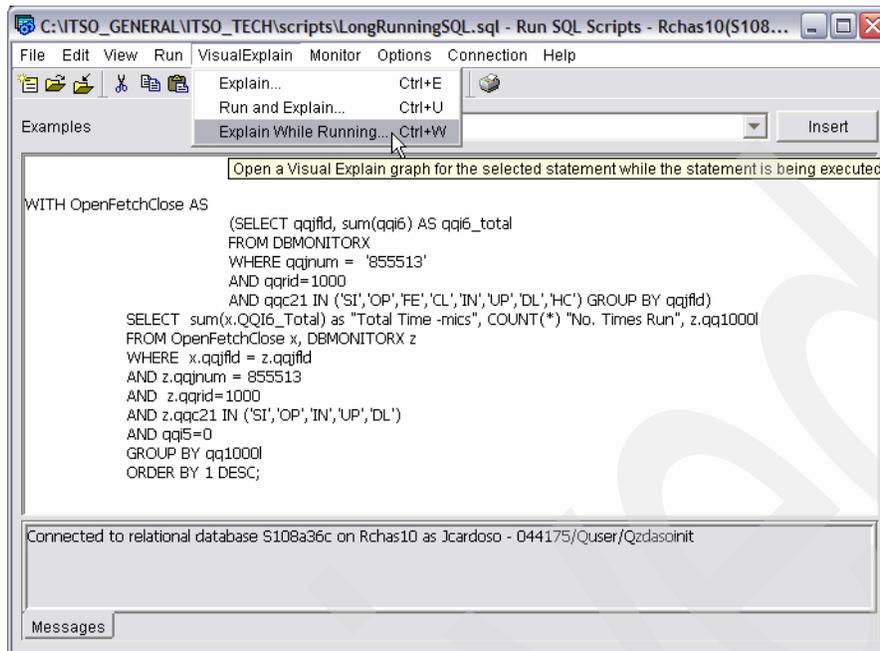


Figure 8-6 Explain While Running menu item of Run SQL Scripts window

8.3 Navigating Visual Explain

The Visual Explain graphics window (Figure 8-7) is presented in two parts. The left side of the display is called the *Query Implementation Graph*. This is the graphical representation of the implementation of the SQL statement and the methods used to access the database. The arrows indicate the order of the steps. Each node of the graph has an icon that represents an operation or values returned from an operation.

The right side of the display has the *Query Attributes and Values*. The display corresponds to the object that has been selected on the graph. The query attributes and values correspond

to the final results icon. The vertical bar that separates the two sides is adjustable. Each side has its own window and is scrollable.

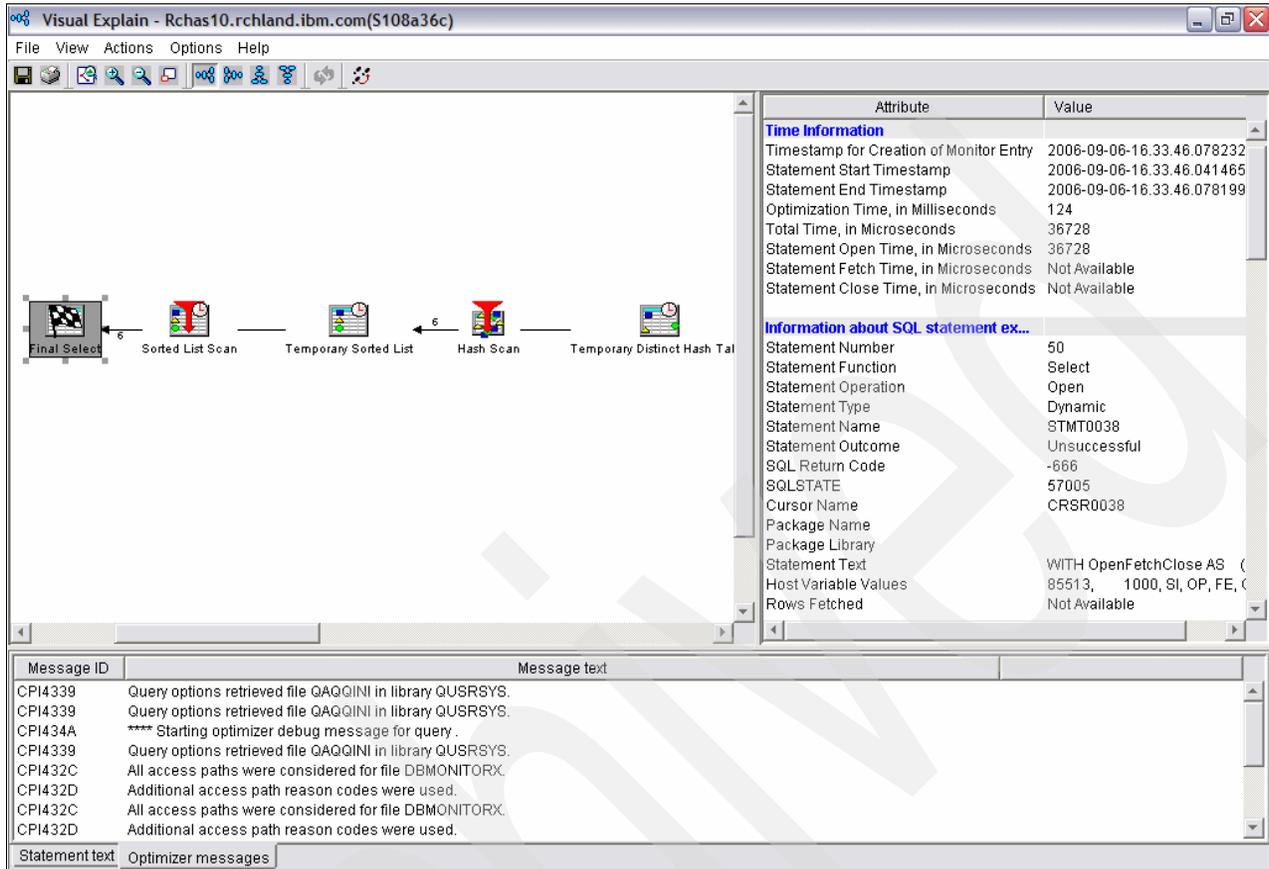


Figure 8-7 Visual Explain Query Implementation Graph and Query Attributes and Values

The default settings cause the display to be presented with the final result icon (a checked flag) on the left of the display. Each icon on the display has a description and the estimated number of rows to be used as input for each stage of the implementation.

Clicking any of the icons causes the Query Attributes and Values display to change and presents the details that are known to the query for that part of the implementation. You might find it helpful to adjust the display to see more of the attributes and values. Query attributes and values are discussed further in 8.3.5, “Visual Explain query attributes and values” on page 294.

When you right-click any of the icons on the display, an action menu is displayed. The action menu has options to assist with query information. It can provide a shortcut to table information to be shown in a separate window. You can find more details in the Action menu options section of 8.3.2, “Menu options” on page 288.

By moving the mouse pointer over the icon, a window appears with summary information about the specific operation. Refer to Figure 8-8.

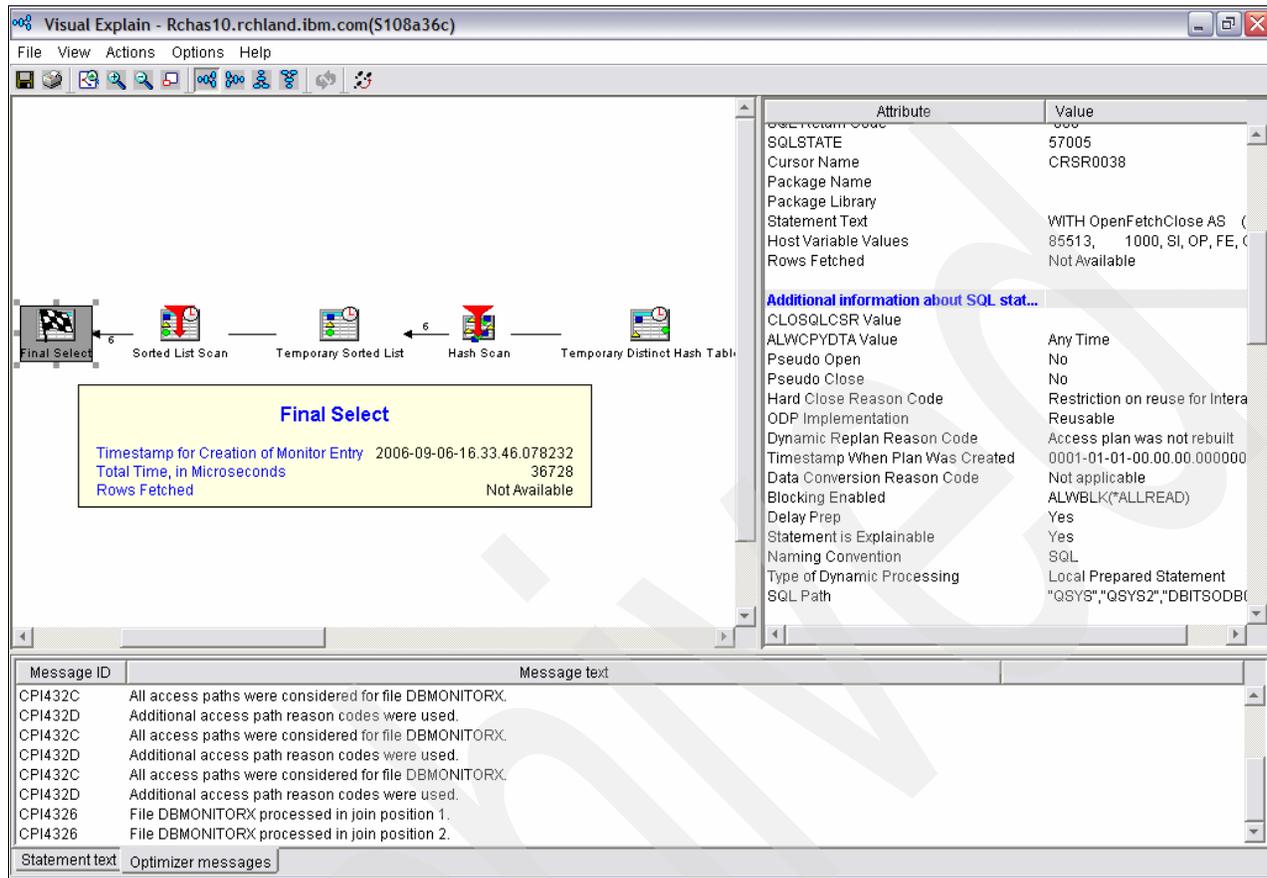


Figure 8-8 Final Select Flyover window

8.3.1 Toolbar

The Visual Explain toolbar (Figure 8-9) helps you to navigate the displays. The first four icons (from left to right after the printer icon) help you to control the sizing of the display. The left-most icon scales the graphics to fit the main window. For many query implementations, this leaves the graphical display too small to be of value. The next two icons allow you to zoom in and out of the graphic image.

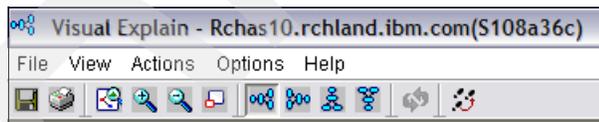


Figure 8-9 Visual Explain toolbar

The fourth icon (Overview) creates an additional window (shown in Figure 8-10) that shows the Visual Explain graphic on a reduced scale. This window has a highlighted area, which represents the part of the image that is currently displayed in the main window.

In the Overview window (as shown in Figure 8-10), you can move the cursor into this highlighted area that is shown in the main window. The mouse pointer changes so you can

drag the highlighted area to change the section of the overall diagram that is shown in the main window.

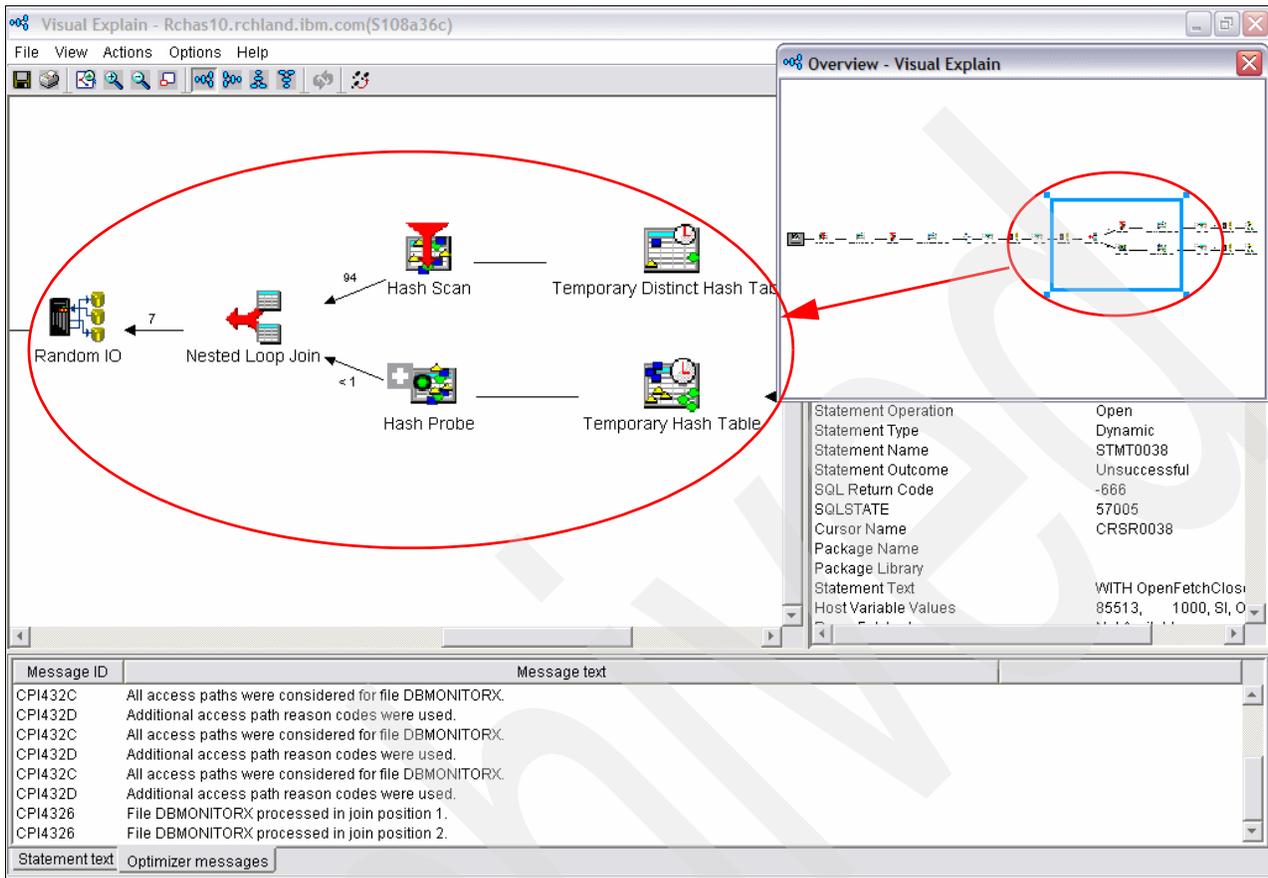


Figure 8-10 Visual Explain Overview window

The default schematic shows the query with the result on the left, working across the display from right to left, to allow you to start at the result and work backward. The next four icons on the Visual Explain toolbar allow you to rotate the query implementation image. The icons are:

- ▶ Starting from the right, leading to the result on the left (default view)
- ▶ Starting from the left, leading to the result on the right
- ▶ Starting at the bottom, leading to the result at the top
- ▶ Starting from the top, leading to the result at the bottom

Try these icons to see which style of presentation you prefer. Starting in V5R1, a frame at the bottom of the main Visual Explain window was added. In this frame, you can see two tabs. The *Statement Text* tab shows the analyzed SQL statement. Also in V5R1, when Visual Explain is used, it activates the Include Debug Messages in Job Log option and conveniently presents those messages under the *Optimizer Messages* tab.

The second to the last icon (two cycling arrows), Refresh the Visual Explain picture, allows you to refresh the Visual Explain graphical picture with runtime information. This icon is selectable if you select Explain While Running option from the Run SQL Script window. It is not available if you select Visual Explain Only option nor Run and Explain option from the Run SQL Script window.

You can use the last icon (three steps), Statistics and Index Advisor (new in V5R2), to look at what the query optimizer recommends for you on the indexes and statistic collection.

In the Index Advisor area, the query optimizer analyzes the row selection in the query and determines, based on default values, if creation of a permanent index improves performance. If the optimizer determines that a permanent index is beneficial, it returns the key columns necessary to create the suggested index. This advice is shown in the Index Advisor window as shown in Figure 8-11.

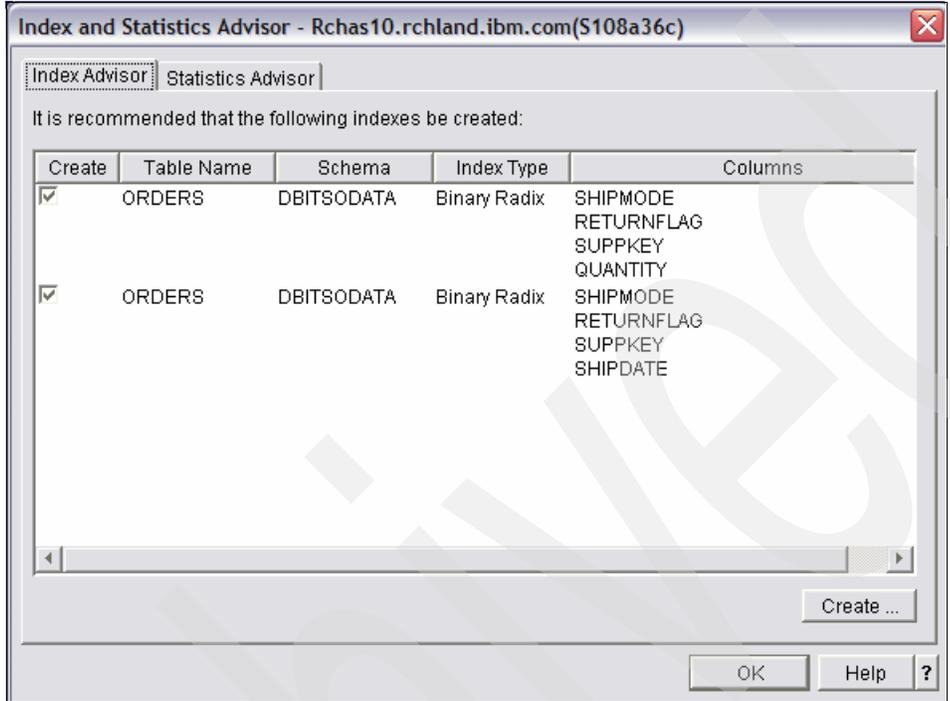


Figure 8-11 Index Advisor in Visual Explain

You can select the index that you want and click the **Create** button to create the index selected as shown in Figure 8-12.

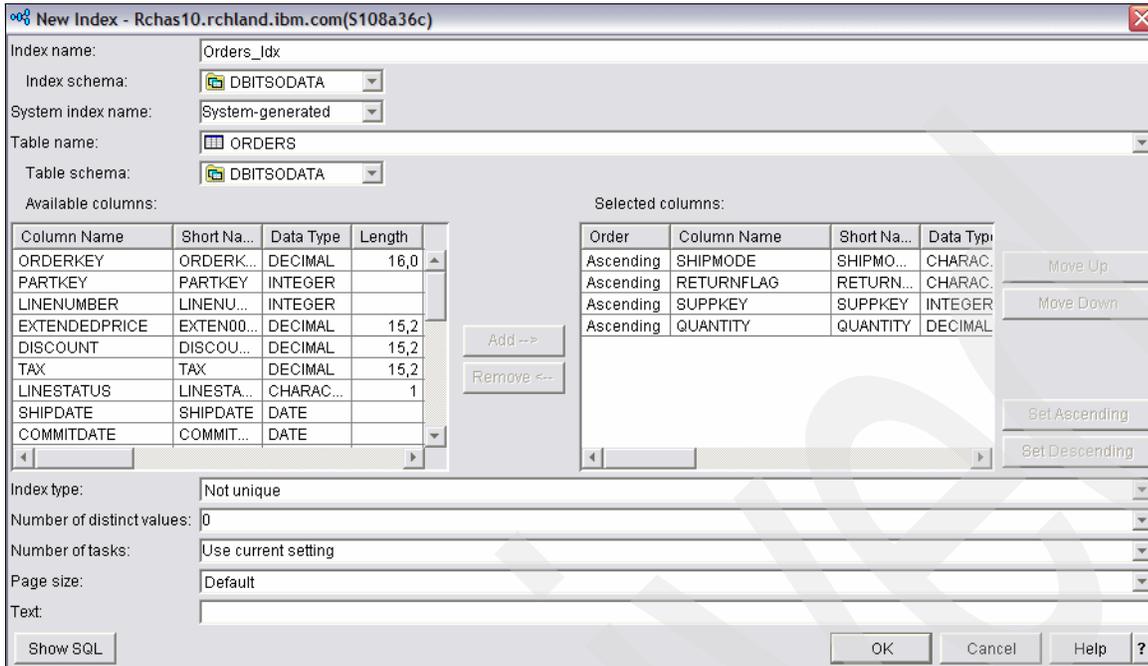


Figure 8-12 Create Index window

You have to enter an index and schema name. The type of index is assumed to be binary radix with non-unique keys.

Note: The Create Index menu item is available from any icon where an index is advised (for example, table scan, key positioning, key selection) in addition to the temp index icon. This is one of the easy-to-use features of Visual Explain. It gives you the ability to easily create an index that the optimizer has suggested.

Use the Statistics Data window (shown in Figure 8-13) to view statistical data for the columns in the selected table. The statistical information can be used by the query optimizer to determine the best access plan for a query. The data presented here can be used to decide whether to collect more statistics in the run immediately mode or background mode, and also

allows you to find the estimated statistics collection time for a selected item before you actually submit the statistics collection task.

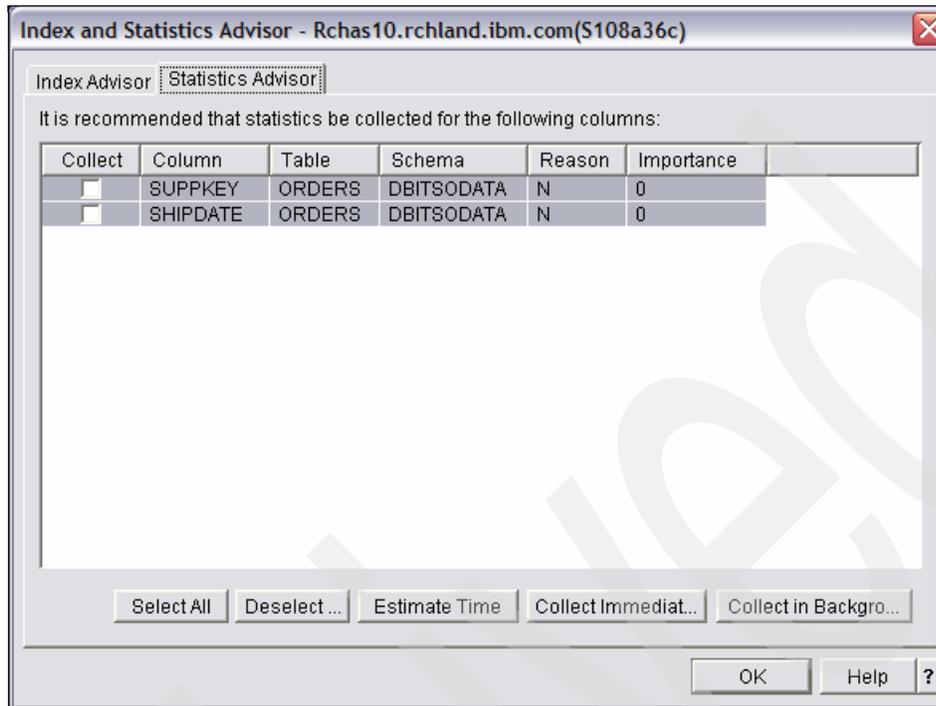


Figure 8-13 Statistics Advisor in Visual Explain

8.3.2 Menu options

The menu options above the toolbar icons are File, View, Actions, Options, and Help. Starting with V5R1, the ability to either print or save the Visual Explain output as an SQL Performance Monitor file was added as shown in Figure 8-14.

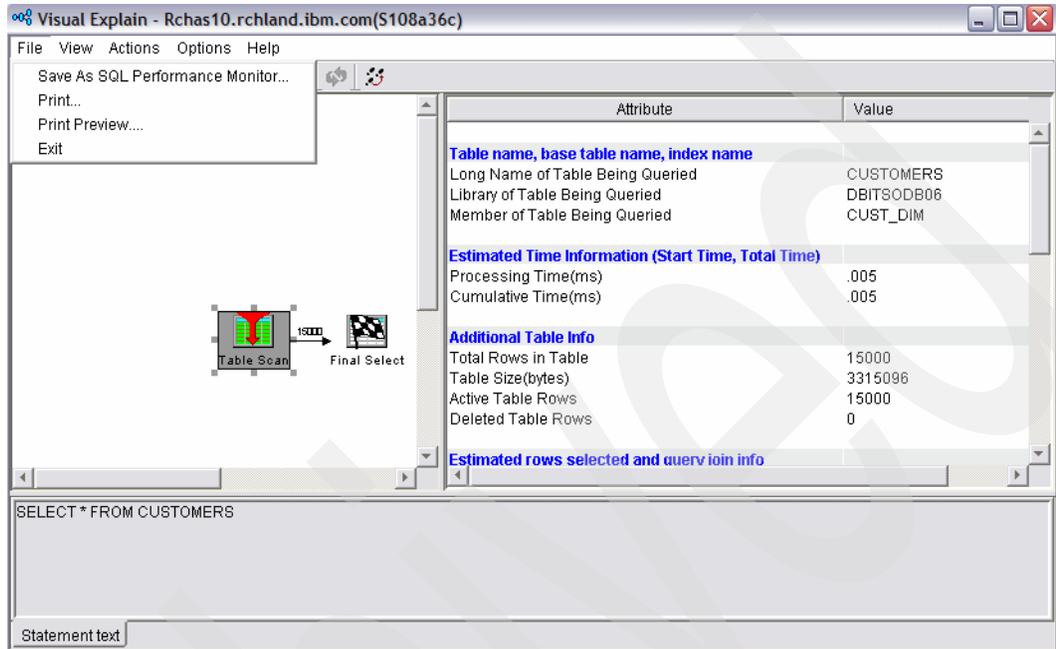


Figure 8-14 The File menu options

View menu options

The View options generally replicate the toolbar icons (shown in Figure 8-15).

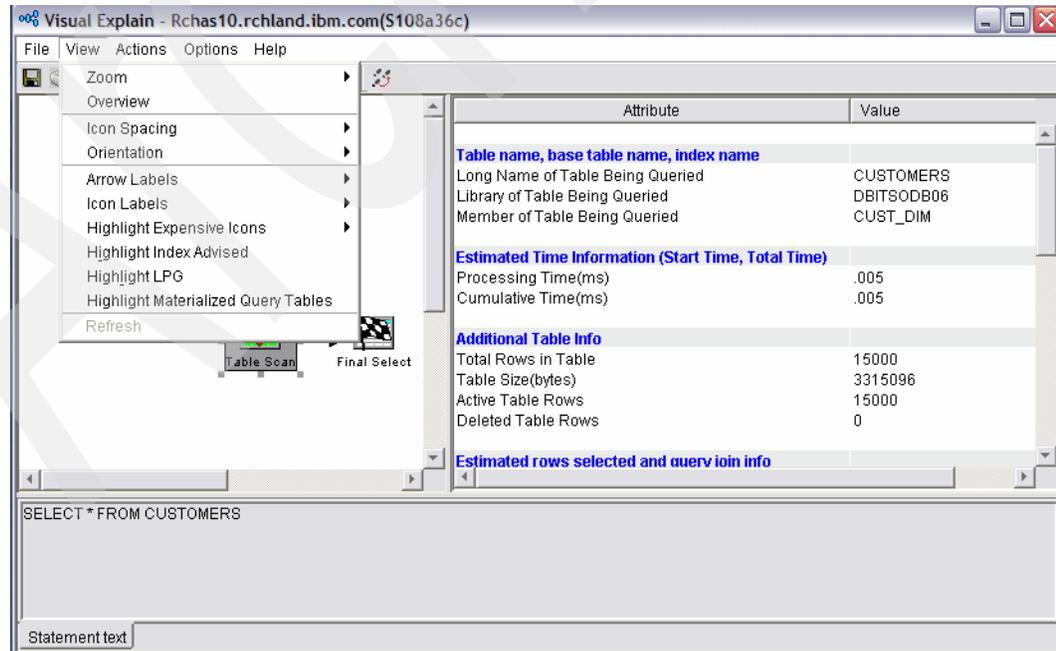


Figure 8-15 The View menu options

The View menu options are:

- ▶ Zoom allows you to increase or decrease the icon size or even fit all icons of the graph into the Visual Explain window.
- ▶ Overview allows you to display the entire graph in the Overview window.
- ▶ Icon spacing allows you to either increase or decrease the space between icons, both horizontally and vertically.
- ▶ Orientation provides you with the flexibility to orient the graph with the final result icon, either on the left, right, top or bottom.
- ▶ Arrow labels allow you to show or hide the estimated number of rows, processing time, or the degree of parallelism that the query is processing at each stage of the implementation.
- ▶ Icon labels allow you to show or hide the description of the icons and the object name associated with the icon.
- ▶ You can highlight expensive icons by number of returned rows and processing time.
- ▶ You can also highlight advised indexes and icons in your graph where the optimizer recommends that you create a permanent index. The icon is highlighted in the graph and the relevant text fields are highlighted in the icon attributes and values table in the right pane of Visual Explain window.
- ▶ The optimizer can also use the Look Ahead Predicate Generation to minimize the random I/O costs of a join. To highlight predicates that used this method, select the Highlight LPG menu option.
- ▶ You can also use the Highlight Materialized Query Table menu option to highlight icons where Materialized Query Tables were used in the queries.
- ▶ For long-running queries, you can refresh the visual explain graph with runtime statistical information before the query is complete. Refresh also updates the appropriate information in the icon attributes section shown on the right pane of the Visual Explain window. In order to use the Refresh option, select **Explain while Running** from the Run SQL Scripts window.

Actions menu options

The Actions menu item replicates the features that are available on the display (shown in Figure 8-16).

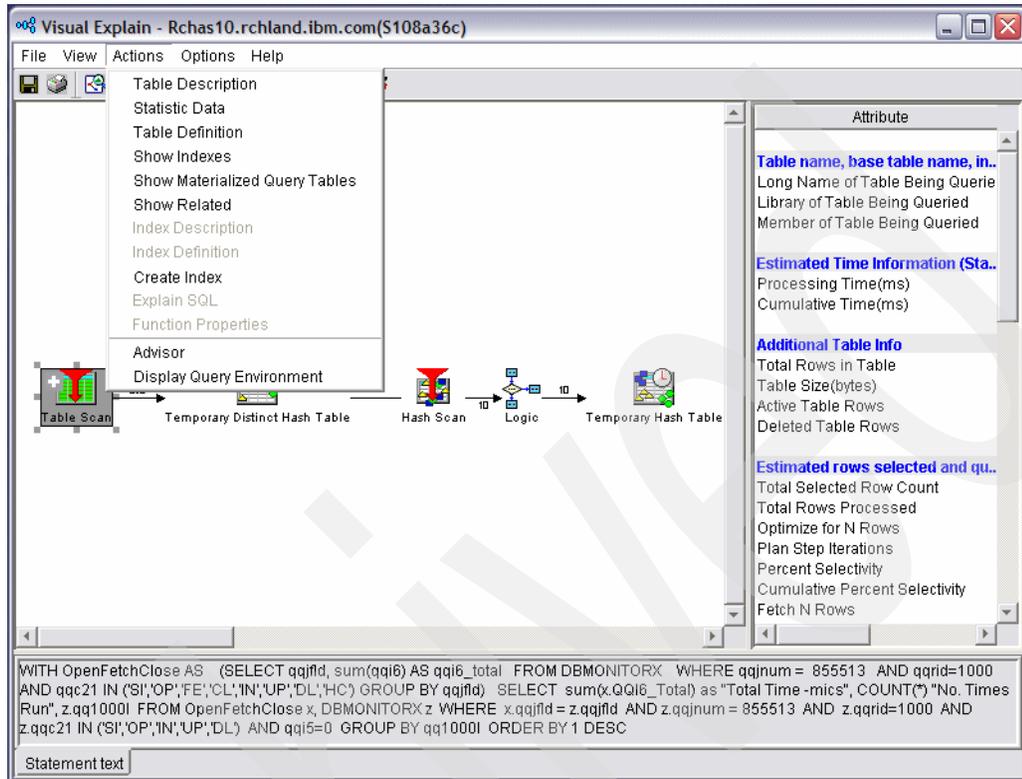


Figure 8-16 The Actions menu option

You might find the following action menu items selectively on different icons:

- ▶ **Table Description:** The Table Description menu item takes you into the graphical equivalent of the DSPFD command. From here, you can learn more information about the file. The description has several tabs to select to find further information. A limited number of changes can be made from the different tab windows. Displays table information returned by the Display File Description (DSPFD) command.
- ▶ **Statistics Data:** You can use the Statistics Advisor function to identify missing or stale statistics and to specify how the statistics will be collected.
- ▶ **Table Definition:** The Table Definition (Table properties in V5R1) menu item opens the same window that is used to create the table. It enables you to change the definition of the table including columns, constraints, and partitions.
- ▶ **Show Indexes:** Displays a list of the indexes for the selected table.
- ▶ **Show Materialized Query Tables:** Display a list of the MQTs for the selected table.
- ▶ **Show Related:** Display a list of objects related to the selected item.
- ▶ **Index Description:** The Index Description attributes can be accessed to obtain further information about the index. Several changes are allowed to an index from these windows, including access path maintenance settings.
- ▶ **Index Definition:** The Index Definition window shows the columns that exist in the table. A sequential number is placed next to the columns that form the index, with an indication

of whether the index is ascending or descending. The display also shows the type of index.

- ▶ **Create Index:** From the temporary index icon, the Create Index menu item takes you to a window where the attributes of the temporary index are completed. Simply click OK to create a permanent index.
- ▶ **Table Properties:** Displays object properties.
- ▶ **Explain SQL:** Display SQL information for the selected object.
- ▶ **Function Properties:** Display function properties.
- ▶ **Advisor:** Show statistic and index advisor.
- ▶ **Display Query Environment:** Displays environment settings used during the processing of this query.

8.3.3 Controlling the diagram level of detail

The Options menu provides you with flexibility to view the Visual Explain graph in your preferred way. As shown in Figure 8-17, you can select how much detail you want to see on the Visual Explain graphs. The menu options enable you to change the level of detail.

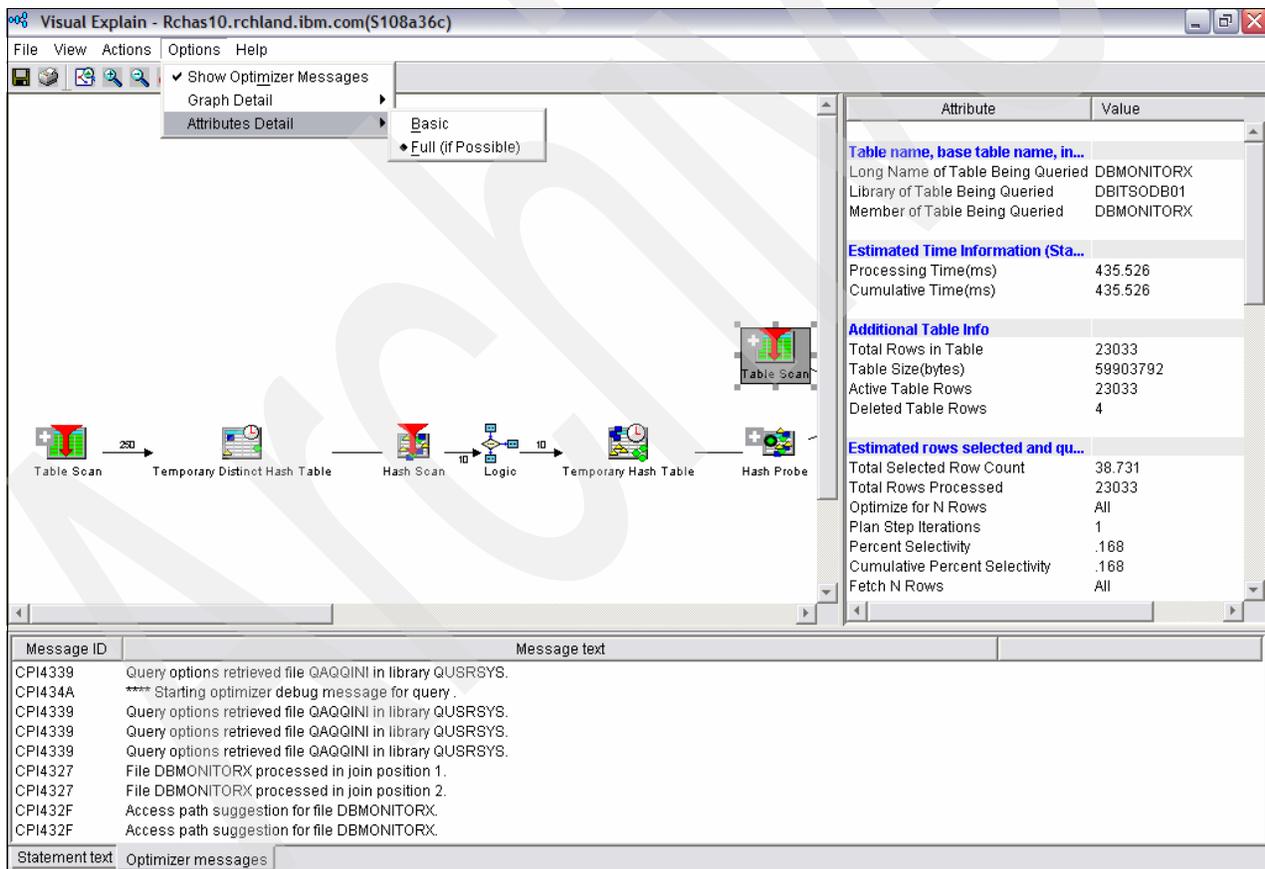


Figure 8-17 The Options menu

Click **Options** → **Graph Detail** → **Basic** to see only the icons that are directly related to the query. Or click **Options** → **Graph Detail** → **Full** to see also the icons that are indirectly related to the query, such as table scans performed to build temporary indexes. This option is usually used by the developers in the System i Lab in Rochester, MN. If you select Full, a

white cross is displayed next to some icons where you can right-click the icon and select **Expand** or **Expand to window** to view a subgraph of the query operation as shown in Figure 8-18.

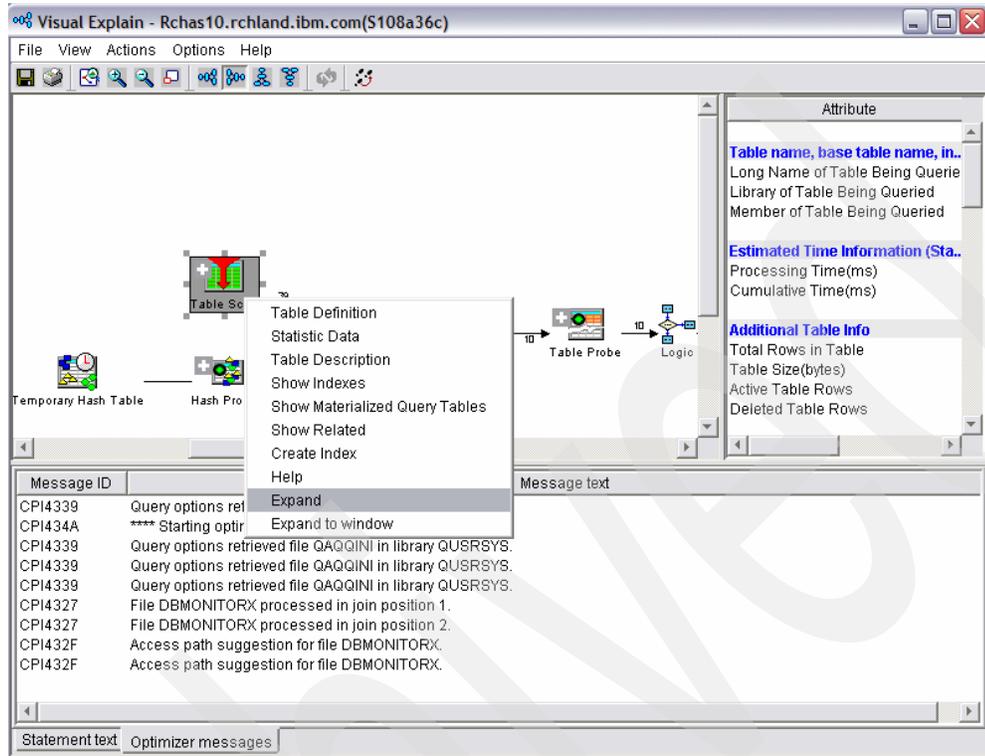


Figure 8-18 Controlling the diagram level of detail: *Full level (part 1 of 2)

management environment (Figure 8-20) where the query was executed. You can also obtain this information from the Display Query Environment selection in the Actions menu.

Attribute	Value
System Name	RCHAS10
Job Name	QZDASOINIT
Job User	QUSER
Job Number	046154
Memory Pool Size	18418278400
Date format	ISO
Date separator	-
Time format	ISO
Time separator	.
Decimal point	.

Figure 8-20 Environment Settings

8.3.5 Visual Explain query attributes and values

The query attributes and values show more information about the optimizer implementation of the query. If you select an icon from the Query Implementation graph, you obtain information about that icon, as well as that part of the query implementation.

We selected a few of the query implementation icons to show you the query attributes and values. This way you can see exactly how much information Visual Explain collects. Prior to Visual Explain, the information was often available, but never in one place.

Table name, base table name, index name

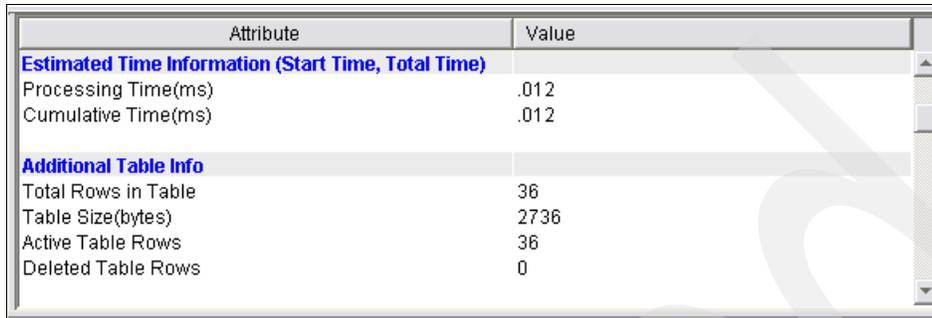
The section in Figure 8-21 shows the name and library or schema of the table being selected.

Attribute	Value
Table name, base table name, index name	
Long Name of Table Being Queried	REVENUESUMMARY
Library of Table Being Queried	DBITSODB06
Member of Table Being Queried	REVEN00001
Name of Table Being Queried	REVEN00001
Optimizer used Materialized Query Table	Yes

Figure 8-21 Table name

Estimated processing time and table info

The estimated processing time (shown in Figure 8-22) shows the time that the optimizer expects to take from this part of the query. The Additional Table Info section shows information about the rows and table size.

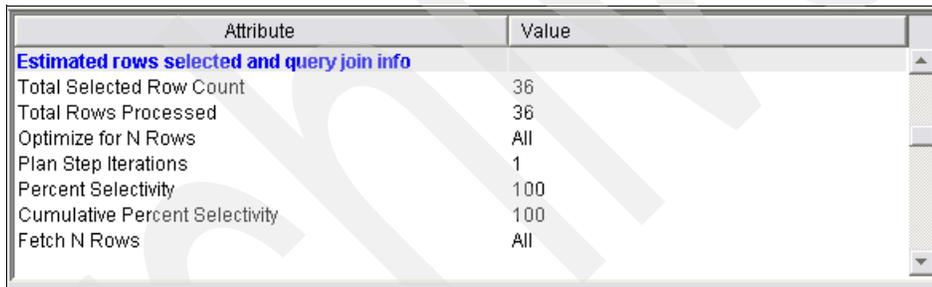


Attribute	Value
Estimated Time Information (Start Time, Total Time)	
Processing Time(ms)	.012
Cumulative Time(ms)	.012
Additional Table Info	
Total Rows in Table	36
Table Size(bytes)	2736
Active Table Rows	36
Deleted Table Rows	0

Figure 8-22 Estimated processing time and table information

Estimated rows selected and query join info

The estimated rows selected (shown in Figure 8-23) shows the number of rows that the optimizer expects to output from this part of the query. If the query is only explained, it shows an estimate of the number of rows. If it is run and explained, it shows the number of rows that are selected.



Attribute	Value
Estimated rows selected and query join info	
Total Selected Row Count	36
Total Rows Processed	36
Optimize for N Rows	All
Plan Step Iterations	1
Percent Selectivity	100
Cumulative Percent Selectivity	100
Fetch N Rows	All

Figure 8-23 Estimated rows selected and query join info

Estimated Cost Information About the Plan Performed

This section (as shown in Figure 8-24) indicates whether the query is CPU or I/O bound. Queries can be CPU-intensive or I/O-intensive. When a query's constraint resource is the CPU, it is called *CPU bound*. When a query's constraint resource is the I/O, it is called *I/O bound*. A query that is either CPU or I/O bound gives you the opportunity to review the query attributes being used when the query was processing. If a symmetric multiprocessor (SMP) is

installed on a multiprocessor system, you should review the DEGREE parameter to ensure that you are using the systems resources effectively.

Attribute	Value
Estimated Cost Information About the Plan Performed	
Processing Time(ms)	.012
I/O Or CPU Bound	CPU Bound
CPU Cost(ms)	.012
I/O Cost(ms)	0
I/O Count	0
PreLoad Relation	No
Memory Used(bytes)	2736
Share of Memory Available(bytes)	27966108
Memory Constrained	No
Cumulative Memory Constrained	No

Figure 8-24 Estimated Cost Information

Information about the index scan performed

This display provides essential information about the index that was used for the query, including the reason for using the index, how the index is being used, and static index attributes. It also specifies the access method or methods used such as Index Scan - Key positioning, Index Scan - Key Selection, and Index Only Access. To find a description about the different reason codes, refer to the manual DB2 Universal Database for iSeries Database Performance and Query Optimization for V5R4, which is available in the Information Center at:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/rzajq/rzajq.pdf>

SMP parallel information

The Symmetric Multiprocessing (SMP) information (shown in Figure 8-25) shows the degree of parallelism that occurred on this particular step. It might appear for more than one icon, because multiple steps can be processed with differing degrees of parallelism.

DB2 SMP Parallel Information	
Parallel Degree Used	1
Parallel Degree Requested	1
Max Capable Parallel Degree	2
Max Capable Cumulative Parallel Degree	2
Max Capable I/O Degree	1

Figure 8-25 SMP parallel information

Note:

- ▶ The SMP parallel information is relevant only when the DB2 SMP licensed feature is installed and that you have setup the QQRVDEGREE system value.
- ▶ For more information about SMP and QQRVDEGREE system value, refer to Chapter 11, “Environmental settings that affect SQL performance” on page 379 or you can also refer to DB2 Symmetric Multiprocessing for iSeries - Database Parallelism within i5/OS (SMP) at:

<http://www-03.ibm.com/servers/eserver/iseriess/db2/db2sym.html>

Index advised information

The Index advised section (Figure 8-26) tells you whether the query optimizer is advising the creation of a permanent index. If an index is being advised, the number and names of the columns to create the index are suggested. This is the same information that is returned by the CPI432F optimizer message. If the Highlight Index Advised option is set, advised index information, such as base table name, library, and involved columns, are easily identifiable. Subsequently, if you click at the Index Advisor Toolbar, you will be presented with Index and Statistic Advisor window as shown at the bottom left of Figure 8-26.

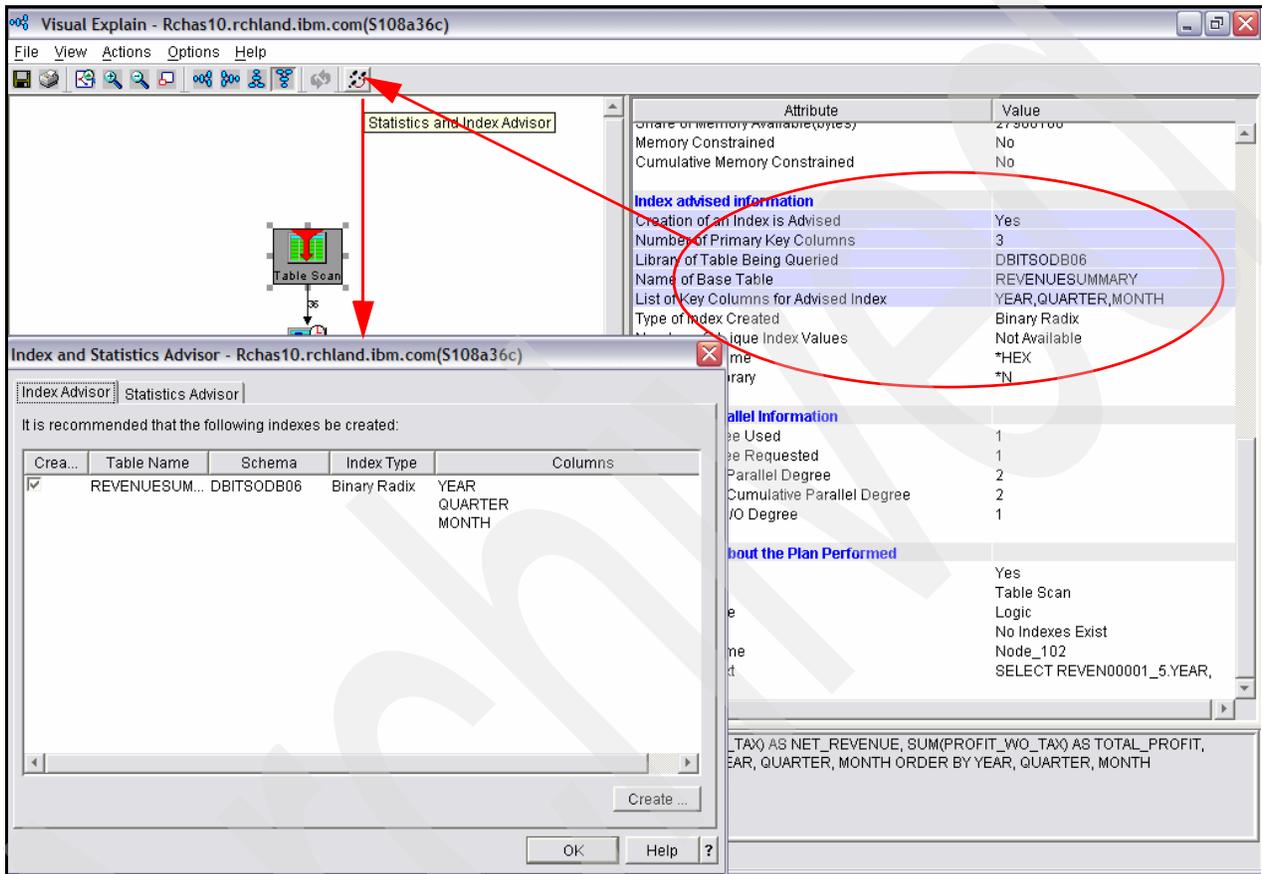


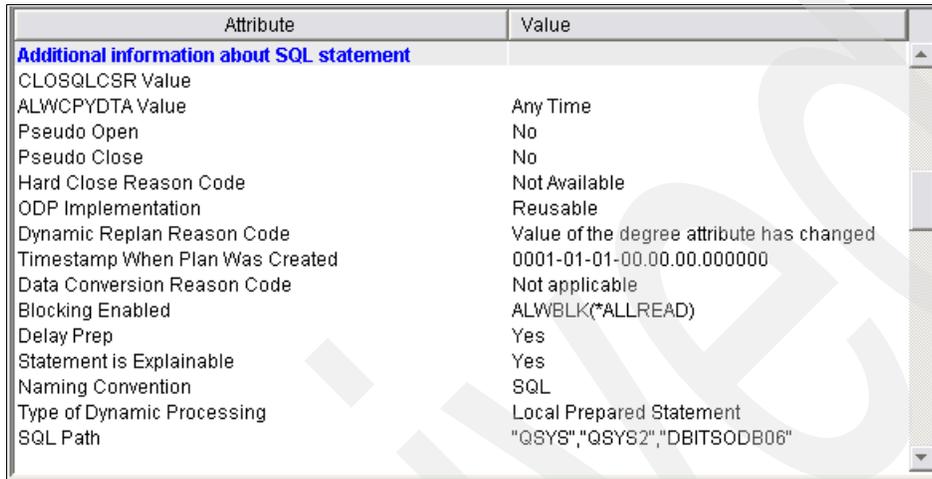
Figure 8-26 Index advised

It is possible for the query optimizer to not use the suggested index, if one is created. This suggestion is generated if the optimizer determines that a new index might improve the performance of the selected data by one microsecond.

Additional information about SQL statement

The display in Figure 8-27 shows information about the SQL environment that was used when the statement was captured. The SQL environment parameters can impact query performance. Many of these settings are taken from the Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC) driver settings.

Statement is Explainable specifies whether the SQL statement can be explained by the Visual Explain tool.



Attribute	Value
Additional information about SQL statement	
CLOSQLCSR Value	
ALWCPYDTA Value	Any Time
Pseudo Open	No
Pseudo Close	No
Hard Close Reason Code	Not Available
ODP Implementation	Reusable
Dynamic Replan Reason Code	Value of the degree attribute has changed
Timestamp When Plan Was Created	0001-01-01-00.00.00.000000
Data Conversion Reason Code	Not applicable
Blocking Enabled	ALWBK(*ALLREAD)
Delay Prep	Yes
Statement is Explainable	Yes
Naming Convention	SQL
Type of Dynamic Processing	Local Prepared Statement
SQL Path	"QSYS","QSYS2","DBITSODB06"

Figure 8-27 Additional information

8.4 Using Visual Explain with Database Monitor data

Performance Monitor data is query information that has been recorded by one of the DB2 Universal Database for iSeries Performance Monitors into a database table that can be analyzed later. Multiple Performance Monitors might run on the iSeries at the same time. They can either record information for individual jobs or for the entire system. Each one is individually named and controlled. Any given job can be monitored by a maximum of one system monitor and one job monitor.

You can start Performance Monitors in iSeries Navigator or by using a CL command. With iSeries Navigator, the SQL Performance Monitors component is used to collect Database Monitor data. If you want to use Visual Explain with the data collected with an SQL Performance Monitor, then you must choose the Detailed Monitor collection when setting up the SQL Performance Monitor in iSeries Navigator.

If you intend to use Visual Explain on the Database Monitor data collected with the CL commands, the data must be imported into iSeries Navigator as detailed data.

Using Visual Explain

In iSeries Navigator, click **Databases** and expand the database that you want to use. Click **SQL Performance Monitors** to obtain a list of the SQL Performance Monitors that are

currently on the system. Right-click the name that you want, and select **Show Statements** (Figure 8-28).

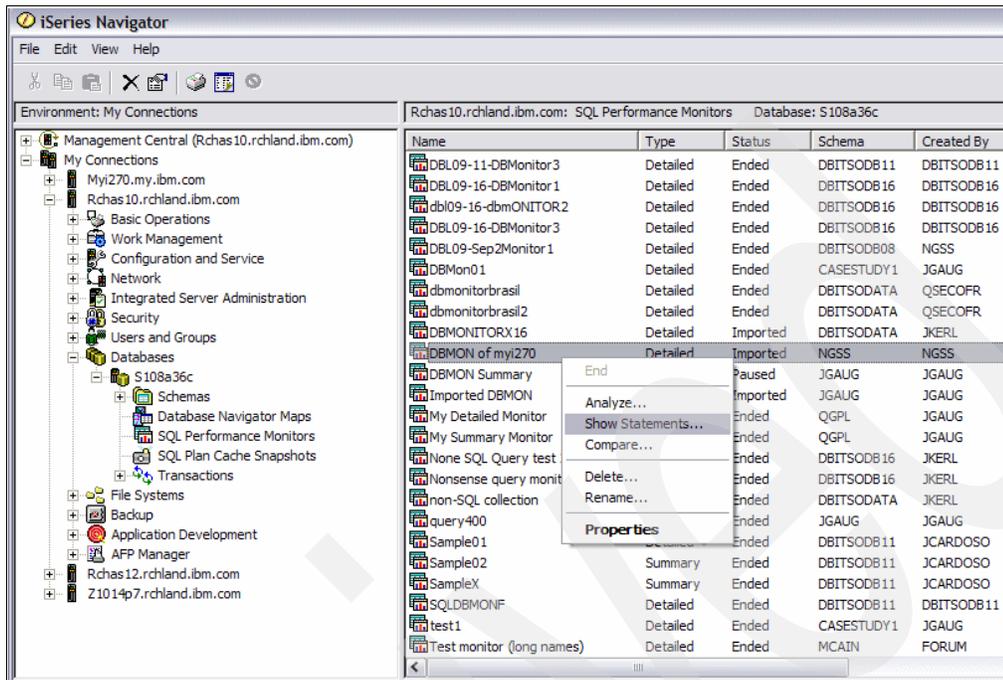


Figure 8-28 Show Statements from SQL Performance data

The Show Statement window with filtering options is shown in Figure 8-29. When you are satisfied with your filtering selection, click **Retrieve** to list the SQL statements. An *explainable*

statement is an SQL statement that can be explained by Visual Explain. Because Visual Explain does not process all SQL statements, some statements might not be selected.

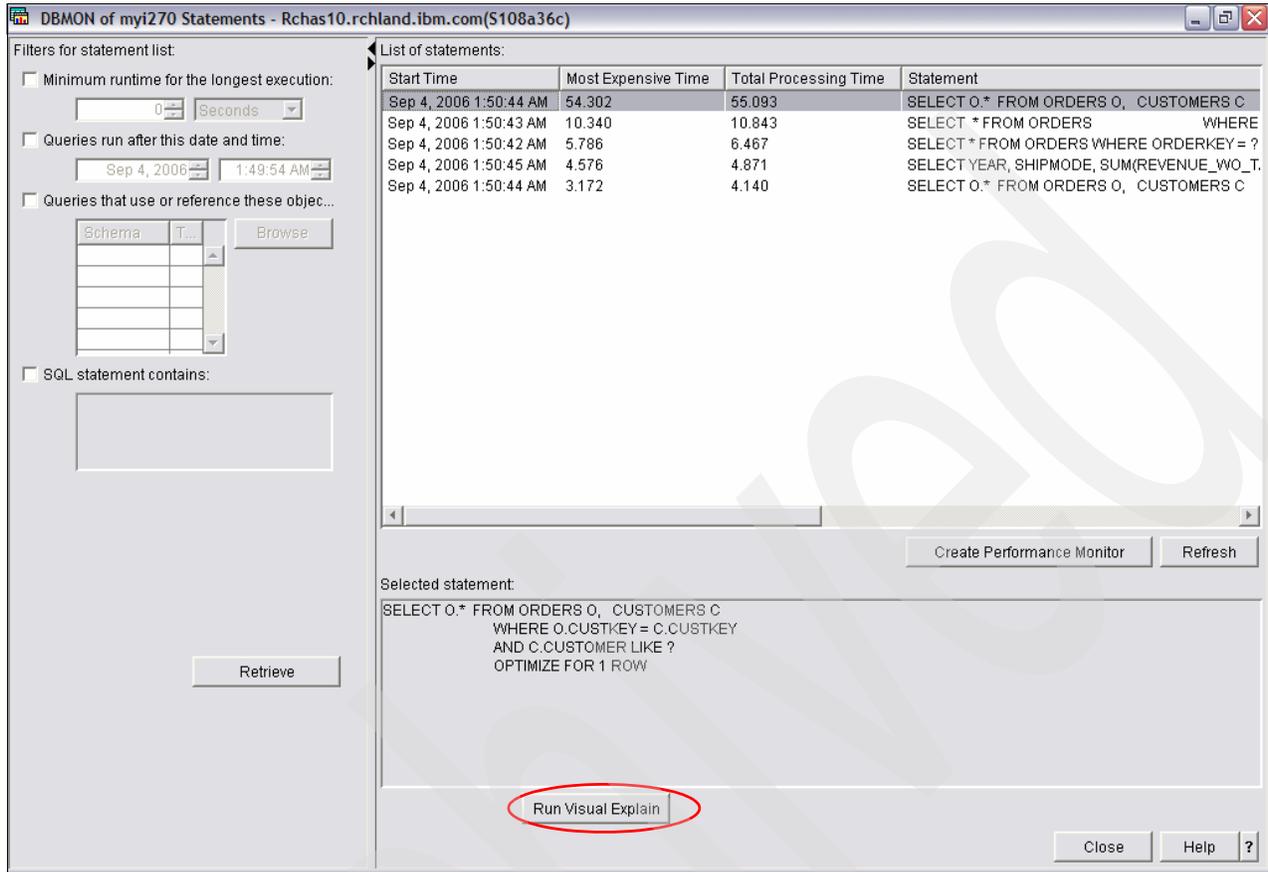


Figure 8-29 SQL statements in Show Statements window

Note: Query optimizer information is generated only for an SQL statement or query request when an open data path (ODP) is created. When an SQL or query request is implemented with a reusable ODP, then the query optimizer is not invoked. Therefore, there is no feedback from the query optimizer in terms of monitor data or debug messages. Also, the statement is not explainable in Visual Explain. The only technique for analyzing the implementation of a statement in reusable ODP mode is to look for an earlier execution of that statement when an ODP was created for that statement.

To use Visual Explain on any of the statements, select the statement from the display. The full SQL statement appears in the lower part of the display for verification. Click **Run Visual Explain** (Figure 8-29) to analyze the statement and prepare a graphical representation of the query.

Exit the Visual Explain window and the Explainable Statements window when you have completed your analysis. You might either retain the performance data or remove it from the system at this time, depending on your requirements.

8.5 Using Visual Explain with imported data

You can import Database Monitor data into Visual Explain and then use the tool to help with diagnosing problems further. Visual Explain can be used against current active jobs and against data collected in Performance Monitors either by iSeries Navigator or using the STRDBMON command.

To import a Database Monitor from another system or the same system, you can use iSeries Navigator and perform the following steps:

1. Select the system where the data is held. Click **Databases** → **your relational database**. Right-click **SQL Performance Monitors** and select **Import** as shown in Figure 8-30.

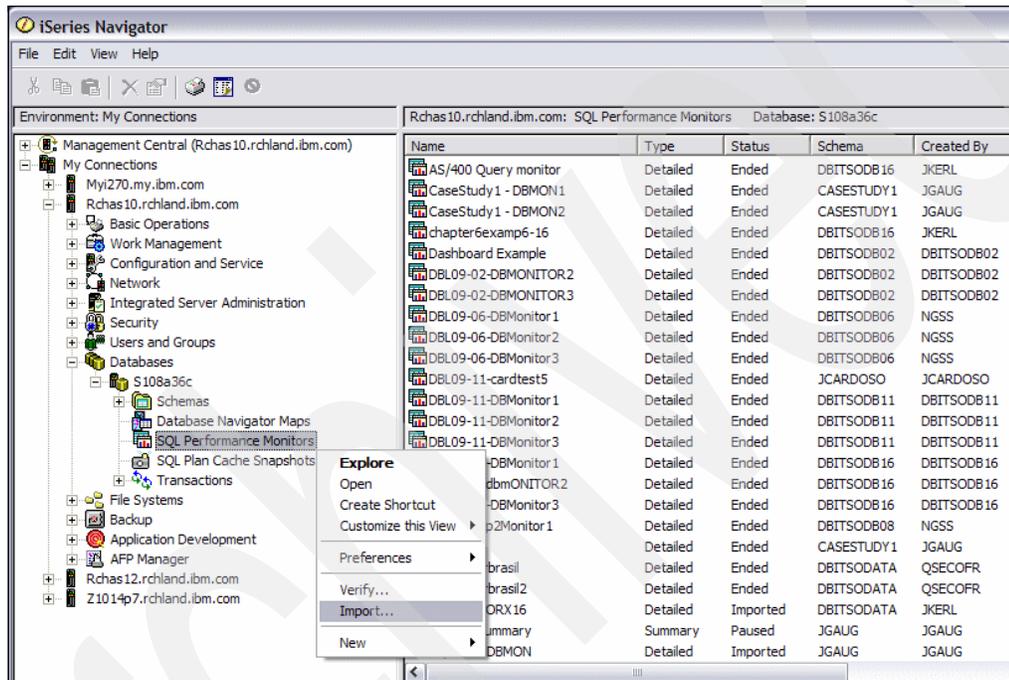


Figure 8-30 Selecting to import a Database Monitor

2. In the Import SQL Performance Monitor Files window (Figure 8-31), specify the name of the monitor and specify the file name and schema in which it resides. For Type of monitor, select either Summary or Detailed depending on how you collected the Database Monitor data.



Figure 8-31 Import SQL Performance Monitor Files window

3. After the monitor is imported, it is displayed in the right pane of the iSeries Navigator window as shown in Figure 8-32.

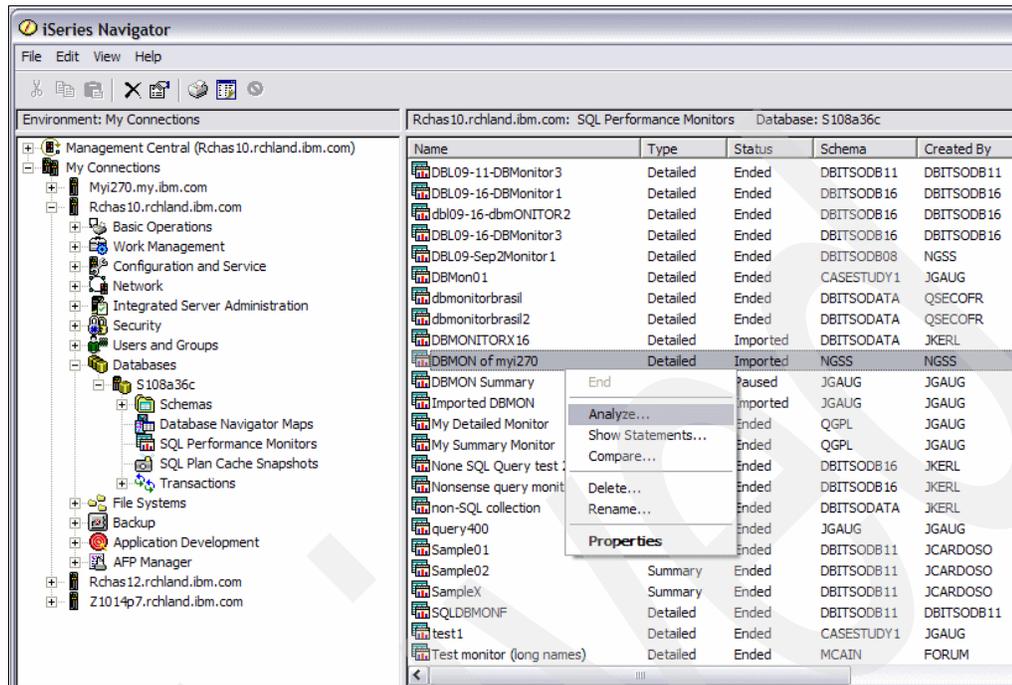


Figure 8-32 SQL Performance Monitors window

After you import the monitor, you can choose either Analyze result or Show Statements.

Note: The Show Statements in V5R4 is the equivalent of List Explainable Statements in V5R3.

8.5.1 Show Statements

When you see the SQL Performance Monitor of interest in the iSeries Navigator window, right-click and select **Show Statements**. A window opens with Filtering options:

- ▶ **Minimum runtime for the longest execution:** Select this to include queries that exceed a certain amount of time. Select a number and then a unit of time.
- ▶ **Queries run after this date and time:** Select this to include queries run at a specified date and time. Select a date and time.
- ▶ **Queries that use or reference these objects:** Select this to include queries that use or reference certain objects. Click **Browse** to select objects to include.
- ▶ **SQL statement contains:** Select this to include only those queries that contain a specific type of SQL statement. For example, specify SELECT if you only want to include queries that are using SELECT.

Once you have filled in the filter selection, click **Retrieve** to list the statements as shown in Figure 8-33. You can select an SQL statement by clicking the required row. You can sort on the columns to help you look for statements of interest, like **Most Expensive Time**. You can also click **Create Performance Monitor** button to create performance data specifically for the statements of your interest.

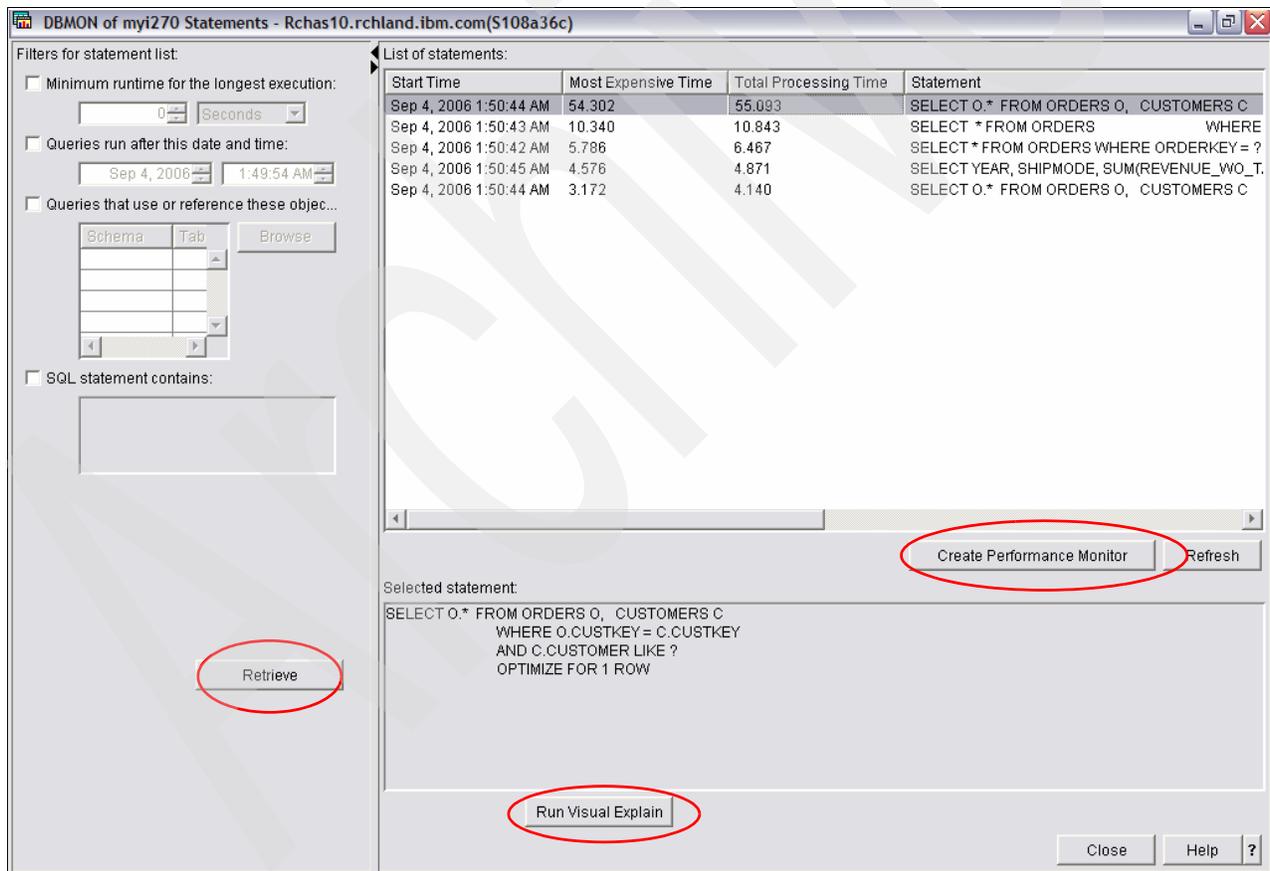


Figure 8-33 Show Statements sorted by Most Expensive Time

If you highlight a specific row, the involved SQL statement appears in the bottom half of the window. To obtain the visual explanation, click the **Run Visual Explain** button. After you select an SQL statement for analysis, you can use Visual Explain to understand the implementation the optimizer chose, the time it took to run, any indexes advised, the

environmental variables that were used, and the resources that were available at the time the job ran as shown in Figure 8-34.

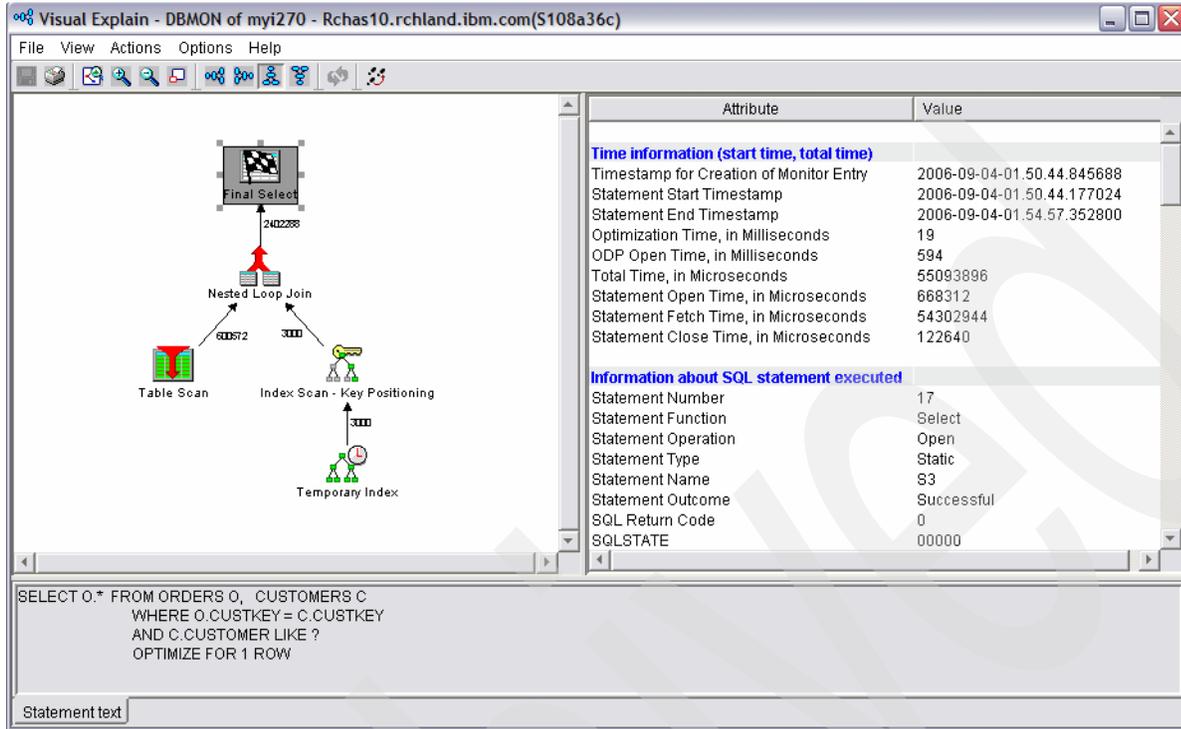


Figure 8-34 Final Select Visual Explain window

8.6 Using Visual Explain with SQE Plan Cache and Plan Cache Snapshot

SQE Plan Cache and Plan Cache Snapshot are newly introduced in V5R4. In this section, we discuss how you can integrate Visual Explain with both SQE Plan Cache and Plan Cache Snapshot to view the implementation of optimizer in graphical representation.

8.6.1 Using Visual Explain with SQE Plan Cache

To access Visual Explain from SQE Plan Cache, in iSeries Navigator, perform the following steps:

1. Click **Databases** and expand the database that you want to use.

SQL statements that fulfill your selection criteria. The related SQL statements appear in the Show Statements window as shown in Figure 8-36.

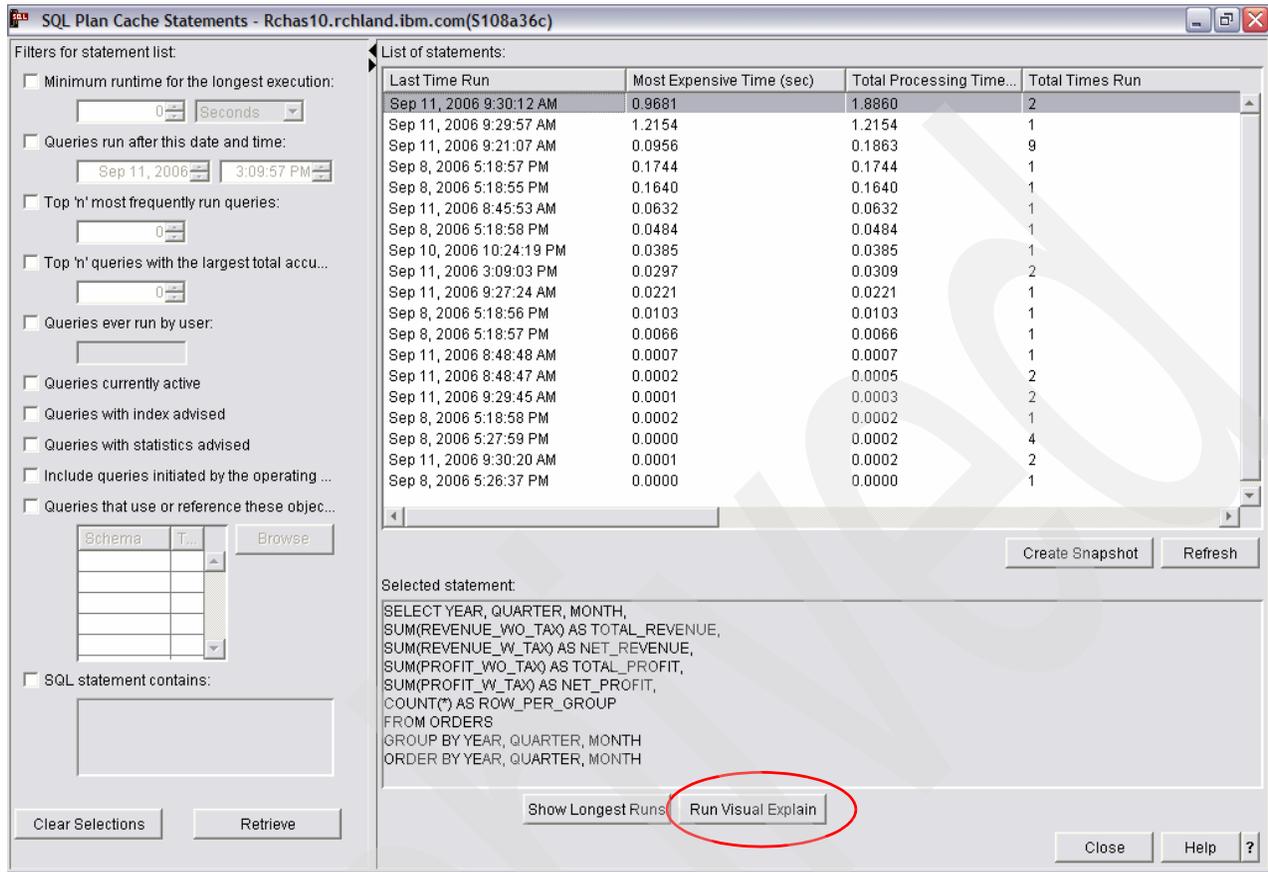


Figure 8-36 Plan Cache Show Statements window with filtering options

- The statements are sorted by Total Processing Time. Highlight the statement which you would like to analyze and click **Visual Explain** button. The Visual Explain window appears (as shown in Figure 8-37), allowing you to examine closely how the optimizer implements the query. You can look at the optimizer's recommendation such as index advised in

Figure 8-37, so that you can make the decision to implement the recommendation in order to enhance your query performance.

The screenshot shows the Visual Explain interface for an SQL query. The execution plan on the left consists of the following steps from bottom to top: Table Scan (600572), Temporary Distinct Hash Table, Hash Scan, Temporary Sorted List (360), Sorted List Scan, and Final Select. The right pane displays a table of attributes and values:

Attribute	Value
Index advised information	
Creation of an Index is Advised	Yes
Number of Primary Key Columns	3
Library of Table Being Queried	DBITSODB08
Name of Base Table	ORDERS
List of Key Columns for Advised Index	YEAR,QUARTER,MONTH
Type of Index Created	Binary Radix
Number of Unique Index Values	Not Available
ACS Table Name	*HEX
ACS Table Library	*N
DB2 SMP Parallel Information	
Parallel Degree Used	1
Parallel Degree Requested	1
Max Capable Parallel Degree	1
Max Capable Cumulative Parallel Degree	1
Max Capable I/O Degree	20
Information About the Plan Performed	
Scrollable	Yes
Plan Name	Table Scan
Plan Step Type	Logic
Reason Code	No Indexes Exist
Plan Step Name	Node_11
Statement Text	SELECT

The statement text at the bottom of the window is:

```
SELECT YEAR, QUARTER, MONTH,
SUM(REVENUE_WO_TAX) AS TOTAL_REVENUE,
SUM(REVENUE_W_TAX) AS NET_REVENUE,
SUM(PROFIT_WO_TAX) AS TOTAL_PROFIT,
SUM(PROFIT_W_TAX) AS NET_PROFIT,
COUNT(*) AS ROW_PER_GROUP
FROM ORDERS
GROUP BY YEAR, QUARTER, MONTH
ORDER BY YEAR, QUARTER, MONTH
```

Figure 8-37 Visual Explain for SQE Plan Cache

8.6.2 Using Visual Explain with SQE Plan Cache Snapshot

To access to Visual Explain from SQE Plan Cache Snapshot, perform the following steps:

1. In iSeries Navigator, click **Databases** and expand the database that you want to use. Expand **SQL Plan Cache Snapshots** to list all the snapshots you have created.

Right-click the SQL Plan Cache Snapshot of your interest and select **Show Statements** as shown in Figure 8-38.

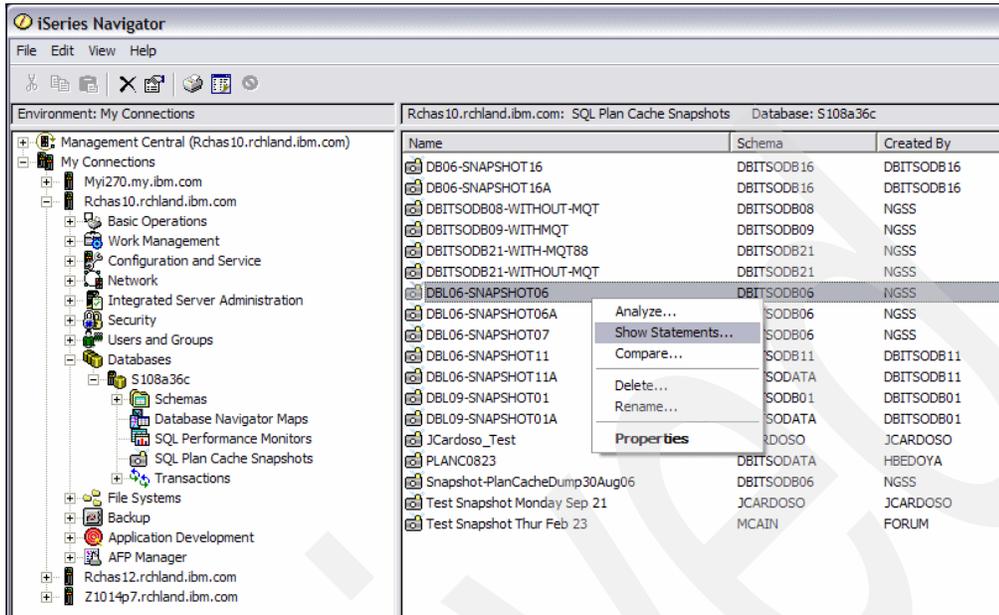


Figure 8-38 Show Statements on Plan Cache Snapshot

- The Show Statements window opens up with filtering options (Figure 8-39). Specify your filtering criteria and click the **Retrieve** button. SQL statements which fulfilled your selection criteria are listed and sorted by Total Processing Time as shown in Figure 8-39.

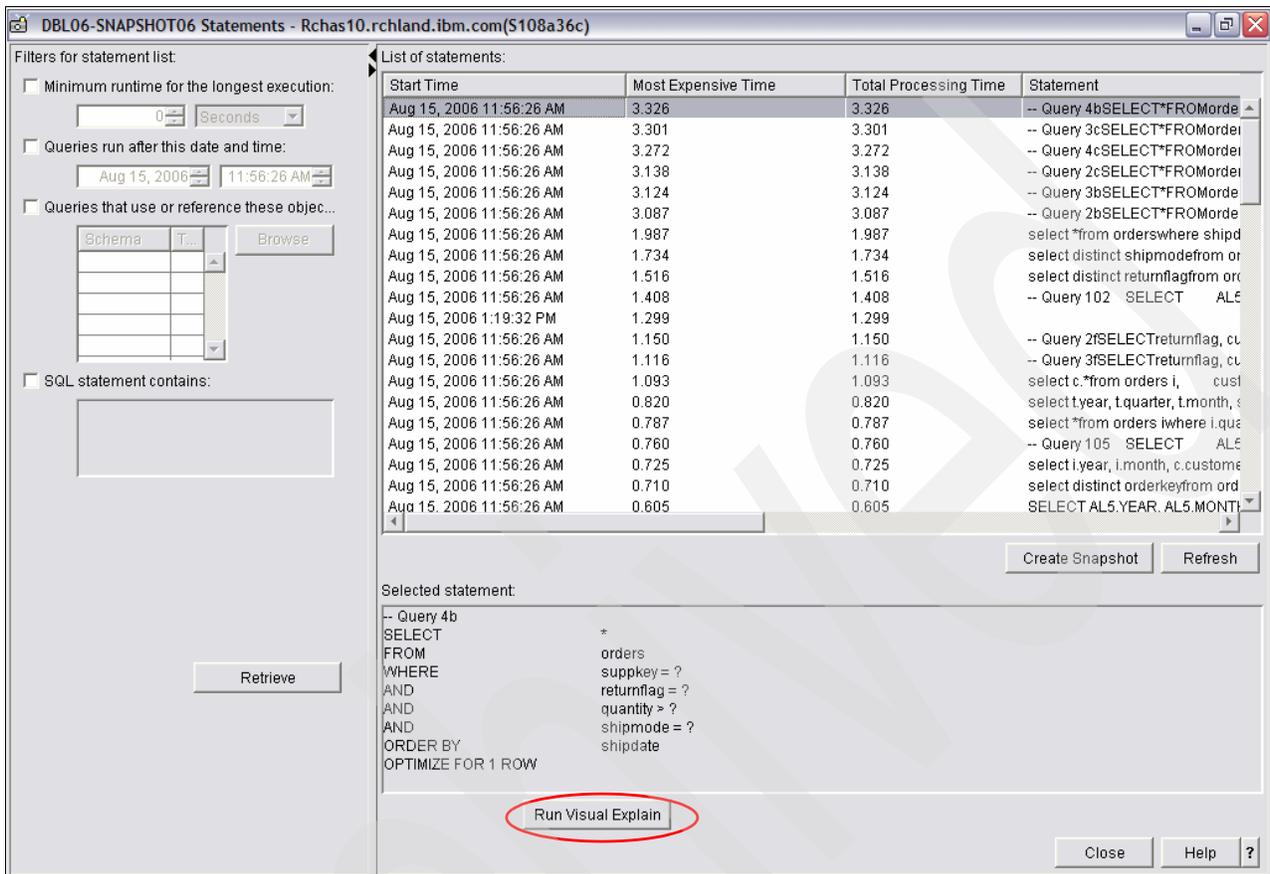


Figure 8-39 Plan Cache Snapshot Show Statements window with filtering options

- Highlight the SQL statement of your interest and click the **Run Visual Explain** button. This calls up the Visual Explain window presenting the graphical implementation of the optimization on the left pane, coupled with the related facts on the right pane as shown in Figure 8-40.

In addition to Show Statements, you can also select the Analyze function, which opens up the Analysis Overview Dashboard. From the Dashboard, you can highlight the items of your interest, click **Statements** to list out the SQL statements and access Visual Explain on the SQL statement of your interest.

Note: For more information about using the Dashboard to analyze the Plan Cache Snapshot, refer to Chapter 7, "SQE Plan Cache and SQE Plan Cache Snapshots" on page 237.

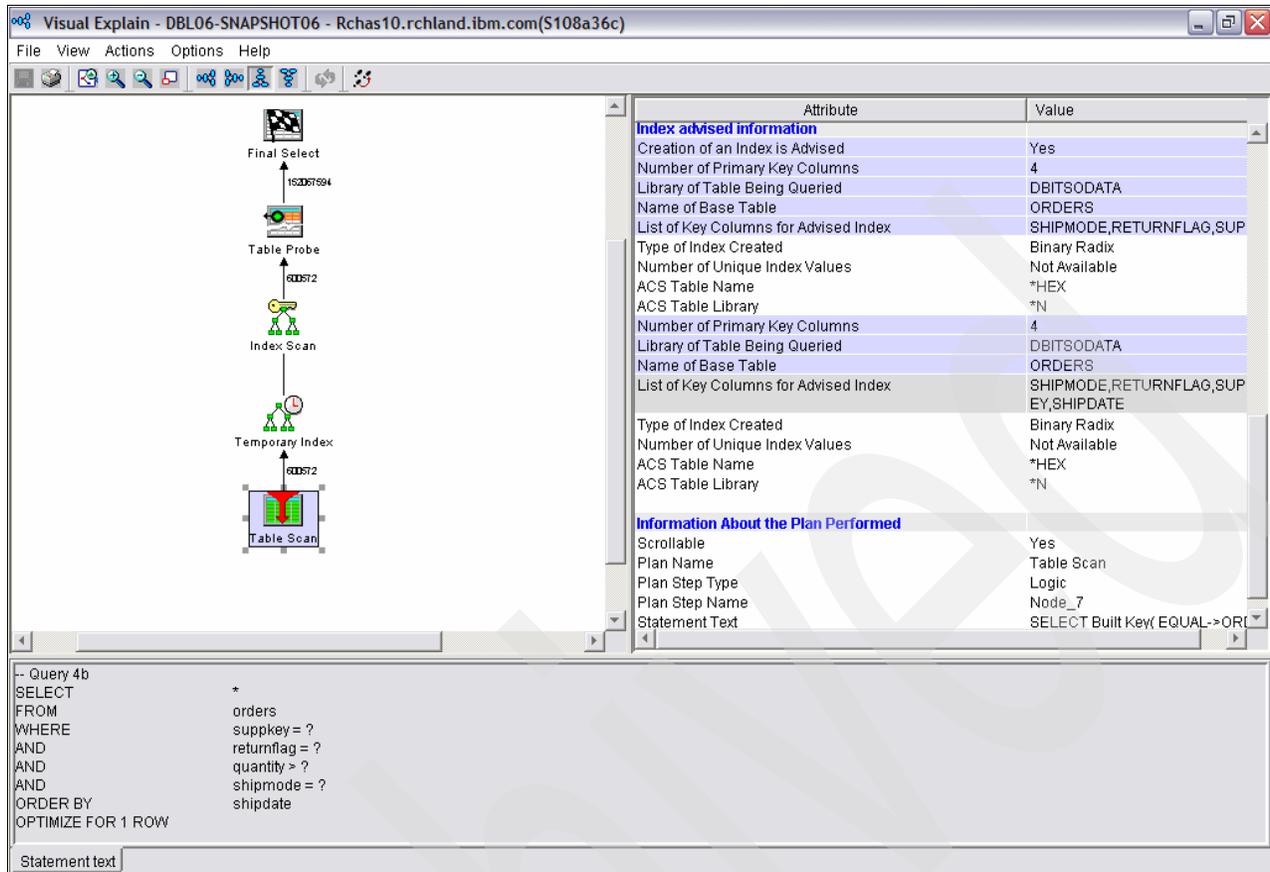


Figure 8-40 Visual Explain for SQE Plan Cache Snapshot

Note: For more information about SQE Plan Cache and SQE Plan Cache Snapshot, refer to Chapter 7, “SQE Plan Cache and SQE Plan Cache Snapshots” on page 237.

8.7 Non-SQL interface considerations

Obviously, the SQL Performance Monitor can capture implementation information for any SQL-based interface. Therefore, any SQL-based request can be analyzed with Visual Explain. SQL-based interfaces range from embedded SQL to Query Manager reports to ODBC and JDBC.

The query optimizer creates an access plan for all queries that run on the iSeries server. Most queries use the SQL interface. They generate an SQL access plan, either directly (SQL Script window, STRSQL command, SQL in high-level language (HLL) programs) or indirectly (Query/400).

Other queries do not generate identifiable SQL statements (Query/400, OPNQRYF command) and cannot be used with Visual Explain via the SQL Performance Monitor. In this instance, the name SQL, as part of the SQL Performance Monitor, is significant.

A request from any DB2 for i5/OS SQL interface can be analyzed with Visual Explain.

The statements that do not generate SQL and, therefore, that cannot be used with Visual Explain via the SQL Performance Monitor include:

- ▶ Native database access from a high level language, for example, Cobol, RPG, and so on
- ▶ Query/400 or Query for iSeries
- ▶ Open Query File (OPNQRYF) command
- ▶ OS/400 Create Query API (QQQRY)

Note: The Memory Resident or Summary Database Monitor is not supported by Visual Explain.

8.8 The Visual Explain icons

Table 8-1 lists the icons that you might find on the Visual Explain query implementation chart.

Note that you can obtain this information by right-clicking on the icons from the Visual Explain graph and choosing the help text.

Table 8-1 Visual Explain icons

	The <i>Final Select</i> icon displays the original text and summary information about how the query was implemented.
	The <i>Table</i> icon indicates that a table was accessed. See the Select icon for more details.
	The <i>Table Scan</i> icon indicates that all rows in the table were paged in, and selection criteria was applied against each row. Only those rows that meet the selection criteria were retrieved. To obtain the result in a particular sequence, you must specify the ORDER BY clause.
	The <i>Table Probe</i> icon indicates that data from the table must be processed and selected for this query. The table is probed using a key derived from the ordinal number or relative record number associated with each row in the table. The ordinal number is used to calculate the pages of data that need to be retrieved and brought into main memory to satisfy this probe request. The ordinal number used for the probe operation was provided by another data access method previously processed for this query.
	The <i>Index</i> icon indicates that an index object was used to process this query.
	The <i>Index Scan</i> icon indicates that the entire index will be scanned, which causes all of the entries in the index that are to be paged into main memory to be processed. Any selection criteria whose predicates match the key columns of the index can then be applied against the index entries. Only those key entries that match the specified key selection are used to select rows from the corresponding table data.
	The <i>Index Probe</i> icon indicates that the selection criteria whose predicates matched the leading key columns of the index were used to probe directly into the index. The probe selection minimizes the number of key entries that must be processed and paged into main memory. Additional key selection can be applied against the non-leading key columns of the index to further reduce the number of selected key entries. Only key entries that match the specified probe and key selection are used to select rows from the corresponding table data.
	The <i>Parallel Table Scan</i> icon indicates that a table scan access method was used and multiple tasks were used to fill the rows in parallel. The table was partitioned, and each task was given a portion of the table to use.



The *Skip Sequential Table Scan* icon indicates that a bitmap was used to determine which rows were selected. No CPU processing was done on non-selected rows, and I/O was minimized by bringing in only those pages that contained rows to be selected. This icon is usually related to the Dynamic Bitmap or Bitmap Merge icons.



The *Skip Sequential Parallel Scan* icon indicates that a skip sequential table scan access method was used and multiple tasks were used to fill the rows in parallel. The table was partitioned, and each task was given a portion of the table to use.



The *Derived Column Selection* icon indicates that a column in the row selected had to be mapped or derived before selection criteria could be applied against the row. Derived column selection is the slowest selection method.



The *Parallel Derived Column Selection* icon indicates that derived field selection was performed, and the processing was accomplished using multiple tasks. The table was partitioned, and each task was given a portion of the table to use.



The *Index Key Positioning* icon indicates that only entries of the index that match a specified range of key values were “paged in”. The range of key values was determined by the selection criteria whose predicates matched the key columns of the index. Only selected key entries were used to select rows from the corresponding table data.



The *Parallel Index Key Positioning* icon indicates that multiple tasks were used to perform the key positioning in parallel. The range of key values was determined by the selection criteria, whose predicates matched the key columns of the index. Only selected key entries were used to select rows from the corresponding table data.



The *Index Key Selection* icon indicates that all entries of the index were paged in. Any selection criteria whose predicates match the key columns of the index was applied against the index entries. Only selected key entries were used to select rows from the table data.



The *Parallel Index Key Selection* icon indicates that multiple tasks were used to perform key selection in parallel. The table was partitioned, and each task was given a portion of the table to use.



The *Encoded-vector Index* icon indicates that access was provided to a database file by assigning codes to distinct key values, and then representing these values in an array (vector). Because of their compact size and relative simplicity, encoded-vector indexes (EVIs) provide for faster scans.



The *Parallel Encoded-vector Index* icon indicates that multiple tasks were used to perform the EVI selection in parallel. This allows for faster scans that can be more easily processed in parallel.



The *Encoded-vector Index Scan* icon indicates that the entire EVI will be scanned causing all of the distinct values represented in the index to be processed. Any selection criteria, whose predicates match the key columns of the EVI can then be applied against the distinct values represented in the index. Only those distinct values that match the specified key selection are then used to process the vector and generate either a temporary row number list or temporary row number bitmap.



The *Encoded-vector Index Probe* icon indicates that the selection criteria whose predicates matched the leading key columns of the EVI were used to probe directly into the distinct values represented in the index. Only those distinct values that match the specified probe selection are then used to process the vector and generate either a temporary row number list or temporary row number bitmap.



The *Sort Sequence* icon indicates that selected rows were sorted using a sort algorithm.



The *Grouping* icon indicates that selected rows were grouped or summarized. Therefore, duplicate rows within a group were eliminated.



The *Nested Loop Join* icon indicates that queried tables were joined together using a nested loop join implementation. Values from the primary file were joined to the secondary file by using an index whose key columns matched the specified join columns. This icon is usually after the method icons used on the underlying tables (that is, index scan-key selection and index scan-key positioning).



The *Hash Join* icon indicates that a temporary hash table was created. The tables queried were joined together using a hash join implementation where a hash table was created for each secondary table. Therefore, matching values were hashed to the same hash table entry.



The *Temporary Index* icon indicates that a temporary index was created, because the query either requires an index and one does not exist, or the creation of an index will improve performance of the query.



The *Temporary Hash Table* icon indicates that a temporary hash table was created to perform hash processing.



The *Temporary Table* icon indicates that a temporary table was required to either contain the intermediate results of the query, or the queried table could not be queried as it currently exists and a temporary table was created to replace it.



The *Dynamic Bitmap* icon indicates that a bitmap was dynamically generated from an existing index. It was then used to determine which rows were to be retrieved from the table. To improve performance, dynamic bitmaps can be used in conjunction with a table scan access method for skip sequential or with either the index key position or key selection.



The *Bitmap Merge* icon indicates that multiple bitmaps were merged or combined to form a final bitmap. The merging of the bitmaps simulates boolean logic (AND/OR selection).



The *DISTINCT* icon indicates that duplicate rows in the result were prevented. You can specify that you do not want any duplicates by using the `DISTINCT` keyword, followed by the selected column names.



The *UNION Merge* icon indicates that the results of multiple subselects were merged or combined into a single result.



The *Subquery Merge* icon indicates that the nested `SELECT` was processed for each row (`WHERE` clause) or group of rows (`HAVING` clause) selected in the outer level `SELECT`. This is also referred to as a *correlated subquery*.



The *Hash Table Scan* icon indicates that the entire temporary hash table will be scanned and all of the entries contained with the hash table will be processed. A hash table scan is generally considered when optimizer is considering a plan that requires the data values to be collated together but the sequence of the data is not required. The use of a hash table scan allows the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary hash table.



The *Hash Table Probe* icon indicates that the selection criteria that match the key columns used to create the temporary hash table will be probed to find all of the matching values stored within the hash table. A hash table probe is generally considered when determining the implementation for a secondary table of a join. The use of a hash table probe allows the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary hash table. An additional benefit of using a hash table probe is that the data structure of the temporary hash table usually causes the table data to remain resident within main memory after creation, reducing paging on the subsequent probe operation.



The *Temporary Distinct Hash Table* icon indicates that a temporary distinct hash table was created in order to perform hash processing. A distinct hash table is a data structure that is identical to the temporary hash table, except all duplicate data is compressed out of the temporary being created. The resulting hash table can then be used to perform distinct or aggregate operations for the query.



The *Distinct Hash Table Scan* icon indicates that the entire temporary distinct hash table will be scanned and all of the entries contained with the hash table will be processed. A distinct hash table scan is generally considered when optimizer is considering a plan that requires the data values to be collated together and all duplicate removed but the sequence of the data is not required. The use of a distinct hash table scan allows the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary distinct hash table. An additional benefit of using a distinct hash table scan is that the data structure of the temporary distinct hash table usually causes the table data within the distinct hash table to remain resident within main memory after creation. This benefit reduces the paging on the subsequent scan operations.



The *Distinct Hash Table Probe* icon indicates that the selection criteria that match the key columns used to create the temporary distinct hash table will be probed to find all of the matching values stored within the hash table. The use of a distinct hash table probe allows the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary distinct hash table. An additional benefit of using a distinct hash table probe is that the data structure of the temporary distinct hash table usually causes the table data to remain resident within main memory after creation. This benefit reduces the paging on the subsequent probe operation.



The *Temporary Sorted List* icon indicates that a temporary sorted list was created in order to perform a sequencing operation. A sorted list is a data structure where the table data is collated and sorted based upon the value of a column or columns referred to as the sort key. The sorted list can then be used to return the data in a specified sequence or to perform probe operations using the sort key to quickly retrieve all of the table data that matches a particular sort key.



The *Sorted List Scan* icon indicates that the entire temporary sorted list will be scanned and all of the entries contained with the sorted list will be processed. A sorted list scan is generally considered when optimizer is considering a plan that requires the data values to be sequenced based upon the sort key of the sorted list. The use of a sorted list scan allows the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary sorted list. An additional benefit of using a sorted list scan is that the data structure of the temporary sorted list usually causes the table data within the sorted list to remain resident within main memory after creation. This benefit reduces the paging on the subsequent scan operations.



The *Sorted List Probe* icon indicates that the selection criteria that match the key columns used to create the temporary sorted list is probed to find all of the matching values stored within the sorted list. A sorted list probe is generally considered when determining the implementation for a secondary table of a join when either the join condition uses an operator other than equal or a temporary hash table is not allowed in this query environment. The use of a sorted list probe allows the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary sorted list. An additional benefit of using a sorted list probe is that the data structure of the temporary sorted list usually causes the table data to remain resident within main memory after creation. This benefit reduces the paging on the subsequent probe operation.



The *Temporary List* icon indicates that a temporary list was created. The temporary list was required to either contain the intermediate results of the query, or the queried table could not be queried as it currently exists and a temporary list was created to replace it. The list is an unsorted data structure with no key. The data contained within the list can only be retrieved by a scan operation.



The *List Scan* icon indicates that the entire temporary list will be scanned and all of the entries will be processed.



The *Temporary Row Number List* icon indicates that a temporary row number list was created in order to help process any selection criteria. A row number list is a data structure used to represent the selected rows from a table that matches any specified selection criteria. Since the selected rows are represented by a sorted list of row numbers, multiple lists can be merged and combined to allow for complex selection and processing to be performed without having any paging occur against the table itself.



The *Row Number List Scan* icon indicates that the entire row number list will be scanned and all of the entries will be processed. Scanning a row number list can provide large amounts of savings for the table data associated with the temporary row number list. Since the data structure of the temporary row number list guarantees that the row numbers are sorted, it closely mirrors the row number layout of the table data, ensuring that the paging on the table will never revisit the same page of data twice.



The *Row Number List Probe* icon indicates that a row number list was used to verify that a row from a previous operation in the query matches the selection criteria used to create the temporary row number list. The use of a row number list probe allows the optimizer to generate a plan that can process the rows in the table in any manner regardless of any specified selection criteria. As the rows are processed, the ordinal number from the row is used to probe into the row number list to determine if that row matches the selection criteria. This is generally found when an index is used to satisfy the *ORDER BY* from a query and a separate viable index exists to process the selection criteria.



The *Bitmap Scan* icon indicates that the entire bitmap will be scanned and all of the entries that represent selected rows will be processed. Scanning a bitmap can provide large amounts of savings for the table data associated with the temporary bitmap. Since the data structure of the temporary bitmap mirrors the row number layout of the table data, the bitmap can be used to efficiently schedule paging of the table for all selected rows.



The *Bitmap Probe* icon indicates that a bitmap was used to verify that a row from a previous operation in the query matches the selection criteria used to create the temporary bitmap. The use of a bitmap probe allows the optimizer to generate a plan that can process the rows in the table in any manner regardless of any specified selection criteria. As the rows are processed, the ordinal number from the row is used to probe into the bitmap to determine if that row matches the selection criteria. This is generally found when an index is used to satisfy the *ORDER BY* from a query and a separate viable index exists to process the selection criteria.



The *Index Scan* icon indicates that the entire temporary index will be scanned causing all of the entries in the index to be paged into main memory to be processed. Any selection criteria whose predicates match the key columns of the index can then be applied against the index entries. Only those key entries that match the specified key selection are used to select rows from the corresponding table data.



The *Index Probe* icon indicates that the selection criteria whose predicates matched the leading key columns of the index were used to probe directly into the temporary index. The probe selection minimizes the number of key entries that must be processed and paged into main memory. Additional key selection can be applied against the non-leading key columns of the temporary index to further reduce the number of selected key entries. Only those key entries that matched the specified probe and key selection are used to select rows from the corresponding table data.



The *Temporary Correlated Hash Table* icon indicates that a temporary correlated hash table was created in order to perform hash processing. A hash table is a data structure where the table data is collated based upon the value of a column or columns referred to as the *hash key*. The hash table can then be used to perform probe operation using the hash key to quickly retrieve all of the table data that matches a particular hash value. Because this is a correlated hash table, the hash table needs to be rebuilt prior to any scan or probe operations being performed.



The *Correlated Hash Table Scan* icon indicates that the entire temporary hash table will be scanned and all of the entries contained with the hash table will be processed. A correlated hash table scan is generally considered when optimizer is considering a plan that requires the data values to be collated together but the sequence of the data is not required. In addition, the some of the values used to create the correlated hash table can change from one scan to another. The use of a correlated hash table scan allows the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary correlated hash table. An additional benefit of using a correlated hash table scan is that the data structure of the temporary correlated hash table usually causes the table data within the hash table to remain resident within main memory after creation. This benefit reduces the paging on the subsequent scan operations.



The *Correlated Hash Table Probe* icon indicates that the selection criteria that match the key columns used to create the temporary correlated hash table will be probed to find all of the matching values stored within the hash table. A correlated hash table probe is generally considered when determining the implementation for a secondary table of a join. The use of a hash table probe allows the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary correlated hash table. An additional benefit of using a correlated hash table probe is that the data structure of the temporary correlated hash table usually causes the table data to remain resident within main memory after creation. This benefit reduces paging on the subsequent probe operation.



The *Temporary Correlated List* icon indicates that a temporary correlated list was created. The temporary correlated list was required to either contain the intermediate results of the query, or the queried table could not be queried as it currently exists and a temporary correlated list was created to replace it. The list is an unsorted data structure with no key that must be rebuilt prior to any scan operation being performed.



The *Correlated List Scan* icon indicates that the entire temporary list will be scanned and all of the entries will be processed.



The *Temporary Buffer* icon indicates that a temporary buffer was created to store the intermediate rows of an operation. The temporary buffer is generally considered at a serialization point within a query to help facilitate operations such as parallelism. The buffer is an unsorted data structure, but it differs from other temporary data structures in that the buffer does not have to be fully populated in order allow its results to be processed.



The *Buffer Scan* icon indicates that the entire temporary buffer will be scanned and all of the entries will be processed.



The *Table Random Pre-Fetch* icon indicates that the pages required for the table probe operation will be requested synchronously in the background prior to the actual table probe operation being performed. The system attempts to manage the paging for the table probe to maintain that all of the pages of data necessary to perform the table probe operation stay resident within main memory until they are processed. The amount of pre-fetch paging that is performed by this data access method is dynamically controlled by the system based upon memory consumption and the rate at which rows continue to be processed.



The *Table Clustered Pre-Fetch* icon indicates that the pages required for the table probe operation will be requested synchronously in the background prior to the actual table probe operation being performed. The system attempts to manage the paging for the table probe to maintain that all of the pages of data necessary to perform the table probe operation stay resident within main memory until they are processed. The amount of pre-fetch paging that is performed by this data access method is dynamically controlled by the system based upon memory consumption and the rate at which rows continue to be processed.



The *Index Random Pre-Fetch* icon indicates that the pages required for the index probe operation will be requested synchronously in the background prior to the actual index probe operation being performed. The system attempts to manage the paging for the index probe to maintain that all of the pages of data necessary to perform the index probe operation stay resident within main memory until they are processed. The amount of pre-fetch paging that is performed by this data access method is dynamically controlled by the system based upon memory consumption and the rate at which rows continue to be processed.



The *Logic* icon indicates that the query needed to perform an operation or test against the data in order to generate the selected rows.



The *Fetch N Rows* icon indicates that a limit was placed upon the number of selected rows. The fetch n rows access method can either be used to implement a user specified limit on the selected rows or it can be combined with other access methods by the optimizer to satisfy complex implementation plans.



The *Lock Row for Update* icon indicates that an update lock was acquired for the associated table data in order to perform an update or delete operation. To minimize contention between queries, the optimizer attempts to place the lock row for update operation such that the lock is not acquired and held for a long duration.



The *User-defined table function* icon indicates that a user-defined function that returns a table was used. A table function can be referenced in an SQL FROM clause in the same way that a table or view can be referenced.



The *Select* icon indicates a point in the query where multiple results are brought together into a single result set. For example, if a query is the union of two different select statements, at the point before the union occurs, the Select icon indicates the points where the select statements finished and the union is about to occur. This icon also represents the default information for an operation that is unknown or not defined elsewhere with Visual Explain. It can help to represent tables or insert, update and delete operations for a query. The summary information for this icon contains any available information to help describe the operation being performed and what the icon represents.



The *Incomplete Information* icon indicates that a query could not be displayed due to incomplete information.

Archived



Index Advisor

In this chapter we introduce the new Index Advisor tool which offers you three different levels of information access, that is, *Database*, *Schema* and *Table*. We further describe the Index Advisor repository table and the Index Advisor tool. We also discuss the various possible interfaces that give you access to the Index Advisor information. This new tool coupled with the index advised information will help you in optimizing your queries.

9.1 What is the Index Advisor

The Database component of iSeries Navigator is the major driver of DB2 UDB simplification in V5R4 with the addition of new tools and streamlining of existing tools. This new combination of tools is known as the *DB2 OnDemand Performance Center*.

Index Advisor is one of the new tools in V5R4. Index advice from the DB2 UDB query optimizer is not a new capability in V5R4. Feedback on suggested indexes in past releases could be found in optimizer debug messages within a joblog or a database monitor collection. However, this feedback was only available if someone manually started a database monitor or turned on debug messaging. Even if this was done, an analyst had to deal with the complexities of extracting the advised index details of various joblogs and database monitor files. The new Index Advisor eliminates these hurdles with the click of a mouse. No user intervention such as starting a database monitor is needed since the query optimizer automatically logs index advice for all queries to a repository in V5R4. Right-click your Database name in the iSeries Navigator tree and select the "Index Advisor" task and you are given the output displayed here.

A further enhancement in V5R4 is that the index advice provided by the SQE query optimizer is more intelligent and complete. The index advice given in past releases only focused on filtering criteria of a query and ignored the join, grouping and ordering criteria. Again, this design required manual intervention from an analyst to take the DB2 index advice and then review the associated query to supplement the DB2 index advice. Look closely at the Index Advisor output and you will see that the optimizer also provides advice on the type of index to create. Some queries may benefit from a traditional radix index while others would benefit from an encoded vector index structure.

9.2 System Wide Index Advised Table

In this section we will discuss a place to find index advised information. The table *QSYS2.SYSIXADV* (i5/OS format *QSYS2/QYQIXADV*) provides summary information about the table and schema on which the indexes are advised.

The Index Advisor Repository is QSYS2/SYSIXADV. The details of this system table are shown in Table 9-1.

Table 9-1 *SYSIXADV* system table

Column name	System column name	Data type	Description
TABLE_NAME	TBNAME	VARCHAR(258)	Table over which an index is advised
TABLE_SCHEMA	DBNAME	CHAR(10)	Schema containing the table
SYSTEM_TABLE_NAME	SYS_TNAME	CHAR(10)	System table name on which the index is advised
PARTITION_NAME	TBMEMBER	CHAR(10)	Partition detail for the index
KEY_COLUMNS_ADVISED	KEYSADV	VARCHAR(16000)	Column names for the advised index
LEADING_COLUMN_KEYS	LEADKEYS	VARCHAR(16000)	Leading, Order Independent keys. the keys at the beginning of the KEY_COLUMNS_ADVISED field which could be reordered and still satisfy the index being advised.

Column name	System column name	Data type	Description
INDEX_TYPE	INDEX_TYPE	CHAR(14)	Radix (default) or EVI
LAST_ADVISED	LASTADV	TIMESTAMP	Last time this row was updated
TIMES_ADVISED	TIMESADV	BIGTINT	Number of times this index has been advised
ESTIMATED_CREATION_TIME	ESTTIME	INT	Estimated number of seconds for index creation
REASON_ADVISED	REASON	CHAR(2)	Coded reason why index was advised
LOGICAL_PAGE_SIZE	PAGESIZE	INT	Recommended page size for index
MOST_EXPENSIVE_QUERY	QUERYCOST	INT	Execution time in seconds of the query
AVERAGE_QUERY_ESTIMATE	QUERYEST	INT	Average execution time in seconds of the query
TABLE_SIZE	TABLE_SIZE	BIGINT	Number of rows in table when the index was advised
NLSS_TABLE_NAME	NLSSNAME	CHAR(10)	NLSS table to use for the index
NLSS_TABLE_SCHEMA	NLSSDBNAME	CHAR(10)	Schema name of the NLSS table
MTI_USED	MTIUSED	BIGINT	Number of times an MTI that matched the advised definition was used by the database because a matching permanent index did not exist
MTI_CREATED	MTICREATED	INT	Number of times this specific index advice was used by the database to create a MTI
LAST_MTI_USED	LASTMTIUSE	TIMESTAMP	Last time an MTI was used by the database because a matching permanent index did not exist

The descriptions of the QSYS2/SYSIXADV columns are as follows:

- ▶ **Table for Which Index was Advised:** The optimizer is advising creation of a permanent index over this table. This is the long name for the table. The advice was generated because the table was queried and no existing permanent index could be used to improve the performance of the query.
- ▶ **Schema:** Schema or library name for the table.
- ▶ **Short Name:** System table name on which the index is advised.
- ▶ **Partition:** Partition detail for the index.
 - <blank>, which means for all partitions
 - For Each Partition
 - Specific name of the partition
- ▶ **Keys Advised:** Column names for the advised index. The order of the column names is important. The names should be listed in the same order on the CREATE INDEX SQL statement, unless the leading, order-independent key information indicates that the ordering can be changed.

- ▶ **Leading Keys Order Independent:** The keys at the beginning of the KEY_COLUMNS ADVISED field that could be reordered and still satisfy the index being advised.
- ▶ **Index Type Advised:**
 - Binary Radix (default)
You can find more information about Binary Radix indexes on the Web at:
<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/rzajq/rzajqbinary.htm>
 - Encoded Vector (not unique).
You can find more information about EVIs on the Web at:
<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/rzajq/whatareevi.htm>
- ▶ **Last Advised for Query Use:** The time stamp representing the last time this index was advised for a query.
- ▶ **Times Advised for Query Use:** The cumulative number of times this index has been advised. This count should stop increasing when a matching permanent index is created. The row of advice will remain in this table until the user removes it.
- ▶ **Estimated Index Creation Time:** Estimated time required to create this index.
- ▶ **Reason Advised:** Coded reason why index was advised.

Note: These are the same reason codes found in QQR COD in Detailed Performance Monitor record type 3020.

- I1 - Row selection
- I2 - Ordering/Grouping
- I3 - Row selection and Ordering/Grouping
- I4 - Nested Loop join
- I5 - Row selection using bitmap processing
- ▶ **Logical Page Size Advised (KB):** Recommended page size to be used on the PAGESIZE keyword of the CREATE INDEX SQL statement when creating this index.
- ▶ **Most Expensive Query Estimate:** Execution time in seconds of the longest-running query that generated this index advice.
- ▶ **Average of Query Estimates (seconds):** Average execution time in seconds of all queries that generated this index advice.
- ▶ **Rows in Table when Advised:** Number of rows in the table for which the index is being advised, for the last time this index was advised.
- ▶ **NLSS Table Advised:** The sort sequence table in use by the query that generated the index advice. For more detail about sort sequences refer to:
 - <http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/rzajq/usesortseq.htm>
 - <http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/db2/rbafzmtsrtsequence.htm#sortsequence>
- ▶ **NLSS Schema Advised:** The library of the sort sequence table.
- ▶ **MTI USED:** The number of times that this specific MTI has been used by the optimizer. The optimizer will stop using a matching MTI when a permanent index is created.

- ▶ **MTI CREATED:** The number of times that this specific MTI has been created by the optimizer. MTIs do not persist across system IPLs.
- ▶ **MTI LAST USED:** The time stamp representing the last time this specific MTI was used by the optimizer to improve the performance of a query. The MTI Last Used field can be blank, which indicates that an MTI that exactly matches this advice has never been used by the queries that generated this index advice.

In Example 9-1 there is an example of a custom query that you could use to view the System-Wide Index Advised table, QSYS.SYSIDXADV, and ask the question, "Which tables had the most indexes advised?"

Example 9-1 System-Wide Index Advised table - QSYS.SYSIDXADV

```
SELECT
  TABLE_NAME,
  TABLE_SCHEMA,
  KEY_COLUMNS_ADVISED,
  INDEX_TYPE,
  TIMES_ADVISED,
  REASON_ADVISED,
  LAST_ADVISED
FROM QSYS2.SYSIXADV
WHERE TABLE_SCHEMA NOT IN ('QSYS','QUSRSYS')
ORDER BY TIMES_ADVISED Desc, TABLE_NAME;
```

The results of this custom query is shown in Figure 9-1. The table on which an index was advised most was ACCOUNTEJB in schema TRADE6DB, 5,892 times. The reason for this index advise is I1, Row selection. Also notice that EVIs are advised, see row with table name ORDERS, in column INDEX_TYPE the text Encoded Vector.

TABLE_NAME	TABLE_SCHEMA	KEY_COLUMNS_ADVISED	INDEX_TYPE	TIMES_ADVISED	REASON_ADVISED	LAST_ADVISED
ACCOUNTEJB	TRADE6DB	PROFILE_USERID, ACCOUNTID	RADIX	5892	I1	2006-02-13 04:53:16.049964
QAYPSYSGRP	QMGT	OWNER, CHANGEDDATE	RADIX	601	I3	2006-06-27 11:00:18.458783
QAYPSJDFN	QMGT	OWNER, CHANGEDDATE	RADIX	526	I3	2006-06-27 11:00:18.458783
QAYPSJDS	QMGT	OWNER, CHANGEDDATE	RADIX	526	I3	2006-06-27 11:00:18.983133
QAYPSJDT	QMGT	OWNER, CHANGEDDATE	RADIX	526	I3	2006-06-27 11:00:18.983133
QAYPSJES	QMGT2	OWNER, CHANGEDDATE	RADIX	526	I3	2006-06-27 11:00:18.983133
QAYPSJET	QMGT2	OWNER, CHANGEDDATE	RADIX	526	I3	2006-06-27 11:00:18.983133
QAYPSJEVT	QMGT	OWNER, CHANGEDDATE	RADIX	526	I3	2006-06-27 11:00:18.458783
QAYPSJDS	QMGT	OWNER, STATUS	RADIX	525	I1	2006-06-27 11:00:18.983133
QAYPSJDT	QMGT	OWNER, STATUS	RADIX	525	I1	2006-06-27 11:00:18.983133
QAYPSJES	QMGT2	OWNER, STATUS	RADIX	525	I1	2006-06-27 11:00:18.983133
QAYPSJET	QMGT2	OWNER, STATUS	RADIX	525	I1	2006-06-27 11:00:18.983133
ORDERS	DBITSODATA	CUSTKEY	ENCODED VECTOR	312	I1	2006-08-15 11:55:15.007238
SALES	COUNTRYUSA	TYPE, SALESDATE, STOREID	RADIX	243	I1	2006-04-24 07:50:17.113486
SALES	COUNTRYUSA	TYPE, SALESDATE, CURRENCY, BRAN...	RADIX	232	I1	2006-04-24 07:50:17.113486
ORDERS	DBITSODATA	SHIPDATE	ENCODED VECTOR	210	I1	2006-08-15 11:55:13.434266
CUSTOMERS	DBITSODATA	CUSTOMER, CUSTKEY	RADIX	180	I1	2006-08-15 11:55:14.482908
ORDERS	DBITSODATA	CUSTKEY	RADIX	177	I1	2006-08-15 11:55:14.482908
ORDERS	DBITSODATA	SHIPDATE	RADIX	165	I1	2006-08-15 11:55:13.434266
REGIONS	COUNTRYUSA	REGIONNAME	RADIX	141	I1	2006-04-28 11:27:56.741770
ORDERS	CASESTUDY1	CUSTKEY	RADIX	133	I1	2006-08-29 16:59:57.333278
CUSTOMERS	DBITSODATA	CUSTOMER	RADIX	131	I1	2006-08-15 11:55:02.021053
CUSTOMERS	CASESTUDY1	CUSTOMER, CUSTKEY	RADIX	124	I1	2006-08-29 16:59:57.333278

Figure 9-1 System Wide Index Advised table

The following is the difference between the information returned from a custom query written for the System-Wide Index Advised table and one written for "The 3020 record: index advised (SQE)" on page 190:

- ▶ The System-Wide Index Advised table is summary information and provides the *big picture* and is a good starting point for checking for large number of advised indexes for a particular table.

- ▶ In order to determine if a number of indexes advised have been made for the same table, you will have to modify the query in Example 9-1 to be ordered by table name and number of times advised.
- ▶ You will notice that no specific instance, statement, or job information is contained within this table, so you cannot drill down to the specific SQL statement which generated any of the index advice.
- ▶ Remember that the System-Wide Advised table is not automatically *pruned*, cleared of old data, so it must be done manually via the iSeries Navigator *Index Advisor* or using SQL *Delete*.
- ▶ For more comprehensive advised index information reporting, use a custom query to process the Detailed Performance Monitor data record 3020. Section 6.4.9, “Index advised” on page 214 provides examples of how to write custom queries to get to the level of detail required to determine the who, what, when and why but more importantly which indexes you should create.

Note: The source code used in the SQL script examples in this chapter can be found in the downloadable material of the book found in Appendix B, “Additional material” on page 467.

9.3 Levels of Index Advisor access

Index Advisor is accessible at three levels. They are shown in the following sections of this chapter:

- ▶ “Index Advisor access at Database level” - also known as System Wide Index Advisor
- ▶ “Index Advisor access at Schema level”
- ▶ “Index Advisor access at Table level”

Note: The optimizer provides feedback to your SQL queries in the form of an Access Plan that is populated to the Plan Cache or Program Object. At the same time, the feedback is also populated to QSYS2/SYSIZADV system table. The Index Advised information provided on Database, Schema and Table level are retrieved from QSYS2/SYSIXADV system table.

9.3.1 Index Advisor access at Database level

To access Index Advisor at Database level, select **iSeries Navigator** → **Database** and right-click the database name. Then select **Index Advisor** as shown in Figure 9-2.

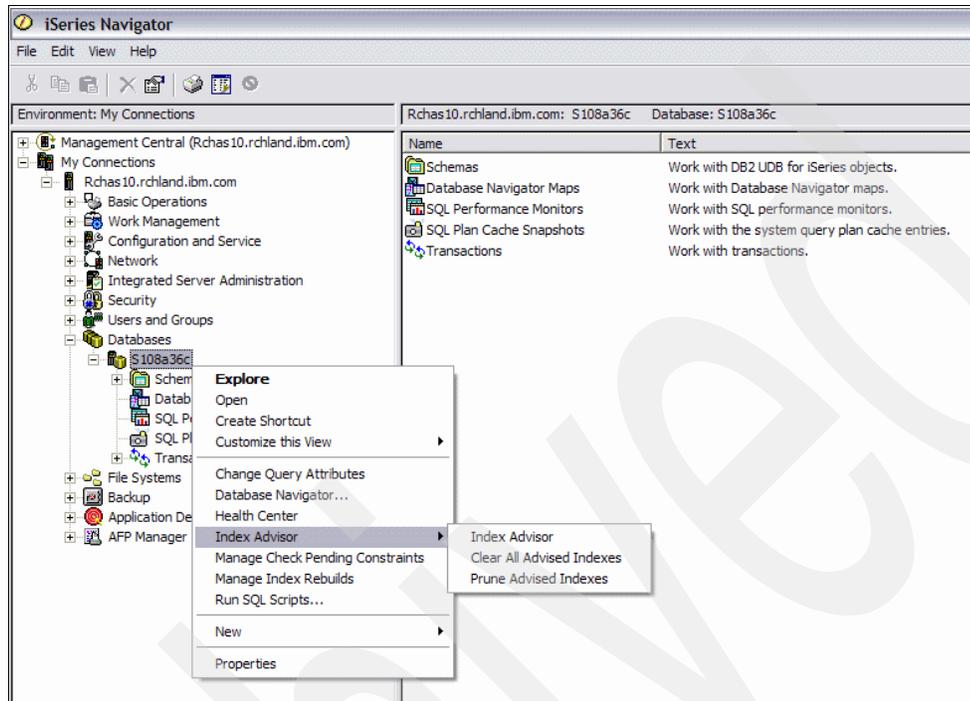


Figure 9-2 Index Advisor access at Database level

9.3.2 Index Advisor access at Schema level

To access Index Advisor at Schema level, select **iSeries Navigator** → **Database** → **database name** → **Schema**. Right-click your schema name and select **Index Advisor** as shown in Figure 9-3.

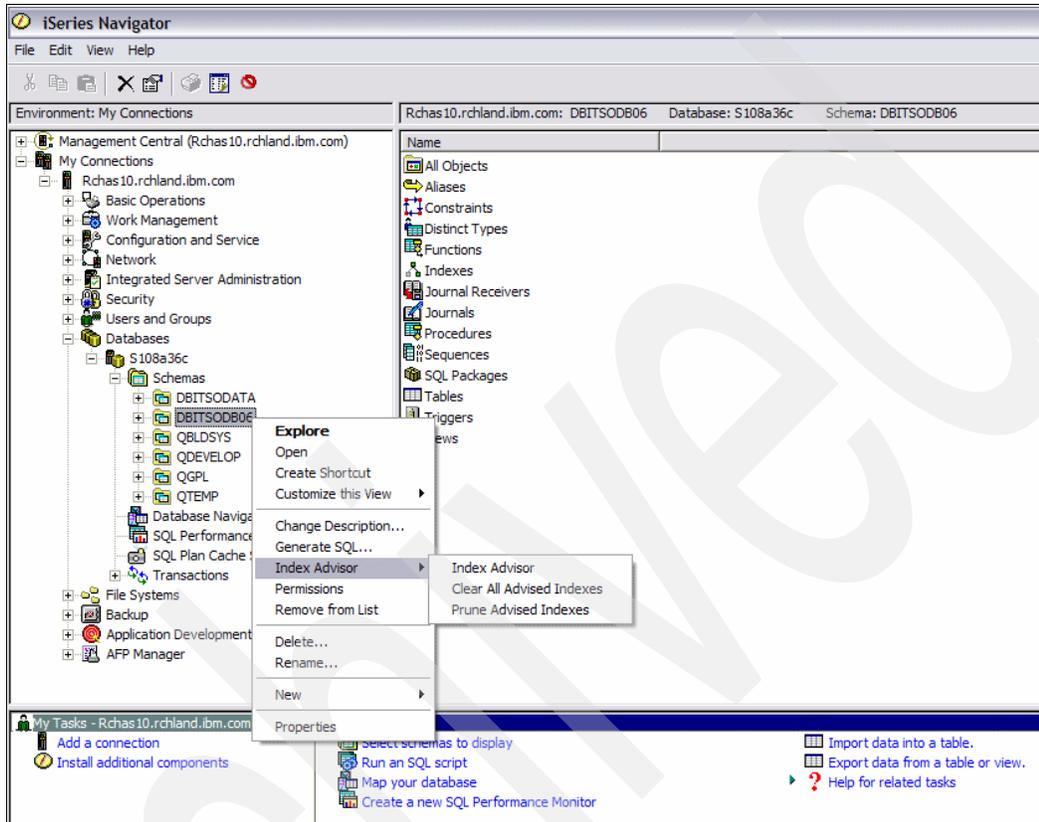


Figure 9-3 Index Advisor access at Schema level

9.3.3 Index Advisor access at Table level

To access Index Advisor at Table level, use **iSeries Navigator** → **Database**. Select the database name → **Schema**. Then select the schema name → **Tables** and right-click your table name. Then select **Index Advisor** as shown in Figure 9-4.

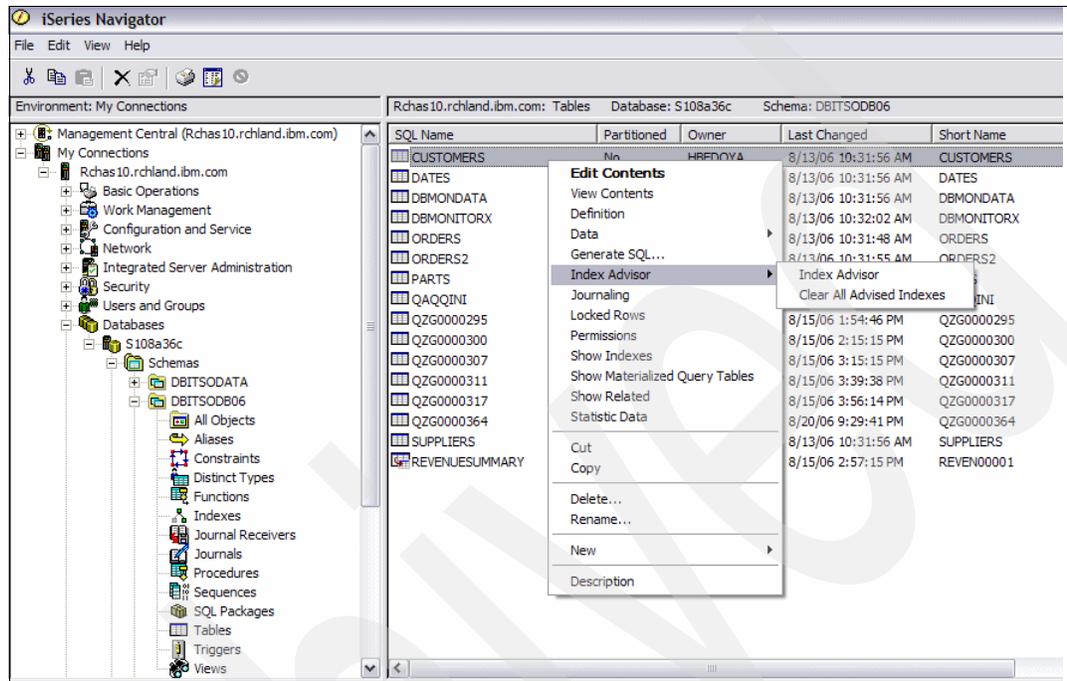


Figure 9-4 Index Advisor access at Table level

Note:

- ▶ CQE only provides basic advice based on local selection predicates.
- ▶ SQE provides complex advice based on all parts of the query, it supports predicates like Group By, Order By and Joins.
- ▶ Currently SQE does not advise indexes for OR predicates.

9.4 Index Advisor interface in iSeries Navigator

The index Advisor interface in iSeries Navigator offers three main functions as shown in Figure 9-3:

- ▶ **Index Advisor: Work with Index Advisor**
- ▶ **Clear All Advised Indexes:** Clear advised indexes from index advisor table
- ▶ **Prune Advised Indexes:** Prune advised indexes from index advisor table for tables that no longer exist

The QSYS2/SYSIXADV system table is populated by the database but is not maintained. As indexes are created or tables are dropped, the old advise is not removed automatically by the system. It can be considered an historical table and as such will keep accumulating rows, so you may want to clear out the table occasionally. You can use the Clear Advised indexes menu option. For the database-wide and schema-wide views you have an additional option to

Prune Advised Indexes. This option removes all index advised rows where the table that the advise was created for no longer exists. There's also a menu option to remove the selected row of index advised from the list.

When you select **Index Advisor** → **Index Advisor** regardless of which level you are at (Database, Schema or Table), the information is populated in the format shown in Figure 9-5.

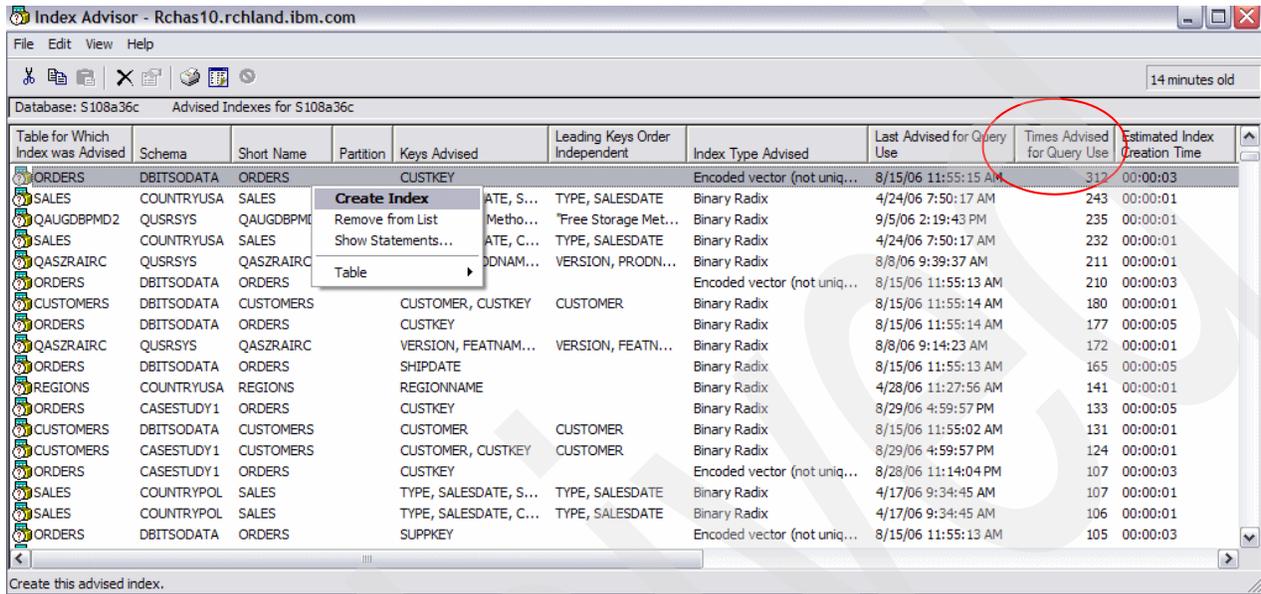


Figure 9-5 Index Advisor

The rows are sorted by Times Advised for Query Use column so that the most frequently advised indexes are first in the list. To resort the list by any other column, select that column header.

You can also see additional statistics for the last time the index was advised and the estimated time it might take to create the index. These pieces of information can help you gauge the impact of creating the new index. If the index is recommended many times but was last recommended a month ago, and the table is one that would be considered historical, you might conclude that there is no compelling reason to act on the advice. However, if the advice was recommended five minutes ago, is advised many times and the table is one of your key production tables, it qualifies as a great candidate.

Index Type Advised column tells you the type of index recommended (binary radix or EVI). Reason Advised column shows you the reason the optimizer recommended the index. These columns combine to give you a more complete picture of the optimizer's advice.

Another important piece of information is the list of key columns being advised for this index, along with the leading key-order independent columns. If there are keys in the leading key-order independent column, it means that the index could have the key columns specified in any order. When the Index Advisor specifies leading key-order independent columns, there is an opportunity to collapse advise to cover more queries. Since indexes must be maintained by the database, reducing their number can improve performance.

When determining whether to create, drop or replace indexes based on this index advice, consider all of the indexes being advised for this table, as well as the existing table.

Note: At table level, information is displayed if there is an index advised, otherwise you may find the window returns no row.

When you have decided to create an index, right click the **table name** → select **Create Index** Alternatively you can select **Show Statements** to determine more of the SQL statement before creating an index. You also can select **Remove from List** to remove the entry from the advisor list.

Note: The Table option allows you to perform all the table-related functions such as Edit Content, View Content, Definition, Data and so on. For more information, refer to *Advanced Functions and Administration on DB2 Universal Database for iSeries*, SG24-4249.

From Figure 9-5, when you select **Show statements**, you can perform further statement analysis on the statements shown in Figure 9-6.

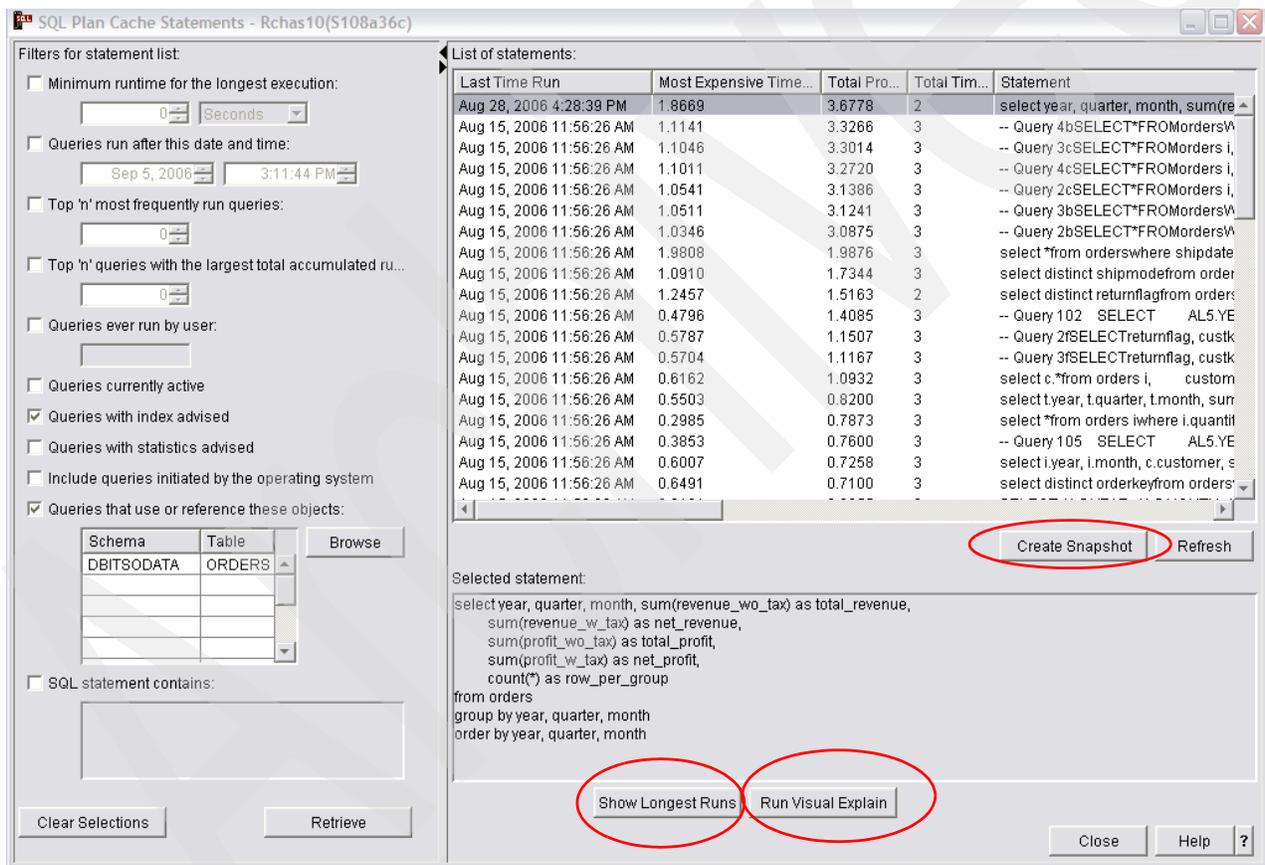


Figure 9-6 SQL Plan Cache Statements

From Figure 9-6, you can also activate Create Snapshot, Show Longest Run (result shown in Figure 9-7) or Run Visual Explain function from this interface.

Time Run	Processi...	Records ...	User Na...	Job Name	Job User	Job Num...	CPU Tim...	Synchron...	Synchron...	Asynchri
Aug 15, 2...	1.1141	2	HBEDOYA	QZDASOI...	QUSER	040465	0.6864	0	0	0
Aug 15, 20...	1.1085	2	HBEDOYA	QZDASOI...	QUSER	040465	0.6842	0	0	0
Aug 15, 20...	1.1039	2	HBEDOYA	QZDASOI...	QUSER	040465	0.6801	0	0	0

Figure 9-7 Show Longest Run

Note: For more information about Creating Snapshot, refer to Chapter 7, “SQE Plan Cache and SQE Plan Cache Snapshots” on page 237 and for more information about Visual Explain, refer to Chapter 8, “Analyzing database performance data with Visual Explain” on page 275.

9.5 Interfaces to Index Advised information

You can access Index Advised information from various iSeries Navigator screen interfaces such as:

- ▶ From the Detailed SQL Performance Monitor
- ▶ From the SQE Plan Cache screen interface
- ▶ From the SQE Plan Cache Snapshot screen interface
- ▶ From the Visual Explain screen interface
- ▶ From the Debug messages

Note:

- ▶ The Index Advised information used by these interfaces is retrieved from the Access Plan feedback that the optimizer provides instead of QSYS2/SYSIXADV.
- ▶ For more information about Creating Snapshots, refer to Chapter 7., “SQE Plan Cache and SQE Plan Cache Snapshots”; and for more information about Visual Explain, refer to Chapter 8., “Analyzing database performance data with Visual Explain”.

9.5.1 Access to Index Advised information from Detailed SQL Performance Monitor screen interface

To access Index Advised information from SQE Plan Cache Snapshot screen interface, perform the following steps:

1. In the **SQL Performance Monitors** right pane, right-click the detailed performance monitor name and select **Analyze** as shown in Figure 9-8.

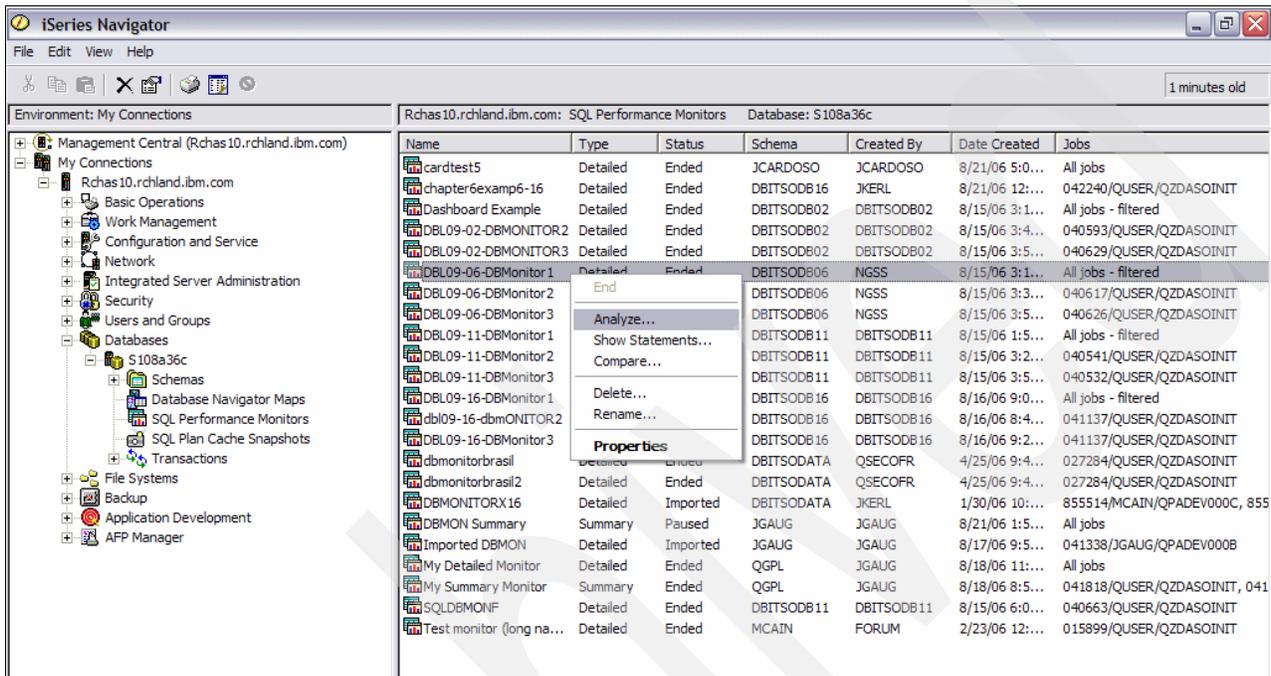


Figure 9-8 Analyzing Detailed Performance Monitor

- As shown in Figure 9-9, in the Analysis Overview Dashboard window, expand the **Overview** folder if it is not expanded and highlight **Index Creates Advised**. Then select **Summary**.

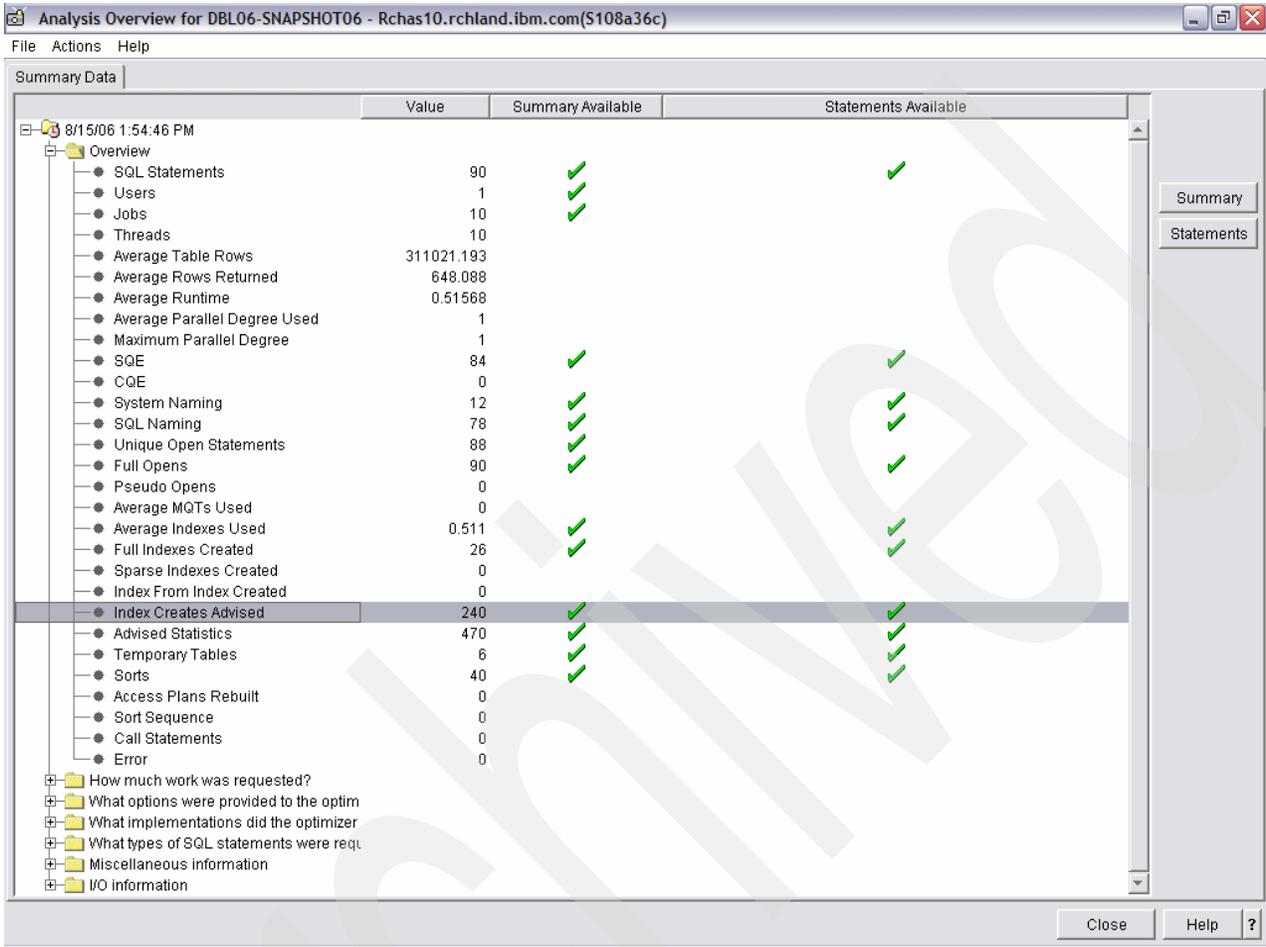


Figure 9-9 The Analysis Overview Dashboard

- You get Index Creates Advised - Summary window in Figure 9-10. Scroll to the right for more detail. The rows are sorted by the Maximum Runtime column.

Most Expensive Use	Index Creates Advised	Rows In Base Tab...	Estimated Nonparallel Create Time	Average Indexes Used	Time To Create Index
2006-08-15 15:16:37.056569	2	307786.000		1.000	
2006-08-15 15:16:37.222612	1	600572.000		1.000	
2006-08-15 15:16:37.129800	3	405381.333		1.000	
2006-08-15 15:16:36.991114	2	600572.000		1.000	
2006-08-15 15:16:36.960197	1	600572.000		1.000	
2006-08-15 15:15:15.505730	1	80.000		1.000	1.000

Average MQTs Used	Maximum Runtime	Average Runtime	Minimum Runtime	Total Runtime	Maximum Open Time	Maximum Fetch Time	Maximum Close Time
0.000	0.390288	0.390288	0.390288	0.390288	0.071608	0.317632	0.001048
0.000	0.285584	0.285584	0.285584	0.285584	0.035952	0.248760	0.000972
0.000	0.213752	0.213752	0.213752	0.213752	0.091200	0.121576	0.000976
0.000	0.176128	0.176128	0.176128	0.176128	0.061816	0.113312	0.001000
0.000	0.097184	0.097184	0.097184	0.097184	0.029200	0.067048	0.000936
0.000	0.045000	0.045000	0.045000	0.045000	0.042008	0.000288	0.002704

Figure 9-10 Index Creates Advised - Summary

- If you select **Statement** option, you get the Index Creates Advised - Statements window in Figure 9-11. Scroll to the right for more detail. The rows are sorted by the Runtime column.

Time	Reason Code	Number Of Advised Key Columns	Advised Index Keys
2006-08-15 15:16:37.210916	Record Selection	2	CUSTOMER, CUSTKEY
2006-08-15 15:16:37.210921	Record Selection	1	CUSTKEY
2006-08-15 15:16:37.210974	Record Selection	1	CUSTKEY
2006-08-15 15:16:37.120665	Record Selection	1	CUSTKEY
2006-08-15 15:16:37.120716	Record Selection	2	CUSTKEY, CUSTOMER
2006-08-15 15:16:37.044855	Record Selection	3	YEAR, MONTH, RETURNFLAG
2006-08-15 15:16:37.044904	Record Selection And ORDER BY Or GROUP BY	3	YEAR, MONTH, SHIPMODE
2006-08-15 15:15:15.529213	Record Selection And ORDER BY Or GROUP BY	4	"Save Library", "Name", "Date Started", "Time Started"
2006-08-15 15:16:37.254355	Record Selection And ORDER BY Or GROUP BY	2	YEAR, SHIPMODE
2006-08-15 15:16:36.983439	Record Selection	1	ORDERKEY

Array	System Base Table	System Base Table Member	Runtime	Statement Outcome	SQLSTATE	SQLCODE	Operation	Statement Text
	CUSTOMERS	CUST_DIM	0.091200	Successful	00000	0	OPEN	SELECT O.* FROM ORDERS O, CUS
	ORDERS	ITEM_FACT	0.091200	Successful	00000	0	OPEN	SELECT O.* FROM ORDERS O, CUS
	ORDERS	ITEM_FACT	0.091200	Successful	00000	0	OPEN	SELECT O.* FROM ORDERS O, CUS
	ORDERS	ITEM_FACT	0.071608	Successful	00000	0	OPEN	SELECT O.* FROM ORDERS O, CUS
	CUSTOMERS	CUST_DIM	0.071608	Successful	00000	0	OPEN	SELECT O.* FROM ORDERS O, CUS
	ORDERS	ITEM_FACT	0.061816	Successful	00000	0	OPEN	SELECT * FROM ORDERS
	ORDERS	ITEM_FACT	0.061816	Successful	00000	0	OPEN	SELECT * FROM ORDERS
	QAUGDBPMD2	QAUGDBPMD2	0.042008	Successful	00000	0	OPEN	SELECT SYSSUM_NAME_00001, SYSS
	ORDERS	ITEM_FACT	0.035952	Successful	00000	0	OPEN	SELECT YEAR, SHIPMODE, SUM(REVE
	ORDERS	ITEM_FACT	0.029200	Successful	00000	0	OPEN	SELECT * FROM ORDERS WHERE OR

Figure 9-11 Index Creates Advised - Statements

With these details, you can perform an index review and further performance analysis.

Note: For more information about the Dashboard tool, refer to Chapter 5., “Analyzing SQL performance data using iSeries Navigator”.

9.5.2 Access to Index Advised information from SQE Plan Cache screen interface

To access Index Advised information from SQE Plan Cache screen interface, perform the following steps:

1. In the iSeries Navigator, select **Database** and select the **database name**. Then select **Schema**, right-click **SQL Plan Cache Snapshot** and select **SQL Plan Cache** → **Show Statements** as shown in Figure 9-12.

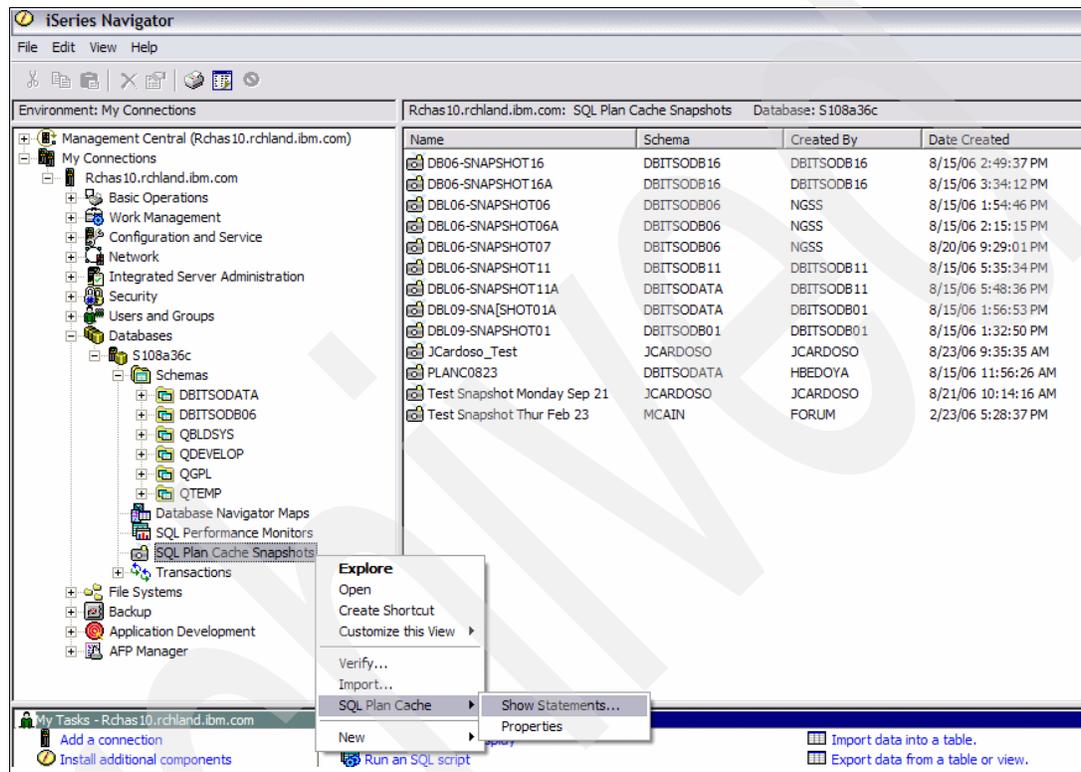


Figure 9-12 Accessing Index Advised information from SQE Plan Cache screen interface

2. You are presented with the SQL Plan Cache Statements window in Figure 9-13. Select **Queries with index advised** → **Retrieve** and you see the list of statements where indexes are advised. You can analyze further by choosing **Show Longest Runs**, **Run**

Visual Explain or even Create Snapshot and import into other system to carry out further analysis.

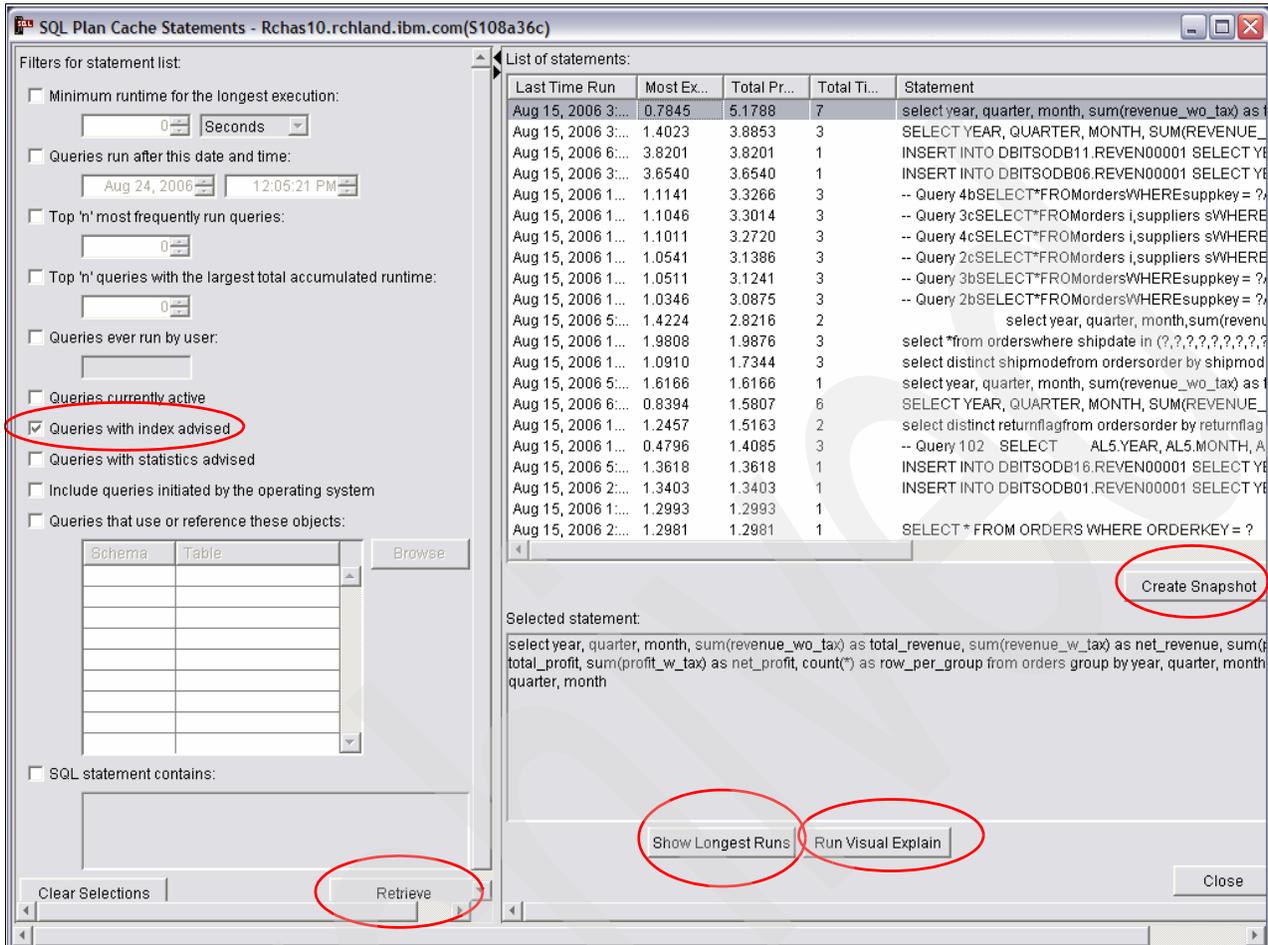


Figure 9-13 Select Queries with index advised

9.5.3 Access to Index Advised information from SQE Plan Cache Snapshot screen interface

To access Index Advised information from SQE Plan Cache Snapshot screen interface, perform the following steps:

1. In the SQL Plan Cache Snapshot right pane, right-click the snapshot name and select **Analyze** as shown in Figure 9-14.

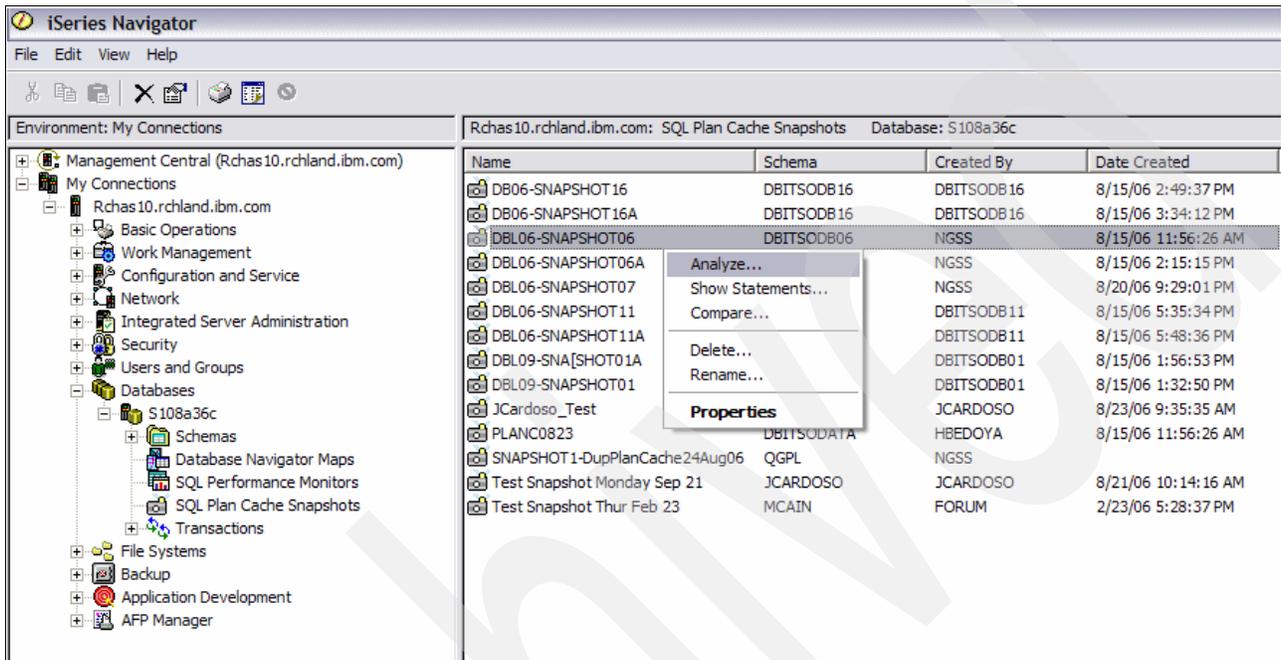


Figure 9-14 Path to SQL Plan Cache Snapshot Analysis

2. In the **Dashboard** window in Figure 9-15, expand **Overview** folder if it is not expanded and highlight **Index Created advised**. Then select **Summary**.

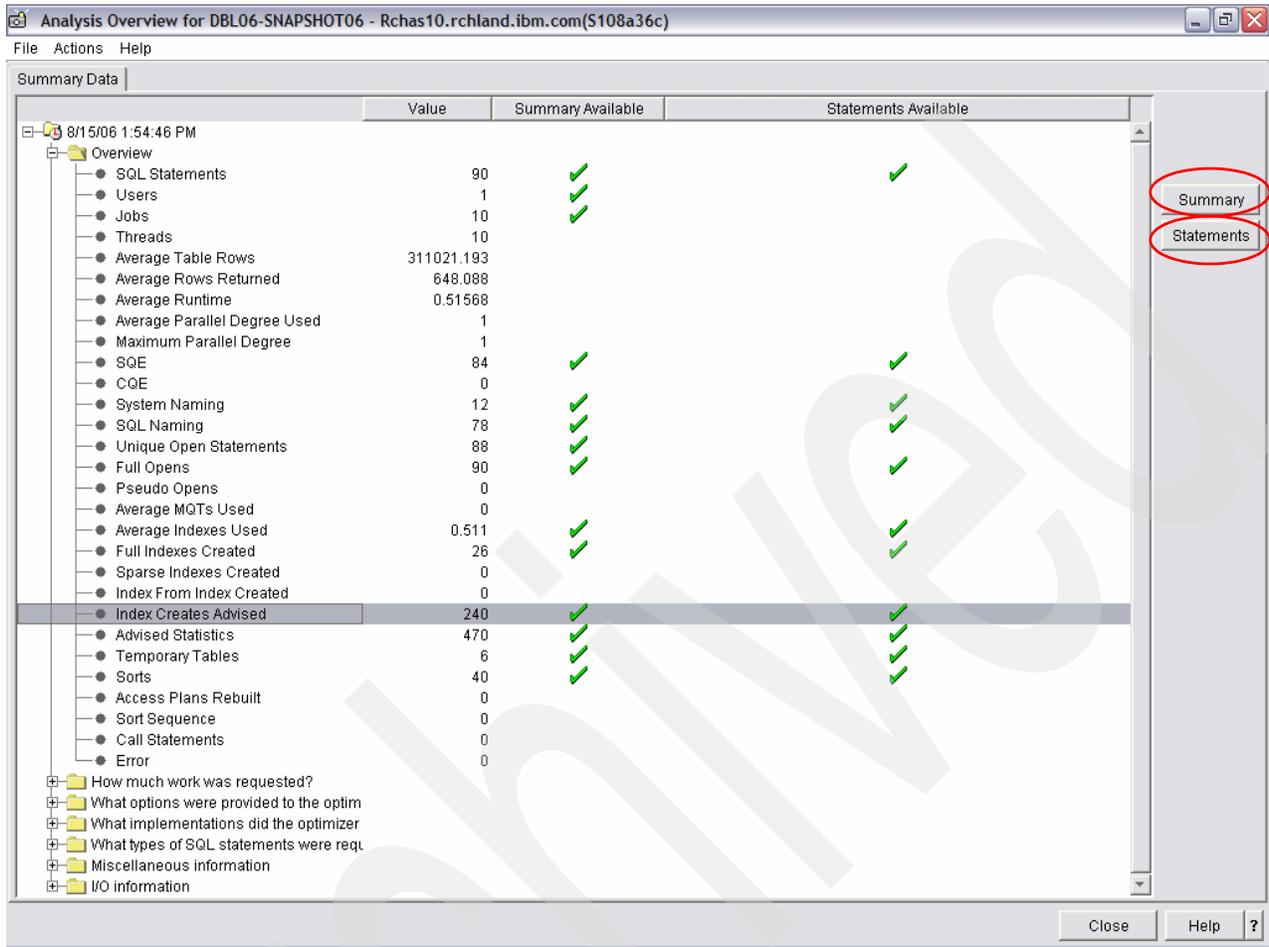


Figure 9-15 The Dashboard

- In Figure 9-16, you see the Index Creates Advised-Summary window populated with information. Scroll to the right to view more details. The rows are sorted by Maximum Runtime column.

DBL06-SNAPSHOT06 - Index Creates Advised - Summary - Rchas10.rchland.ibm.com(S108a36c)						
File Actions Help						
Most Expensive Use	Index Creates Advised	Rows In Base Table	Estimated Nonparallel Create Time	Average Indexes Used	Time To Create Index	
2006-08-15 13:19:32.677015		6	405381.333	1.000		
2006-08-15 11:56:26.316648		2	600572.000	1.000	1.000	
2006-08-15 11:56:26.316648		3	400714.666	1.000	2.000	
2006-08-15 11:56:26.316648		3	400714.666	1.000	2.000	
2006-08-15 11:56:26.316648		2	300786.000	1.000	2.000	
2006-08-15 11:56:26.316648		2	600572.000	1.000	1.000	
2006-08-15 11:56:26.316648		1	600572.000	1.000	1.000	
2006-08-15 11:56:26.316648		1	600572.000	1.000		
2006-08-15 11:56:26.316648		1	600572.000	1.000	1.000	

DBL06-SNAPSHOT06 - Index Creates Advised - Summary - Rchas10.rchland.ibm.com(S108a36c)									
File Actions Help									
Temporary Indexes Reused	Average MQTs Used	Maximum Runtime	Average Runtime	Minimum Runtime	Total Runtime	Maximum Open Time	Maximum Fe		
0	0.000	1.299312	0.680574	0.061836	1.361148				
0	1.000	1.108888	1.108888	1.108888	1.108888	3.326664			
0	2.000	1.100488	1.100488	1.100488	1.100488	3.301464			
0	2.000	1.090674	1.090674	1.090674	1.090674	3.272024			
0	2.000	1.046200	1.046200	1.046200	1.046200	3.138600			
0	1.000	1.041384	1.041384	1.041384	1.041384	3.124152			
0	1.000	1.029186	1.029186	1.029186	1.029186	3.087560			
0	0.000	0.758172	0.758172	0.758172	0.758172	1.518344			
0	0.000	0.662552	0.662552	0.662552	0.662552	1.987656			
0	0.000	0.578144	0.578144	0.578144	0.578144	1.734432			
0	1.000	0.469509	0.469509	0.469509	0.469509	1.408528			
0	0.000	0.383586	0.383586	0.383586	0.383586	1.150760			

Figure 9-16 Index Creates Advised-Summary window

- From Figure 9-15, when you highlight **Index Created advised**, select **Statements**. You see the Index Creates Advised - Statements window as shown in Figure 9-17. Scroll to the right for more information. The rows are sorted by Runtime column.

DBL06-SNAPSHOT06 - Index Creates Advised - Statements - Rchas10.rchland.ibm.com(S108a36c)						
File Actions Help						
Time	Reason Code	Number Of Advised Key Columns	Advised Index Keys			
2006-08-15 11:56:26.316648	Record Selection	4	SHIPMODE, RETURNFLAG, SUPPKEY, QUANTITY			
2006-08-15 11:56:26.316648	Record Selection And ORDER BY Or GROUP BY	4	SHIPMODE, RETURNFLAG, SUPPKEY, SHIPDATE			
2006-08-15 11:56:26.316648	Record Selection	3	RETURNFLAG, SUPPKEY, QUANTITY			
2006-08-15 11:56:26.316648	Record Selection	1	SUPPKEY			
2006-08-15 11:56:26.316648	Record Selection And ORDER BY Or GROUP BY	3	RETURNFLAG, SUPPKEY, SHIPDATE			
2006-08-15 11:56:26.316648	Record Selection	4	RETURNFLAG, SHIPMODE, SUPPKEY, QUANTITY			
2006-08-15 11:56:26.316648	Record Selection	1	SUPPKEY			
2006-08-15 11:56:26.316648	Record Selection And ORDER BY Or GROUP BY	4	RETURNFLAG, SHIPMODE, SUPPKEY, SHIPDATE			

DBL06-SNAPSHOT06 - Index Creates Advised - Statements - Rchas10.rchland.ibm.com(S108a36c)								
File Actions Help								
LSS Table	System Base Table Library	System Base Table	System Base Table Member	Runtime	Statement Outcome	SQLSTATE	SQLCODE	Operation
DBITSODATA		ORDERS	ITEM_FACT	3.326664	Successful	00000		0 OPEN
DBITSODATA		ORDERS	ITEM_FACT	3.326664	Successful	00000		0 OPEN
DBITSODATA		ORDERS	ITEM_FACT	3.301464	Successful	00000		0 OPEN
DBITSODATA		SUPPLIERS	SUPP_DIM	3.301464	Successful	00000		0 OPEN
DBITSODATA		ORDERS	ITEM_FACT	3.301464	Successful	00000		0 OPEN
DBITSODATA		ORDERS	ITEM_FACT	3.272024	Successful	00000		0 OPEN
DBITSODATA		SUPPLIERS	SUPP_DIM	3.272024	Successful	00000		0 OPEN
DBITSODATA		ORDERS	ITEM_FACT	3.272024	Successful	00000		0 OPEN
DBITSODATA		ORDERS	ITEM_FACT	3.138600	Successful	00000		0 OPEN
DBITSODATA		SUPPLIERS	SUPP_DIM	3.138600	Successful	00000		0 OPEN

Figure 9-17 Index Creates Advised - Statements window

9.5.4 Access to Index Advised information from Visual Explain screen interface

Run SQL Scripts is a component of iSeries Navigator and it is available under the Database icon as shown in Figure 9-18. To locate the Database icon, you must establish a session on your selected iSeries server using the iSeries Navigator icon.

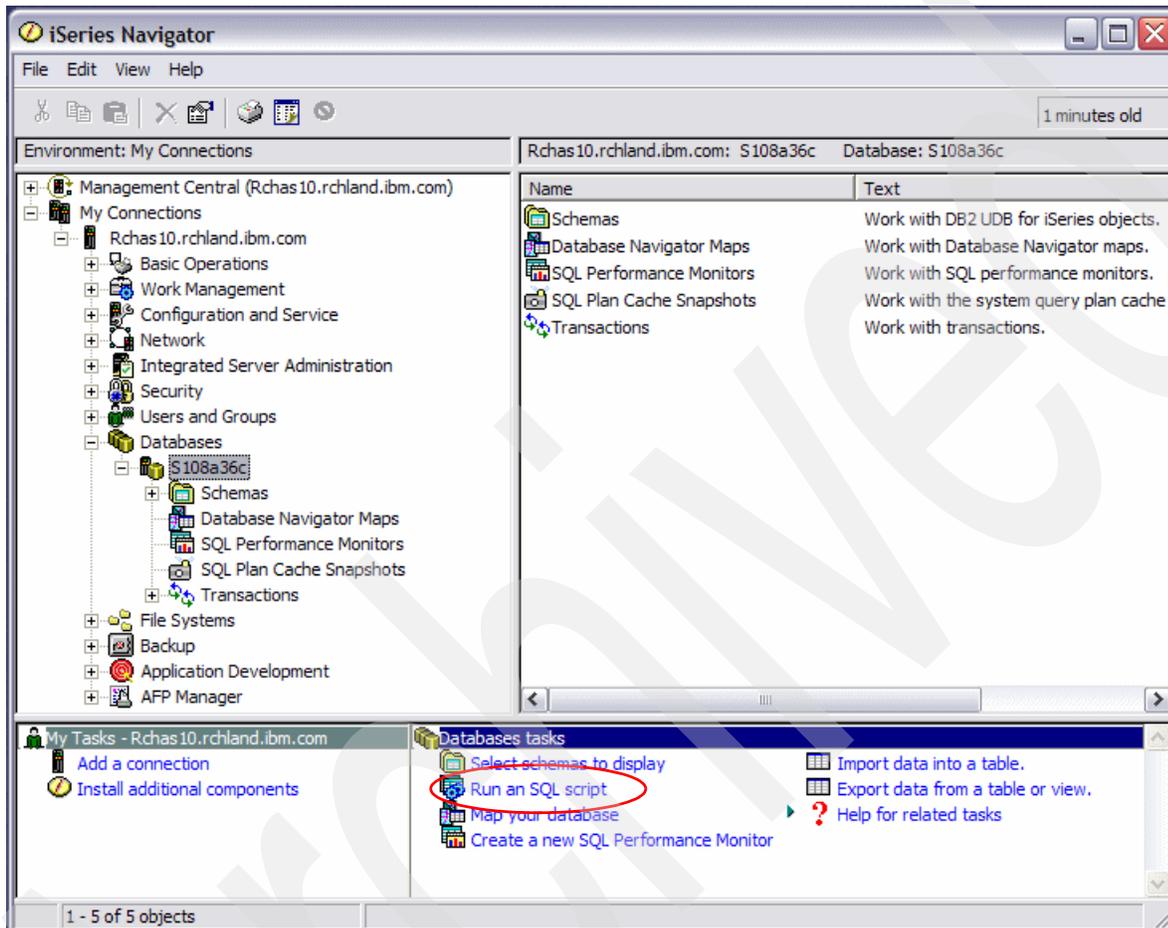


Figure 9-18 Path to Run SQL Scripts

As shown in Figure 9-19, from the SQL Scripts Center, you can access Visual Explain directly, either from the menu or from the toolbar.

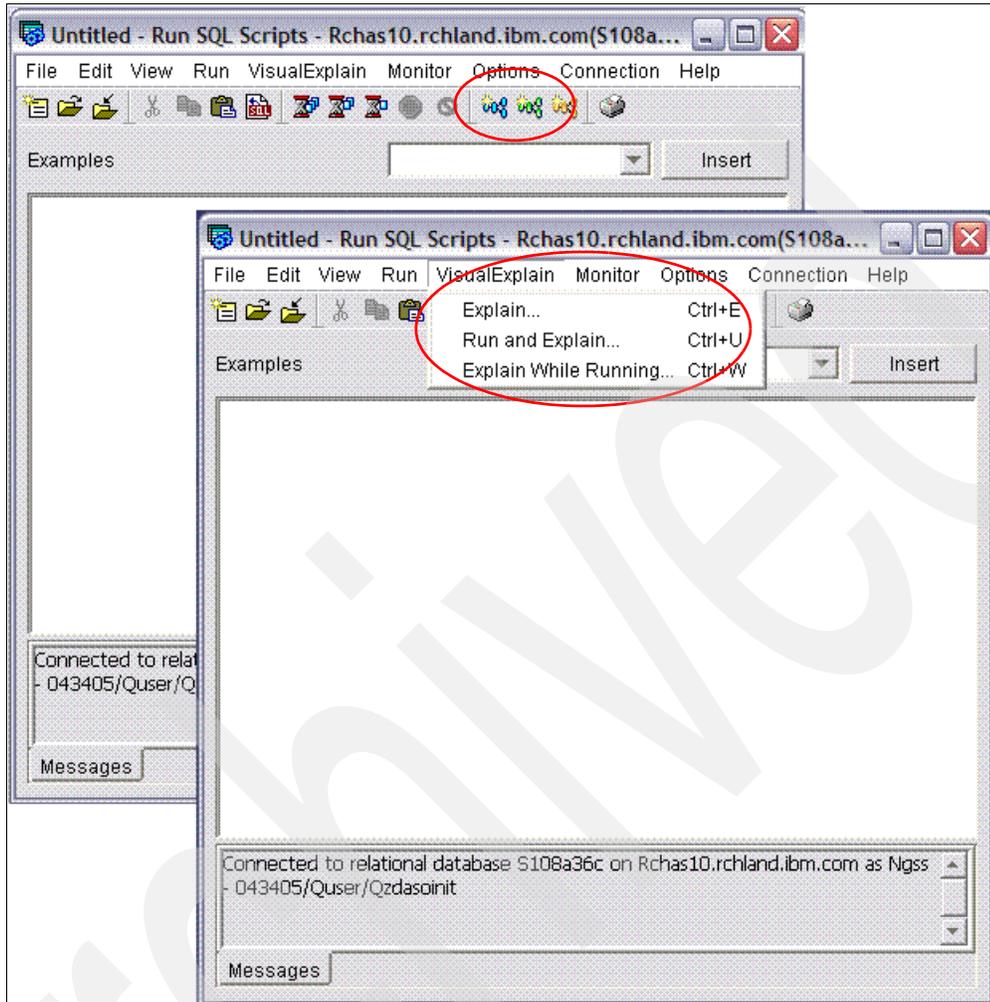


Figure 9-19 Access Visual Explain from toolbar or icon

Another way to access Visual Explain is through the Show Statement window as shown in Figure 9-20. You can access the Show Statements window from by selecting **SQL Performance Monitor**. Then right-click your database monitor data and select **Show Statements**. Alternatively, select **SQL Plan Cache Snapshot** and right-click your snapshot

and select **Show Statements** (refer to Figure 9-18 for the SQL Performance Monitor folder and SQL Plan Cache Snapshot folder).

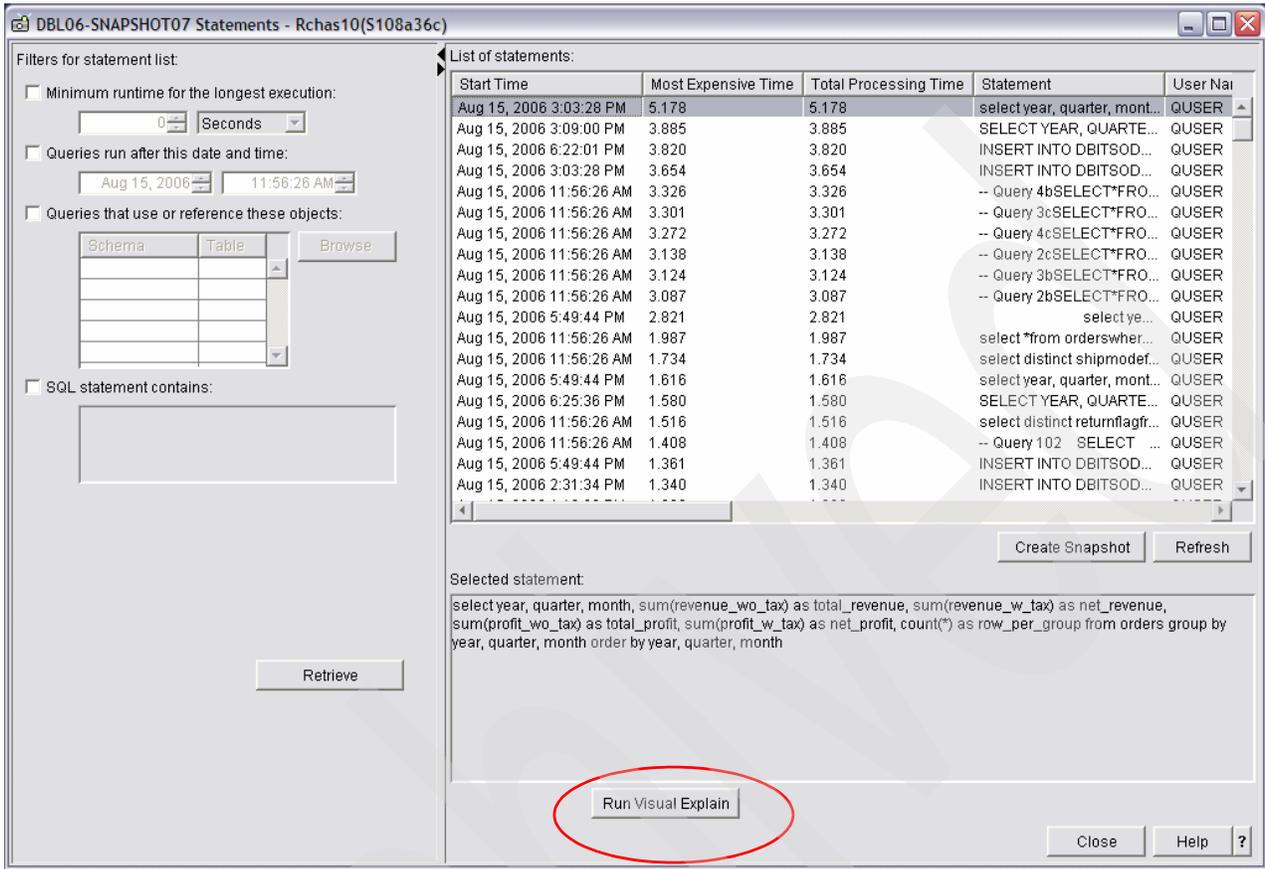


Figure 9-20 Show Statements window

Note: Refer to Chapter 8., “Analyzing database performance data with Visual Explain” for more information about what Visual Explain is and how to use Visual Explain to integrate with Database Monitors and SQE Plan Cache Snapshots. Alternatively, you can also refer to Chapter 7., “SQE Plan Cache and SQE Plan Cache Snapshots” on the integration of plan cache and plan cache snapshot to Visual Explain.

When you are in Visual Explain as shown in Figure 9-21, select **View** and enable **Highlight Index Advised**. The Table Scan (Index Advised section) is immediately highlighted. Click the highlighted icon once (Table Scan icon in this case) and the index related information is

displayed on the right pane. Scroll down and locate the Index advised information section for more details.

The screenshot shows the Visual Explain interface for a query. The left pane contains a menu with options like 'Zoom', 'Overview', 'Icon Spacing', 'Orientation', 'Arrow Labels', 'Icon Labels', 'Highlight Expensive Icons', 'Highlight Index Advised' (checked), 'Highlight LPG', 'Highlight Materialized Query Tables', and 'Refresh'. A tooltip for 'Highlight Index Advised' says 'Highlight icons where index creation is advised'. The main pane shows a query plan diagram with steps: Table Scan (circled in red), Temporary Distinct Hash Table, Hash Scan, Temporary Sorted List, Sorted List Scan, and Final Select. The right pane shows a table of performance metrics. The 'Index advised information' section is circled in red.

Attribute	Value
Estimated Cost Information About the ...	
Processing Time(ms)	1536.25
I/O Or CPU Bound	I/O Bound
CPU Cost(ms)	219.774
I/O Cost(ms)	1536.25
I/O Count	1229
PreLoad Relation	No
Memory Used(bytes)	2740549.64
Share of Memory Available(bytes)	27405496
Memory Constrained	No
Cumulative Memory Constrained	No
Index advised information	
Creation of an Index is Advised	Yes
Number of Primary Key Columns	3
Library of Table Being Queried	DBITSODB06
Name of Base Table	ORDERS
List of Key Columns for Advised Index	YEAR,QUARTER,MONTH
Type of Index Created	Binary Radix
Number of Unique Index Values	Not Available
ACS Table Name	*HEX
ACS Table Library	*N
Information About the Plan Performed	
Scrollable	Yes
Plan Name	Table Scan
Plan Step Type	Logic
Reason Code	No Indexes Exist
Plan Step Name	Node_11
Statement Text	SELECT

Statement text

```
select year, quarter, month, sum(revenue_wo_tax) as total_revenue, sum(revenue_w_tax) as net_revenue, sum(profit_wo_tax) as total_profit, sum(profit_w_tax) as net_profit, count(*) as row_per_group from orders group by year, quarter, month order by year, quarter, month
```

Figure 9-21 View options in Visual Explain

As shown in Figure 9-22, when you select **Actions** from the pull-down menu, there are three index related options you can perform:

- ▶ **Show Index:** Display a list of the indexes for the selected table
- ▶ **Create Index:** Create index based on advised or temporary index
- ▶ **Advisor:** Show Statistics and Index Advisor

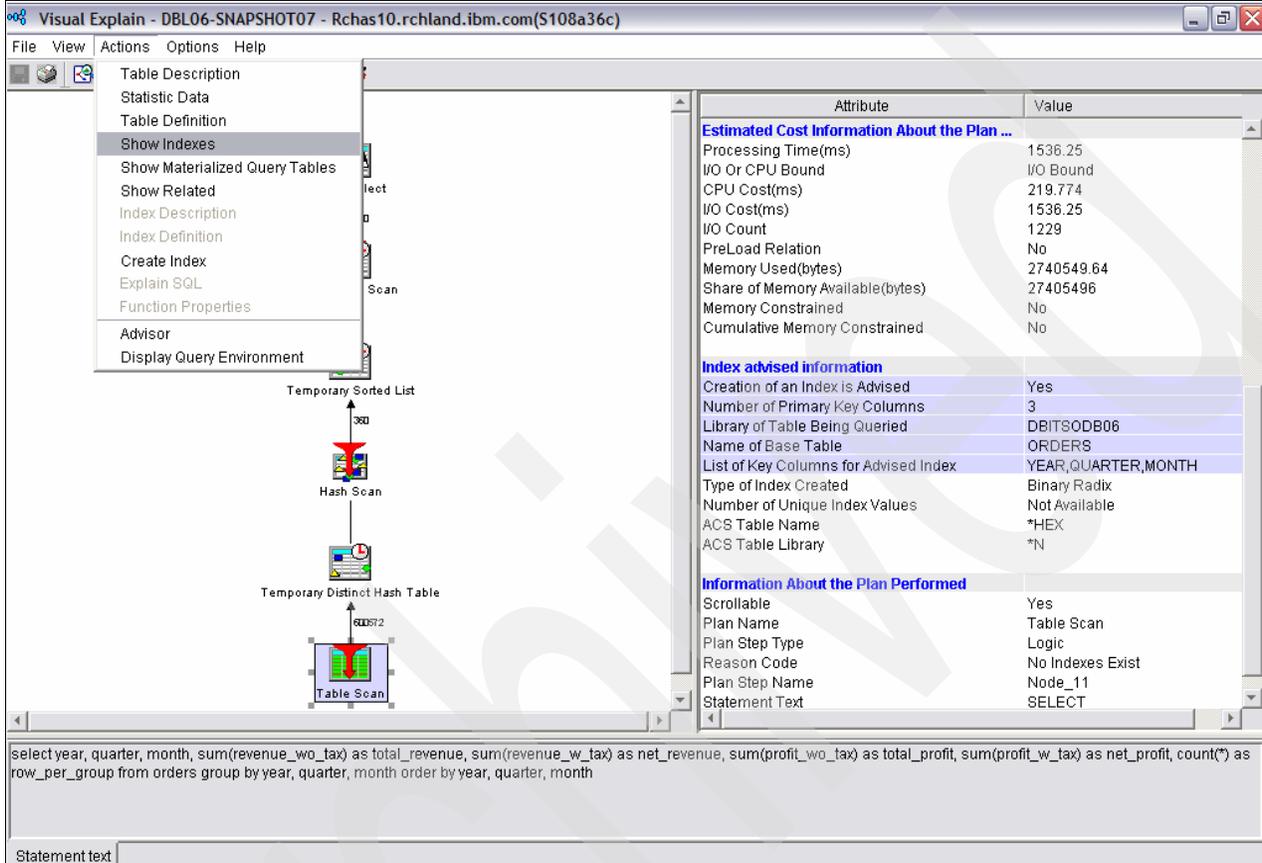


Figure 9-22 Action options in Visual Explain

Show index

From Figure 9-22, when you select **Actions** → **Show Indexes**, an empty window as shown in Figure 9-23 is presented since the Orders table contains no index in our Table Scan example.

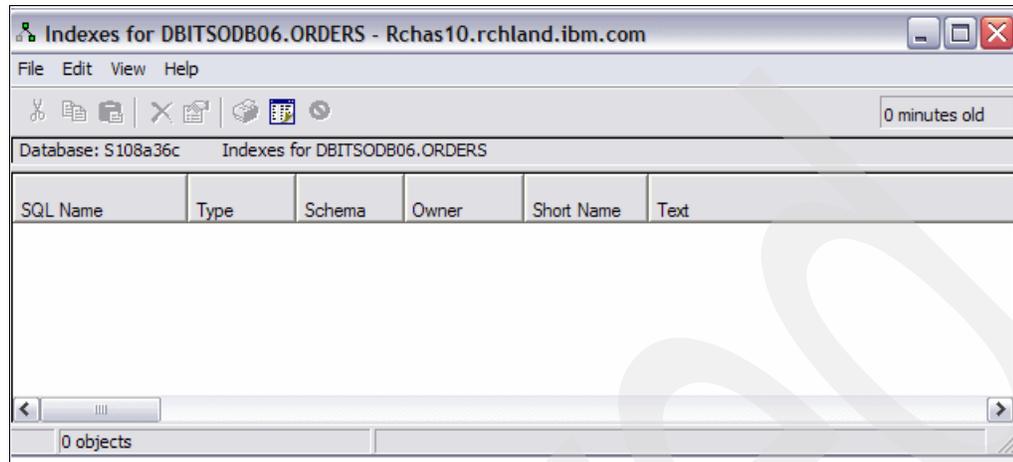


Figure 9-23 Show Index

Create Index

When you select **Actions** → **Create Index**, you are presented with a New Index window as shown in Figure 9-24.

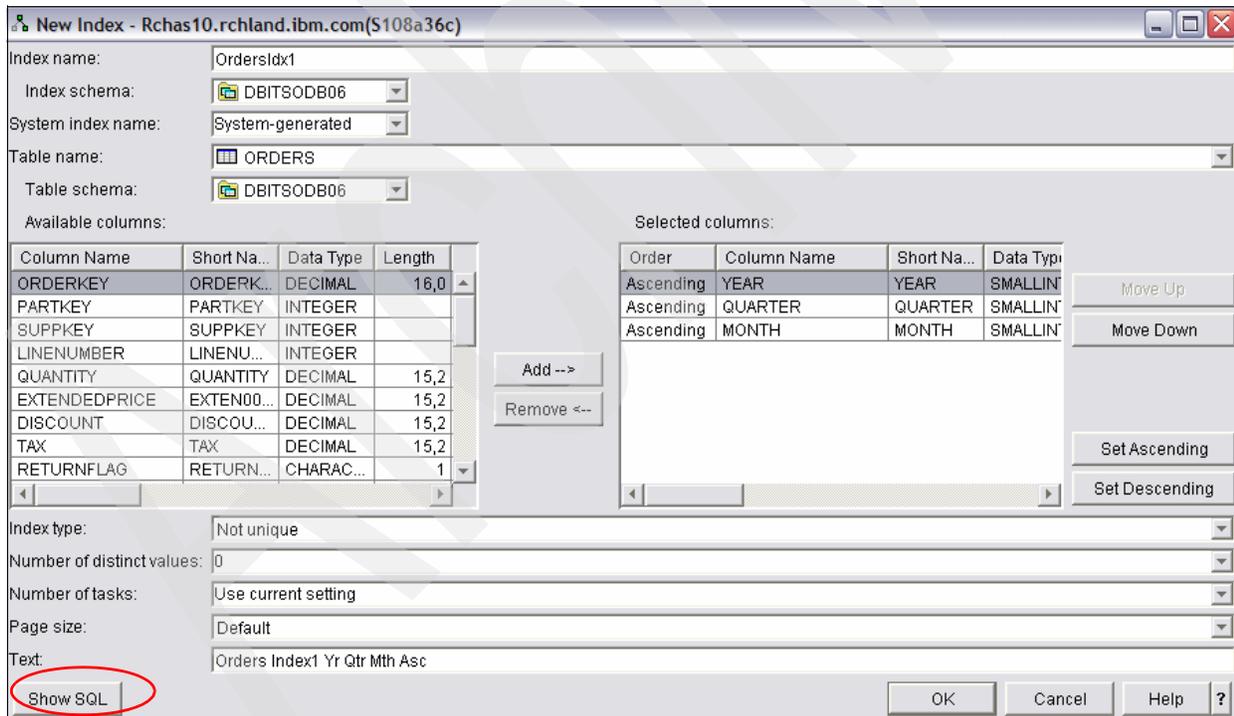


Figure 9-24 New Index

Here are some of the fields you have to fill into the New Index window:

- ▶ **Index name:** The SQL name for the object and must be unique in the schema.
- ▶ **Index schema:** Names the schema in which the index resides.

- ▶ **Table name:** Select the name of the table that this index references.
- ▶ **Selected columns:** You can use Move up or Down to order the column sequence. Additionally, you also can use Set Ascending or Descending to order the priority of the columns. This are the columns that make up the keys of the index.
- ▶ **Index type:** Specify the types of index. The types are:
 - **Not unique:** The table can contain multiple rows with the same value of the index key.
 - **Encoded Vector (Not unique):** The resulting index is an encoded vector index. Encoded vector indexes (EVIs) keep track of the distinct values that can be found in the key columns of a table. EVIs can improve data warehouse performance queries as well as business applications queries. An EVI cannot be used to ensure any expected ordering of records and cannot be used for positioning of an open data path.
 - **Unique:** Prevents the table from containing two or more rows with the same value of the index key.
 - **Unique where not null:** Prevents the table from containing two or more rows with the same non-null value of the index key. Multiple rows with null values are allowed.
- ▶ **Number of distinct values:** Specifies the number of distinct key values expected for this index. The entry size of the vector of an EVI has a direct affect on the size and performance of the index. An index with a 4 byte vector entry size consumes four times as much space as the same index with a 1 byte vector. Also, the scanning time during query processing of a 4 byte vector entry size EVI may be 4 times that of the same EVI with a 1 byte vector entry size. The possible values are:
 - **Not specified:** The number of distinct key values is not specified.
 - **1 to 255:** Number between 1 and 255 for an EVI allocates a vector with a 1 byte entry for each row in the table. An encoded vector index with a 1 byte vector entry length restricts the maximum number of distinct key values that the index can contain to 255.
 - **256 to 65355:** This causes the EVI to allocate a vector with a 2 byte entry for each row in the table, and restricts the maximum number of distinct key values that the index can contain to 65355.
 - **Greater than 65356:** This causes the EVI to allocate a vector with a 4 byte entry for each row in the table, and restricts the maximum number of distinct key values that the index can contain to 2147483647.

Note: For non-EVIs, specifies the estimated number of rows the table will contain. This value does not have to be exact, but may be helpful to the query optimizer.

- ▶ **Page size:** The page size is used by the system to determine the size of each page of the index. This page size is the amount of bytes of the index that can be moved into the job's storage pool from the auxiliary storage for a page fault. Possible values are:
 - Default - page size is determined by the total length of the key, or keys. This is the value that we recommend.
 - 8 KB
 - 16 KB
 - 32 KB
 - 64 KB
 - 128 KB
 - 256 KB
 - 512 KB

When you are satisfied with the entries made, click **OK** to create the index or you can also click **Show SQL** to view the SQL statement used to build the index.

Advisor

From Figure 9-22, when you select **Actions** → **Advisor**, you are presented with a window as shown in Figure 9-25.

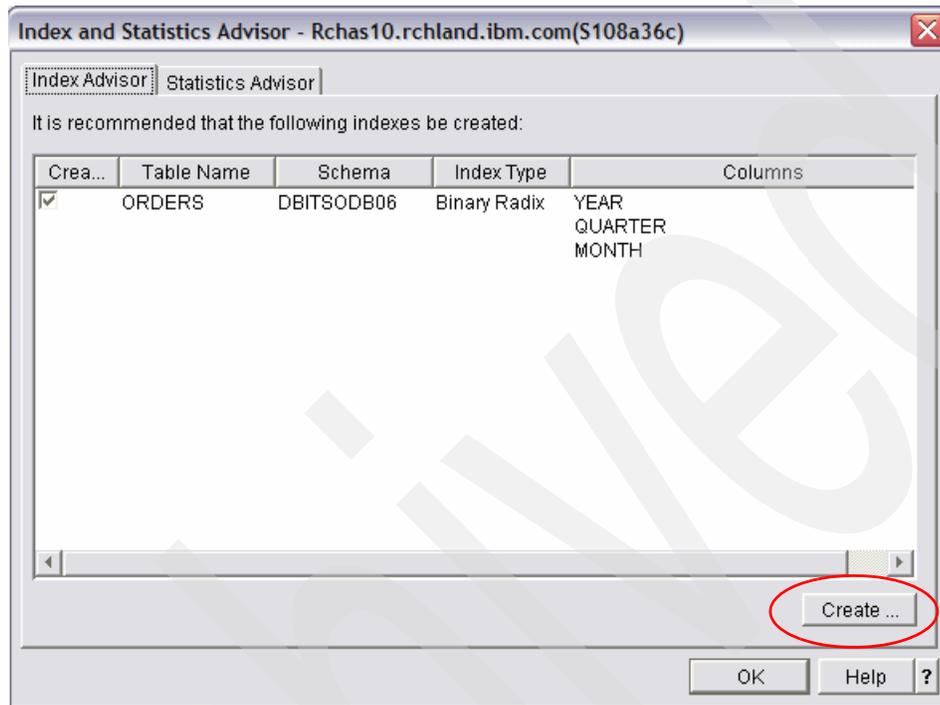


Figure 9-25 Index Advisor

You can also use the **Create** button to create the index advised on-the-fly.

9.5.5 Access to Index Advised information from the Debug messages

Query optimizer debug messages issue informational messages to the joblog about the implementation of a query. These messages explain what happened during the query optimization process. For example, you can learn:

- ▶ Why an index was or was not used
- ▶ Why a temporary result was required
- ▶ Whether joins and blocking are used
- ▶ What type of index was advised by the optimizer
- ▶ Status of the job's queries
- ▶ Indexes used
- ▶ Status of the cursor

There are four methods of directing the system to generate debug messages while executing your SQL statements:

- ▶ Selecting the option in Run SQL Scripts interface of iSeries Navigator
- ▶ Using the Start Debug (STRDBG) CL command
- ▶ Setting the QAQQINI table parameter
- ▶ Using Visual Explain

By looking at the first level debug messages and reviewing second level text behind the debug messages, you can identify changes that might improve the performance of the query such as the index advised.

Note: The debug messages are no longer being enhanced for queries that go through SQE. Thus, Visual Explain is a better tool to review optimizer's recommendations such as index advised, compared to Joblog or Debug messages.

We use this set of SQL statements in Example 9-2 for all of the four methods and observe the outcome.

Example 9-2 SELECT statements that causes index advised

```
SELECT YEAR, MONTH, RETURNFLAG, PARTKEY, QUANTITY  
FROM ORDERS  
WHERE YEAR = 2002 AND MONTH = 6 AND RETURNFLAG = 'R';
```

Selecting the option in Run SQL Scripts interface of iSeries Navigator

Run SQL Script is found in the Database function. Now perform the following steps:

1. Click **Run SQL Scripts** as shown in Figure 9-26 to activate the Run SQL Script window.

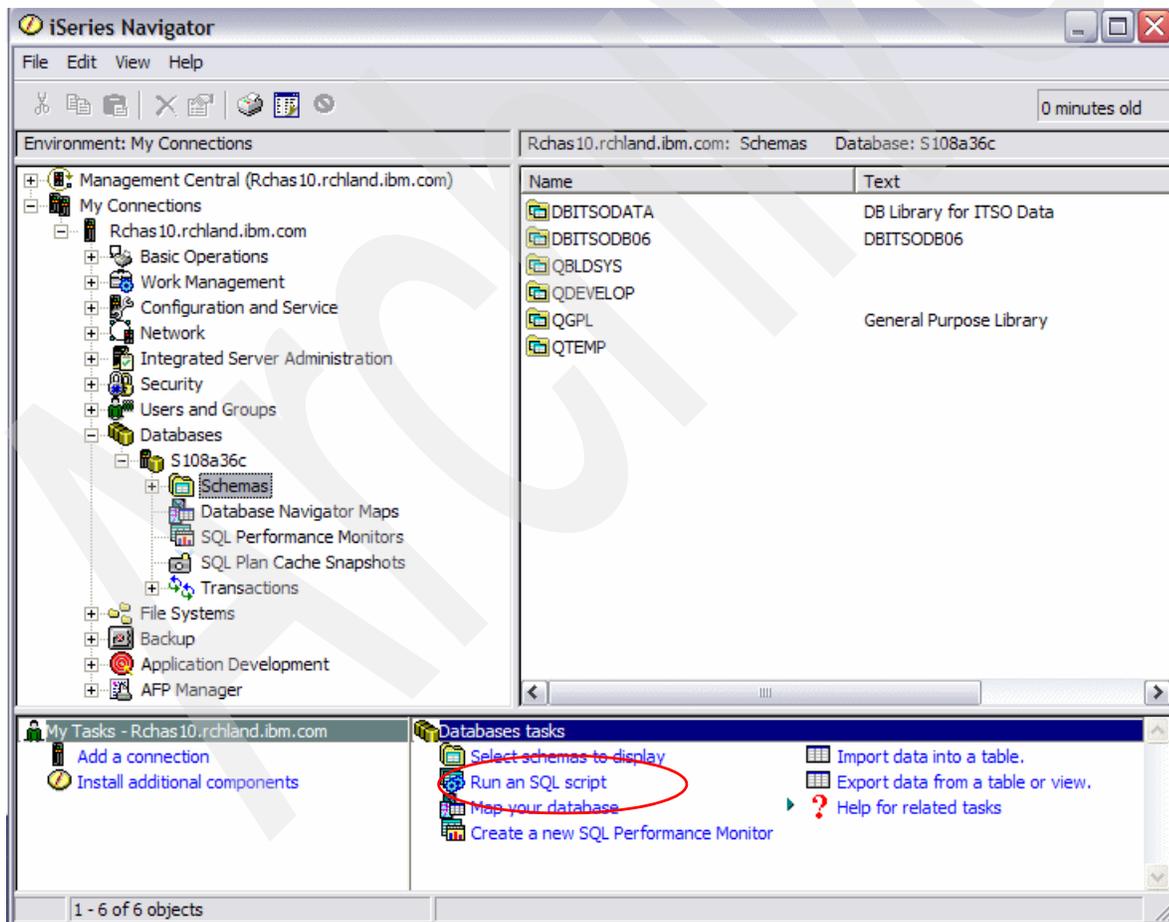


Figure 9-26 Run SQL Scripts of Database function

- To view debug messages in Run SQL Scripts, from the **Options** menu, select **Include Debug Messages in Job Log** as shown in Figure 9-27.

Click the Run All icon to execute both the SQL statements.

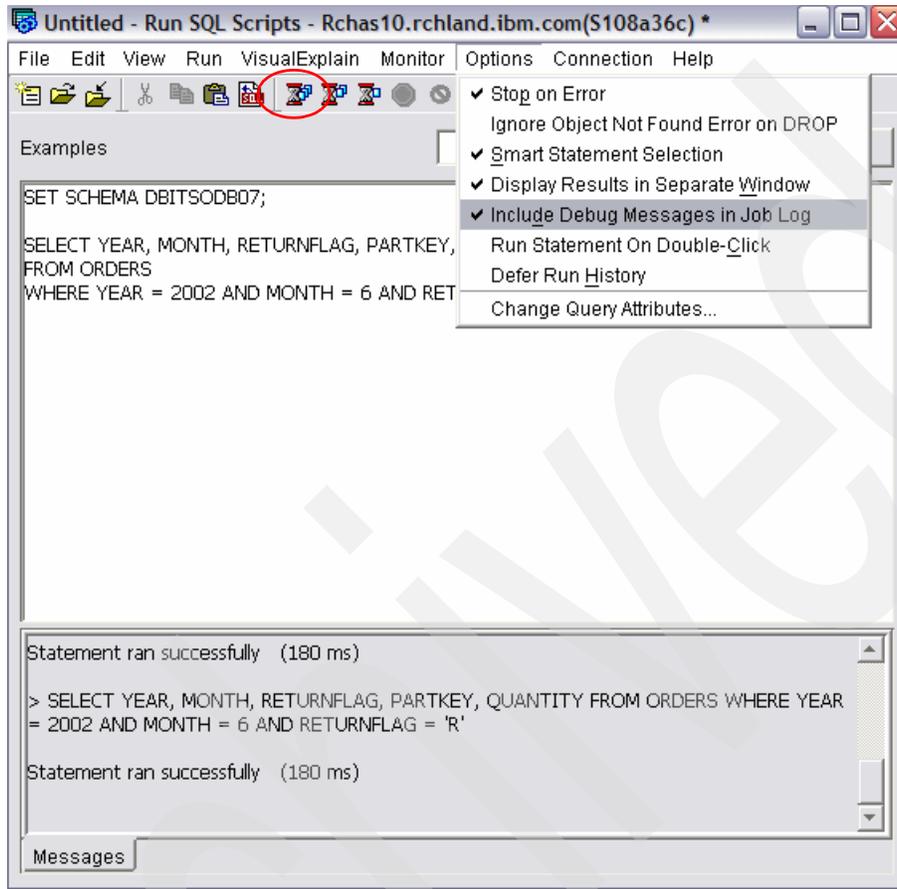


Figure 9-27 Include Debug Messages in Job Log

3. As shown in Figure 9-28, The statements ran successfully. Now select **Job Log** from the **View** menu.

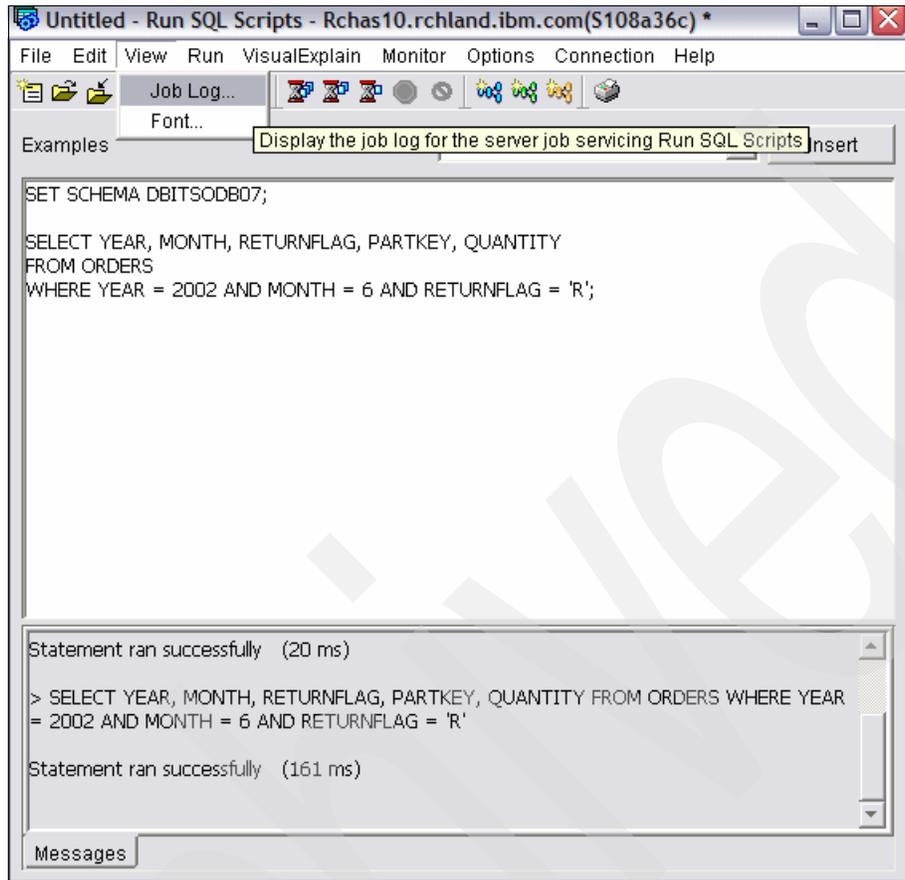


Figure 9-28 Run All and View Job Log

The Job Log window appears as shown in Figure 9-29. Observe optimizer joblogs that appear in the Debug message section:

```
CPI434A **** Starting optimizer debug message for query.  
CPI4339 Query options retrieved file QAQQINI in library QUSRSYS.  
CPI4329 Arrival sequence access was used for file ORDERS.  
CPI432F Access path suggestion for file ORDERS.  
CPI434B **** Ending debug message for query.
```

- As shown in Figure 9-29, highlight and right-click the CPI432F Access path suggestion for the ORDERS line and select **Properties**.

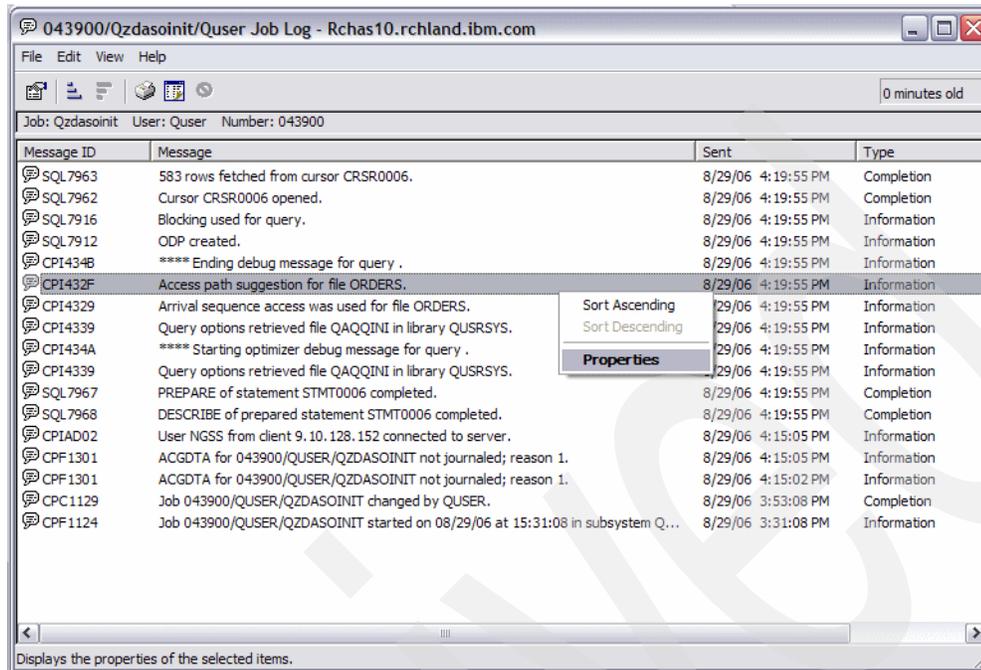


Figure 9-29 Job Log through Run SQL Script window

The CPI432F properties window is shown in Figure 9-30. Observe the Cause explained and the Recovery action recommended on the index.

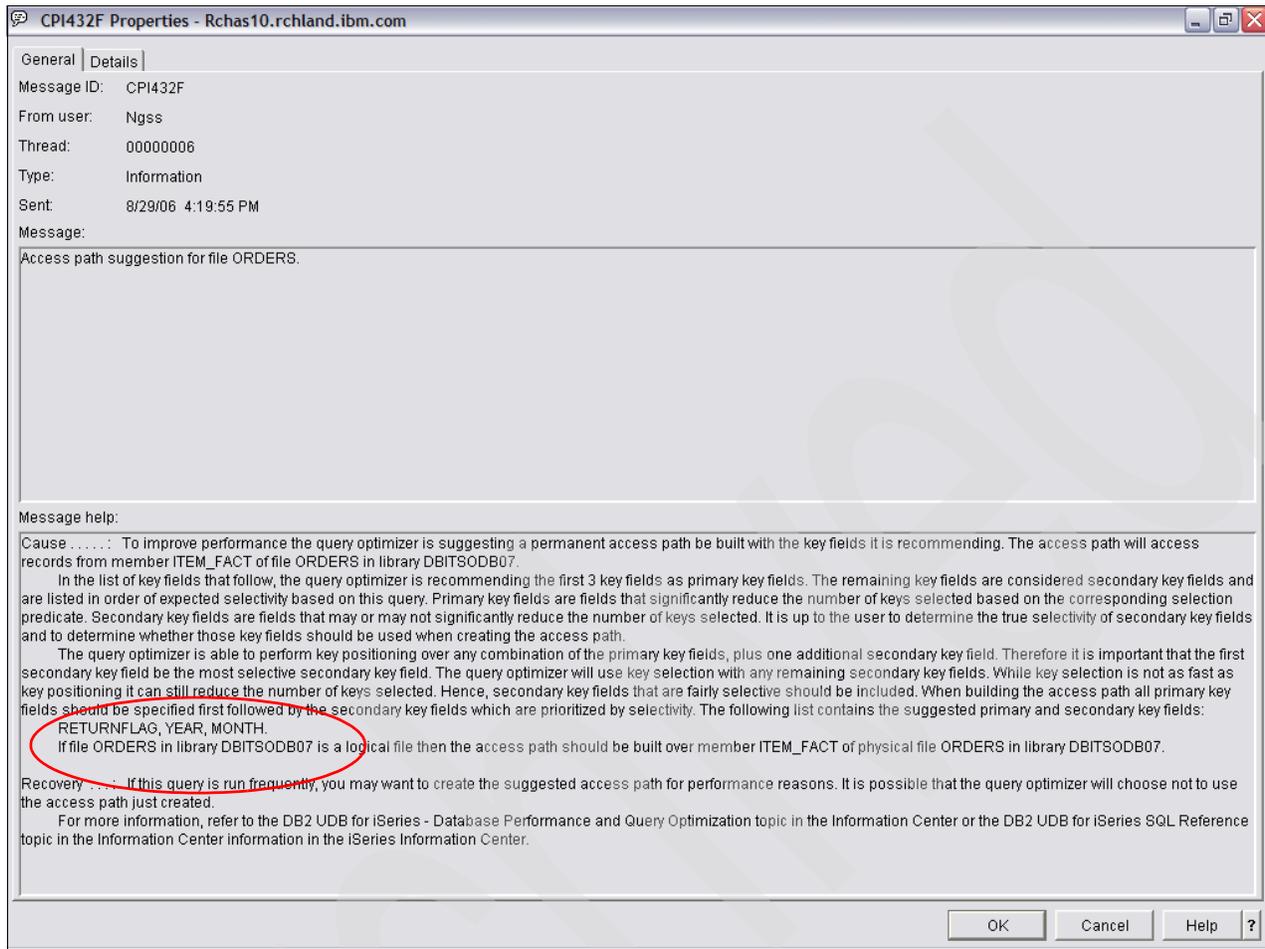


Figure 9-30 CPI4329 Properties

Setting the QAQQINI table parameter

You can also set the QRYOPLIB parameter on the Change Query Attributes (CHGQRYA) command to a user schema where the QAQQINI table exists. Set the parameter on the QAQQINI table to MESSAGES_DEBUG, and set the value to *YES. This option places query optimization information in the job log. Changes made to the QAQQINI table are effective immediately and will affect all users and queries that use this table. Once you change the MESSAGES_DEBUG parameter, all queries that use this QAQQINI table write debug messages to their respective joblogs. Perform the following steps:

1. Create a duplicate of the default QAQQINI file in QUSRSYS into your application schema, for example DBITSODB0, using the following command:

```
CRTDUPOBJ OBJ(QAQQINI) FROMLIB(QUSRSYS) OBJTYPE(*FILE) TOLIB(DBITSODB07)
DATA(*YES)
```

2. Proceed to make changes on the query attribute. In Run SQL Script center select **Options** → **Change Query Attributes** as shown in Figure 9-31.

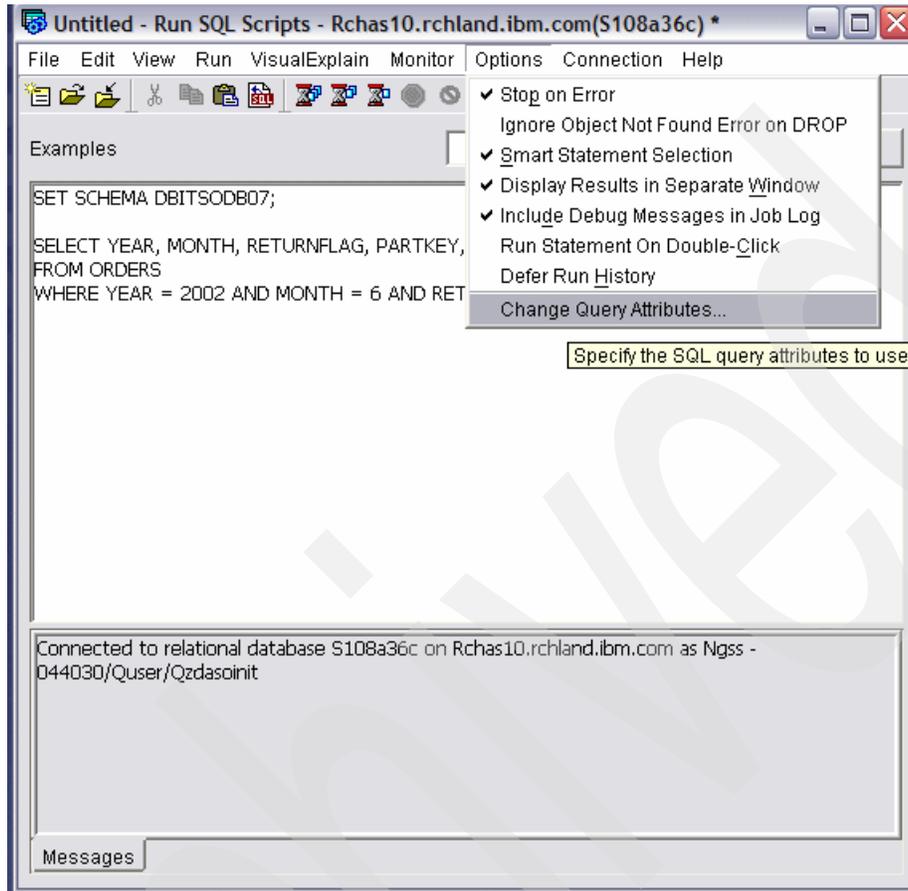


Figure 9-31 Change Query Attributes option in Run SQL Script center

3. Check the Use the query options file located in the following schema box and select your schema name (for example DBITSODB07). Then click **Edit Options** as shown in Figure 9-32.

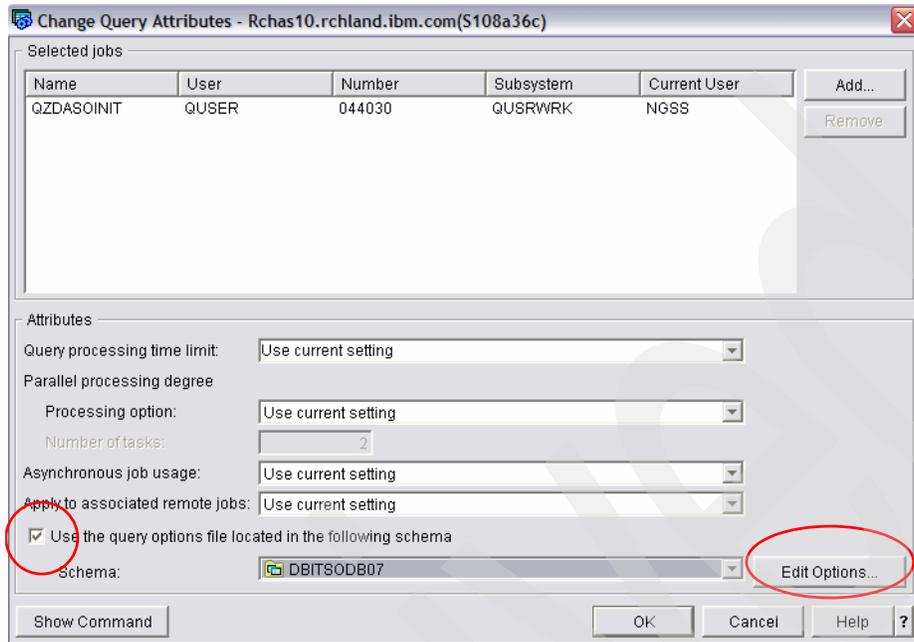


Figure 9-32 Change Query Attribute window - select the QAQQINI file in your schema

4. Update the MESSAGE_DEBUG value from *DEFAULT to *YES as shown in Figure 9-33.

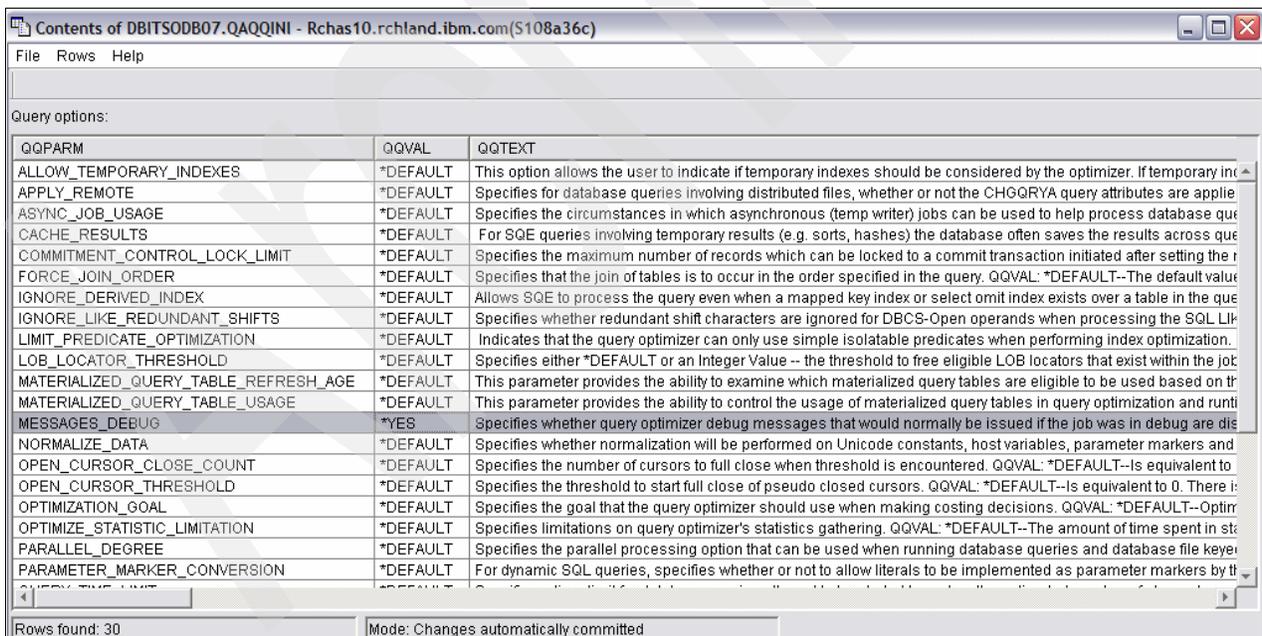


Figure 9-33 DEBUG_MESSAGE setting in QAQQINI file

5. Close the file and click **OK** in the Change Query Attribute window when done. This enables Debug messages to be logged into joblog.

Now you can execute the SQL statements and review the debug messages in joblog for the index advised recommendation. The joblog review steps are similar to what is described in “Selecting the option in Run SQL Scripts interface of iSeries Navigator” on page 347.

Using Visual Explain

In Visual Explain, debug messages are always available. You do not need to turn them on or off. Debug messages appear in the lower portion of the window. You can view detailed messages by double-clicking on a message. This is done by performing the following steps:

1. As shown in Figure 9-34, highlight the same set of scripts in the Run SQL Script window and select the Run and Visual Explain icon.

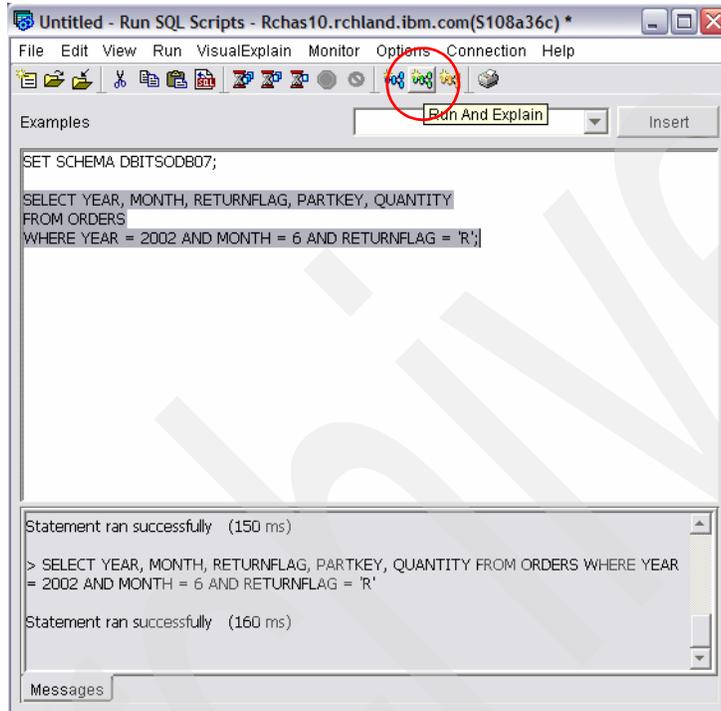


Figure 9-34 Run and Explain

2. You are presented with Visual Explain window as shown in Figure 9-35. Click the **Optimizer messages** button to view the messages presented by Optimizer (it is identical to other debug messages shown in “Selecting the option in Run SQL Scripts interface of iSeries Navigator” on page 347 and “Setting the QAQQINI table parameter” on page 351). You can also select **View** → **Highlight Index Advised**, the Index Advised section (Table scan in this case) is automatically highlighted. The index information is presented on the right pane. You can now examine the information in detail before deciding to create the index advised by optimizer.

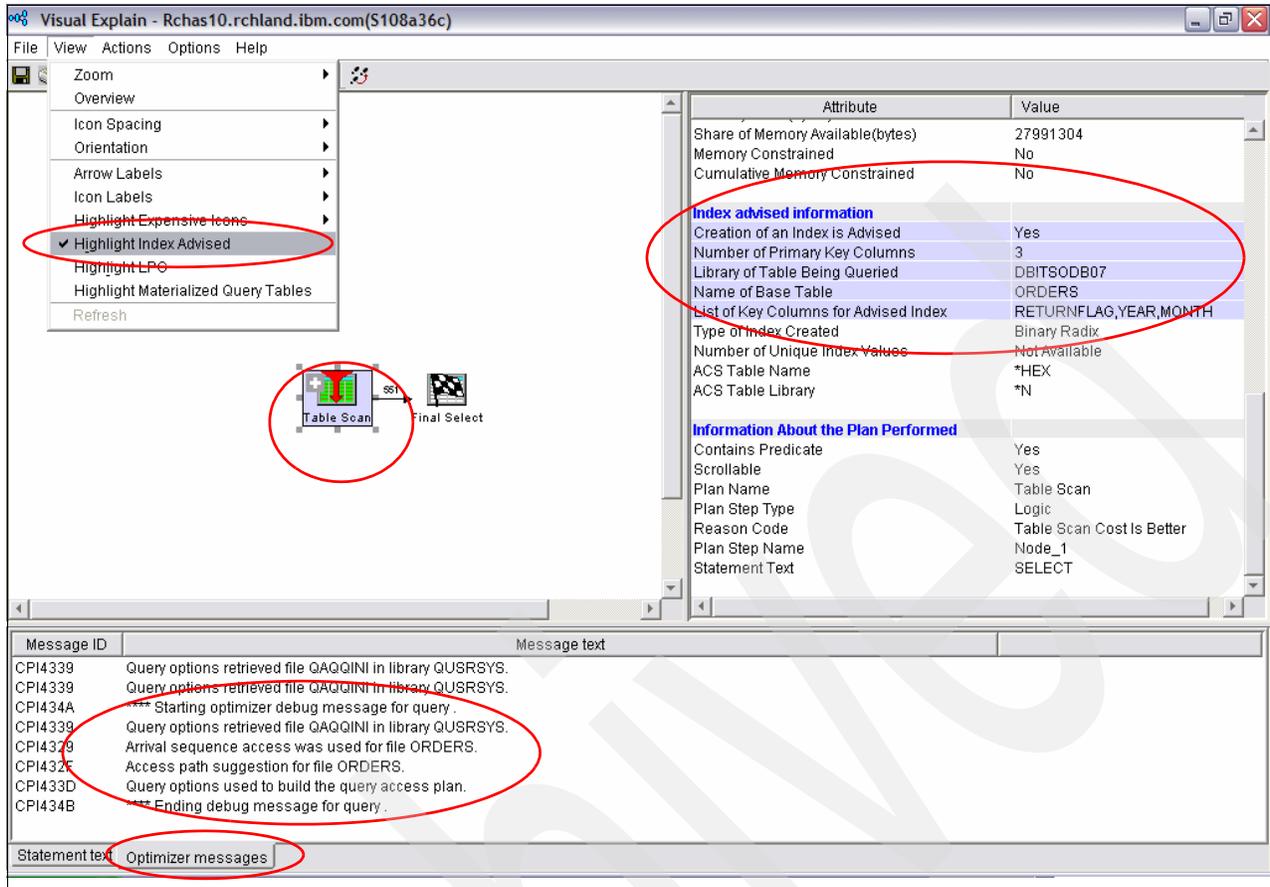


Figure 9-35 Optimizer Messages in the Visual Explain window

9.6 Temporary Indexes

A temporary index is a temporary object that allows the optimizer to create and use a radix index for a specific query. The temporary index has all of the same attributes and benefits as a radix index that is created by a user through the CREATE INDEX SQL statement or Create Logical File (CRTLF) CL command. Additionally, the temporary index is optimized for use by the optimizer to satisfy a specific query request. This includes setting the logical page size and applying any selection to the creation to speed up the use of the temporary index after it has been created.

The temporary index can be used to satisfy a variety of query requests:

- ▶ Ordering
- ▶ Grouping/Distinct
- ▶ Joins
- ▶ Record selection

In this section, we discuss CQE - Temporary Indexes and SQE - Temporary Indexes in more detail.

9.6.1 CQE - Temporary Indexes

The index advised in CQE are basic advise and it covers Radix index only. The advice is based on table scan and local selection columns. The temporary index creation information provides you with insight to fine-tune your SQL performance. Further more, Visual Explain always try and tie pieces together to advise you with a good index and presented to you in a graphical picture as shown in Figure 9-36.

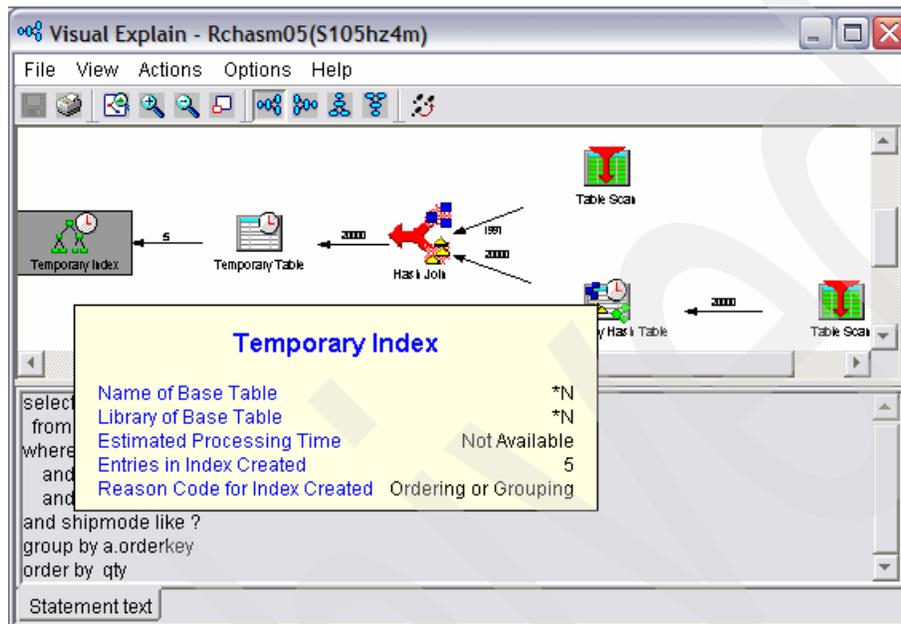


Figure 9-36 CQE Temporary Index in Visual Explain

Note: CQE Temporary indexes are not reused and cannot not be shared across jobs. They also cannot be shared among SQL statements within the same job, and they are deleted after the SQL statement is completed.

9.6.2 SQE - Temporary Indexes

The index advised in SQE is of robust advise. The support includes Radix and EVI indexes and is based on all parts of the query. Multiple indexes can be advised for the same query and the temporary index creation (autonomic indexes) provides you with good insight to optimizing your SQL queries.

Note: SQE Automatic temporary index creation is a new feature in V5R4. The temporary indexes are reused and can be shared across jobs and queries.

Maintained Temporary Indexes (MTIs)

Maintained Temporary Indexes (MTIs) are created and used by the SQE optimizer in V5R4, under situations where a matching permanent index does not exist. The SQE optimizer watches query requests and learns. It has autonomic capability to create an index based on:

- ▶ Observing “n” number of queries
- ▶ Assessing the benefits of creating and using an index
- ▶ Optimizer’s own index advice

The MTIs indexes are:

- ▶ Temporary
- ▶ Maintained while any cursor is open - maintenance is delayed when all cursors are closed
- ▶ Shared between queries and jobs
- ▶ Primarily created on smaller tables
- ▶ Not used for statistics

This new feature of MTIs in V5R4 allows additional queries to use SQE (for example. Sensitive cursors), and it also allows DB2 UDB for iSeries to “tune” itself.

Note: For more information about Sensitive Cursor, you can refer to Chapter 11, “Environmental settings that affect SQL performance” on page 379.

The MTI existence and usage information has been externalized to the customer by way of the iSeries Navigator Index Advisor and Show Indexes facilities.

Viewing the MTI information in the context of index advice can help explain performance fluctuations. In one common scenario, performance is shown to be degraded after an IPL, but improves over time as queries are executed. MTIs do not persist across an IPL. By accessing the Show Indexes detail for the table, you can confirm that MTIs exist. Further, the index advisor information details how often a specific MTI has been created and used and how recently it was used.

When paired with other index advice environmental information such as Times Advised and Average Query Estimate, a better index strategy can be determined. If an MTI becomes a crucial part of an index strategy, it might be the perfect time to change the permanent indexes to avoid using the MTI altogether. This can be accomplished by simply launching the Create Index dialog from the index advisor. The optimizer will recognize the existence of a matching permanent index and discard the MTI. This change will be reflected in the index advisor table via the MTI Last Used time stamp.

After making any change to an index strategy, the usage information within the permanent indexes can be reset. Resetting the usage statistics makes it easier to evaluate the value of the current set of permanent indexes. Because an implied maintenance cost is associated with each index, having as few permanent indexes as possible is preferable.

Use this command to reset Index statistics:

```
CHGOBJD OBJ(schema/index) OBJTYPE(*FILE) USECOUNT(*RESET)
```

Additionally, after an index strategy is changed, it might be useful to clear out the existing index advice for that schema or table. There are many ways to clear the advice, both within iSeries Navigator and from Run SQL Scripts. After clearing the advice and index statistics, the index strategy is ready to be evaluated. These techniques work in a performance analysis environment, on a development system or on production machines.

Interfaces to Maintained Temporary Indexes (MTIs) information

To access to MTIs information, perform the following steps:

1. From the iSeries Navigator Schemas folder view, right-click the schema of choice and select **Index Advisor** → **Index Advisor** to launch the Index Advisor (Figure 9-37).

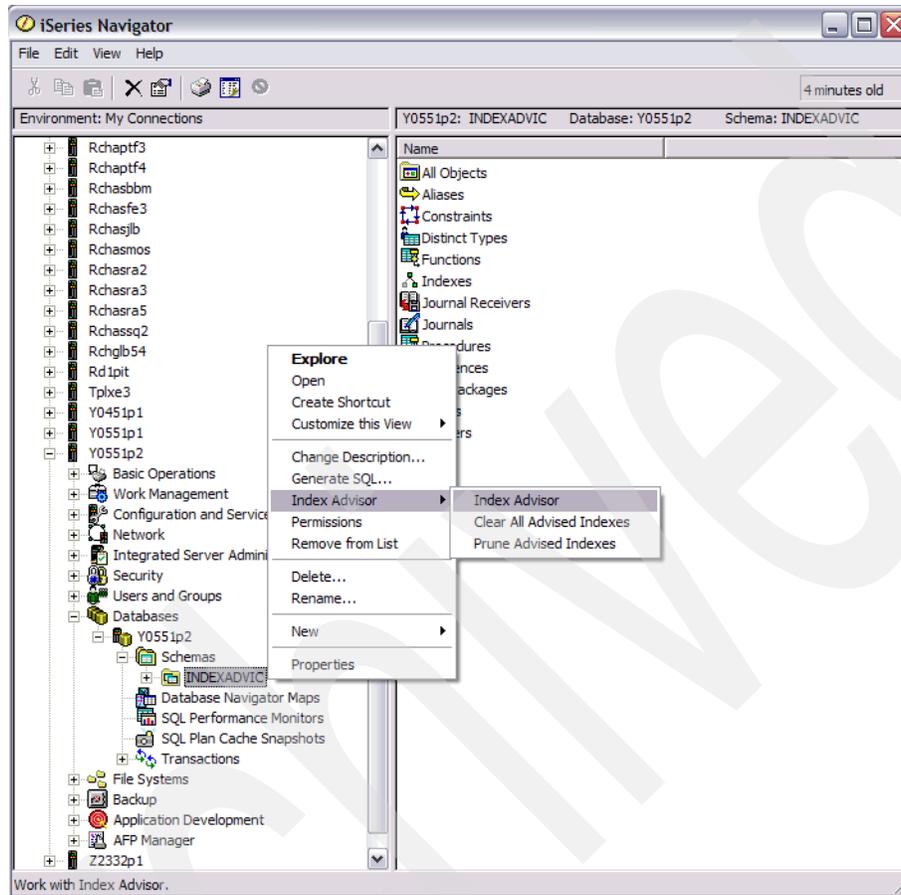


Figure 9-37 Path to Index Advisor at Schema level

The Index Advisor window appears as shown in Figure 9-38. The MTIs information appears on the far right side. The table can be resorted by selecting any column header. The table shown here is sorted by MTI USED.

Index Advisor - Y0551p2
 Database: Y0551p2 Advised Indexes for INDEXADVIC
 1 minutes old

Table for Which Index was Advised	Schema	Short Name	Keys Advised	Leading Keys Order Independent	Times Advised	MTI USED	MTI CREATED	MTI LAST USED
EMPLOYEE	INDEXADVIC	EMPLOYEE	LASTNAME		9 F	6	2	7/27/06 4:57:56 PM
SALES	INDEXADVIC	SALES	SALES_PERSON		12 F	4	2	7/27/06 4:57:56 PM
EMPLOYEE	INDEXADVIC	EMPLOYEE	JOB, WORKDEPT	JOB	3 F	2	1	7/27/06 4:57:56 PM
EMPLOYEE	INDEXADVIC	EMPLOYEE	LASTNAME, FIRSTNAME, SALARY, BONUS, COMM		2 C	0	0	
SALES	INDEXADVIC	SALES	SALES_DATE, REGION, SALES_PERSON	SALES_DATE	1 F	0	0	
SALES	INDEXADVIC	SALES	SALES_DATE		1 F	0	0	
EMPLOYEE	INDEXADVIC	EMPLOYEE	WORKDEPT, LASTNAME, FIRSTNAME, SALARY, B...	WORKDEPT	2 F	0	0	
SALES	INDEXADVIC	SALES	SALES_PERSON, SALES_DATE		1 C	0	0	
SALES	INDEXADVIC	SALES	SALES_DATE, SALES_PERSON, REGION		3 C	0	0	
SALES	INDEXADVIC	SALES	REGION, SALES_PERSON		3 C	0	0	
SALES	INDEXADVIC	SALES	SALES_DATE, REGION, SALES_PERSON	SALES_DATE, R...	1 F	0	0	
EMPLOYEE	INDEXADVIC	EMPLOYEE	WORKDEPT, LASTNAME	WORKDEPT	4 F	0	0	

1 - 12 of 12 objects

Figure 9-38 Index Advisor window with MTIs information

Tip: The row of index advised that registers a high number of MTI Used value, it makes a great candidate for permanent index to be created. This information serves as an index advised information.

- If you are interested in the indexes advise of a particular table, right-click the row of index advice and select **Table** → **Show Indexes** as shown in Figure 9-39.

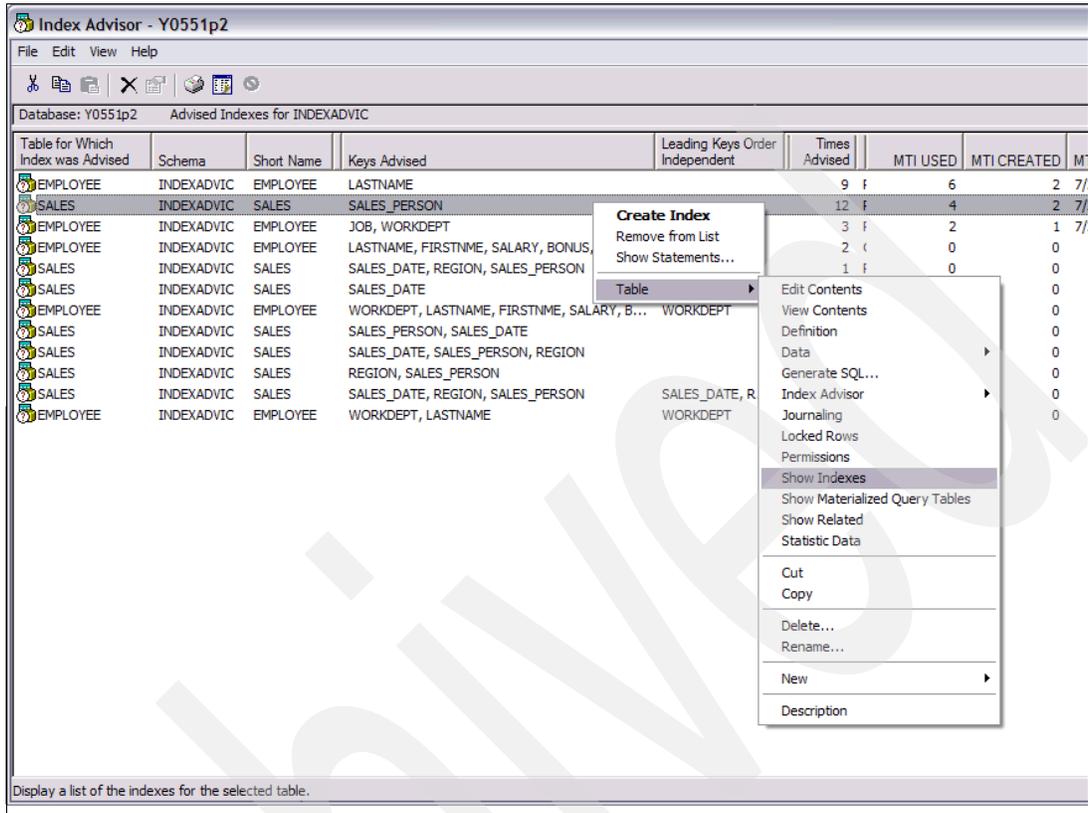


Figure 9-39 Table Show Indexes from the Index Advisor window

- As shown in Figure 9-40, the Show Indexes window is launched. You can use the information presented in the window for easy comparison between existing permanent indexes and MTIs over this table to the indexes being advised.

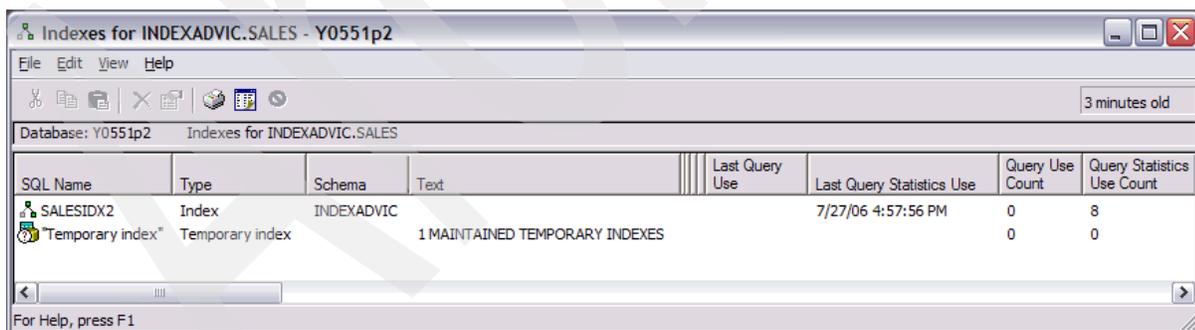


Figure 9-40 MTI details in Show Indexes window

Note:

- ▶ MTIs detail in Show Indexes window is summary information, for specific information about MTIs, refer to columns MTI USED, MTI CREATED and MTI LAST USED in QSYS2/SYSIXADV system table in 9.2, “System Wide Index Advised Table”.
- ▶ PTF information:
 - The Index Advisor MTI enabling Server PTFs were included in SF99504: 540 DB2 UDB for iSeries Group Level #4.
 - The iSeries Navigator enabling Client PTFs ship after September 1, 2006.
- ▶ Related articles:
 - DB2 for i5/OS Redefines On Demand for Indexing found on the Web at:
<http://www.ibmssystemsmag.com/ME2/Audiences/dirmod.asp?sid=&nm=&type=Publishing&mod=Publications%3A%3AArticle&mid=8F3A7027421841978F18BE895F87F791&tier=4&id=00D363297CDB4F54BAF9FA42E9286218&AudID=1E8FEE745A284521B6CFB3FD70B49099>
 - The Optimizer Takes Its Own Advice found on the Web at:
<http://www.ibmssystemsmag.com/ME2/Audiences/dirmod.asp?sid=&nm=&type=Publishing&mod=Publications%3A%3AArticle&mid=8F3A7027421841978F18BE895F87F791&AudID=1E8FEE745A284521B6CFB3FD70B49099&tier=4&id=15C863E9852B45F6960150532D43F2A5>

Archived

SQL performance analysis: a methodology

In this, the final chapter of Part 2, “Gathering, analyzing, and querying database performance data” on page 31, we will tie everything together.

In the previous chapters of this SQL Performance Diagnosis book, we have discussed the new features and enhancements that became available with DB2 for i5/OS for V5R4. The Analysis Overview dashboard is one of the new tools that provides a graphical interface to view and analyze SQL database performance. Other new tools include the SQE Plan Cache analysis tools and the system-wide Index Advisor. Visual Explain has received a number of enhancements as well. In order to allow a wide range of IT professionals to use the new database tools and functions, the new tools have been seamlessly integrated into the iSeries Navigator GUI interface. All of these tools have been linked together in some fashion in order to provide you with an easy way to view and analyze all of the information required to identify and diagnose an SQL Performance problem.

In the preceding chapters, we have introduced each of the new tools or interfaces. We also have described how to use them, how to identify an SQL problem, how to analyze it and finally how to view some recommendations on ways you might resolve the problems. For an experienced SQL performance analyst, this may be all that is needed to make an analyst productive in an i5/OS SQL environment. The analyst modifies the methodology that they have used in the past to allow the use of the new tools and continues their job as usual.

But for most of us who do not perform SQL Performance analysis on a daily basis, we need a roadmap or methodology on how to handle a possible SQL performance problem. Therefore, in this chapter, we will suggest some steps to use to follow the problem to its resolution through the use of the analysis tools.

10.1 Performance methodology

First, let us take a step back and review the entire performance methodology process. In reality, performance problem diagnosis (performance troubleshooting) is a small part of a larger methodology which is meant to keep the problem from occurring. Prevention is the best medicine. An overall performance methodology is made up of five separate parts:

- ▶ **Planning for performance and scalability:** Document the projected system topology, database design, the projected workload, and define measurable performance and scalability objectives. Perform initial capacity planning and sizing estimates.
- ▶ **Making solution design choices and performance trade-offs:** Understand topology choices (types of client, SQL interfaces, and so on), feature choices (such as, MQT, Index strategies, procedures/triggers/functions, HLL) and their performance implications.
- ▶ **Performance tuning:** Plan for an initial tuning period to maximize confidence and reduce risk, using the tools and techniques from the monitoring and troubleshooting phases to meet the objectives set in the planning phase. Focus on effective tuning of SQL and its environment using management of system resources: CPU, memory, disk I/O, and network bandwidth.
- ▶ **Monitoring and maintaining performance:** Maintain a performance profile of the workload metrics and resource utilizations on the database and servers. Monitor over time to observe trends before they become a problem.
- ▶ **Performance troubleshooting:** When performance issues arise, use a disciplined and organized approach to solve the problem using the performance tools, data, and performance tuning guidelines.

This book is written to address the last item in the list, performance troubleshooting. Since we have reviewed the exciting new tools and the interfaces to SQL performance feedback, the only thing that we need is a basic blueprint on how to use them when we encounter an SQL performance problem. Let us now focus only on the last item in the list above, since this book is all about the use of the tools and enhancements in V5R4 of i5/OS which enable effective performance troubleshooting and analysis.

10.2 Performance troubleshooting

This is the part of the performance methodology that guides us on how to identify if there is a problem, how to quantify the problem, and which analysis to use to discover resolutions to the problems.

When attacking a performance problem, we take the following steps in order to progress from initial problem discovery to resolution.

- ▶ Problem source determination
- ▶ Performance data capture
- ▶ Performance analysis process

10.2.1 Problem source determination

In this part of the analysis methodology, we take a problem report record and ask specific questions that we will use later to narrow down the scope or source of the problem. This step is required in order to later determine what tool to use to enter the performance analysis process. The types of questions that should be asked include those discussed in Chapter 1,

“Determining whether you have an SQL performance problem” on page 3. Further examples are listed below.

- ▶ Is it a single user or system-wide?
- ▶ What are the symptoms?
- ▶ What do I need to capture to identify the source of the problems?
- ▶ What can be done prior to starting monitors?
- ▶ Which tools will give the best picture of the problem?
- ▶ Which feedback mechanism may have captured this problem?

Identify resources that affect performance

During source problem determination it is essential that system-wide and environmental causes are eliminated prior to continuing the analysis process. This section identifies the hardware, software and environmental attributes that should be checked prior to examining SQL performance attributes.

- ▶ Hardware
 - CPU
 - Memory
 - Disk
 - Network
- ▶ Software
 - System (for example, OS)
 - Middleware
 - Application
- ▶ Environmental
 - Steady state
 - Constantly changing

10.2.2 Performance data capture

Once identification of a problem has occurred, we should ensure that the problem is SQL-related by eliminating system configuration, operational, and environmental causes. This might be done ahead of time by a system performance process, which monitors the state of the overall workload, capacity, and operation of the system as a whole.

If there is currently no process for monitoring the system's performance, then we would have to perform system tuning. Once that is done and eliminated from being the cause of our SQL problem, we can turn to evaluating SQL performance tools to narrow the symptoms of the problem.

The following questions are the last to be answered before selecting the SQL Performance tool to begin the analysis of the problem.

- ▶ What monitors should be started and what is the scope? System, jobs, users?
- ▶ When should the monitors be started and ended?
- ▶ How should these monitors be started?
- ▶ What level of data detail?

When we answer these questions we are ready to select whether we choose to start the analysis at a detailed or summary level. Also, with the ability to quickly review the types of access plans most recently used for certain SQL statements by viewing the Plan Cache via the new analysis tools, we may not have to capture data via performance monitor to identify where the problem is occurring. Chapter 7, “SQE Plan Cache and SQE Plan Cache Snapshots” on page 237 will give you insight into interrogating recent SQL statement

execution. You will want to first review Chapter 5, “Analyzing SQL performance data using iSeries Navigator” on page 117 to familiarize yourself with the new Analysis Overview interface.

If you find, based on the answers to the questions above that you must capture more information about the SQL during execution, then you will need to read Chapter 3, “Overview of tools to analyze database performance” on page 33 to gain a better understanding of the use and capabilities of each of the SQL Performance tools for DB2 for i5/OS.

10.2.3 Performance analysis process

This section outlines the steps that we should take in approaching the actual analysis of a performance problem once it has been identified.

Analysis process

The following is the list of the process steps:

- ▶ All reported problems should be recorded and entered into the analysis process
- ▶ The approach should be hierarchical: High level (system or user) to low level (statements or instances)
- ▶ The scope should be wide (for example, all operations) to narrow (for example, fetches, and so on)
- ▶ Total runtime to specific time to perform a function or operation
- ▶ Search for deviation from typical runtime or excessively long runtimes for small amounts of work (for example, ten seconds to fetch 100 rows)
- ▶ Appearance of unexplained long inactivity during high resource utilization workloads

Specific problem analysis

Using one of the tools identified in Chapter 3 to narrow down the problem area, we are now ready to focus on the specific problem. The following list is a sample of the types of specific problems an analyst will focus on at this step in the process.

- ▶ An SQL statement in a job by local/remote user takes longer to execute?
- ▶ All jobs are taking longer today
- ▶ Remote users are complaining about certain jobs
- ▶ My daily query was submitted and didn't returned any results
- ▶ Will an index help this query? What kind of index?
- ▶ How can I improve the performance of this query?

10.3 Application of the tools to the methodology

Now let us see how new set of tools fits in to the SQL performance troubleshooting methodology.

In 10.2, “Performance troubleshooting” on page 364, we identified three parts. The steps that we are about to introduce fit into the parts as follows:

- ▶ Problem source determination (Step 1)
- ▶ Performance data capture (Step 2 & 3)
- ▶ Performance analysis process (Step 4-8)

In problem source determination, we attempt to determine as closely as possible the cause of the current SQL performance problem. We first have to eliminate environmental and system

attributes as a possible cause. In Chapter 13, “Using Collection Services data to identify jobs using system resources” on page 423, we discuss Collection Services and its use to monitor and analyze system resources that affect every job that executes on the system. Therefore, this is the place where you should start your problem determination search and ask the question “What effect is the overall system performance having on the identified SQL Problem?”

Step 1: Ensure Collection Services eliminates system resources as a possible cause of the SQL Problem

If IT operations does not collect system statistics on a periodic basis, you will have to perform a one-time analysis of the system by collection data during the time the SQL Problem occurs and analyze the impact of resources on your SQL job or jobs. If you are not familiar with, Collection Services and its data collection and reporting read Chapter 13. For more information, go to Information Center on the Web at:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/rzahx/rzahxcollectdata.cs.htm>

Once we have eliminated system resources and performance as a source of interference with the execution of our SQL jobs and the source of the problem, we can turn our focus to the SQL jobs and generally how the database is handling them.

Step 2: Examine the SQE plan cache

In Chapter 1, we identified the overall components of a database work request. By using Collection Services, we have reduced the possibility that other system components are the cause of our SQL performance problem (for example, communications, system memory or disk constraints, and so on). We are ready to focus on the database component.

In Chapter 3 we learned that the SQE Plan Cache allows quick analysis of plans that were created by SQE. If the SQL was executed by SQE we may be able to quickly drill down into the cache and find the plan for the statement that is having the problem. For example we could be using user filter by longest-running query.

The SQE Plan Cache viewer allows the use of comprehensive filtering to focus in on the time, job, user, objects and so on to reduce the volume of data to analyze to find the SQL problem.

If we suspect we have found the problem in the plan cache we want to make sure we dump this information into a persistent object. For this reason our next step would be to save this information into a Plan Cache snapshot by using the same filtering options. Once we have the Plan Cache snapshot we would proceed to Step 4.

Step 3: If the problem is not apparent from the SQE plan cache, start a detailed performance monitor and recreate the issue

There are a few reasons why the problem SQL may not have been found in the plan cache. One of the possible reasons is that it was executed by CQE. Another possible reason is that it may have already been pruned from the cache by the time you analyzed it. In any case, if a problem is not evident from the analysis of the SQE Plan cache, then recreate the problem while collecting a detailed SQL Performance monitor. Even if the statement of interest is found in the Plan Cache, remember that it is just a snapshot. If we need to capture data over a period of time, for example, multiple executions of the same SQL statement, then a Detailed SQL Performance monitor will be required.

Step 4: Review the items in the Overview Analysis dashboard for any key indicators of a problem

Does anything stick out as being excessive or out of the ordinary, in comparison to the rest of the data captured? Do you have an SQL Performance Monitor containing the same SQL statements from a period of satisfactory performance? If so, run the Compare utility to highlight any differences. Chapter 5 highlights several of the analysis overview areas that you may want to review for problems.

Step 5: Run Summary or Statements reports, or both, for any of the focus areas identified to get to the root SQL statement or statements behind the problem

The drill down capability of the tools allow you to continually narrow your focus and dig deeper into the problem. With the V5R4 tools, there are no “dead end” reports. Details on several of the analysis reports are presented in Section 5.2, “In-depth analysis reports” on page 128. You can also use the system wide Index Advisor, explained in Chapter 9, “Index Advisor” on page 319, at this stage to check for any recommended indexes on the tables in your identified query or queries.

Step 6: Run Visual Explain to review the implementations of any statements found in step 5

Once you have identified any problem statements, you can use Visual Explain to review the details of the specific statement’s implementation. From Visual Explain you can choose to highlight indexes, expensive operations, and so on. Review Chapter 8, “Analyzing database performance data with Visual Explain” on page 275 for the full discussion on the Visual Explain tool.

Step 7: Make modifications to the query, create indexes and so on

Use the Work with Statement option from within the analysis reports to bring your statement into a new SQL Script to test any changes in real time. For example, you could modify the SQL here, or create new indexes.

Step 8: Recreate your application after implementing changes

If you still find a problem, return to Step 2 to gather a new SQL Performance monitor that covers the changes made in Step 7. Remember, SQL Performance analysis is an iterative process. If at first you do not succeed, try, try again! The new toolset makes that easier than ever.

Figure 10-1 shows a flowchart of this suggested methodology.

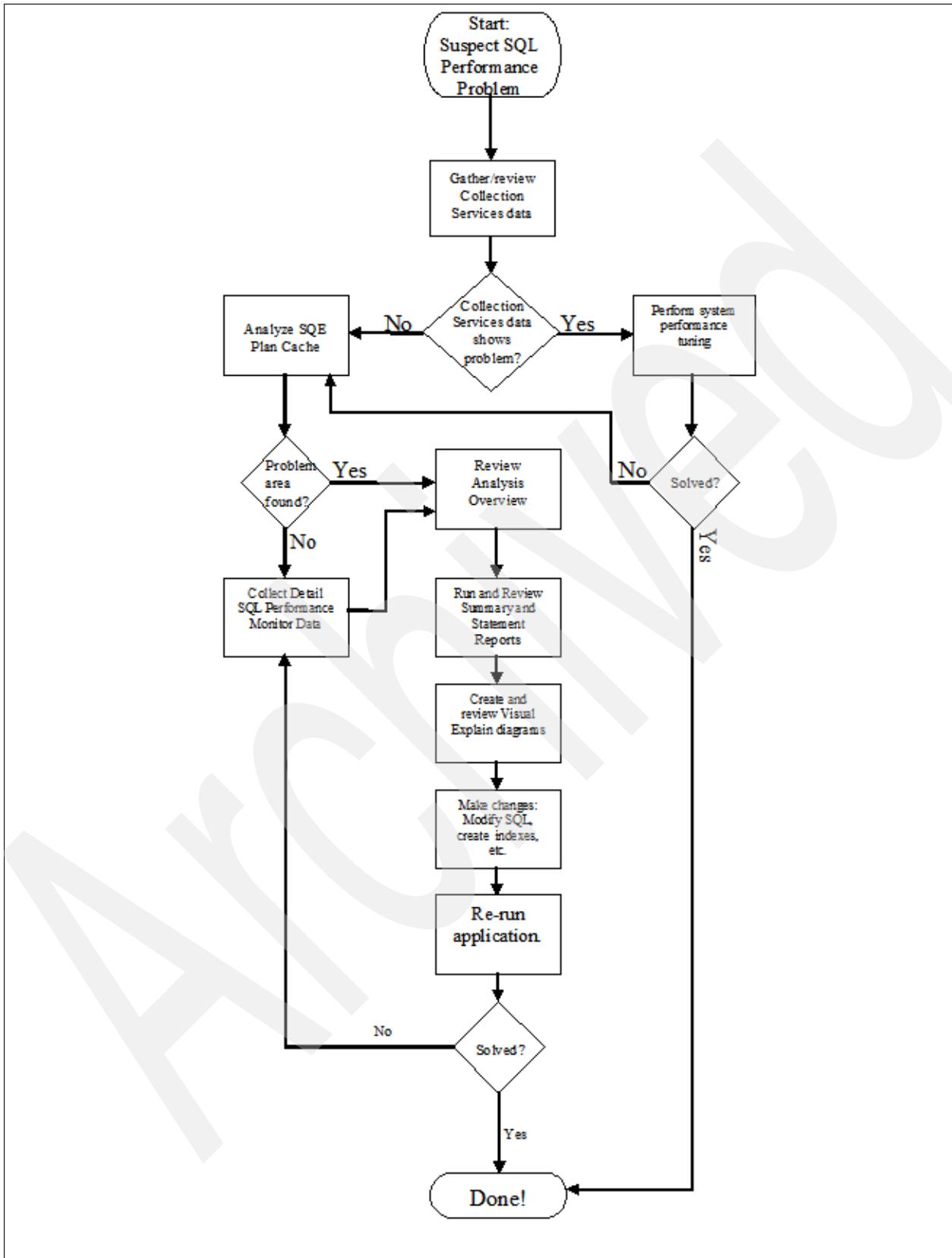


Figure 10-1 SQL Performance Analysis methodology flowchart

10.4 Example of using the methodology

A user has noticed that an application which tallies up sales for each sales person in the company is starting to perform slower every day. We also know that business has been good and there has been an increased volume in sales. Let us follow the methodology steps through this hypothetical situation.

1. Assume for the purposes of this example that we have reviewed the Collection Services data and have found no evidence of needing system-wide tuning.
2. We review the plan cache and sort by total run time. We find a statement at the top which is taking the greatest run time. We see that this is a query joining the employee and department table to the sales file so it could be a culprit. We create a plan cache snapshot by clicking the Create Snapshot button shown in Figure 10-2.

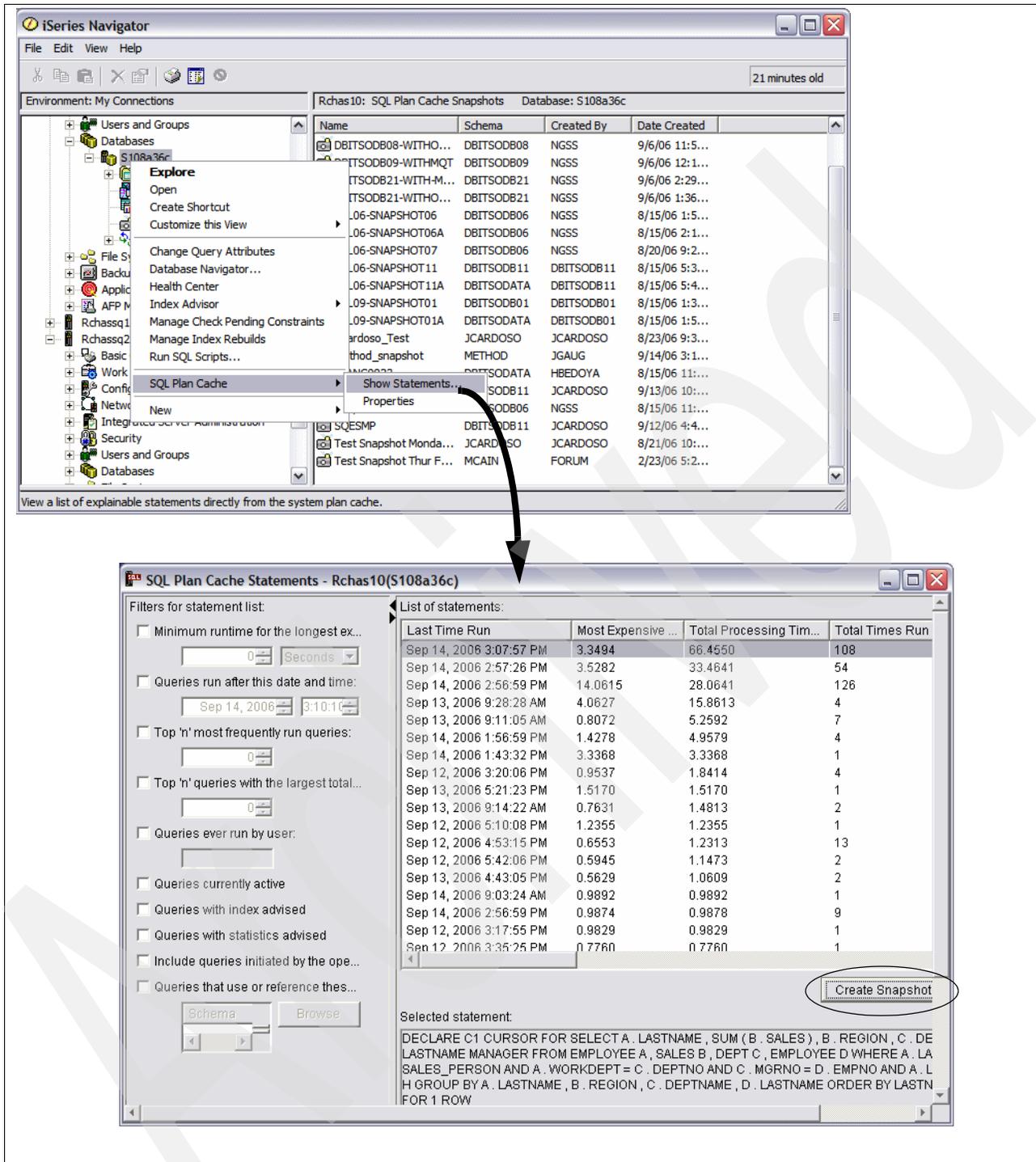


Figure 10-2 Viewing the SQE Plan Cache

- Since we found a statement to investigate in the plan cache, there is no need to collect an SQL Performance monitor at this time. So we will proceed to the next step. After creating the snapshot, you should select SQL Plan Cache Snapshots from the left pane to view the snapshots as shown in upper window of Figure 10-3, then you select the newly-created snapshot in the right pane.

- We select **Analyze** to open up the plan cache snapshot Analysis Overview dashboard. We take a look at the areas of interest. We find that there are several index recommendations for our statement as well as a couple of table scans (Figure 10-3).

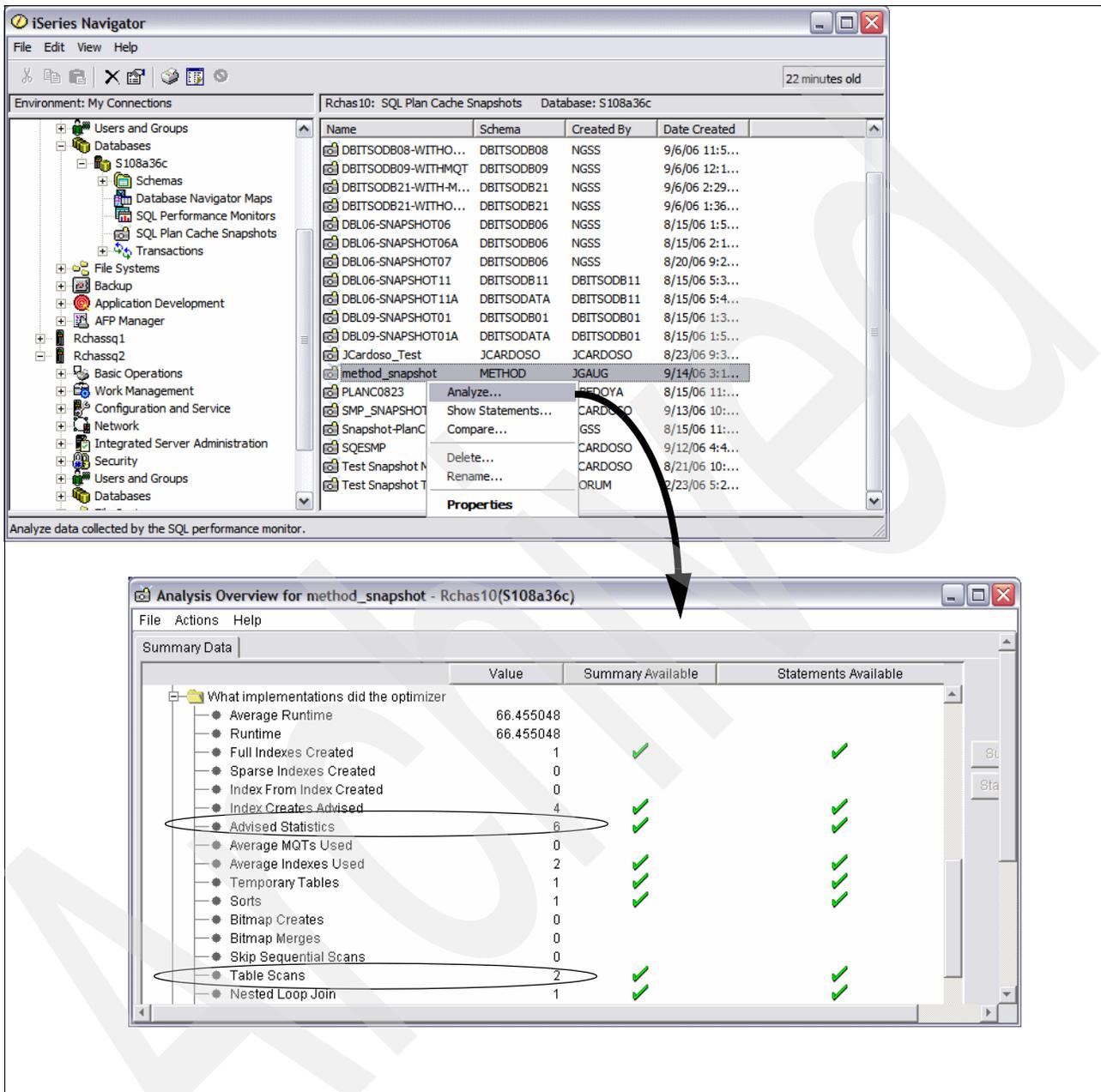


Figure 10-3 Analyzing the Plan Cache snapshot

- At this point, we decide to review the index advised and table scan reports, Figure 10-4. Another place we could get this information is the system wide Index Advisor, explained in Chapter 7.

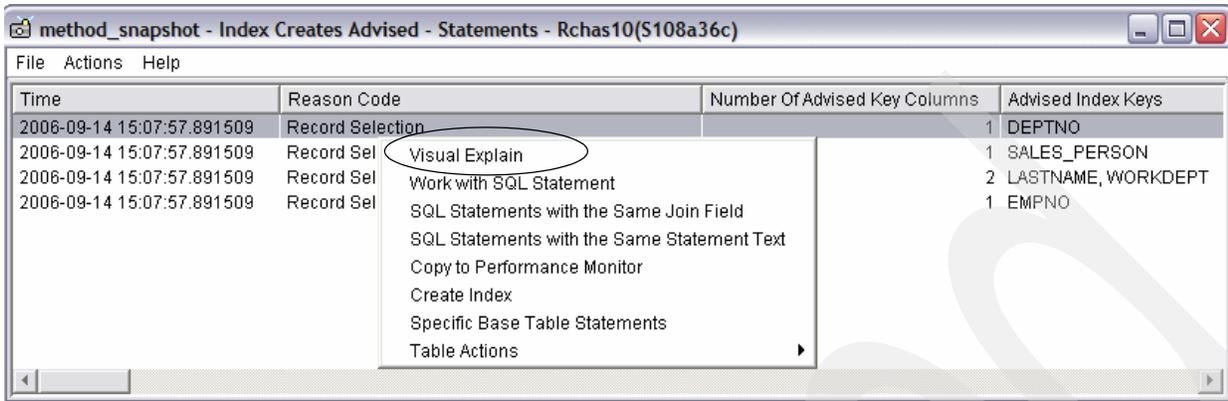


Figure 10-4 Running Visual Explain from the index advised report

- We could have created the recommended indexes from either the index advised or table scan reports, or even from the system wide index advisor. For illustrative purposes, let us proceed to the Visual Explain diagram and ask that the index advice be highlighted by selecting **View** → **Highlight Index Advised** (Figure 10-5).

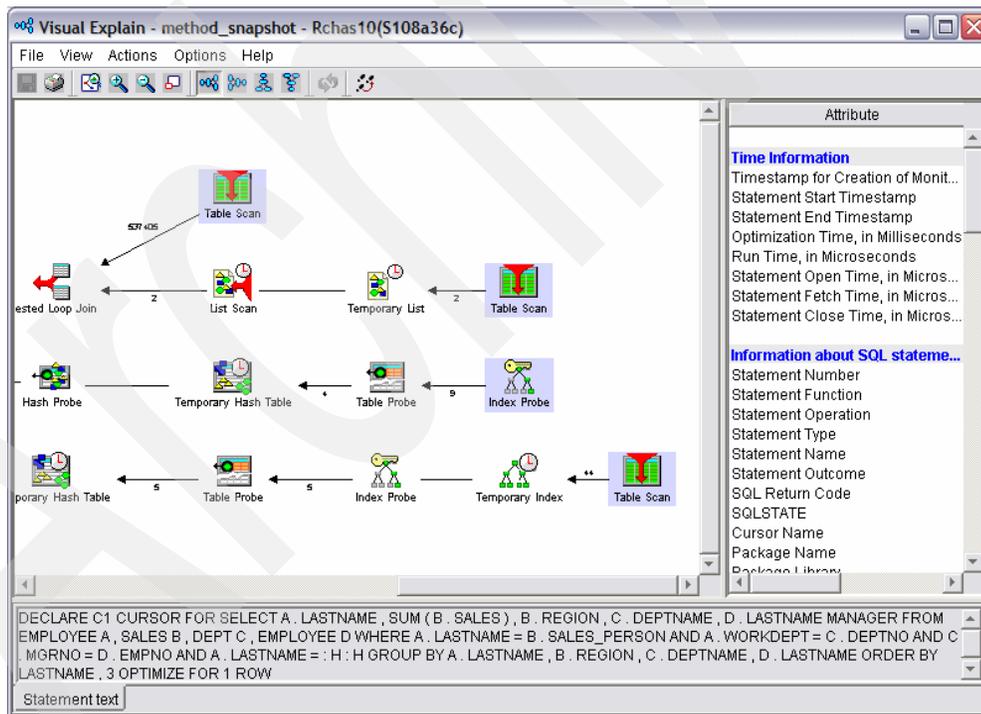


Figure 10-5 Visual Explain diagram with advised indexes highlighted

- There are four tables in our query. But, three of them are small, so we decide to build only the index which is advised over the large Sales table (Figure 10-6).

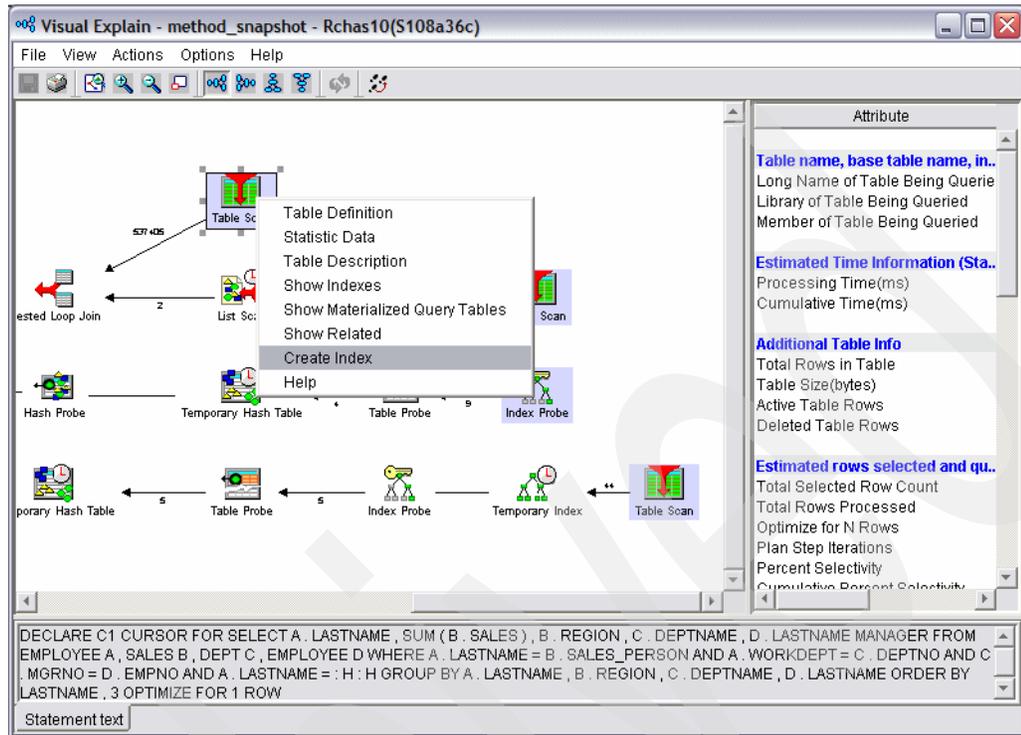


Figure 10-6 Right click the highlighted table scan to create the advised index

- We re-run the application and it find its performance has improved significantly. We could optionally collect a SQL Performance monitor at this stage to quantify the improvement made by our index, and impress our CIO with our query analysis and problem solving skill.

In Figure 10-7 we see that the run time has improved from the 66 second time seen in the plan cache snapshot to less than half a second. Nice work!

Category	Value	Summary Available	Statements Available
What implementations did the optimizer			
● Average Runtime	0.001211		
● Runtime	0.375536		
● Full Indexes Created	2	✓	✓
● Sparse Indexes Created	0		
● Index From Index Created	0		
● Index Creates Advised	3	✓✓	✓✓
● Advised Statistics	510	✓✓	✓✓
● Average MQTs Used	0		
● Average Indexes Used	3.333	✓✓	✓✓
● Temporary Tables	99	✓✓	✓✓
● Sorts	100	✓✓	✓✓
● Bitmap Creates	1	✓	✓
● Bitmap Merges	0		
● Skip Sequential Scans	1	✓	✓
● Table Scans	0		
● Nested Loop Join	99	✓	✓
● Hash Join	0		
● Index Group By	00	✓	✓

Figure 10-7 Nice work

If we step back and think about it, this makes sense. When the database was first created, the sales table was small and it performed fine without any indexes. But as sales grew and the size of the table increased, it became more and more expensive to find a particular sales person's data within the table. Therefore, the index, keyed by sales person, was just what was needed.

Archived

Additional tips

In this part, we provide additional tips to help prevent database performance problems. We present tips regarding indexing strategy and optimizing your SQL statements. We will also discuss some environmental settings that affect SQL Performance. The last chapter of the book will address Collection Services to identify non-SQL performance problems.

This part contains the following chapters:

- ▶ Chapter 11, “Environmental settings that affect SQL performance” on page 379
- ▶ Chapter 12, “Tips to proactively prevent SQL performance problems” on page 407
- ▶ Chapter 13, “Using Collection Services data to identify jobs using system resources” on page 423

Archived



Environmental settings that affect SQL performance

SQL queries optimized by DB2 for i5/OS generate access plans to retrieve the data. These access plans are impacted by the environment in which the query is generated. Various environmental factors affect the costing of access plans and therefore which access plan is selected to retrieve the data.

This chapter discusses some environmental attributes that may affect the access plan as following:

- ▶ Optimization goal
- ▶ Sensitive cursors
- ▶ SMP (Symmetrical multiprocessing) Parallel Degree

11.1 Introduction

DB2 UDB for i5/OS can be accessed by different types of SQL applications such as the following:

- ▶ Web/J2EE applications connecting DB2 for i5/OS via JDBC
- ▶ Web/.Net applications via .Net Provider
- ▶ Client Server application via JDBC, ODBC, CLI
- ▶ Client Server application via ADO/OLEDB provider
- ▶ High Level Languages with Embedded SQL statements

As you can see in Figure 11-1 an access plan is an intersection of various factors.

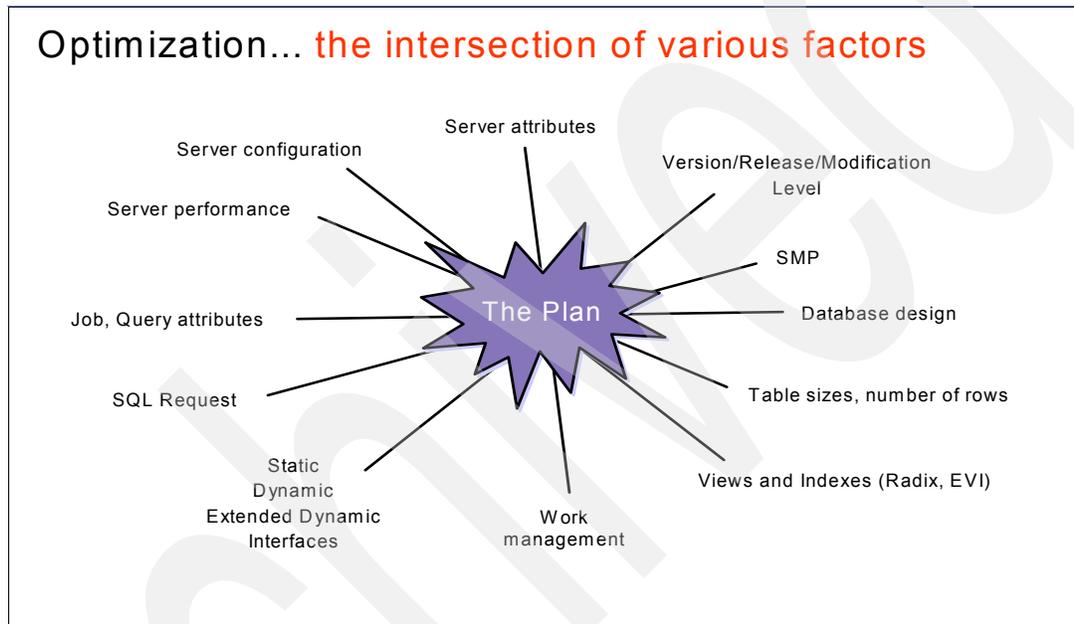


Figure 11-1 The Access Plan - The intersection of various factors

The performance of such applications depends on these factors. We classify these factors as following:

Server factors (Hardware and Software)

- ▶ Server model/ Architecture
- ▶ Main storage size
- ▶ Number of processors (CPUs)
- ▶ Server configuration (subsystems, pools, activity levels, etc.)
- ▶ Server attributes (system values such as QPFRADJ, QQRVDEGREE)
- ▶ Server Performance
- ▶ Work Management
- ▶ Subsystem configuration (number of connections, job queues)
- ▶ Version/Release/Modification Level
- ▶ SMP

Job and Query attribute factors

- ▶ Allow copy data (ALWCPYDTA)
- ▶ Query time limit
- ▶ QAQQINI
- ▶ Naming Convention
- ▶ Job Description (Library list, priority, job queue, output queue)
- ▶ Data source configuration (JDBC/ODBC)
- ▶ Optimization goal
- ▶ CCSID

Database factors

- ▶ Database Design
- ▶ Table Sizes (number of rows)
- ▶ Indexes (Radix, EVI)
- ▶ Views
- ▶ MQTs
- ▶ Statistics
- ▶ CCSID

Request factors

- ▶ SQL statement
- ▶ Static SQL
- ▶ Dynamic SQL
- ▶ Extended Dynamic SQL
- ▶ CQE
- ▶ SQE
- ▶ Result set size
- ▶ Optimization goal

If you change one or more of these factors, the access plan and its performance may also change. The change can be better or worse than usual. Therefore, you have to understand at least the basic aspects of each factor mentioned previously.

If you really want to understand a given access plan and its respective performance, you must know all factors and configurations at the moment of the query optimization process and its execution.

11.2 Optimization goal

It is very common for users to run queries which display results to front-end interfaces. Many times users cancel the query just after viewing the first page of output data. In this front-end scenario users cannot wait too long for large result sets. On the other hand, users also submit queries where output is be written to a file or report, or the interface queues the output data. Normally in this batch scenario users know in advance that the queries are supposed to take some time to build the result set. DB2 for i5/OS has two Optimization Goal attributes to treat each one of these scenarios:

- ▶ FIRST I/O - *FIRSTIO value or *Optimize for N rows* clause
- ▶ ALLIO - *ALLIO value or *Optimize for ALL rows* clause

11.2.1 What is the goal?

Before we technically explain what is the Optimization goal on DB2 for i5/OS, we are going to explain it in a non-technical manner.

People are always moving from one place to another. What criteria do people use to choose transportation? Why don't people always use airplanes? Is the airplane the fastest transportation to move from one place to another?

Who will arrive at the destination first? One who moves by bicycle or one who moves by airplane? The answer is: It depends!

It depends on how far is the destination.

As shown in Figure 11-2 if the distance between one place to another is just ten meters, then one who moves by bicycle will win. By the time that the airplane has warmed up its engines the bicycle has moved the 10 meters. Therefore, the bicycle is the best choice for such short distances.

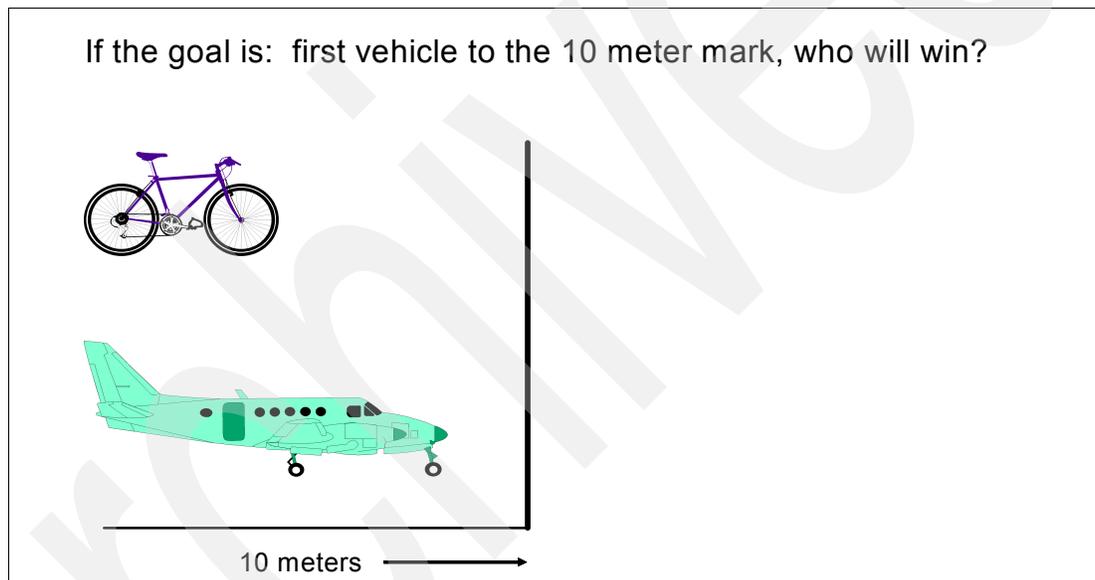


Figure 11-2 First vehicle to 10 meter mark, who will win?

As shown in Figure 11-3, if the distance between one place to another is one hundred kilometers, then one who moves by airplane will win. Therefore, the airplane is the best choice for long distances.

If the goal is: first vehicle to the 100 kilometer mark, who will win?

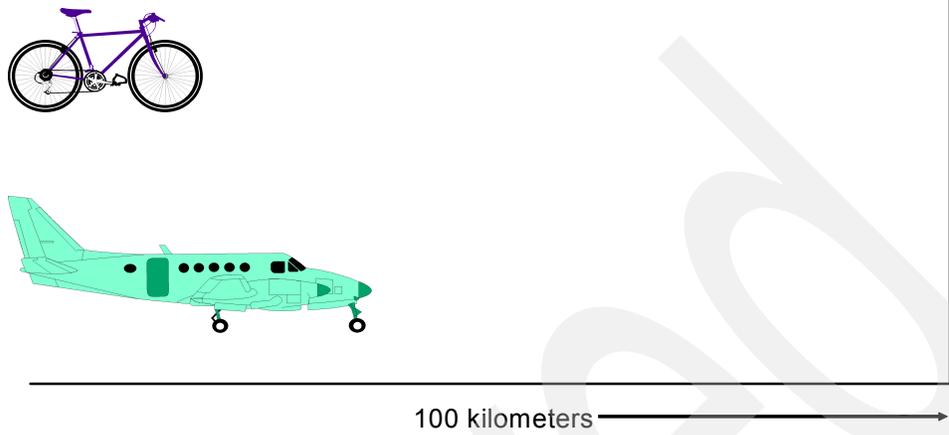


Figure 11-3 First vehicle to the 100 kilometer mark, who will win?

In conclusion, you usually know the distance between one place to another in advance. Actually you do not need to know the accurate distance. But, normally you have an idea if the distance is short, medium or long. Therefore, you can decide if you walk, if you go by bicycle, by car or by plane. How about going by rocket? Again, It depends! Are you going to the moon? Can you go to the moon by airplane?

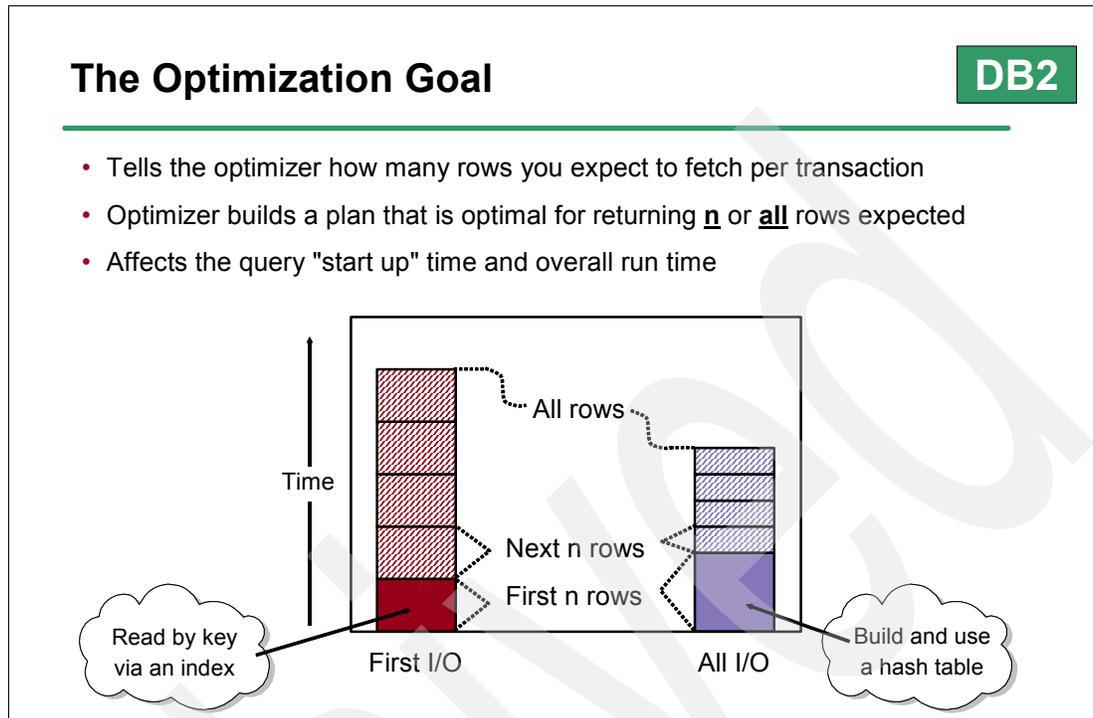
Now we are going to talk about the Optimization Goal of the DB2 for i5/OS.

The decision of what transportation would be the better choice depends on the distance that one knows in advance. Think about queries in a similar way, the Optimizer builds a plan dependent on how many rows you expect to fetch per transaction. It builds a plan that is optimal for returning *n* or *all* rows expected. Therefore, you have to answer the following questions:

- ▶ How much do you know in advance about your query?
- ▶ How much can you tell the optimizer about your query?
- ▶ How many rows do you expect to fetch per transaction?

Of course, you do not have to know an accurate number of how many rows your queries are going to fetch. But, you usually have an idea whether the result sets are going to be small or

big. As shown in Figure 11-4, the answers to these questions affects the query “start up” time and overall time.



When setting the Optimization Goal you can use the following:

▶ ***DEFAULT**

- Optimization goal is determined by the interface (ODBC, SQL precompiler options, OPTIMIZE FOR nnn ROWS clause).

▶ ***FIRSTIO**

- All queries will be optimized with the goal of returning the first page of output as fast as possible. This goal works well when the control of the output is controlled by a user who is most likely to cancel the query after viewing the first page of output data. Queries coded with an OPTIMIZE FOR nnn ROWS clause will honor the goal specified by the clause.

Note: Be aware that usually the optimizer looks for an index when you use First I/O.

▶ ***ALLIO**

- All queries will be optimized with the goal of running the entire query to completion in the shortest amount of elapsed time. This is a good option when the output is being written to a file or report, or the interface is queuing the output data. Queries coded with an OPTIMIZE FOR nnn ROWS clause will honor the goal specified by the clause.

11.2.2 Setting the Optimization Goal

The Optimization Goal can be set via the following:

- ▶ Optional SQL statement clause
 - OPTIMIZE FOR n ROWS
 - OPTIMIZE FOR ALL ROWS
- ▶ OPTIMIZATION_GOAL parameter of the QAQQINI options file –or- ODBC/JDBC connection attributes
 - *FIRSTIO
 - *ALLIO
- ▶ Default for dynamic interfaces is First I/O
 - ODBC, JDBC, STRSQL, dynamic SQL in programs
 - CQE - 3% of expected result set
 - SQE - 30 rows
- ▶ Otherwise default is ALL I/O
 - Extended dynamic, RUNSQLSTM, INSERT + subSELECT, CLI, static SQL in programs
 - All of expected result set

Important: The optimization goal will affect the optimizer's decisions:

- ▶ Use of indexes, SMP, temporary intermediate results like hash tables
- ▶ Tell the optimizer as much information as possible
- ▶ If the application fetches the entire result set, use *ALLIO

The following SQL script in Example 11-1 uses Optimization Goal FIRST I/O. See the *Optimize for 10 rows* clause:

Example 11-1 SQL statement with optimization goal First I/O and Optimize for 10 rows clause

```
SELECT year, part
FROM orders i, parts p
WHERE i.partkey = p.partkey
AND year = 2002
AND returnflag like 'R'
Optimize for 10 rows;
```

Figure 11-5 shows the Visual Explain graph with the access plan chosen by the optimizer. See that the Optimizer used the index and considered the Optimization Goal FIRST I/O for 10 rows.

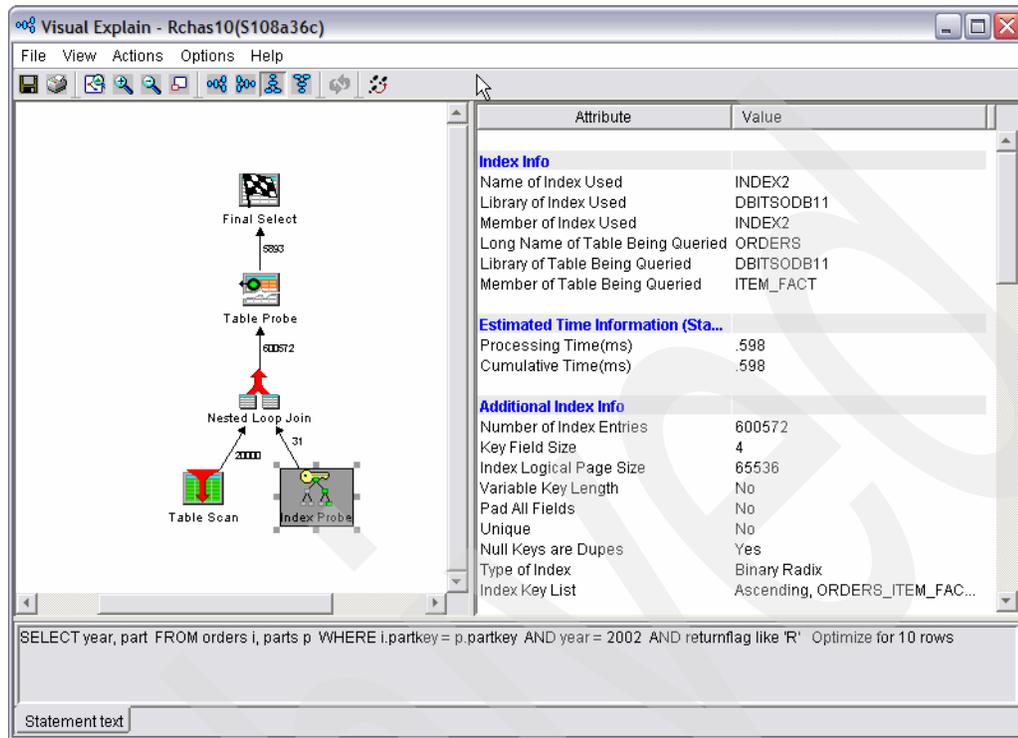


Figure 11-5 Sample of Visual Explain of the SQL statement with optimization goal First I/O and Optimize for 10 rows clause

Now see the following SQL script in Example 11-2 that uses Optimization Goal ALL I/O with the *Optimize for ALL rows* clause:

Example 11-2 SQL statement with optimization goal ALL I/O and Optimize for ALL rows clause

```
SELECT year, part
FROM orders i, parts p
WHERE i.partkey = p.partkey
AND year = 2002
AND returnflag like 'R'
Optimize for ALL rows;
```

Figure 11-6 shows the Visual Explain graph with the access plan chosen by the optimizer. See that this time the Optimizer considered the Optimization Goal ALL I/O and it used a table

scan to build a Hash Table. The optimizer took some time to build the Hash Table. Once the Hash table was ready it speeded up the query.

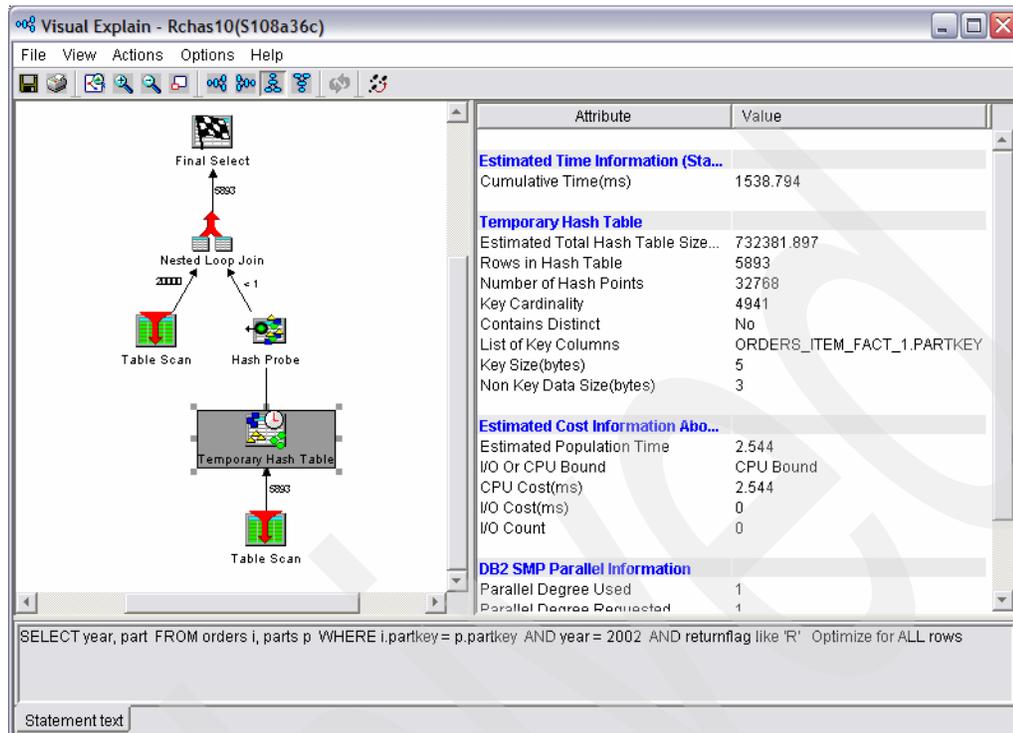


Figure 11-6 Sample of Visual Explain of the SQL statement with optimization goal ALL I/O and Optimize for ALL rows clause

11.3 Sensitive Cursors

Cursor sensitivity controls whether or not the result set returned by a cursor will include changes (Insert/Update/Delete) made to a table after the cursor was opened.

There are three types of settings for Sensitive Cursors

- ▶ **ASENSITIVE**
 - The Default setting. ASENSITIVE cursor allows DB2 to choose to implement the cursor as Sensitive or Insensitive depending on how the *select-statement* is optimized. Updateable cursors are always implemented as Sensitive. If the cursor is read-only, the cursor behaves as INSENSITIVE.
- ▶ **SENSITIVE**
 - Specifies that changes made to the database after the cursor is opened are visible in the result table. The cursor has some level of sensitivity to any updates or deletes made to the rows underlying its result table after the cursor is opened. The cursor is always sensitive to positioned updates or deletes using the same cursor. Additionally, the cursor can have sensitivity to changes made outside this cursor. If the database manager cannot make changes visible to the cursor, then an error is returned. The database manager cannot make changes visible to the cursor when the cursor implicitly becomes read-only.

► INSENSITIVE

- Specifies that once the cursor is opened, it does not have sensitivity to inserts, updates, or deletes performed by this or any other activation group. If INSENSITIVE is specified, the cursor is read-only and a temporary result is created when the cursor is opened. In addition, the SELECT statement cannot contain a FOR UPDATE clause and the application must allow a copy of the data (ALWCOPYDTA(*OPTIMIZE) or ALWCOPYDTA(*YES)).

For example, a sensitive cursor declared as “SELECT * FROM orders” would return all the current orders as well as any new orders inserted into the database after the cursor was opened. For example, if there were six orders when a sensitive cursor was opened and two new orders inserted after the cursor was opened by another application, then a sensitive cursor will try to include all eight orders in its result set (see Table 11-1).

Table 11-1 Sensitive cursor result set

A122	5	Hammer
A123	500	Screw - 1/4
A125	10	Saw
A126	100	Carriage Bolt
A127	6	Hammer
A130	22	Electric Drill
A131	1	Table Saw
A132	200	Clamp

The result set contents returned by an insensitive cursor in the same application scenario are displayed in Table 11-2.

Table 11-2 Insensitive cursor result set

A122	5	Hammer
A123	500	Screw - 1/4
A125	10	Saw
A126	100	Carriage Bolt
A127	6	Hammer
A130	22	Electric Drill

11.3.1 Performance and query optimization impacts

The cursor sensitivity setting obviously has an impact on the access plan created by the query optimizer. The access plan for a sensitive cursor cannot include algorithms or data access techniques that make a copy of the table data. For instance, hash tables are often used in the implementation of join and grouping queries, but since they contain a copy of the table data, they cannot be used in the implementation of sensitive queries. This direct correlation between cursor sensitivity and the query optimizer is why the cursor sensitivity support has been enhanced since V5R2. V5R2 included query optimization enhancements that resulted in the optimizer using more algorithms that make a copy of table data. The end result was fewer cursors returning sensitive result sets than in previous releases. To solve

this problem, programming interfaces were enhanced to give application developers greater control of the cursor sensitivity setting.

The default setting of `ASENSITIVE` is the best performing option because it allows the query optimizer to use its complete set of algorithms when deciding on the best method for implementing a query. Utilization of an algorithm that makes a copy of the data (e.g., hashing) can drastically improve performance and other queries where performance is better when copies of the data are avoided. The `ASENSITIVE` setting gives the query optimizer the freedom to choose the best performing method. The `SENSITIVE` and `INSENSITIVE` cursor settings can force the optimizer into a plan that is sub-optimal from a performance point of view. The `SENSITIVE` setting eliminates the usage of temporary data copies by the optimizer; this also prevents parallel processing since the DB2 Symmetric MultiProcessing (DB2 SMP) feature makes copies of the table data. The `INSENSITIVE` setting forces a copy of the table data whether it is good for performance or not. Thus, the `INSENSITIVE` and `SENSITIVE` cursor settings should be used only when the associated cursor behavior is absolutely required by the application.

11.3.2 Cursor sensitivity programming interfaces

This section contains the programming details on how to control the cursor sensitivity setting with the iSeries integrated middleware. Please be aware that when controlling the cursor sensitivity at a connection level that sensitivity is applied to all cursors opened within that connection. This means that if `SENSITIVE` is the connection level setting, the database manager must be able to implement all of the cursors in that connection as sensitive cursors. Otherwise, an error will be returned.

Note: *Updateable cursors* are always implemented as `SENSITIVE` cursors, so the sensitivity settings described below are ignored for updateable cursors.

iSeries access ODBC driver

With the ODBC driver, sensitivity can be controlled at an individual cursor level with the `SQL_ATTR_CURSOR_SENSITIVITY` attribute on the `SQLSetStmtAttr` API. It should be noted that ODBC specifications do not support a value of `Asensitive`; instead, `Unspecified` is the value used on the `SQLSetStmtAttr` API to specify an `Asensitive` cursor sensitivity behavior.

The `CURSORSENSITIVITY` connection keyword can be specified to control the cursor sensitivity setting at a connection level with one of the following values:

- ▶ 0 - `Asensitive`
- ▶ 1 - `Insensitive`
- ▶ 2 - `Sensitive`

The keyword applies only to applications that use the ODBC 2.0 cursor type statement attribute (which associates sensitivity and scrollability). The connection keyword is ignored in applications that use ODBC 3.0 attributes for sensitivity.

Static cursors are always `Insensitive`. The sensitivity settings only apply to dynamic and forward-only cursors.

The following ODBC code snippet in Example 11-3 shows how to control the cursor sensitivity at a connection level with the `CURSORSENSITIVITY` connection keyword and at a statement level with the `SQLSetStmtAttr` API. When using the `SQLSetStmtAttr` API to change attributes, changing some cursors attributes such as cursor type can also cause the sensitivity setting to

be changed. These indirect changes of the sensitivity setting are documented in the ODBC specifications.

Example 11-3 ODBC Code Example

```
connectStr = "DRIVER={iSeries Access ODBC
Driver};SYSTEM=myiSvr;UID=myid;PWD=mypwd;" +
"CURSORSENSITIVITY=2;"
rc = SQLDriverConnect(hdbc, NULL, (SQLCHAR *)connectStr, SQL_NTS, NULL, 0, NULL,
SQL_DRIVER_NOPROMPT);
...

SQLAllocHandle(SQL_HANDLE_STMT, hdbc,&hstmt1);
SQLSetStmtAttr(hstmt1, SQL_ATTR_CONCURRENCY, SQL_CONCUR_VALUES, 0);
SQLSetStmtAttr(hstmt1, SQL_ATTR_CURSOR_SENSITIVITY, SQL_SENSITIVE, 0);
SQLExecDirect(hstmt1, "SELECT orderid, status FROM orders", SQL_NTS);
...
```

iSeries OLE DB provider

The OLE DB provider only allows Cursor Sensitivity to be controlled at the connection level with the connection property, "Cursor Sensitivity." The connection property values are the same as the ODBC connection keywords. This new connection property cannot be changed after a connection has been opened.

Example 11-4 is a coding example showing how to set the new connection property either on the connection string or with the Properties method.

Example 11-4 OLE DB Code Example

```
cn.ConnectionString = "Provider=IBMDA400;Data Source=MyiSeries;Cursor
Sensitivity=2"

cn.Properties("Cursor Sensitivity") = 0 '/// ASENSITIVE
cn.Open...
```

Like the ODBC driver, static cursors are always implemented with the Insensitive attribute. Forward only cursors can never be sensitive with the OLE DB provider. The next release will allow forward only cursors to be defined as sensitive.

Toolbox JDBC Driver

The Toolbox JDBC driver also has a "Cursor sensitivity" connection property that supports the property values of: asensitive, sensitive and insensitive.

The sensitive connection property value is only honored for cursors declared with a resultSetType of TYPE_SCROLL_SENSITIVE. Cursors declared with the resultSetType of TYPE_SCROLL_SENSITIVE are implemented as ASENSITIVE by default. The connection property of "sensitive" must be specified in order for cursors with resultSetType, TYPE_SCROLL_SENSITIVE, to be implemented as SENSITIVE cursors.

Insensitive cursors are defined by specifying TYPE_SCROLL_INSENSITIVE for the resultSetType. The only time that the insensitive connection property value is recognized is for cursors declared with a resultSetType of TYPE_FORWARD_ONLY.

TYPE_FORWARD_ONLY cursors are always implemented as ASENSITIVE unless the insensitive connection property value is specified.

The following JDBC coding in Example 11-5 shows how to specify a sensitive cursor setting with the new connection property.

Example 11-5 JDBC Code example

```
// Load the IBM Toolbox for Java JDBC driver.  
  
DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());  
Properties props = new Properties();  
  
props.setProperty("user", "myid");  
props.setProperty("password", "mypass");  
props.setProperty("cursor sensitivity", "sensitive");  
  
Connection connection = DriverManager.getConnection(" jdbc: as400://myiSvr",  
props);  
...  
s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,  
ResultSet.CONCUR_READ_ONLY);  
ResultSet rs = s.executeQuery("SELECT ordid, status FROM orders");  
...
```

Important:

The AS400JDBCDataSource java class has the following *getCursorSensitivity* method which returns the value of the cursor sensitivity property:

```
public String getCursorSensitivity()
```

If the resultSetType is ResultSet.TYPE_FORWARD_ONLY or ResultSet.TYPE_SCROLL_SENSITIVE, the value of this property will control what cursor sensitivity is requested from the database. If the resultSetType is ResultSet.TYPE_SCROLL_INSENSITIVE, this property will be ignored.

Returns the cursor sensitivity. Valid values include:

- ▶ “asensitive”
- ▶ “insensitive”
- ▶ “sensitive”

Also the AS400JDBCDataSource java class has the following *setCursorSensitivity* method which allows you to set the cursor sensitivity to be requested from the database:

```
public void setCursorSensitivity(String cursorSensitivity)
```

If the resultSetType is ResultSet.TYPE_FORWARD_ONLY or ResultSet.TYPE_SCROLL_SENSITIVE, the value of this property will control what cursor sensitivity is requested from the database. If the resultSetType is ResultSet.TYPE_SCROLL_INSENSITIVE, this property will be ignored.

Valid values include:

- ▶ “asensitive”
- ▶ “insensitive”
- ▶ “sensitive”

The default is “asensitive”. This property is ignored when connecting to systems running OS/400 V5R1 and earlier.

Embedded SQL

V5R4 and V5R3 include the ability to specify the three cursor sensitivity settings on the DECLARE CURSOR statement. Refer to Example 11-6.

Example 11-6 Cursor sensitivity setting on the Declare cursor statement

```
EXEC SQL  
DECLARE C1 SENSITIVE SCROLL CURSOR FOR  
SELECT DEPTNO, DEPTNAME, MGRNO  
FROM TDEPT  
WHERE ADMRDEPT = 'A00';
```

Note: V5R2 support only allows the default setting of ASENSITIVE or choosing an INSENSITIVE cursor definition.

For more information, refer to the article available on the Web at:

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0403milligan/index.html>

11.4 SMP (Symmetrical multiprocessing) Degree

In this section we give you the concepts of SMP and how it works cohesively with DB2 for i5/OS to deliver powerful parallelism for database access. Also, we discuss how to turn SMP on and off, and how to manage the degree of SMP that is invoked by a particular job. Feedback from SQE Plan Cache Snapshots, SQL Performance Monitor and Visual Explain are also covered in this section. They provide information to understand how well your jobs are taking advantage of SMP. At the end of this section we mention several sets of tips and considerations related to the use of database parallelism in the i5/OS environment.

11.4.1 iSeries i5/OS Architecture

The architecture and technology of the eServer i5 server running i5/OS provides the foundation for database parallelism. Starting at the bottom of the Figure 11-7.

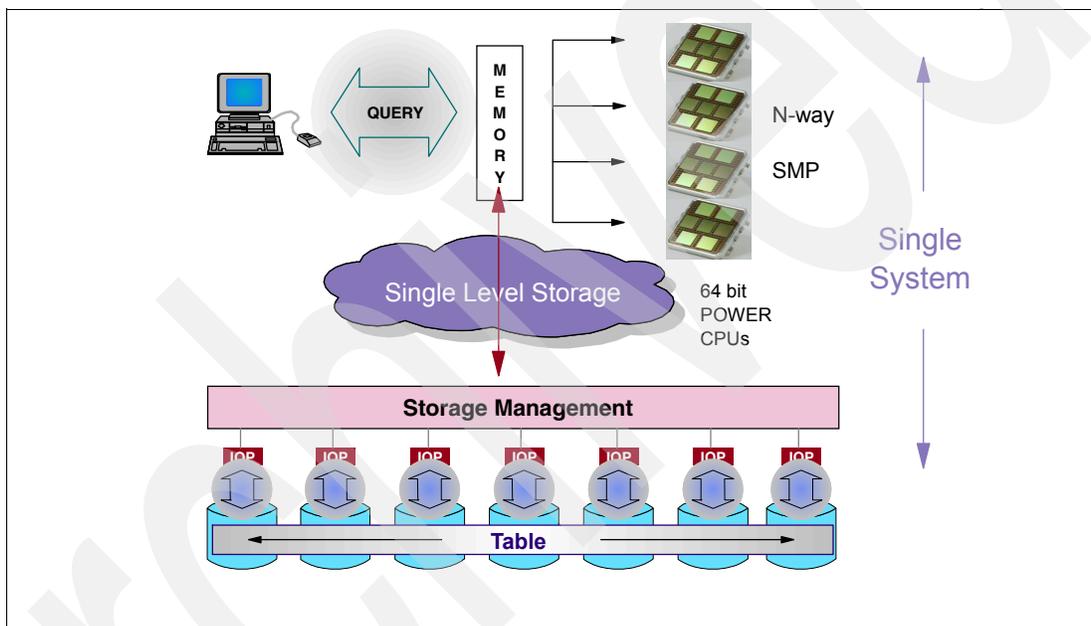


Figure 11-7 iSeries i5/OS Architecture

- ▶ The iSeries independent I/O subsystem, along with i5/OS storage management, allows for synchronous and asynchronous database I/O requests. These requests can make use of parallel operations to access data on multiple disk units simultaneously via the I/O processors (IOPs) and I/O adapters (IOAs), without using the CPU.
- ▶ DB2 UDB for iSeries uses storage management to spread the database objects across all the available disk units. For example, as a table is populated, the space is automatically allocated on the disk units for optimal performance. This spreading of data minimizes contention on any single disk unit, and it also provides the basis for parallel I/O.
- ▶ As the disk units are accessed, the data is brought into memory. The design of the iSeries hardware supports a very large memory system. As a true 64-bit system, i5/OS and DB2 UDB for iSeries can take full advantage of all the available memory. This provides the advantage of using main memory like cache for database objects.
- ▶ iSeries servers have a unique way of addressing storage. It views the disk space on the server and the server's main memory as one large storage area. This way of addressing storage is known as single-level storage. The concept of single-level storage means that the knowledge of the underlying characteristics of hardware devices (in this case, main

storage and disk storage) resides in the System Licensed Internal Code (SLIC). All the storage is automatically managed by the server. No user intervention is ever needed to take full advantage of any storage technology. Programs work with objects; objects are accessed by name, not by address.

- ▶ iSeries servers support multiple 64-bit POWER™ CPUs (currently up to 64). i5/OS can take advantage of multiple CPUs by automatically dispatching work to one or more CPUs.
- ▶ As the query request is processed, the integrated database takes advantage of the advanced server and operating system technologies to exploit symmetric multiprocessing, and thus achieve database parallelism.

11.4.2 What is SMP?

DB2 Symmetric Multiprocessing provides high performance database processing and is an excellent way to take advantage of all the available resources within the eServer i5 platform. This feature relies on the integrated technologies found in i5/OS and is an example of the outstanding value provided by the best business computer available today.

n-way processing

Before we start describing what SMP is let's explain the concept of n-way processing. Within i5/OS, a unit of work is defined as a job, thread, or task. Built into the operating system is the ability to dispatch this work to any one of the available CPUs. This concept has the advantage of allowing more requests to be processed, as more CPUs are made available. As shown in Figure 11-8, any individual job, thread, or task can only run on a single CPU. If additional CPUs are available, they provide little or no help. For example, if one job is executing on a server or LPAR with eight CPUs available, this job will only take advantage of one of the CPUs while the other seven sit idle. To utilize the other CPUs, additional techniques and strategies must be applied. This is where DB2 SMP comes into play.

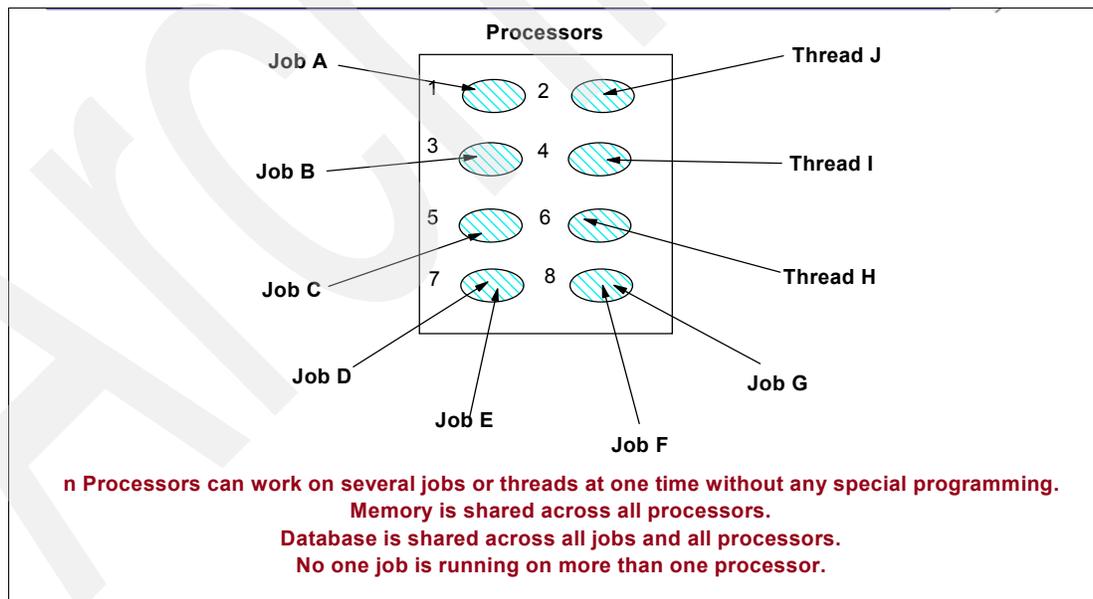


Figure 11-8 n-Way Processing

How SMP works

The DB2 UDB Symmetric Multiprocessing feature provides the optimizer and database engine with additional methods and strategies for retrieving data and processing data in parallel. SMP enables database parallelism on a single server or LPAR where multiple (CPU

and I/O) processors that share memory and disk resource work simultaneously toward achieving a single end result. As shown in Figure 11-9 parallel processing allows the database engine to have more than one (or all) the processors working on a single query simultaneously. The performance of a CPU-bound query can be significantly improved with this feature on multiple-processor servers by distributing the processor load across more than one processor. While using SMP does not require the presence of more than one CPU, database parallelism is most effective when there is more than one physical CPU available to run the tasks or threads. Given that SMP is achieved through the use of the iSeries server and its i5/OS advanced architecture, table partitioning is not required for database parallelism.

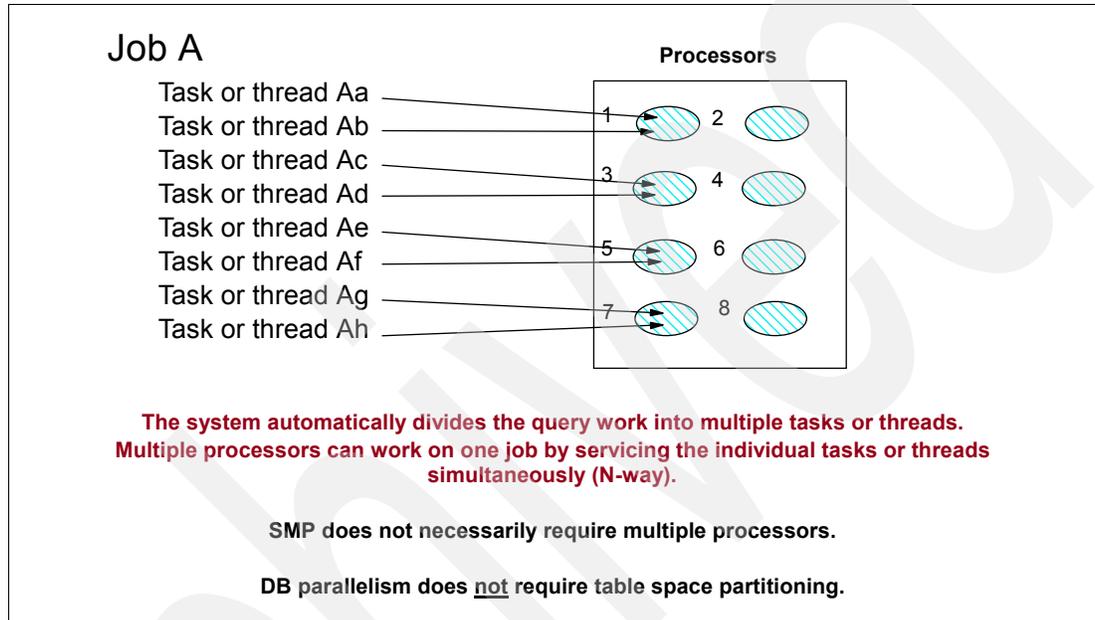


Figure 11-9 How SMP Works

CQE versus SQE

Database parallelism is achieved through the use of SLIC tasks or threads. The Classic Query Engine (CQE) originally provided with SQL on the iSeries server uses *tasks* as a means of achieving parallelism. Instead, the new (as of V5R2) SQL Query Engine (SQE) uses threads. Generally speaking, CQE limits the number of tasks (for a given query request) to the *number of disk units* that contain the data. In contrast, SQE limits the number of threads (for a given query request) to the *number of physical CPUs* available.

Furthermore, CQE requires a large number of tasks to drive parallel I/O and data processing: normally 1 task per disk unit (up to 255 per query or 1024 per server). On the other hand, SQE requires a small number of threads to drive parallel I/O and data processing: normally 1 or 2 threads per CPU. Therefore, SQE can take more advantage of SMP than CQE.

The row selection and column processing is performed in the task or thread. The parent job schedules the work requests to the tasks or threads, and merges the results into the result buffer that is returned to the application.

11.4.3 SMP parallel-enabled functions

By installing and enabling the DB2 SMP feature, the query optimizer is able to consider parallel methods and strategies. However, these parallel methods and strategies are only available for certain database features and functions. Generally speaking, any query request

that is processed by the DB2 optimizer is eligible, regardless of programming interface. A static or dynamic *SQL request from within a high-level language program* is eligible to use SMP.

Restriction: Native, record level access from within high-level language programs is not enabled for SMP. This is one reason why using SQL is an advantage.

Another important use of SMP is to *speed up the creation and maintenance of indexes* (sometimes referred to as *keyed access paths*). Creating indexes with SMP allows the database engine to use all of the available resources to speed up the creation process. This may be an important strategy when an application is unavailable until an index for one of the files accessed by that application is available.

With multiple indexes or keyed logical files over a table or physical file, index maintenance occurs whenever rows are added, changed, or deleted. Normally, each index is maintained synchronously, that is, one at a time. With SMP enabled, blocked INSERT or WRITE operations can benefit from the fact that the database engine maintains each index in parallel. This is accomplished by using one database task to maintain each index – again trading the use of additional resources for a decrease in time. This has the benefit of reducing the overall index maintenance time and the overall INSERT time.

Within i5/OS, DB2 UDB for iSeries has the ability to import data into a table. This function is known as *copy from import file*, and is invoked using the CPYFRMIMPF command. This feature is also parallel-enabled via SMP.

Within i5/OS, DB2 UDB for iSeries has the ability to physically reorganize the data within a table. This function is known as *reorganize physical file member*, and is invoked using the iSeries Navigator command. This feature is also parallel-enabled via SMP.

Integrated within i5/OS is the ability to save and restore objects, including database objects. This function is parallel-enabled, but does not require the SMP feature. Given that the *parallel save and restore support* is accessed via APIs, the recommendation is that you use Backup Recovery and Media Services (BRMS) if parallel save and restore is to be used.

SQL requests and SMP

While INSERT, UPDATE and DELETE operations cannot execute in parallel, most read-only query methods and queries can use parallel methods. This means that if the query is using a

table scan, that particular operation can use parallelism and benefit from SMP. Figure 11-10 shows how an SQL SELECT statement can take advantage of SMP.

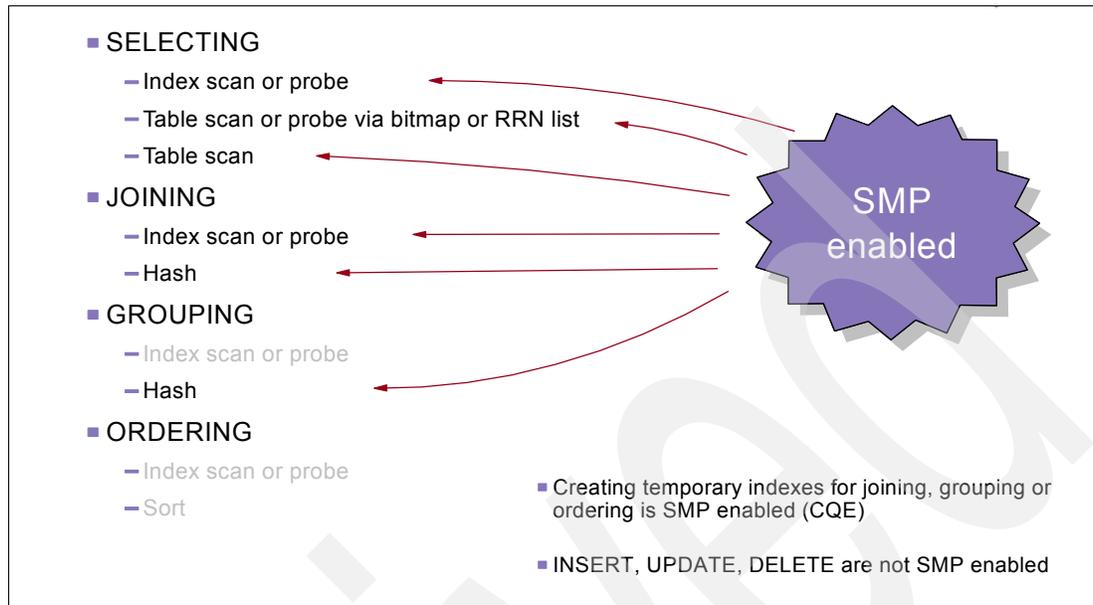


Figure 11-10 SQL Requests and SMP

Some operations, such as ordering the data via a sort, are not performed in parallel and do not benefit from SMP.

If a temporary index is created during the query request, the operation can benefit from SMP.

As queries are analyzed and tuned for performance, it is important to remember which methods and strategies are parallel-enabled and potentially benefit from SMP.

11.4.4 Parallel Database Processing

The use of database parallelism requires the SMP feature to be installed on the server (or LPAR). However, SMP must also be enabled for the job processing the request.

The default setting for the server allows no database parallelism. The parallel processing degree can be set by either the system value QQRVDEGREE, a query option file (QAQQINI), or by the DEGREE parameter on the change query attributes (CHGQRYA) command. In fact, the CHGQRYA command can be issued multiple times within a job to turn parallelism on and off.

There are two forms of database parallelism: I/O parallelism and CPU parallelism. I/O parallelism allows for the accessing of data in parallel, but the processing of that data does not occur in parallel. I/O parallelism utilizes shared memory and disk resources by pre-fetching or pre-loading the data, in parallel, into memory. CPU parallelism utilizes one (or all) of the system processors in conjunction with the shared memory and disk resources in order to reduce the overall elapsed time of a query. CPU parallelism allows for both the accessing of data and the processing of that data in parallel. For example, a table scan can use only parallel I/O to access and bring the data into memory, but only process it with one task. With the SMP feature installed and enabled, CPU parallelism can be used to both access the data in parallel and to process the data in parallel, using multiple tasks or threads.

Setting the parallel degree allows the query optimizer to consider the use of SMP. The optimizer determines whether or not the query will benefit from parallel methods, and builds the appropriate strategy for using database parallelism.

Note: An excellent article on parallel indexing by Kent Milligan is available on the Web at: <http://www.ibm.com/developerworks/db2/library/techarticle/0301milligan/0301milligan.html>

11.4.5 Enabling parallel processing

After you install the DB2 SMP feature, there are several different ways to activate parallel processing so that parallel index builds and maintenance can be performed.

At a system level

By default, the amount of parallel processing is controlled at a system-wide level via the QQRVDEGREE system value. If that system value is set to a value other than *NONE, DB2 UDB will use parallel processing. The CHGSYSVAL (Change System Value) commands can be used to change the setting of the QQRVDEGREE system value. Here are the values that you can specify to enable parallel index processing:

- ▶ *NONE - No parallel processing is allowed for database query processing.
- ▶ *IO - Any number of tasks may be used when the database query optimizer chooses to use I/O parallel processing for queries. CPU parallel processing is not allowed. SQE always considers IO parallelism.
- ▶ *OPTIMIZE - The query optimizer can choose to use any number of tasks or threads for either I/O or CPU parallel processing. Use of parallel processing and the number of tasks or threads used will be determined with respect to the number of processors available in the system, the job's share of the amount of active memory available in the pool in which the job is run, and whether the expected elapsed time for the query is limited by CPU processing or I/O resources.
- ▶ *MAX - The query optimizer can choose to use either I/O or CPU parallel processing to process the query. The choices made by the query optimizer will be similar to those made for parameter value *OPTIMIZE except the optimizer will assume that all active memory in the pool can be used to process the query.

At a job or connection level

If you want to restrict parallel processing to an individual job or connection, then use the CHGQRYA (Change Query Attributes) CL system command or a QAQQINI file to enable parallel processing for a job or connection. These interfaces override the system value setting. Both interfaces can configure a job or connection to use parallel processing with the same values as the system value QQRVDEGREE. Also, at job or connection level you can configure the following values:

- ▶ *SYSVAL - Specifies that the processing option used should be set to the current system value, QQRVDEGREE.
- ▶ *ANY - This value has the same meaning as *IO. The *ANY value is maintained for compatibility with prior releases.
- ▶ *NBRTASKS nn - Specifies the number of tasks or threads to be used when the query optimizer chooses to use CPU parallel processing to process a query. I/O parallelism will also be allowed. This value is not available via the system value. It is used to manually control the degree value. The value is used whether or not parallelism provides a faster elapsed time.

Note: The use of parallel CPU methods requires the SMP feature. When you set the degree parameter value to *NBRTASKS the optimizer will build a plan to use the number of task or threads requested. This value is used for experimentation and testing, and is not necessarily designed for use in a production environment given that it forces the optimizer to use parallelism.

The DEGREE parameter on the CHGQRYA CL command accepts the same values as the system value. If the CHGQRYA command is executed in a job as shown in Example 11-7, then the parallel degree is set to *MAX only for that job, regardless of the system value.

Example 11-7 Changing parallel degree only for a job

```
CHGQRYA DEGREE(*MAX)
```

This command can be issued multiple times within a job to turn parallelism on and off.

The QAQQINI file also lets you specify parallel processing for an individual job or connection. The QAQQINI file can be used to influence the behavior of the database engine - in this case, the parallel processing behavior. These QAQQINI configuration settings can be saved and applied dynamically across multiple database requests.

The fact that the QAQQINI file is just a normal database table means you can dynamically change the values of different attributes as required by your environment. It's also very flexible, because normal database interfaces can be used to change the values of a QAQQINI attribute. The SQL statement in Example 11-8 demonstrates how to use the QAQQINI file to set the parallel processing degree to *OPTIMIZE.

*Example 11-8 Using the QAQQINI file to set the parallel processing degree to *OPTIMIZE*

```
UPDATE MyLib.QAQQINI  
SET QQVAL = '*OPTIMIZE'  
WHERE QPARM='PARALLEL_DEGREE'
```

Note: There is no limit on the number of times that an attribute value can be changed during a job or connection.

Background Database Server Jobs

Some database requests are executed within the background database server jobs, and are recognized on the job list by the name "QDBSRVxx." These jobs pick up the parallel degree value from the system value QQRYDEGREE prior to handling a request. To change the degree value to something other than the system value, use the CHGQRYA command and specify the particular QDBSRVxx job whose degree value you wish to alter.

11.4.6 Feedback

It is very important to have SMP feedback to see how the optimizer is taking advantage of its benefits. This feedback can be found in Visual Explain and in the SQE Plan Cache Snapshots Dashboard or SQL Performance Monitors.

Visual Explain

CQE depicts parallel database methods using specific icons within the query graph as shown in Figure 11-11. See that CQE used a parallel Table Scan to create a temporary Hash Table.

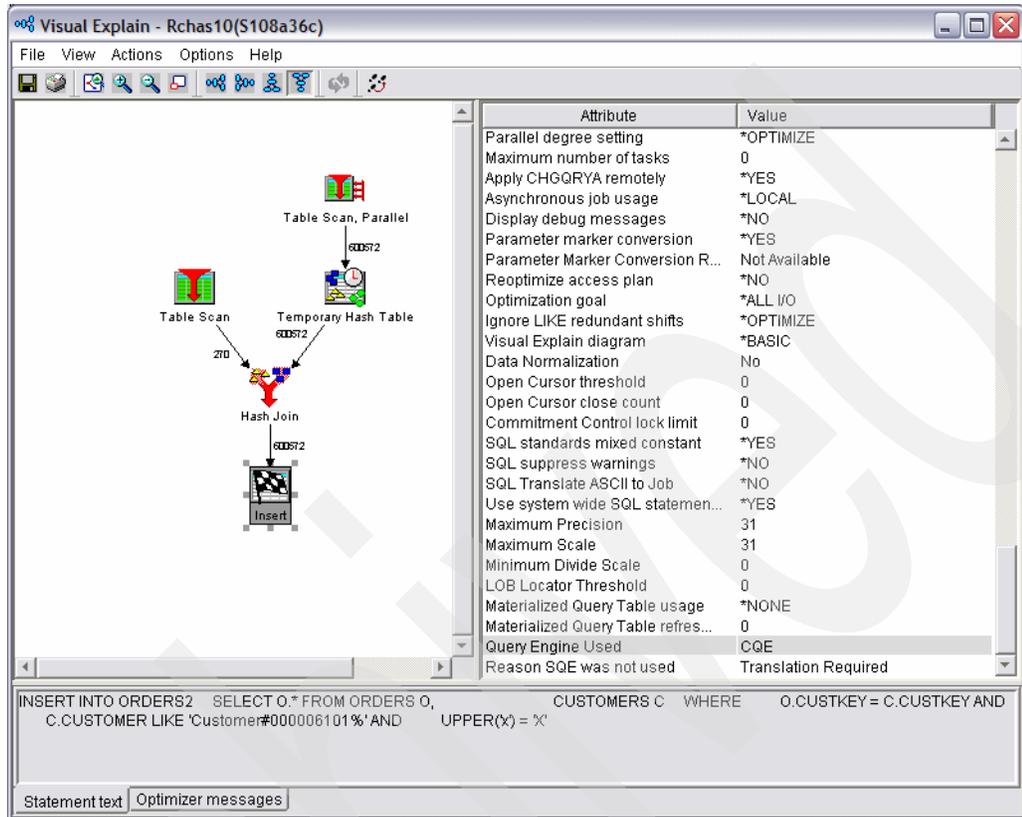


Figure 11-11 Visual Explain of a CQE statement using SMP

SQE depicts database parallelism by using double arrows as shown in Figure 11-12. In these areas, parallel methods and strategies are used to decrease the run time of the query. For example, see that SQE used a parallel table scan to create a temporary hash table.

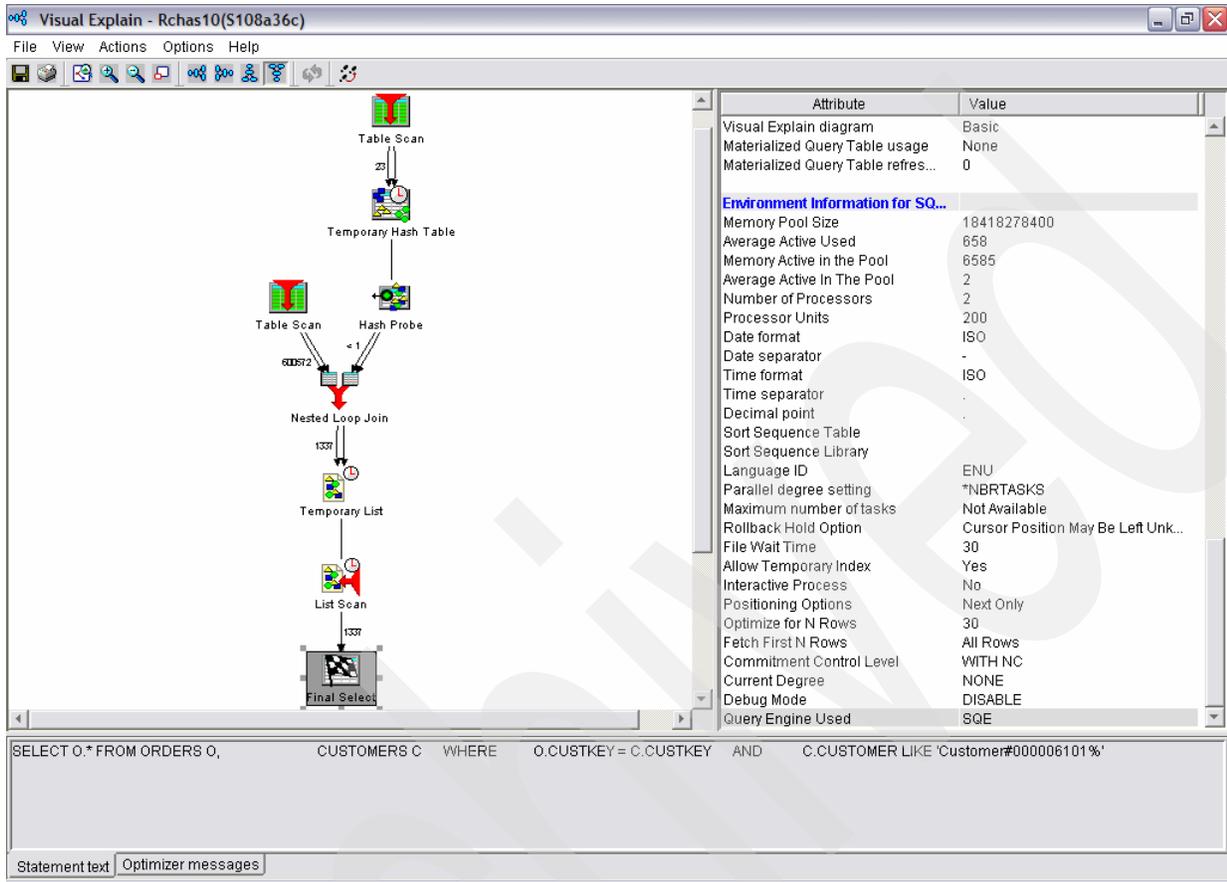
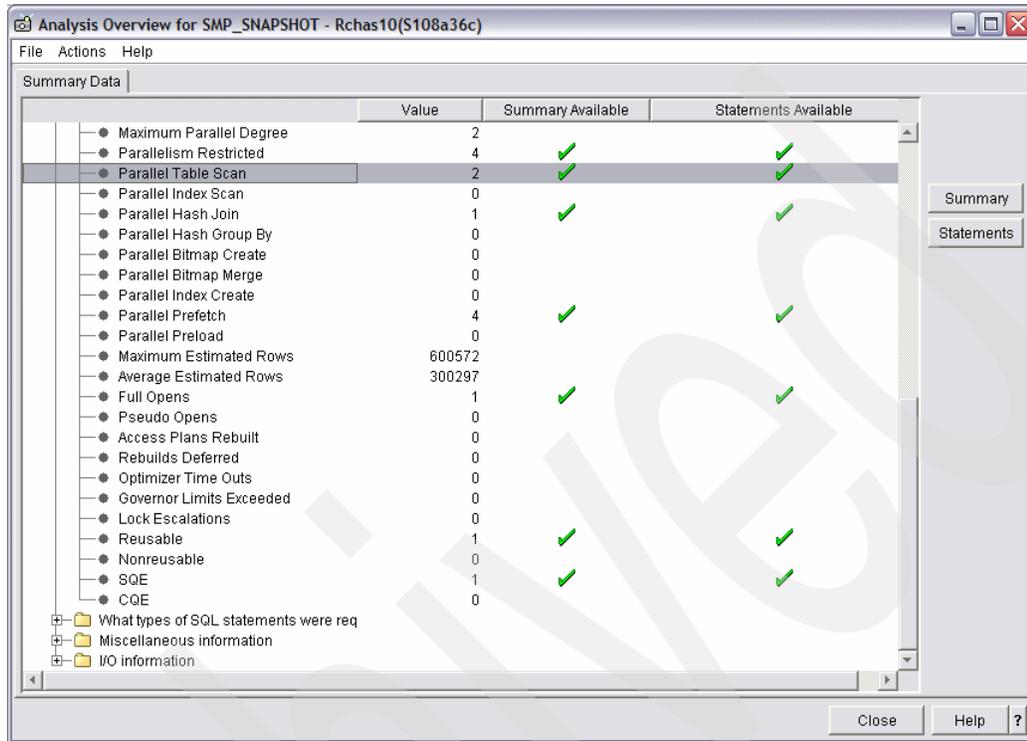


Figure 11-12 Visual Explain showing a SQE statement using SMP

SQE Plan Cache Snapshot

You can see if a SQE query has used SMP from the SQE Plan Cache Snapshot Dashboard as shown in Figure 11-13.



	Value	Summary Available	Statements Available
● Maximum Parallel Degree	2		
● Parallelism Restricted	4		
● Parallel Table Scan	2	✓	✓
● Parallel Index Scan	0		
● Parallel Hash Join	1	✓	✓
● Parallel Hash Group By	0		
● Parallel Bitmap Create	0		
● Parallel Bitmap Merge	0		
● Parallel Index Create	0		
● Parallel Prefetch	4	✓	✓
● Parallel Preload	0		
● Maximum Estimated Rows	600572		
● Average Estimated Rows	300297		
● Full Opens	1	✓	✓
● Pseudo Opens	0		
● Access Plans Rebuilt	0		
● Rebuilds Deferred	0		
● Optimizer Time Outs	0		
● Governor Limits Exceeded	0		
● Lock Escalations	0		
● Reusable	1	✓	✓
● Nonreusable	0		
● SQE	1	✓	✓
● CQE	0		

Figure 11-13 SQE Plan Cache Snapshot Dashboard showing the use of SMP

Database Monitor

The Database Monitor contains data that indicates the use of parallel methods and strategies. This data can be displayed using either handwritten queries or the dashboard provided by the SQL Performance Monitors in iSeries Navigator as shown in Figure 11-14.

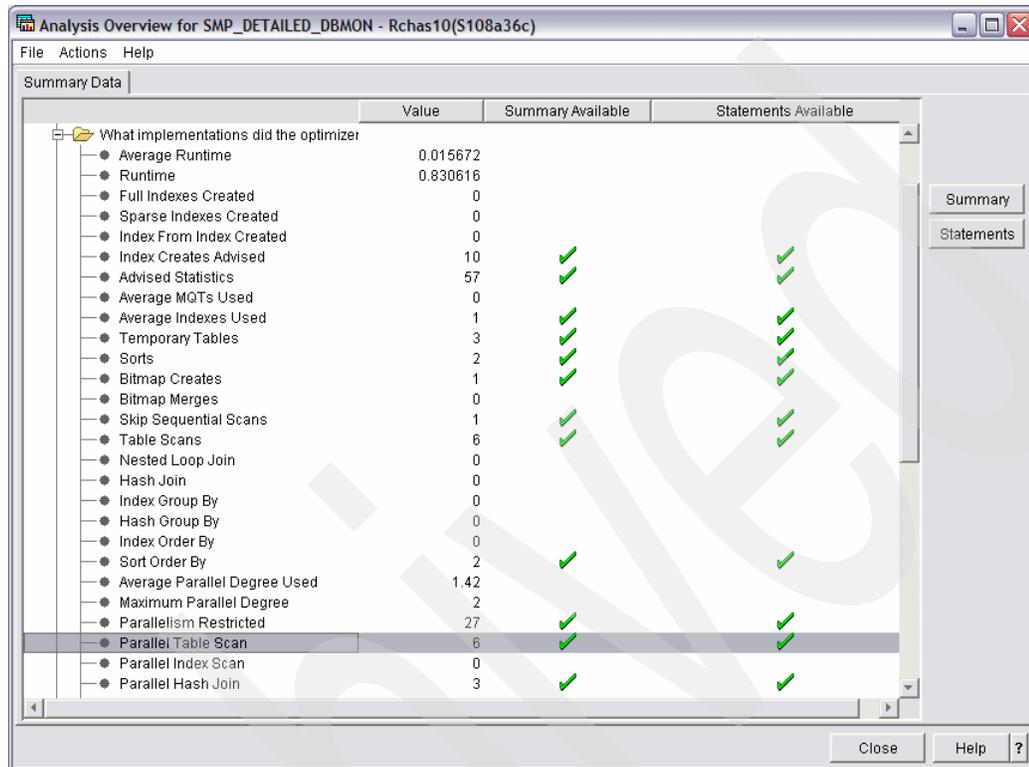


Figure 11-14 SQL Performance Monitor Dashboard showing the use of SMP

11.4.7 Available and balanced resources

To take advantage of SMP, the computing resources must be available and balanced. In other words, the system or LPAR configuration must have a good balance of CPU, memory, and disk units. Otherwise, the optimizer will not be able to make the best use of SMP. For example, a robust set of CPU resources must be supported with a robust set of I/O resources; else the CPUs will be waiting for I/O requests to be fulfilled. On the other hand, if the CPU resources are limited, then the amount of parallelism will also be limited.

Enabling SMP for a given job will allow that job to use multiple tasks or threads to perform the work. Those multiple tasks or threads will consume more resources with the goal of faster response times. Multiple CPUs will allow the tasks or threads to execute in parallel. In other words, more work will be accomplished in the same unit of time.

Distribute adequate resources

Assume a given job demands one unit of resources without SMP. Running 10 of those jobs simultaneously would place 10 units of demand on the system resources.

Now assume a given job demands five units of resources with SMP. Running 10 of those jobs simultaneously would place 50 units of demand on the system resources.

In other words, enabling the optimizer to choose parallel methods and strategies for the job will also allow that job to consume more resources. Running multiple jobs with SMP may

cause even more resources to be consumed. If the resources are not available, the computing resources will be overcommitted and the throughput will be reduced.

Important: When considering the use of SMP, ensure there will be enough resources available to handle the increased demand.

Work Management and SMP

With SMP-enabled workloads, Work Management processes are the same, and yet different. Work Management is the same given that the tasks and threads use the parent job's run priority and memory pool. Work Management is different given that the tasks and threads are working on behalf of the parent job.

To prevent the database engine from consuming all the computing resources for parallelism, the SMP degree also controls the optimizer. During query optimization, the job's fair share of memory is calculated. This fair share helps to determine how aggressive the query plan will be, including the amount of parallelism. All of the parallel degree values except *MAX, cause the optimizer to use a fair share of the job's memory pool. The value *MAX allows the optimizer to consider using all of the job's memory pool, which results in a more aggressive query plan and possibly more use of SMP.

When the parallel degree value is *NONE, *IO, *OPTIMIZE, or *NBRTASKS; the fair share of memory is determined using the following formula:

CQE fair share = memory pool size / max-active value

SQE fair share = memory pool size / min(max-active, max(avg-active, 5))

Where the "avg-active" is either:

- ▶ A 15-minute rolling average of the number of users when the paging option is set to *CALC, or
- ▶ The number of unique users when the paging option is set to *FIXED.

The Max-active value for the memory pool can be viewed and changed via the **WRKSYSSTS** command or iSeries Navigator (by selecting **Work Management** → **Memory Pools**).

11.4.8 SMP considerations

Conditions that prevent parallelism

Parallel access methods may not be used for queries that require any of the following:

- ▶ Use of the *ALL or *RS commitment control level, or repeatable read isolation level
- ▶ Restoration of the cursor position on rollback
- ▶ Scrollable cursor
- ▶ Nested loop join by key implementation (CQE)
- ▶ Update- or delete-capable cursor

Parallel methods can be used on any intermediate temporary result regardless of the interface used to define the query.

There are some application environments in which SMP cannot be employed because parallel database techniques are not allowed, do not help, or are not available. When considering or evaluating the use of SMP, it is important to understand the application environment and attributes.

SMP may be used for some parts of the query plan and not for others. If the query plan uses temporary, intermediate results; these results may be processed with parallel methods. An example is the creation of a temporary index to support the query. To speed up the processing, the temporary index can be created in parallel, but accessed without parallelism.

It is important to remember that SQL does not guarantee the query results will be returned in any particular order. As a matter of fact, the same query executed multiple times may return the results in a different order each time. One reason for this phenomenon is the use of SMP. When running in parallel, the rows can be processed and returned in any order by the multiple tasks or threads. To ensure a consistent ordering of the query results, specify the SQL ORDER BY clause.

Application environments, computing resources

When considering the use and benefits of SMP, remember that some query requests may benefit from SMP and others may not. Similarly, some environments can support parallel database techniques while others may not. For example, if a query request is accessing a very small set of rows via an index, then that particular query will not benefit from SMP. On the other hand, if a query request is accessing a majority of the rows via a table scan, then that particular query may benefit from SMP by allowing the optimizer to run the table scan in parallel.

If the query request is expected to access and process many rows, then allowing the request to run in parallel can increase the response time of the query. This increase in response time is not magical or free, it is the result of using *more* resources.

Following are the application environments that can use and benefit from parallelism:

- ▶ SQL requests that use parallel-enabled methods
- ▶ Longer-running or complex SQL queries
- ▶ Longer-running requests such as index creation
- ▶ Few or no concurrent users running in same memory pool
- ▶ Dedication of most or all resources to specific SQL requests

The additional resources must be available during query optimization and query execution. The optimizer uses the static resources that are in place at the time of query optimization. These resources are: the number of CPUs as well as their relative power, memory pool size, and number of disk units that contain the database objects being accessed. If the resources are not balanced or not sufficient to support SMP, then the optimizer will not use parallel methods and strategies.

When a parallel query plan is executed, this plan will adhere to the rules of work management. The parallel tasks or threads running on behalf of the job will compete for resources just like any other job. If there are not sufficient resources available to support the increase in workload, then the resources will become saturated and the overall throughput may decrease. If no other jobs are competing for resources, then the parallel query plan will be free to use all the available resources and gain the benefits of SMP – namely increased throughput and response time.

The best way to understand the benefits of using SMP is to study the particular workload and environment, or run a benchmark. Short of that, some general guidelines can be used to assess whether or not SMP may be beneficial. A candidate SMP environment should have more than one physical (dedicated) CPU, four to eight gigabytes of memory per CPU, and more than 10 disk units per CPU. The CPU utilization should be below 60% for the interval of time when SMP will be considered. For example, assuming an SQL query workload, a nightly batch environment has four CPUs and the average (total) CPU utilization is 25% during the nightly processing. To make use of the remaining CPU resources, SMP can be enabled to

allow parallel database processing. On the other hand, the daily transaction environment has an average (total) CPU utilization of 85% during the daily processing. During this time interval, the extra computing resources are not available and using SMP will not increase the throughput.

SMP considerations: first time tips

When using or experimenting with SMP for the first time, start with the parallel degree set to *OPTIMIZE and analyze the query plan and run time. Set the MAX ACTIVE value for the memory pool to a number that represents the true number of concurrent, active jobs or threads. This will allow the optimizer to determine a realistic fair share of the memory pool for the jobs.

For jobs that are running alone in a memory pool, start by using *OPTIMIZE, and then move to *MAX. *MAX will allow the optimizer to consider using all the memory in the pool, but will also let the database engine use more CPU resources.

Setting the memory pool's paging option to *CALC will allow the database engine to be more intuitive and more aggressive with I/O requests. The value of *CALC allows more parallel, asynchronous I/O.

Based on the application's behavior and SQL interface, the optimizer can use an optimization goal of First I/O or All I/O. The First I/O optimization goal tends to avoid parallel methods and strategies given that these plans are built to deliver the first n rows of the result set as fast as possible, and a faster query startup time is required. The All I/O optimization goal tends to allow parallel methods and strategies, given that (1) these plans are built to deliver the entire result set as fast as possible, and (2) a slower query startup time is acceptable. Using parallel methods and strategies may require some additional initialization time during query startup in favor of a faster overall query run time.

Conclusion

To receive the benefits of parallel database processing, the DB2 Symmetric Multiprocessing feature must be installed and enabled. This feature is included in the Enterprise Edition of i5/OS or can be ordered separately.

DB2 Symmetric Multiprocessing provides high performance database processing and is an excellent way to take advantage of all the available resources within the eServer i5 platform. This feature relies on the integrated technologies found in i5/OS and is an example of the outstanding value provided by the best business computer available today.

For more information about Symmetric Multiprocessing refer to the existing whitepaper: *DB2 Symmetric Multiprocessing for iSeries: Database Parallelism within i5/OS* on the Web at:

http://www-03.ibm.com/servers/enable/site/education/abstracts/4aea_abs.html



Tips to proactively prevent SQL performance problems

It is important to analyze SQL performance problems and to fix them. Most of this book has been focused on how to analyze an existing problem. But for programmers, it may be even more important to know how to prevent those problems from the beginning in order to achieve the best performance.

In this chapter, we help you to understand what the optimizer considers an optimal index, so that you can predict which indexes are necessary. We also provide further tips and information about coding techniques to use and to avoid with your SQL statements, along with steps you can take to maintain your database and keep it running optimally.

12.1 Indexing strategy

The database has two types of permanent objects: tables and indexes. Tables and indexes include information about the object's structure, size, and attributes. In addition, tables and indexes contain statistical information about the number of distinct values in a column and the distribution of those values in the table. The DB2 for i5/OS optimizer uses this information to determine how to best access the requested data for a given query request.

Since the iSeries optimizer uses cost-based optimization, the more information given about the rows and columns in the database, the better the optimizer is able to create the best possible (least costly and fastest) access plan for the query. With the information from the indexes, the optimizer can make better choices about how to process the request (local selection, joins, grouping, and ordering).

The primary goal of the optimizer is to choose an implementation that quickly and efficiently eliminates the rows that are not interesting or required to satisfy the request. A proper indexing strategy assists the optimizer and database engine with this task.

For a complete documentation on Indexing Strategy refer to the existing whitepaper: Indexing and Statistics Strategy for DB2 for i5/OS found at:

<http://www-03.ibm.com/servers/enable/site/education/ibo/register.html?indxng>

Note: Although you cannot specify indexes in an SQL statement, the optimizer uses them to implement the best access plan to get access to the requested data. You can only specify tables and views in SQL statements.

12.1.1 Access methods

As stated previously, the database has two types of permanent objects (tables and indexes). There are several methods or algorithms that can be used with these objects:

- ▶ Table scan

With a table scan, all rows of a table are processed regardless of the selectivity of the query. Deleted records are examined even though none are selected.

- ▶ Table probe

A table probe operation is used to retrieve a specific row from a table based upon its row number. The row number is provided to the table probe access method by some other operation that generates a row number for the table. This can include index operations as well as temporary row number lists or bitmaps.

- ▶ Index scan

With an index scan, all keys from the index are processed. The resulting rows are sequenced based upon the key columns. This characteristic is used to satisfy a portion of the query request (such as ordering or grouping).

- ▶ Index probe

An index probe reads like an index scan in a keyed sequence, but the requested rows are first identified by a probe operation.

12.1.2 Guidelines for perfect indexes

Typically you create an index for the most selective columns and create statistics for the least selective columns in a query. By creating an index, the optimizer knows that the column is selective, which gives the optimizer the ability to choose that index to implement the query.

In a perfect radix index, the order of the columns is important. In fact, it can make a difference as to whether the optimizer uses it for data retrieval at all. As a general rule, order the columns by placing the equal predicates first, since predicates using the equal (=) operator generally eliminate the largest number of nonparticipating rows. Within the equal predicates, place the most selective of the columns first in the index.

At a minimum, for a given table in a query you want to have a radix index over the selection columns and another over any join columns, or, a single index with the selection criteria followed by the join criteria. Additionally, each of the following indexes may benefit your basic indexing strategy, if applicable:

- ▶ Local selection columns followed by grouping columns
- ▶ Local selection columns followed by ordering columns
- ▶ Ordering columns followed by local selection columns

Note: Indexes consume system resources, so you need to find a balance between query performance and system (index) maintenance.

The query shown in Example 12-1 uses the ITEMS table and finds all the customers who returned orders at year end 2000 that were shipped via air. We illustrate the perfect indexes for this query.

Example 12-1 One table query

```
SELECT CUSTOMER, CUSTOMER_NUMBER, ITEM_NUMBER
FROM ITEMS
WHERE "YEAR" = 2000
      AND "QUARTER" = 4
      AND RETURNFLAG = 'R'
      AND SHIPMODE = 'AIR'
ORDER BY CUSTOMER_NUMBER, ITEM_NUMBER
```

The query has four local selection predicates and two ORDER BY columns.

Following the guidelines, the perfect index places the key columns first that cover the equal predicates ("YEAR", "QUARTER", RETURNFLAG, SHIPMODE), followed by the ORDER BY columns CUSTOMER_NUMBER, ITEM_NUMBER as specified in Example 12-2.

Example 12-2 Perfect index for the one table query example

```
CREATE INDEX MySchema/ItemIdx01
ON MySchema/Items
("YEAR", "QUARTER", ReturnFlag, ShipMode,
Customer_Number, Item_Number)
```

Encoded-vector index guidelines

An encoded-vector index (EVI) cannot be used for grouping or ordering and has a limited use in joins. Single key EVIs can be used to create bitmaps or relative record number (RRN) lists that can be used in combination with binary radix tree indexes. You might use this technique

when the local selection contains AND or OR conditions and a single index does not contain all the proper key columns or a single index cannot meet all of the conditions.

If you look at the query in Example 12-1, you see four local selection predicates and two ORDER BY columns. Following the EVI guidelines, single key indexes are created with key columns covering the equal predicates as shown in Example 12-3.

Example 12-3 EVIs for the one table query example

```
CREATE ENCODED VECTOR INDEX MySchema/ItemsEVI_Year
  ON MySchema/Items ("YEAR");

CREATE ENCODED VECTOR INDEX MySchema/ItemsEVI_Quarter
  ON MySchema/Items ("QUARTER");

CREATE ENCODED VECTOR INDEX MySchema/ItemsEVI_ReturnFlag
  ON MySchema/Items (RETURNFLAG);

CREATE ENCODED VECTOR INDEX MySchema/ItemsEVI_ShipMode
  ON MySchema/Items (SHIPMODE);
```

12.1.3 Additional indexing tips

Keep in mind that indexes are used by the optimizer for the optimization phase or the implementation phase of the query. For example, you might create an index that you do not see is being used by the optimizer for the implementation phase, but it might have been used for the optimization phase. There are some additional considerations to remember where the optimizer might not use an index:

- ▶ Avoid NULL capable columns if expecting to use index only access. When any key in the index is NULL capable, the index-only access method cannot be used with the Classic Query Engine (CQE).
- ▶ Avoid derived expressions in local selection. Access via an index might not be used for predicates that have derived values. Or, a temporary index is created to provide key values and attributes that match the derivative. For example, if a query included one of the following predicates, the optimizer considers that predicate to be a derived value and might not use an index for local selection:

```
T1.ShipDate > (CURRENT DATE - 10 DAYS)
UPPER(T1.CustomerName) = "SMITH"
```

- ▶ Index access is not used for predicates where both operands come from the same table. For example, if the WHERE clause contains the following snippet, the optimizer does not use an index to retrieve the data since it must access the same row for both operands.
T1.ShipDate > T1.OrderDate
- ▶ Consider *index only access (IOA)*. If all of the columns used in the query are represented in the index as key columns, the optimizer can request index only access. With IOA, DB2 for i5/OS does not have to retrieve any data from the actual table. All of the information required to implement the query is available in the index. This might eliminate the random access to the table and drastically improve query performance.
- ▶ Use the most selective columns as keys in the index. Give preference to columns used in equal comparisons.
- ▶ For key columns that are unique, use a unique constraint.
- ▶ Make sure that statistics exist for the most and least selective columns for the query.

12.1.4 Index Advisor

As we have discussed in earlier chapters, the optimizer has several feedback mechanisms to help you identify the need for an index. To help you to identify the indexes advised for a given query request, you can use the following methods:

- ▶ Running and analyzing an SQL statement using Visual Explain Index Advisor in iSeries Navigator
Refer to 9.5.4, “Access to Index Advised information from Visual Explain screen interface” on page 339.
- ▶ Index advised reports available from the Analysis Overview dashboard
5.1.1, “Analysis overview” on page 119.
- ▶ New V5R4 System-wide Index Advisor
Refer to 9.4, “Index Advisor interface in iSeries Navigator” on page 327.

12.2 Coding of your SQL statements

It is not possible to force the optimizer to use a particular index, but you can affect the optimizer’s decision by coding the SQL select statement in a particular manner. In this section, we show you some ways in which you can influence the optimizer.

Note: As you will see, at V5R4, the SQE optimizer compensates for many of these “to-be-avoided” types of SQL statements and is able to use indexes. However, CQE will usually only be able to implement these with a table scan. So, if many of your queries still use CQE, then you will want to pay special attention to this section.

12.2.1 Avoid using logical files in your select statements

It is a common misunderstanding that, by specifying logical files in SQL statements, the query optimizer can be forced to choose this index. This is not true. The specified index can be selected if it meets all the requirements for the best access path. But the query optimizer can choose any other index or even decide to do a table scan.

If you specify a keyed logical file in an SQL select statement, the optimizer takes only the column selection, join information, and any select or omit clauses, and rewrites the query. For each table in a join logical file a specific index is determined.

There is another reason to avoid specifying logical files in an SQL statement. A select statement that contains logical files is rerouted and executed by CQE and does not benefit from the enhancements of the new SQL Query Engine (SQE). The cost of rerouting might cause an overhead of up to 10% to 15% in the query optimization time.

We illustrate this with an example. Suppose that we have a base table ORDERS that contains all order information. Two logical files are created over the table using DDS.

Example 12-4 show the DDS description of the keyed logical file ORDERSL1, with the key fields ORDERDATE and ORDERKEY.

Example 12-4 Logical file ORDERSL1 DDS

R	ORDERSF	PFILE (ORDERS)
K	ORDERDATE	
K	ORDERKEY	

Example 12-5 shows the DDS description of the second logical file ORDERSL2, with the key fields SHIPDATE in descending order, CUSTKEY, and ORDERKEY.

Example 12-5 Logical file ORDERSL2

R	ORDERSF	PFILE (ORDERS)
K	SHIPDATE	DESCEND
K	CUSTKEY	
K	ORDERKEY	

Now we want to select all orders with the order date 14 July, 2004. In the first case, we perform the select explicitly specifying the logical file ORDERSL2, and in the second case, we specify the physical file.

Figure 12-1 shows the Visual Explain diagram for both statements. In both cases, the access path of the keyed logical file is used. But in the first case, an Index Scan - Key Positioning is

performed by CQE, while in the second case, an index probe in combination with a table probe is executed by SQE.

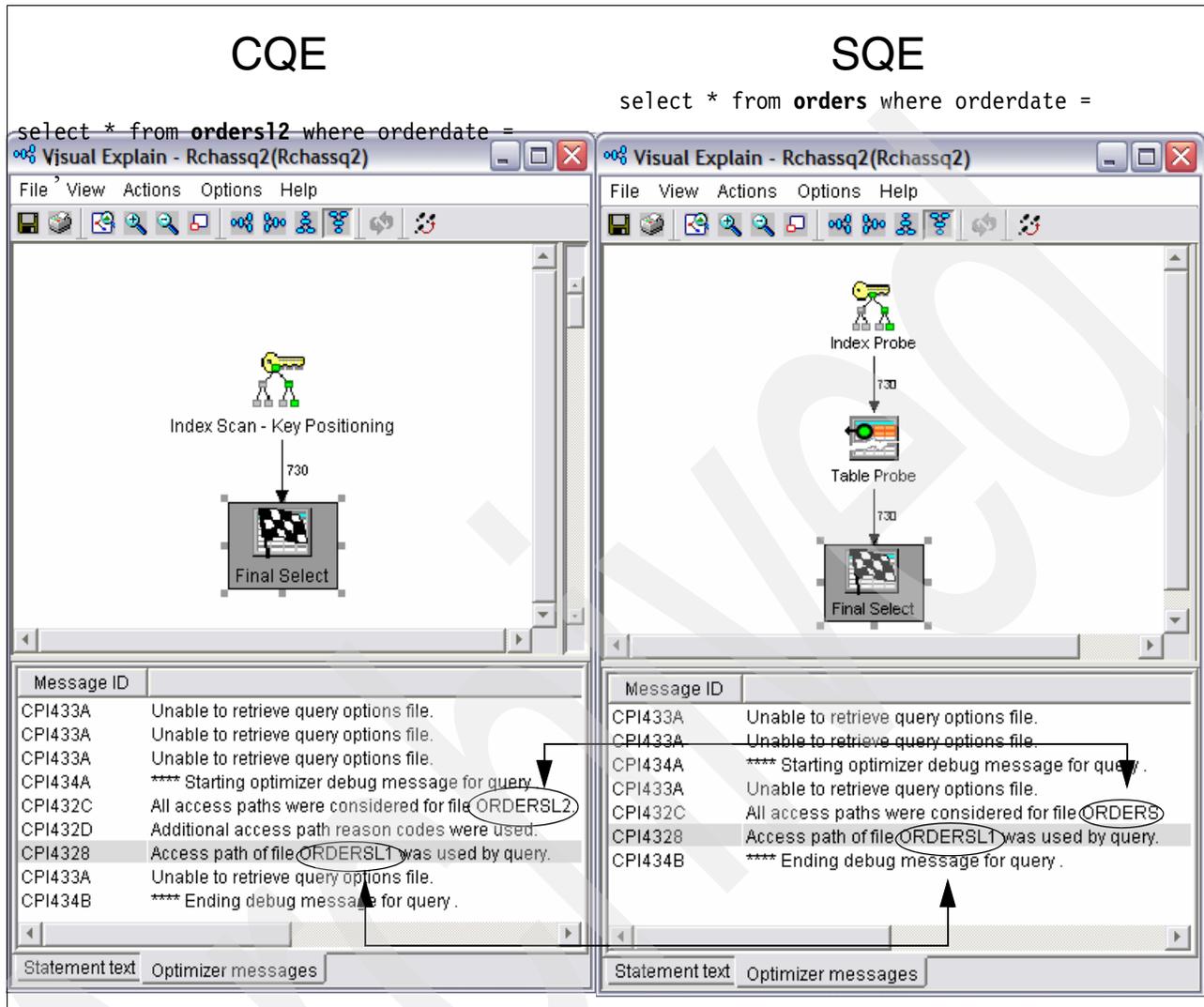


Figure 12-1 Comparing the use of logical and physical files in select statements

12.2.2 Avoid using SELECT * in your select statements

When reading a record with native I/O, all fields and field values are always moved into memory. If you have tables with a lot of columns and you only need information from a few, a lot of unnecessary information must be loaded. With SQL, you can select only the columns that you need to satisfy the data request.

If you specify the required columns in your select statements, the optimizer might be able to perform index only access. With IOA, DB2 for i5/OS does not have to retrieve any data from the actual table. All of the information required to implement the query is available in the index. This might eliminate the random access to the table and substantially improve query performance. Figure 12-2 shows the Visual Explain diagram for the case of SELECT * and

the case of selecting only the required column. Note that in the index-only access diagram, there is no table probe needed.

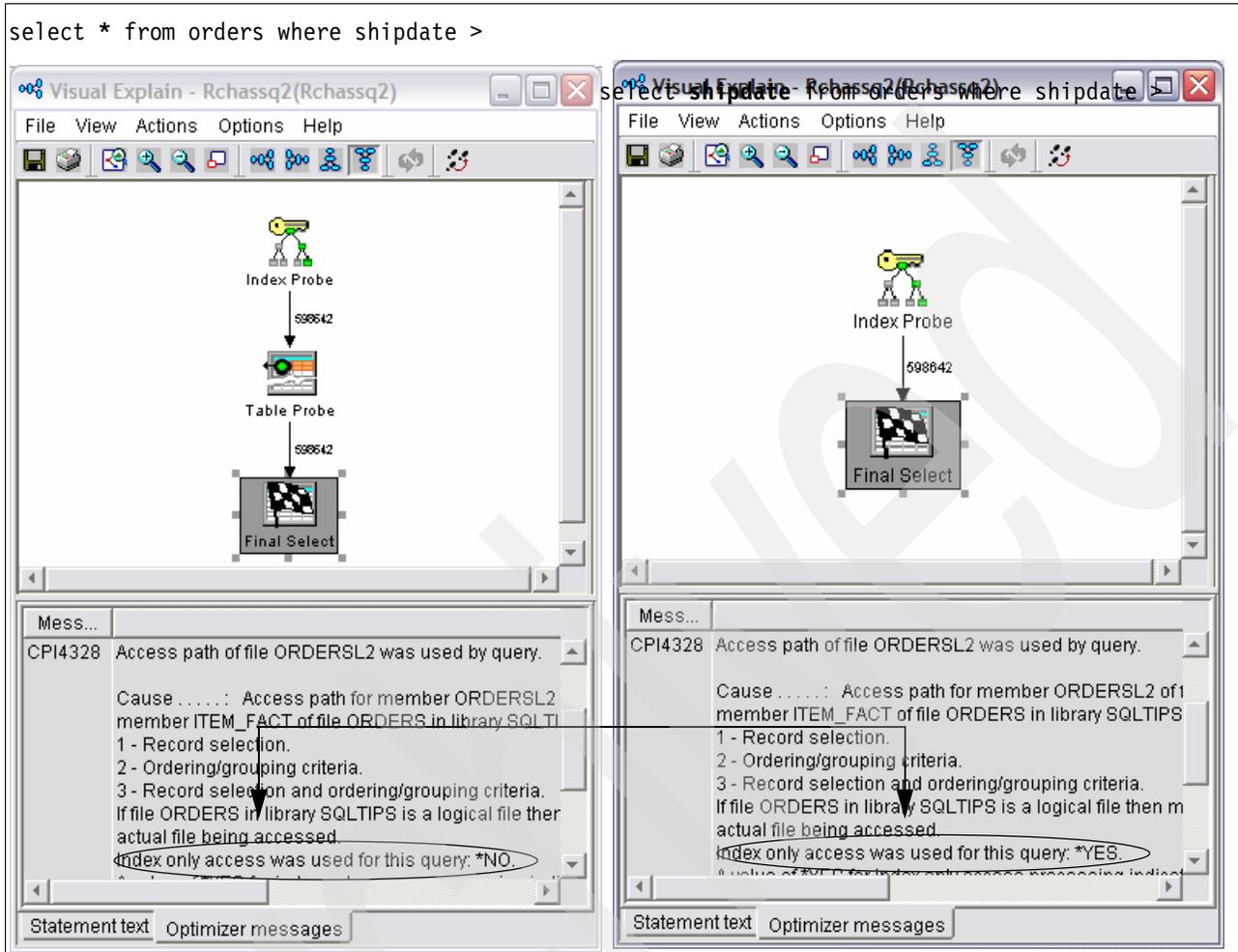


Figure 12-2 Allowing index-only access by specifying the column name rather than SELECT *

12.2.3 Avoid using the relative record number to access your data

For most tables, it is easy to define a primary or unique key. In some cases, such as for transaction tables, it is almost impossible to determine a unique key. With native I/O, the required records are often read using the relative record number. For native I/O, this is the fastest access method.

In SQL, the relative record number can be determined by using the scalar function RRN(). The relative record number is not a defined row in the database table. Therefore, it is not possible to create an index over the relative record number. If you try to read a row using the relative record number, a relative record number scan is performed, which is similar to a table scan.

For tables where you cannot specify a unique key, you must create an additional column to hold a unique value. SQL provides methods, such as identity columns or columns with ROWID data type, where a unique value is automatically generated if a new row is inserted. Identity columns or columns with ROWID data type are ideal for primary or unique key constraints, and it is also possible to create an index over these columns.

Figure 12-3 shows the selection of a specific row using the SQL scalar function RRN() compared to using that row's key.

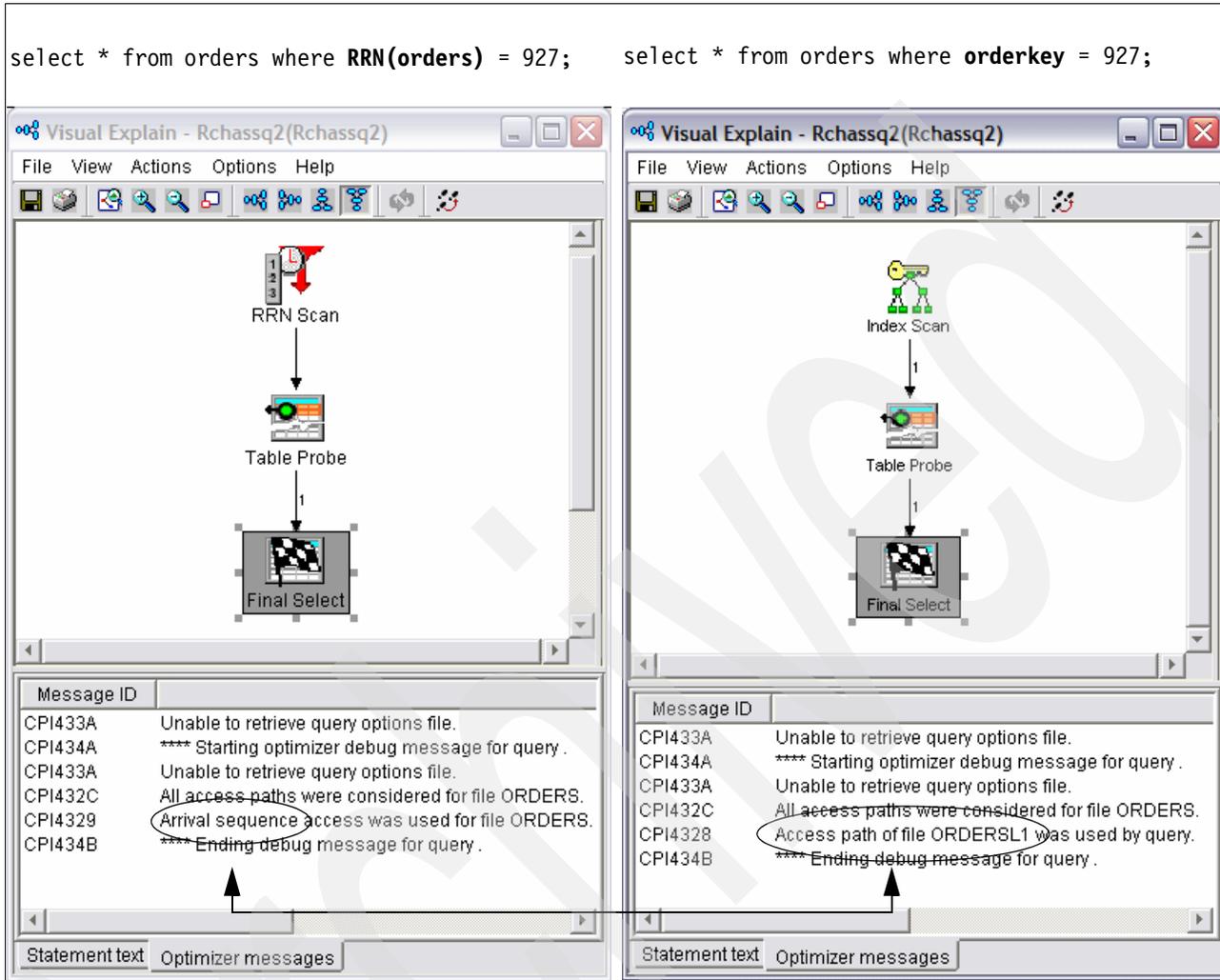


Figure 12-3 Row selection by using the SQL scalar function RRN() compared to using a key

12.2.4 Avoid numeric data type conversion

When a column value and a host variable (or constant value) are being compared, try to specify the same data types and attributes. A query that uses CQE does not use an index for the named column if the host variable or constant value has a greater precision than the precision of the column.

However, if you have different numeric definitions between the column and host variable or constant value, the queries that use SQE *can* use the existing indexes.

If different numeric definitions exist, and no index is available, only SQE is able to suggest an index.

To avoid problems for columns and constants being compared, use:

- ▶ The same data type
- ▶ The same scale, if applicable
- ▶ The same precision, if applicable

Figure 12-4 shows an SQL statement that selects all orders with an order quantity greater than 50. The QUANTITY column is defined with a scale of 2. In the WHERE clause, we used a scale of 4. We executed the query twice. The first time we specified a logical file in order to cause the query to use CQE. The second time we specified the table and the query was executed by SQE.

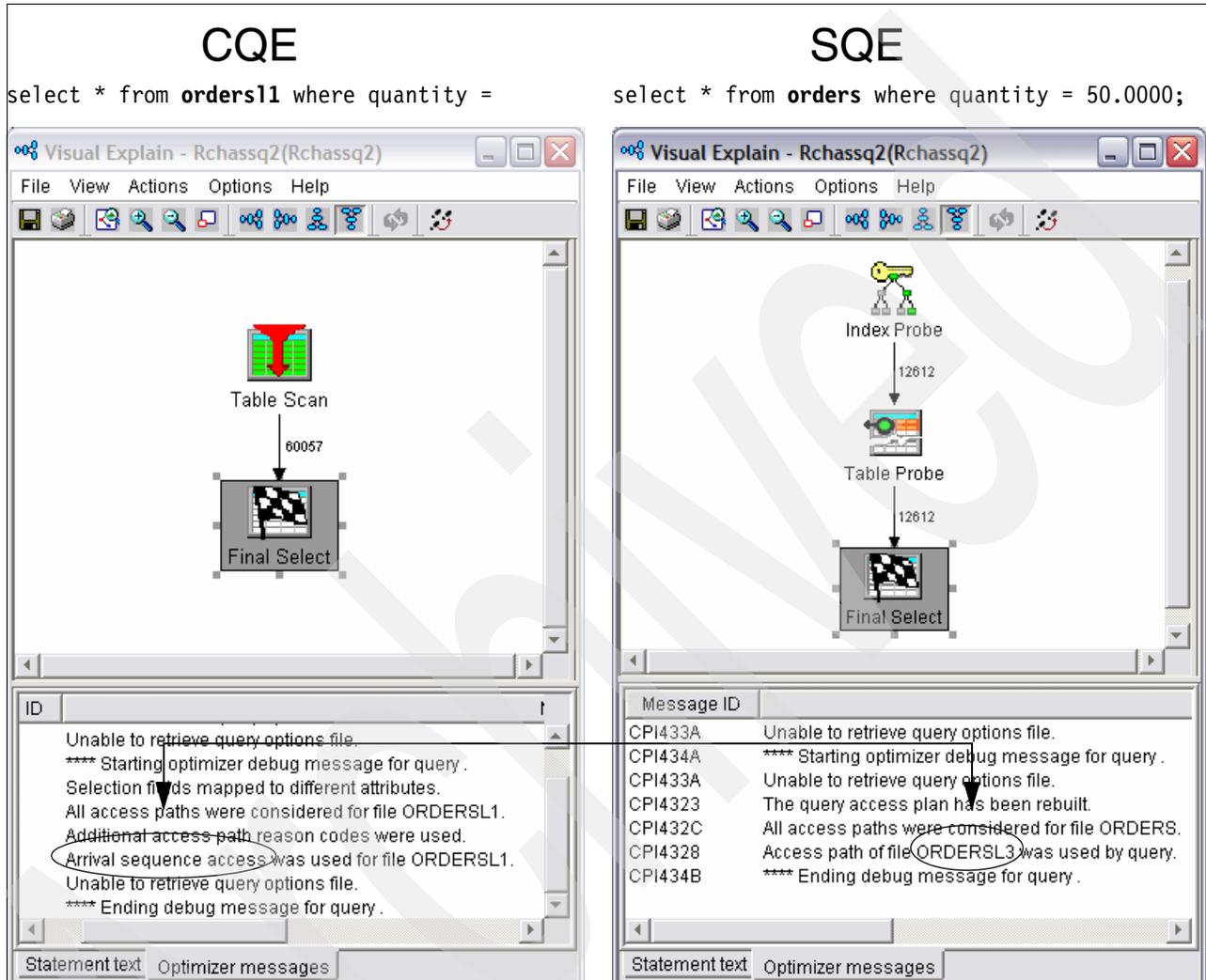


Figure 12-4 Row selection using a different numeric data type in the where clause

With SQE, an index can be used, while with CQE, because of the differing scales, no suitable index can be found and a table scan is performed.

12.2.5 Avoid numeric expressions

When possible, avoid using an arithmetic expression as an operand to be compared to a column in a local selection predicate when your query may be optimized by CQE. CQE may not be able to use an index on a column that is being compared to an arithmetic expression. Furthermore, this may prevent the index from being used for any estimates.

Example 12-6 shows part of an SQL stored procedure that selects the orders for the next seven days using an explicitly specified logical file. The new date is calculated in a numeric expression which is the right operand of the equal predicate.

Example 12-6 Using numeric expressions in an SQL stored procedure

```
CREATE PROCEDURE NUMERIC_EXPRESSION ()
    LANGUAGE SQL
    SPECIFIC NUMERIC_EXPRESSION
    NOT DETERMINISTIC
    MODIFIES SQL DATA
BEGIN

DECLARE OrderCount INTEGER ;

Select Count(ORDERKEY) into OrderCount
from ORDERSL1
Where ORDERDATE = Current_Date + 7 Days;
END;
```

Instead of a numeric expression in the SQL statement, Example 12-7 shows that a host variable is defined and the new date is filled in from a calculation made earlier in the code.

Example 12-7 Using host variables instead of numeric expressions in SQL

```
CREATE PROCEDURE HOST_VARIABLE ()
    LANGUAGE SQL
    SPECIFIC HOST_VARIABLE
    NOT DETERMINISTIC
    MODIFIES SQL DATA
BEGIN

DECLARE OrderCount INTEGER ;
DECLARE NextWeek DATE;

SET NextWeek = Current_Date + 7 Days;

Select Count(ORDERKEY)
into OrderCount
from ORDERSL1
Where ORDERDATE = NextWeek;
END;
```

Figure 12-5 shows the Visual Explain diagram for both SQL Statements. Notice the table scan when using the numeric expression for the CQE query. When a host variable is used instead, CQE is able to utilize an existing index.

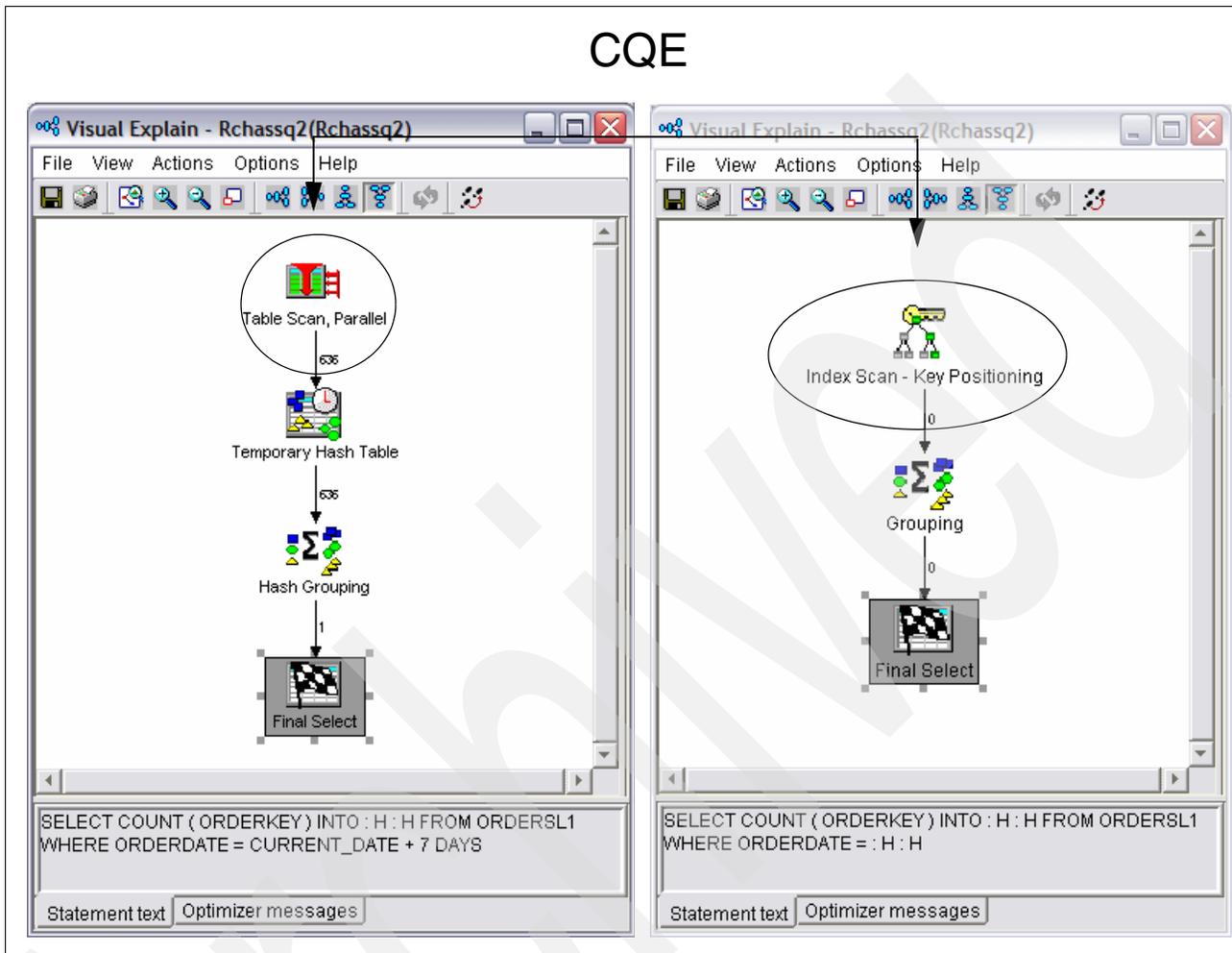


Figure 12-5 Using numeric expressions (right) or host variables (left) in an SQL stored procedure with CQE

As noted, SQE at V5R4 is able to execute the query using an index in either case, as shown in Figure 12-6. So, having a numeric expression in a local selection predicate probably won't affect the performance if your query is using SQE. But, since there are still various reasons

why a query may be routed to CQE, putting the numeric expression in a separate host variable is a good programming practice.

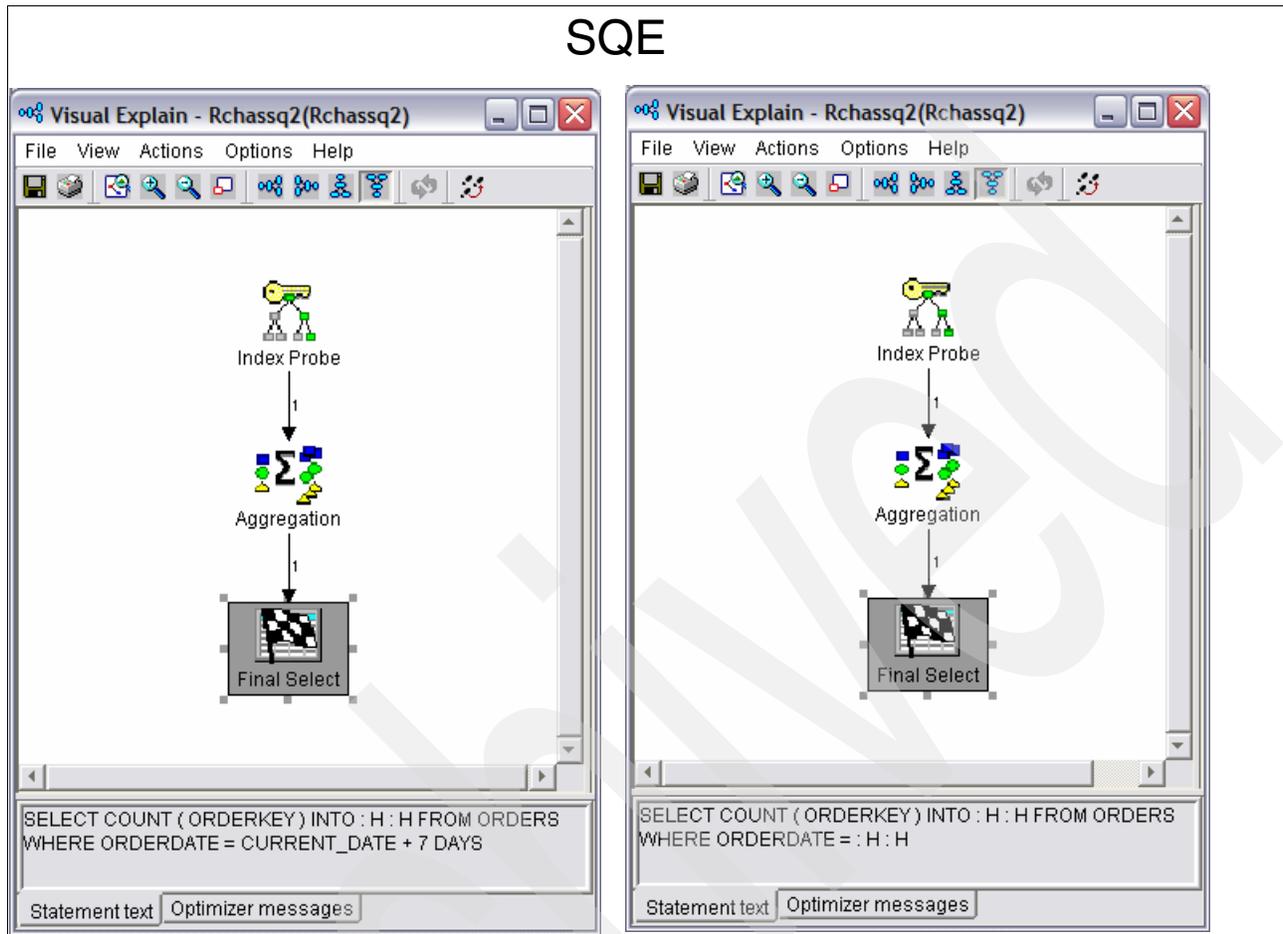


Figure 12-6 SQE implements with an index in either case

12.2.6 Using the LIKE predicate

The percent sign (%) and the underline (_), when used in the pattern of a LIKE predicate, specify a character string that is similar to the column value of rows that you want to select. Such queries can take advantage of indexes when used to denote characters in the middle or at the end of a character string. However, when used at the beginning of a character string, the wildcards can prevent CQE from using any indexes that might be defined in the appropriate column.

Beginning in V5R4, LIKE statements are eligible for optimization by SQE, which *is* able to use an index to implement a LIKE predicate when the search string begins with a wildcard.

Figure 12-7 shows the difference between CQE and SQE when using the LIKE predicate with the wildcard in the first position of the character expression.

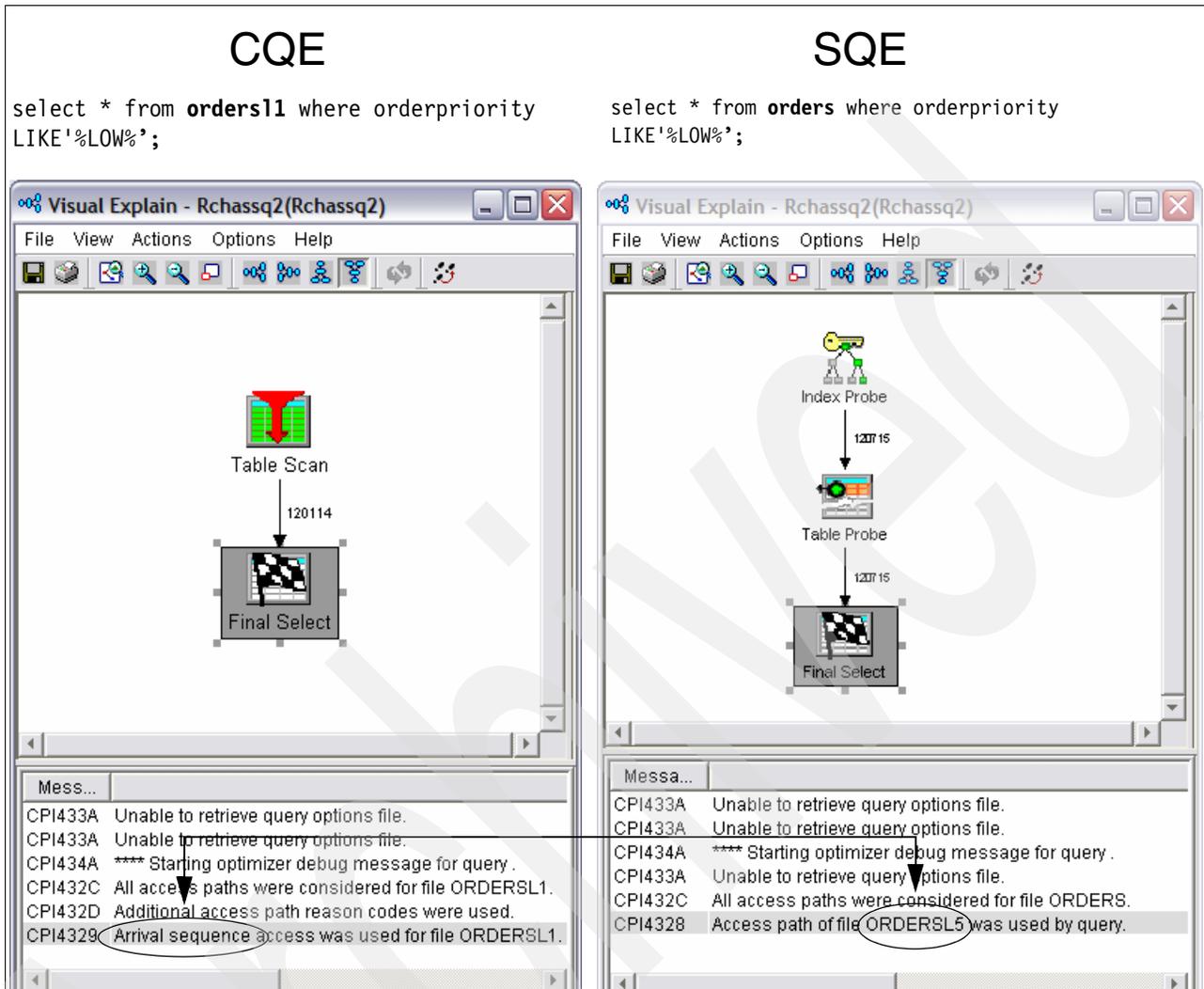


Figure 12-7 Using the LIKE predicate in SQL select statements

Since CQE uses does a table scan, if your query is optimized by CQE for some reason, you may wish to avoid using a LIKE predicate with a wildcard in the first position of the search string.

12.2.7 Avoid scalar functions in the WHERE clause

If a scalar function is used in the WHERE clause, the CQE optimizer might not be able to use an appropriate existing index. In some cases, you can influence the optimizer by rewriting your query in a different manner. For example, if you must select all the orders for a specific year or month, use a range and not the scalar functions YEAR or MONTH.

In the following example, we are getting a count of all orders for the year 2004. As in previous examples, we are specifying the logical file to get a CQE implementation. We first use the scalar function YEAR to select the desired year in the WHERE clause. In the second test, we use a date range instead, using the BETWEEN predicate.

Figure 12-8 shows the Visual Explain diagrams for both SQL statements.

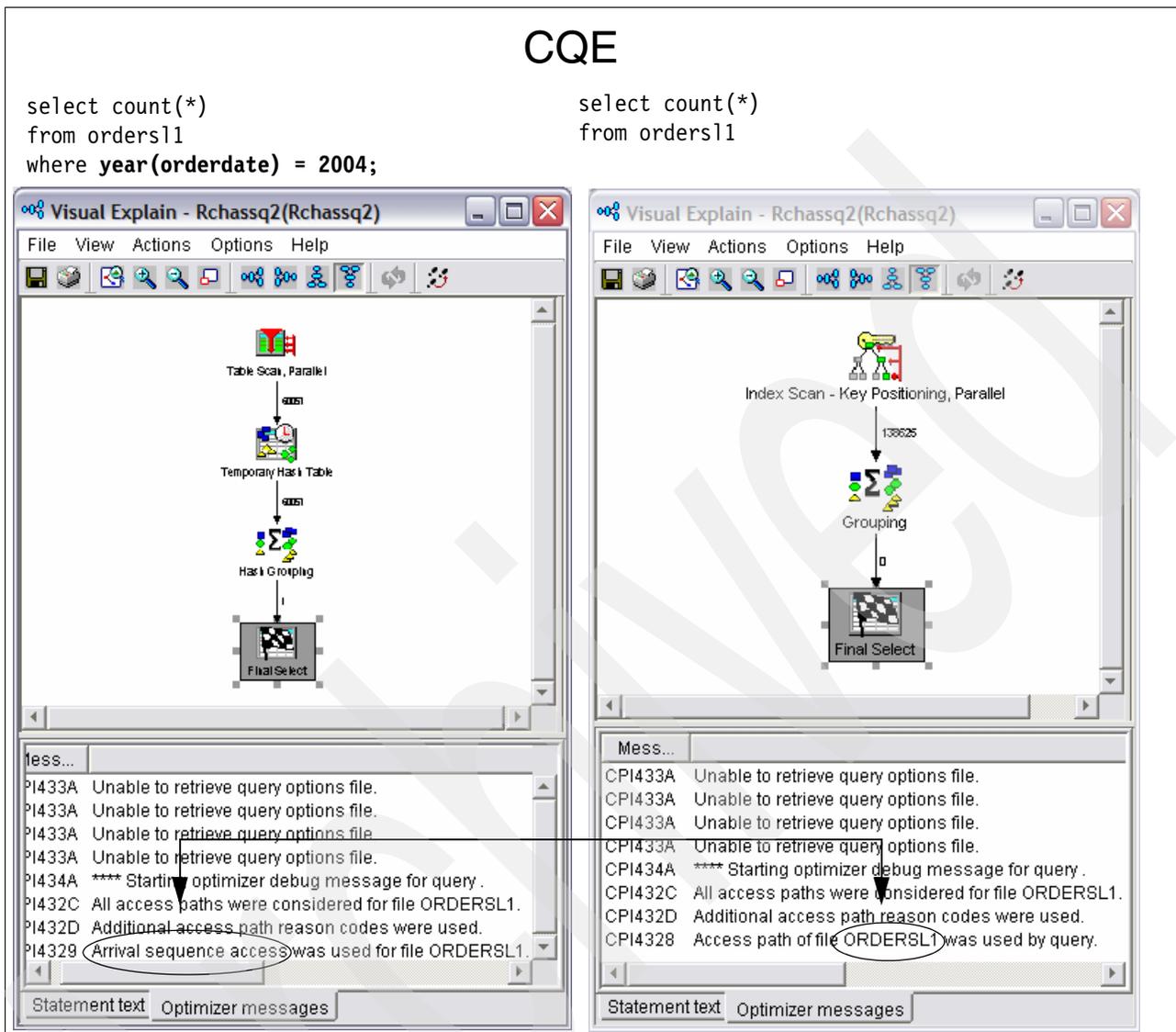


Figure 12-8 Using ranges instead of scalar functions

As you can see, a table scan is done for the case of the scalar function in the WHERE clause, while an index is used for the BETWEEN clause case.

This is another case where the SQE optimizer is now able to handle either case using an index. But, in our testing, the runtime of the SQL statement was significantly better when using the BETWEEN as compared to using the scalar function in the WHERE clause. Therefore, even if your statements are using SQE, you may wish to avoid the use of scalar functions in the WHERE clause to improve performance as shown in Figure .

SQE

select count(*)
from orders

select count(*)
from orders
where **orderdate** between '01/01/04' and

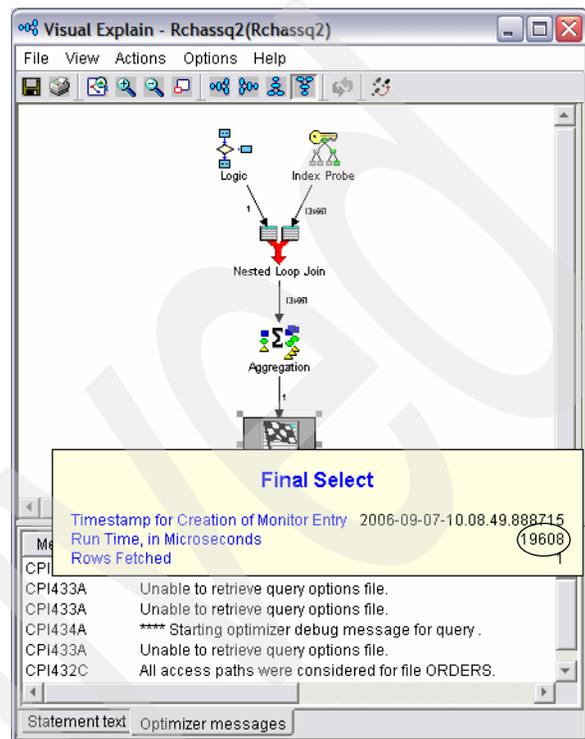
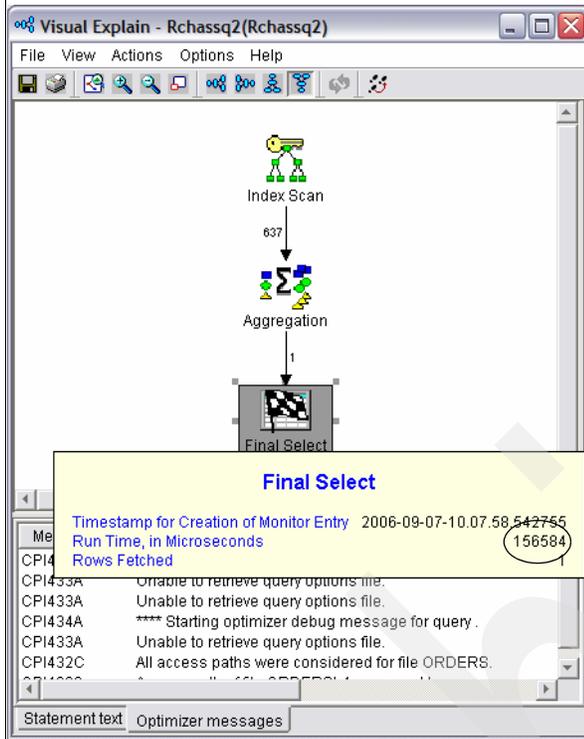


Figure 12-9 SQE can use an index in each case, but performance is better without the scalar function



Using Collection Services data to identify jobs using system resources

This chapter explains how to find jobs using CPU or I/O with Collection Services data and how you can integrate it with the Database Monitor data. This chapter begins by describes how to start Collection Services. Then it guides you in using Collection Services data to find jobs using CPU, to find jobs with high disk I/O counts, and to use it in conjunction with the SQL Performance Monitors. We also discuss using Performance Trace to find jobs that have object locks.

13.1 Relationship of Collection Services, Database Monitor data and Performance Trace

There are times where poor job performance is caused by insufficient system resources. There are many options available to help you to identify and resolve performance problems. In this topic, we discuss using Collection Services, Database Monitor and Performance Trace in helping you to achieve this goal in relation to optimizing your SQL performance.

The Collection Services have reporting facilities that allows you to find the sources of these performance problem. There are a few areas where you need more detail than what Collection Services can provide. In this case, you will need Performance Traces to help you. Once you have identified the sources of system resource constraint and taken the necessary actions to eliminate the bottlenecks, and if job performance still remain as an issue, you should proceed to integrate the Collection Services with Database Monitor data to further address SQL optimization.

To achieve optimum performance, you must recognize the interrelationship among the critical system resources and attempt to balance these resources, CPU, disk, main storage, and for communications, remote lines. Each of these resources can cause a performance degradation. You can make use of Collection Services or even Performance Traces to identify them and take action to eliminate them. The areas to look at are:

- ▶ **Processor load:** Determine if there are too many jobs on the system or if some jobs are using a large percentage of processor time.
- ▶ **Main storage:** Investigate faulting and the wait-to-ineligible transitions.
- ▶ **Disk:** Determine if there are too few arms or if the arms are too slow.
- ▶ **Communications:** Find slow lines, errors on the line, or too many users for the line.
- ▶ **IOPs:** Determine if any IOPs are not balanced or if there are not enough IOPs.
- ▶ **Software:** Investigate locks and mutual exclusions (mutexes).

Note: For more information about Performance Tools, refer to:

- ▶ IBM Redbook - *A Systems Management Guide to Performance Management for i5 and p5 Systems*, SG24-7122
- ▶ Redpaper - *IBM eServer iSeries Performance Management Tools*, REDP-4026

13.2 Collection Services and Database Monitor data

Collection Services allows you to gather performance data with little or no observable impact on system performance. Its data is analyzed using the IBM Performance Tools for iSeries licensed program (5722PT1) or other performance report applications, iSeries Navigator monitors, and the graph history function. If you prefer to view real-time performance data, system monitors provide an easy-to-use graphical interface for monitoring system performance. For more information about iSeries Navigator Monitors, see “iSeries Navigator monitors” in *Systems Management - Performance Version 5 Release 4* on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseres/v5r4/topic/rzaih/rzaih.pdf>

Collection Services collects data that identifies the relative amount of system resource used by different areas of your system. While you are required to use Performance Tools for iSeries, it is not required to gather the data.

In most of the analysis of job-level information in this chapter, we use the Component Report. The Component Report uses Collection Services data to provide information about the same components of system performance as a System Report, but at a greater level of detail. The Component Report helps you find the jobs that are consuming high amounts of system resources, such as CPU, disk, and so on.

Note: While running Collection Services, you should also run Database Monitor on all the jobs. After we identify the job information of the job or jobs that consume high amounts of system resources, such as CPU, disk and so on, we can query the data to find the jobs as shown in 13.5, “Using Performance Data of the Database Monitor to find the query that needs optimization” on page 446.

13.2.1 Starting Collection Services

To analyze Collection Services data, you must first start Collection Services, which you can do in one of the following ways:

- ▶ From the PERFORM menu, using GO PERFORM
- ▶ From iSeries Navigator
- ▶ Using Performance Management APIs
- ▶ In V5R4, using the Start Performance Collection (STRPFRCOL) CL command

Note: Starting Collection Services from the PERFORM menu requires that you install Performance Tools for iSeries, 5722PT1.

In the following sections, we explain the details for each of the methods to start Collection Services to gather data.

PERFORM menu

To start Collection Services, from the PEFORM menu perform the following steps:

1. Enter the following command:

```
GO PERFORM
```

2. On the IBM Performance Tools for iSeries display, type option 2 (Collect performance data).
3. On the Collect Performance Data display, select option 2 (Configure Performance Collection).
4. The Configure Perf Collection (CFGPFRCOL) display is shown. In setting the configuration, you should consider your system resources, for example, whether you have enough storage to handle the data being gathered. You must consider the following two parameters:
 - *Default interval*: This parameter indicates the sample interval time in minutes to collect data.
 - *Collection retention period*: This parameter indicates how long the management collection object (*MGTCOL) should be retained on the system.

Set these parameters based on your system resources and the problem that you are trying to capture. For example, if you have a performance problem that occurs intermittently over a week, you might set the interval time to 15 minutes and the collection retention period to seven days or longer if you want comparison data, but only if you have the resources to maintain the size of the *MGTCOL objects.

Figure 13-1 shows the selection with collection retention period set to seven days.

```
Configure Perf Collection (CFGPFRCOL)

Type choices, press Enter.

Default interval . . . . . 15.00      *SAME, .25, .50, 1.0, 5.0...
Collection library . . . . . QMPGDATA  Name, *SAME
Default collection profile . . . *STANDARDP *SAME, *MINIMUM, *STANDARD...
Cycle time . . . . . 000000      Time, *SAME
Cycle interval . . . . . 24        *SAME, 1-24 hours
Collection retention period:
  Number of units . . . . . 7        *SAME, 1-720, *PERM
  Unit of time . . . . . *DAYS    *HOURS, *DAYS
Create database files . . . . . *YES    *SAME, *YES, *NO
Change PM iSeries library . . . *NO     *NO, *YES

Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

Figure 13-1 Configure Performance Collection CFGPFRCOL

After you set the configuration, press **Enter**.

5. Type option 1 (Start Performance Collection). Specify *CFG for the collection profile.

For more information, see *Systems Management - Performance Version 5 Release 4* on the Web:

<http://publib.boulder.ibm.com/infocenter/series/v5r4/topic/rzahx/rzahx.pdf>

13.2.2 From iSeries Navigator

In iSeries Navigator, select your system name, then select **Configuration and Service**. Right-click **Collection Services** and select **Start Performance Collection** (as shown in Figure 13-2).

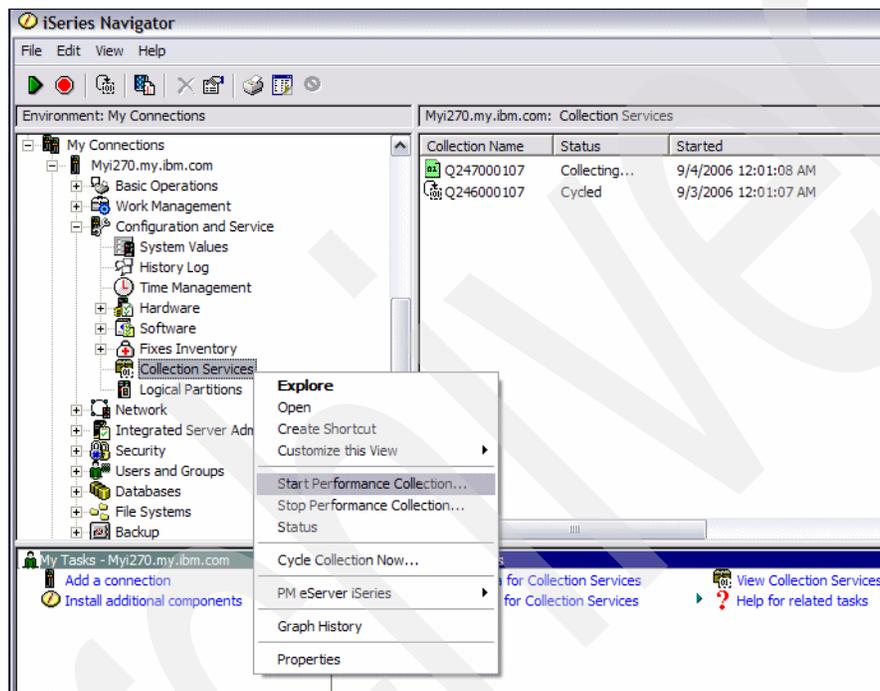


Figure 13-2 Start Performance Collection in iSeries Navigator

In the Start Collection Services window (Figure 13-3), verify the configuration settings and click **OK** to start Collection Services.

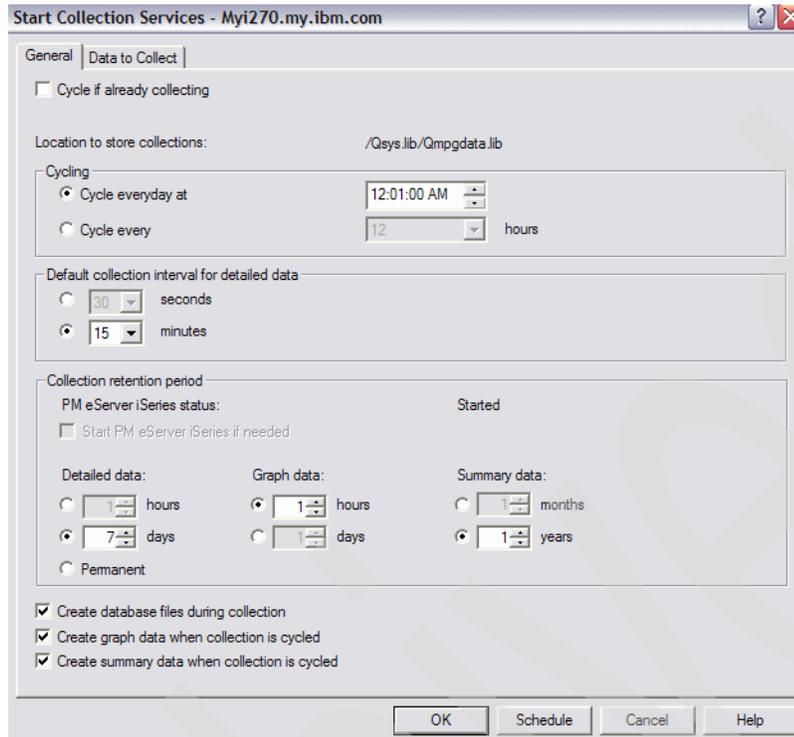


Figure 13-3 Configuring and starting Collection Services in iSeries Navigator

13.2.3 Using Performance Management APIs

You can use Collector APIs to start collecting performance data. The APIs do not require you to have Performance Tools for iSeries installed. To start Collection Services, you can use the following command:

```
CALL QYPSSTRC PARM(' *PFR      ' '*STANDARDP' X'00000000')
```

For more information about the parameters of the API, see “Collector APIs” in *Systems Management - Performance Version 5 Release 4* on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/rzahx/rzahx.pdf>

13.2.4 V5R4 STRPFRCOL command

You can also start V5R4 of Collection Services by using the STRPFRCOL CL command. Then to configure Collection Services, you use the Configure Performance Collection (CFGPFRCOL) CL command.

For more information about the Collection Services CL commands, see *Systems Management - Performance Version 5 Release 4* on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/rzahx/rzahx.pdf>

13.3 Using Collection Services data to find jobs using CPU

You can use Collection Services data to look for jobs using CPU. In this section, we explain how to find jobs that are using CPU by using the following methods:

- ▶ Component Report from the PERFORM menu
- ▶ iSeries Navigator Graph History
- ▶ Management Central System Monitors

Note: When the job information is found, you can use the job information to query the Database Monitor table as explained in 13.5, “Using Performance Data of the Database Monitor to find the query that needs optimization” on page 446.

13.3.1 Finding jobs using CPU with the Component Report

To use the PERFORM menu of IBM Performance Tools for iSeries to gather the Component Report, you must install Performance Tools, product 5722PT1, and then perform the following steps:

1. To access the PERFORM menu, enter the following command:
2. If you want to interactively examine the data, select option **7** (Display performance data). If you choose this option, you must keep in mind your system resources. This job runs interactively and uses CPU and I/O resources. Or you can select option **3** (Print performance report), which submits a job to batch.
3. After you select an option, select the member that you want to investigate, based on the date and time shown. If no members are shown and you have started Collection Services, run the Create Performance Data (CRTPFDRDTA) command against the *MGTCOL object that contains your data or create the files in iSeries Navigator as follows:

```
GO PERFORM
```

- To create the files using the CRTPFDRDTA CL command, find the *MGTCOL object to create the files from by entering the following command:

```
WRKOBJ OBJ(qmpgdata/*ALL) OBJTYPE(*MGTCOL)
```

The attribute of the *MGTCOL object must be *PFR. In the WRKOBJ command shown, you replace *qmpgdata* with the library where you keep your performance data. After the *MGTCOL object is found, you run the following CL command to create the database files:

```
CRTPFDRDTA FROMMGTCOL(library/mgtcolname)
```

- To create the files using iSeries Navigator, select your system name and then select **Configuration and Service**. Right-click **Collection Services**. Then in the right pane, you see a list of the *MGTCOL objects. Right-click the collection name, which is the *MGTCOL object, and select **Create Database Files Now** as shown in Figure 13-4.

For more information, see *Systems Management - Performance Version 5 Release 4* on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseries/v5r4/topic/rzahx/rzahx.pdf>

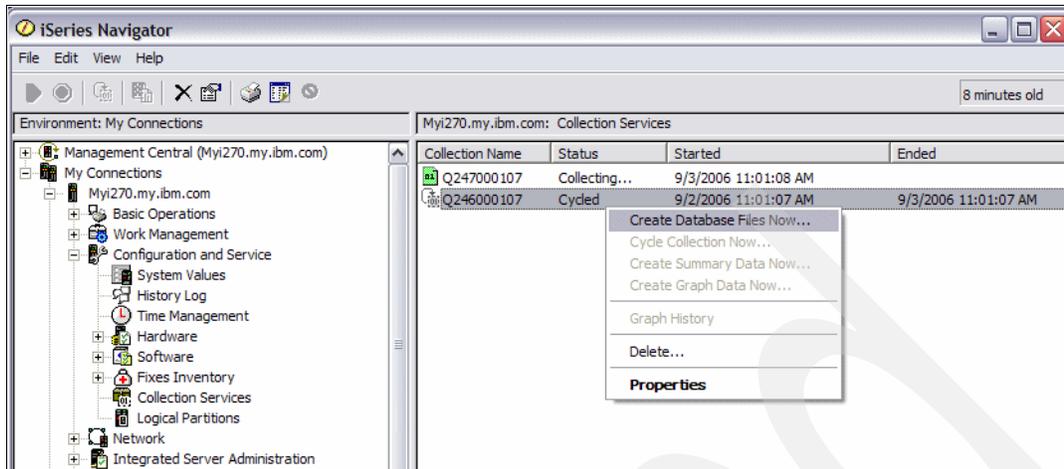


Figure 13-4 Creating files from *MGTCOL objects in iSeries Navigator

The following sections explain how to find the jobs that are using CPU by using either the option to print performance report or the option to display performance data.

Using option 3: Print performance report

When you select option 3, you must specify the library in which you keep your performance data. In most cases, the library is QMPGDATA. You can page up and down until you see a time frame that you want. Type 2 next to the member that you want to use to get a Component Report and then perform the following steps:

1. In the next Select Sections for Report display (Figure 13-5), you can press **F6** to print the entire report to select all the sections to be in your report. Since you must often review all the performance data at the time of high CPU usage, we recommend that you use F6. In

the display shown in Figure 13-5, you see that we choose option 1 to select Job Workload Activity so that we can only see the jobs that are using CPU.

```
                                Select Sections for Report

Member . . . . . : Q247000107

Type options, press Enter. Press F6 to print entire report.
  1=Select

Option   Section
  1      Job Workload Activity
        Component Interval Activity
        Storage Pool Activity
        Disk Activity
        IOP Utilizations
        Local Work Stations
        Remote Work Stations
        Exception Occurrence
        Data Base Journaling Summary
        TCP/IP Activity
        HTTP Server Activity

                                                Bottom

F3=Exit  F6=Print entire report  F12=Cancel
```

Figure 13-5 Select Sections for Report display

2. In the Select Categories for Report display (Figure 13-6), we type option 1 to select Time interval. Time interval is usually the best option if you do not know any other information. If

you want more information, you can press **F6** to print the entire report, but it is best to narrow the information down to a time frame in which you are interested.

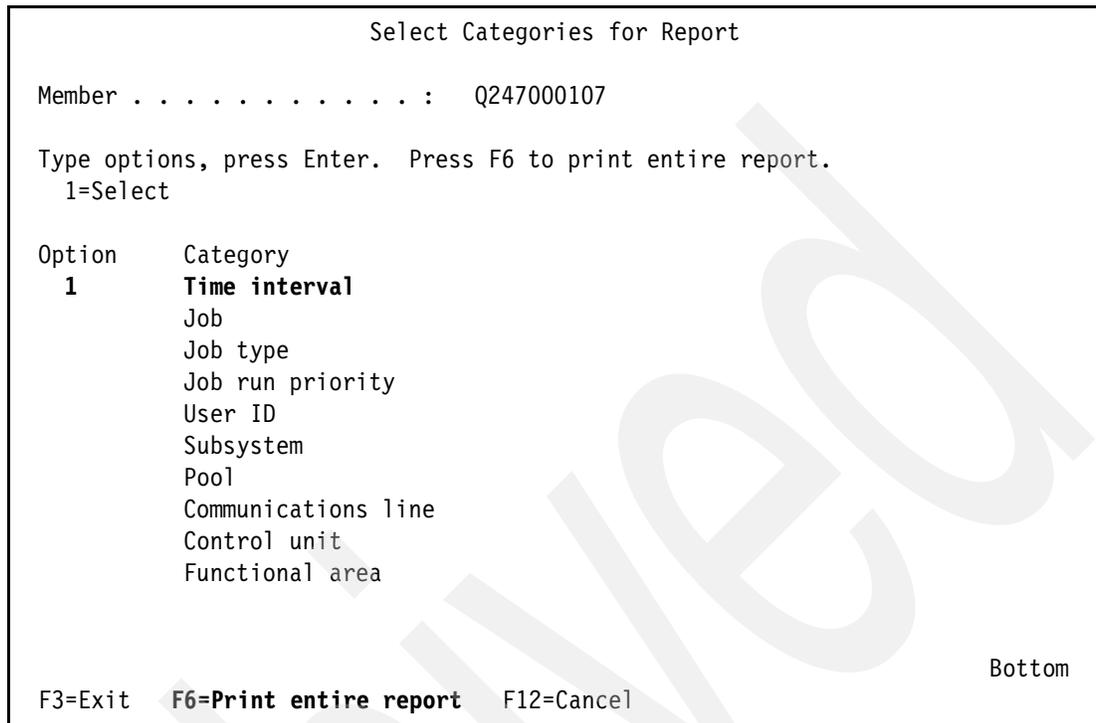


Figure 13-6 Select Categories for Report display

3. If you selected Time interval on the previous display, you see the Select Time Intervals display (Figure 13-7). You can select intervals that you want to look at based on CPU

utilization. In this example, we select two time frames of interest by typing 1 in the Opt column. These time frames were chosen because the concern is the sudden CPU growth.

Select Time Intervals													
Library : QMPGDATA							Performance data : Q247000107						
Type options, press Enter.													
1=Select													
0													
p			Transaction		-CPU Util--		Int	High		Pool			
t	Date	Time	Count	Resp	Tot	Int	Bch	Feat	--Util--	-Fault/Sec-			
	09/04	07:45	0	.00	21	0	21	0	1 0004	0	0 02		1298
	09/04	08:00	12	5.91	28	1	27	0	4 0005	4	8 02		1730
	09/04	08:15	0	.00	29	0	29	0	2 0004	1	4 02		1787
	09/04	08:30	18	.16	36	0	36	0	2 0003	0	2 02		1715
	09/04	08:45	0	.00	66	0	66	0	6 0005	2	20 02		1542
1	09/04	09:00	47	1.23	99	2	97	2	4 0005	1	17 02		1470
1	09/04	09:15	0	.00	100	0	100	0	2 0005	0	9 02		1480
1	09/04	09:30	0	.00	99	0	99	0	3 0004	0	13 02		1395
	09/04	09:45	34	.35	40	0	40	0	3 0005	2	21 02		1850
													Bottom
F3=Exit				F5=Refresh				F12=Cancel					
F13=Sort (date/time)				F14=Sort (count)				F24=More keys					

Figure 13-7 Select Time Intervals display

- In the Specify Report Options display (Figure 13-8), you specify any report title that you want. In this example, we specify CPU Report. Press **Enter**, and a job is submitted to batch.

Specify Report Options			
Type choices, press Enter.			
Report title	CPU Report		
Report detail	*JOB		*JOB, *THREAD
Job description	QPRJOB		Name, *NONE
Library	*LIBL		Name, *LIBL, *CURLIB
F3=Exit		F12=Cancel	

Figure 13-8 Specify Report Options panel

- You then return to the Print Performance Report - Sample data display and see the following message at the bottom of the display:

Job 117667/NGSS/PRTCPT submitted to job queue QBATCH in library QGPL.

This message gives you the submitted job for your report. You can find your report by entering the following command:

```
WRKSBMJOB *JOB
```

- In the Work with Submitted Jobs display, your submitted job is called PRTCPTTRPT. When the PRTCPTTRPT job is in OUTQ status, select option 8 to view the spool file. The report is in the spool file QPPTCPTTR. Type 5 on the line that precedes the QPPTCPTTR file.

In Figure 13-9, you see part of the Component Report showing the jobs that are using CPU. In this example, there are three jobs of concern as highlighted in bold.

```

Display Spooled File
File . . . . . : QPPTCPTTR
Control . . . . .
Find . . . . .
*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8...+...9...+...0..
Component Report
CPU report
Member      :Q247000107 Model/Serial: 270/65-2DF1B Main storage  :1024.0 MB Started:09/04/06 00:01
Library     :QMPGDATA System name :MYI270 Version/Release:5/3.0 Stopped  :09/04/06 09:45
Partition ID:000 Feature Code:22A2-2248-1517 Int Threshold  :15.60 %
Virtual Processors:1 Processor Units:1.0

```

Job Name	User Name/ Thread	Job Number	T y	P Pl	y	CPU Util	DB Cpb Util	Tns /Hour	Rsp	----- Sync	Disk I/O Async	----- Logical	
QSQSRVR	QUSER	114671	B	02	10	.00	.0	0	0	.000	45	1	6
QSQSRVR	QUSER	114679	B	02	20	.00	.0	0	0	.000	612	18	21
QSQSRVR	QUSER	114687	B	02	10	.00	.0	0	0	.000	3	0	0
QSQSRVR	QUSER	114690	B	02	10	.00	.0	0	0	.000	288	78	170
QSYSWRK	QSYS	114548	M	02	00	.00	.0	0	0	.000	35	0	0
QTCPMONITR	QTCP	114612	B	02	10	.00	.0	0	0	.000	1364	251	0
QTOTNTP	QNTP	114649	B	02	10	.00	.0	0	0	.000	444	0	0
QTSMTPLD	QTCP	114663	B	02	35	.00	.0	0	0	.000	90	0	0
QTSMTPSRVD	QTCP	114650	B	02	35	.00	.0	0	0	.000	6	0	0
QUSRWRK	QSYS	114563	M	02	00	.00	.0	0	0	.000	33	0	0
QYPSJSVR	QYPSJSVR	114637	B	02	16	.00	.0	0	0	.000	10710	573	0
QYPSFRCOL	QSYS	117306	B	02	01	.00	.0	0	0	.000	896	656	0
QYUSCMCRMD	QSYS	114646	B	02	50	.00	.0	0	0	.000	1147	0	0
QZDASOINIT	QUSER	117683	C	02	20	.00	79.4	0	0	.000	987	5208	343
QZRCSRVS	QUSER	117686	C	02	20	.00	.0	0	0	.000	352	115	0
QZRCSRVS	QUSER	117687	C	02	20	.00	.0	0	0	.000	33	1	0
QZRCSRVS	QUSER	117689	C	02	20	.00	.0	0	0	.000	157	89	0
Q1PSCH	QPM400	114614	B	02	50	.00	.0	0	0	.000	94	3	310
REPLICA	QNOTES	115839	B	02	20	.00	.0	0	0	.000	1	0	0

```

F3=Exit  F12=Cancel  F19=Left  F20=Right  F24=More keys

```

Figure 13-9 Partial print of Jobs using CPU

The disadvantage of using the printed report versus displaying the data interactively is that the printed Component Report is sorted by the job name, not by CPU. Displaying the data interactively allows you to sort on CPU.

Now that we have the jobs that are using the majority of CPU, we can now look at the Database Monitor data that was running at the same time to investigate what the jobs were doing. 13.5, "Using Performance Data of the Database Monitor to find the query that needs optimization" on page 446 gives examples on how to investigate the SQL.

Option 7: Display performance data

If you select option 7, you must specify the library in which you keep your performance data. In most cases, the library is QMPGDATA. Now perform the following steps:

1. Select the member that you want to investigate, based on the date and time shown. You do this by typing 1 next to the member.
2. After you select the member, use page up and down until you see a time frame you are looking for or until you see a CPU Utilization that concerns you. Then you type 1 next to the interval or intervals that you want to examine.
3. After you select the intervals, you see the Display Performance Data display (Figure 13-10).

Display Performance Data	
Member	Q247000107
Library	QMPGDATA
F4 for list	
Elapsed time :	00:15:00
System	MYI270
Start date	09/04/06
Start time	00:01:08
Partition ID :	000
QPFRAJ	3
QDYNPTYSCD :	1
QDYNPTYADJ :	1
Version	5
Release	3.0
Model	270
Serial number . . . :	65-2DF1B
Feature Code :	22A2-2248-1517
Int Threshold . . . :	15.60 %
Virtual Processors :	1
Processor Units . . :	1.00
CPU utilization (interactive) :	2.35
CPU utilization (other) :	97.57
Interactive Feature Utilization :	2.38
Time exceeding Int CPU Threshold (in seconds) . . :	0
Job count :	59
Transaction count :	47
More...	
F3=Exit	F4=Prompt
F12=Cancel	F24=More keys
F5=Refresh	F6=Display all jobs
	F10=Command entry

Figure 13-10 Display Performance Data display

4. From this display, press **F6** to view all jobs. Then you see a listing of the jobs that were running during the interval as shown in Figure 13-11. You can use **F19** to sort by CPU.

Display Jobs								
Elapsed time . . . : 00:15:00			Member : Q247000107					
			Library : QMPGDATA					
Type options, press Enter.								
5=Display job detail 6=Wait detail								
Option	Job	User	Number	Job Type	CPU Util	Tns Count	Avg Rsp	Disk I/O
	QZDASOINIT	QUSER	117683	BCH	69.11	0	.0	3459
	QYPSJSVR	QYPSJSVR	114637	BCH	8.71	0	.0	4697
	SERVER	QNOTES	115828	BCH	3.05	0	.0	3651
	CFINT01			LIC	2.80	0	.0	0
	EVENT	QNOTES	115831	BCH	2.61	0	.0	215
	HTTP	QNOTES	115834	BCH	2.43	0	.0	257
	QPADEV002B	NGSS	117681	PTH	2.35	47	1.2	4614
	ADMINP	QNOTES	115837	BCH	1.21	0	.0	650
	RMTMSAFETA	SK		LIC	1.03	0	.0	7
	AUDPRDJOB	AUDUSR	117621	BCH	.95	0	.0	178
								More...
F3=Exit		F12=Cancel		F15=Sort by job		F16=Sort by job type		
F19=Sort by CPU				F24=More keys				

Figure 13-11 Listing of jobs using CPU

In this example, it appears as though there is a job that is using the majority of CPU. If this is unusual and Database Monitor data was gathered during this interval, you can use the examples as shown in 13.5, “Using Performance Data of the Database Monitor to find the query that needs optimization” on page 446 to isolate the SQL, if any, that this job was running during this interval.

13.3.2 Finding jobs using CPU with iSeries Navigator Graph History

Graph history provides a graphical view of performance data collected over days, weeks, months, or years with Collection Services. You do not need to have a system monitor running to view performance data. As long as you use Collection Services to collect data, you can

view the Graph History window. To access the graph history in iSeries Navigator, select the system name, then select **Configuration and Service** and perform the following steps:

1. Right-click **Collection Services** and select **Graph History** as shown in Figure 13-12.

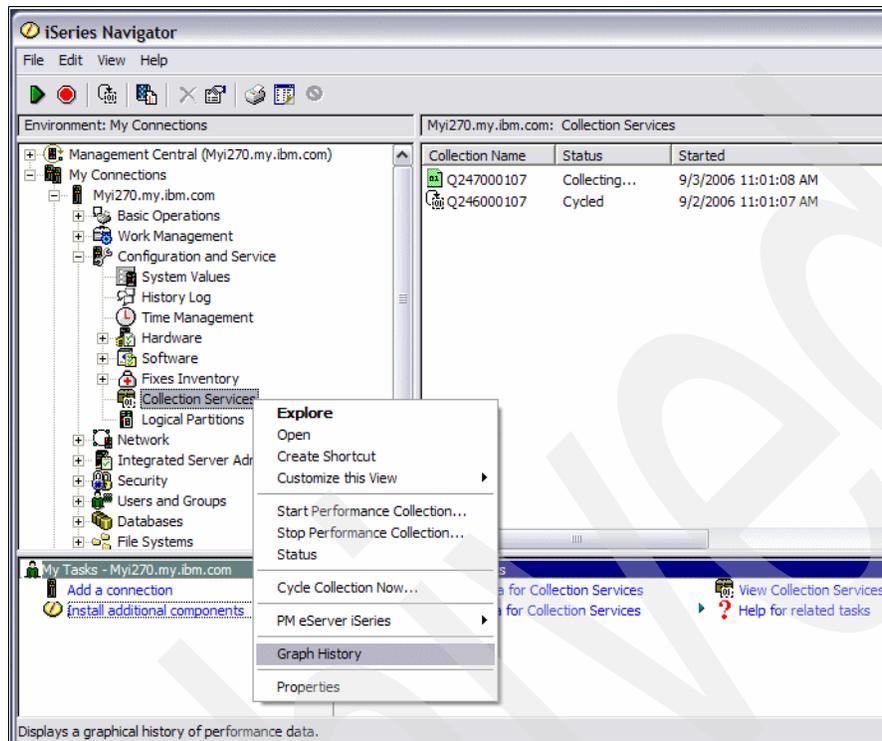


Figure 13-12 Selecting Graph History

2. The Graph History window opens. When you click the drop-down list for Metric, you see several options to help you find the jobs using CPU as shown in Figure 13-13. In this

example, we select the CPU Utilization (Database Capability) option. You also need to specify the time frame for which you want to see graph history.

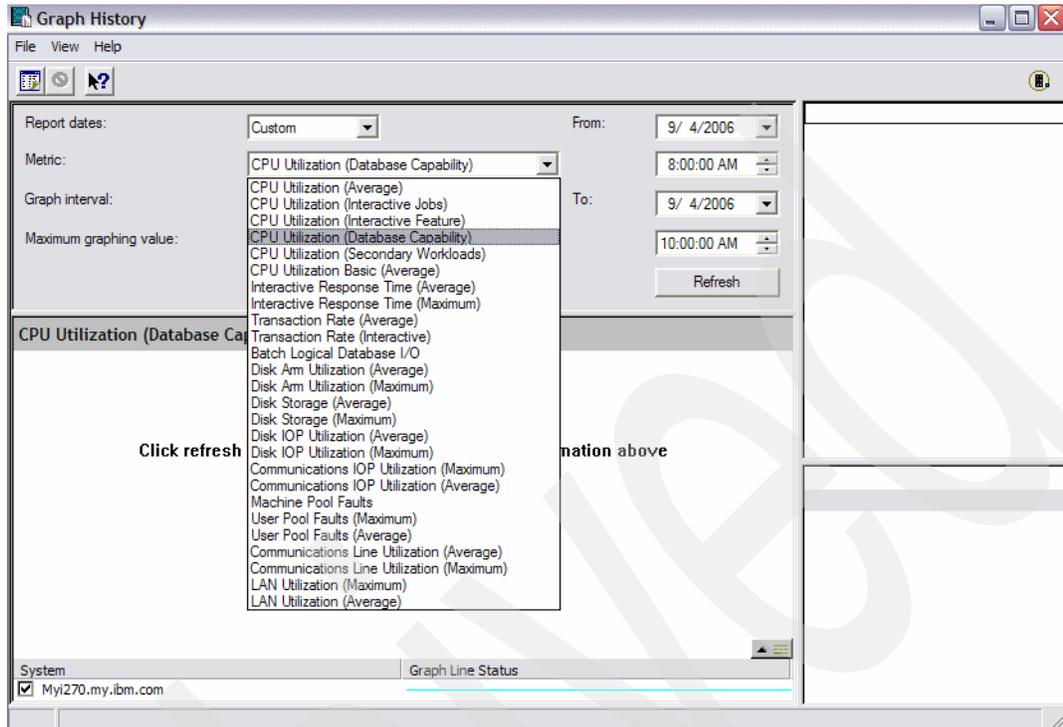


Figure 13-13 Graph History metric options

3. After you launch a graph history, a window opens that shows a series of graphed collection points. These collection points on the graph line are identified by three different graphics that correspond to the three levels of data that are available:
 - A *square collection point* represents data that includes both the detailed information and properties information.
 - A *triangular collection point* represents summarized data that contains detailed information.
 - A *circular collection point* represents data that contains no detailed information or properties information.

You can select any of the bars on the graph to view information about the job in the box underneath it. Figure 13-14 shows information after a point on the graph was selected. It shows the bargraph in the upper right corner that shows job names sorted by CPU. In the example shown, the bar turns black after being selected, and the first three lines of information in the box underneath the graph are the job name, user name, and job number. This information is useful to be used in 13.5, “Using Performance Data of the

Database Monitor to find the query that needs optimization” on page 446 to indicate whether the job was running SQL, and if so, the SQL statement that was run.

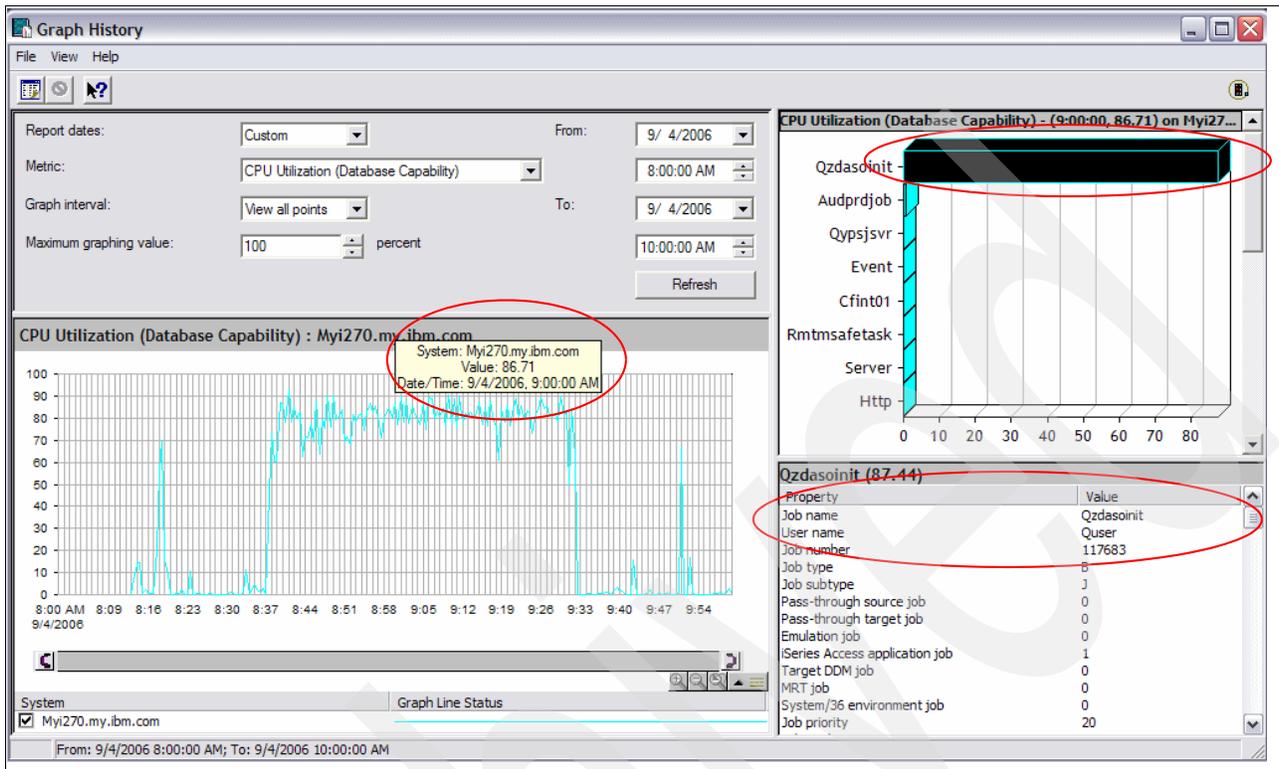


Figure 13-14 Displaying job using CPU with Graph History data

For more information about using Graph History, refer to *System Management - Performance Version 5 Release 3* on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseres/v5r4/topic/rzahx/rzahx.pdf>

13.3.3 Finding jobs using CPU with Management Central System Monitors

The system monitors display the data stored in the collection objects that are generated and maintained by Collection Services. The system monitors display data as it is collected, for up to one hour. To view longer periods of data, use Graph History as explained in 13.3.2, “Finding jobs using CPU with iSeries Navigator Graph History” on page 436. You can change the frequency of the data collection in the monitor properties, which overrides the settings in Collection Services.

To set up a system monitor, you must define the metrics that you want to track and the action that the monitor should take when the metrics reach the specified levels. To define a system monitor that looks for jobs using CPU, complete the following steps:

1. In iSeries Navigator, expand **Management Central** → **Monitors**. Right-click **System Monitor** and select **New Monitor** (Figure 13-15).

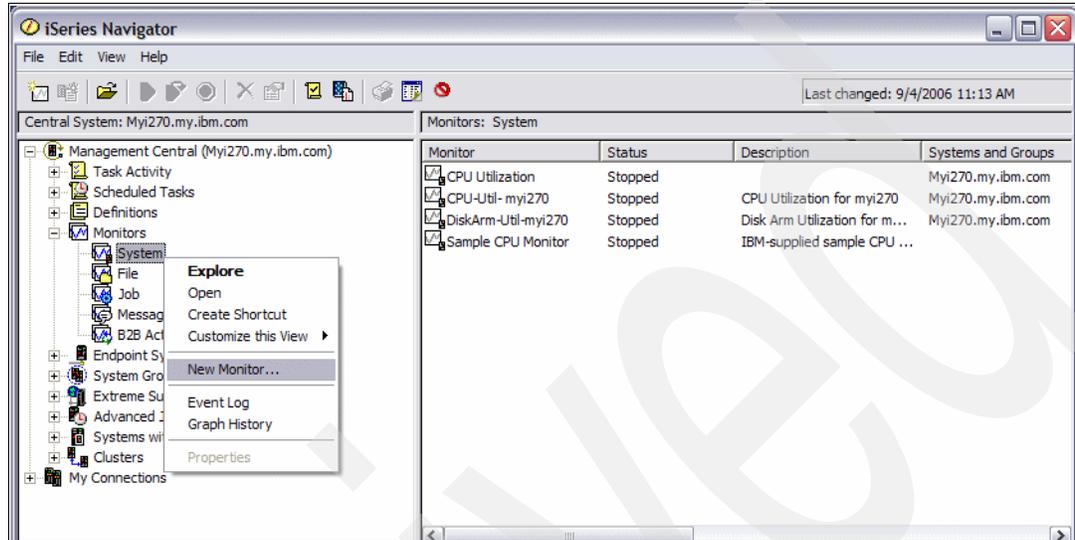


Figure 13-15 Create New Monitors

2. On the General page, enter a name and description for the monitor.

3. On the Metrics page (Figure 13-16), enter the following values:
 - a. From the list of Available Metrics, select **CPU Utilization (Average)** and click **Add**. CPU Utilization (Average) is now listed under Metrics to monitor, and the bottom portion of the window shows the settings for this metric.
 - b. Add three other Available Metrics: CPU Utilization (Interactive Jobs), CPU Utilization (Interactive Feature) and CPU Utilization (Database Capability).
 - c. For Collection interval, specify how often you want to collect this data. This value overrides the Collection Services setting. For this example, we specify 30 seconds.
 - d. To change the scale for the vertical axis of the monitor's graph for this metric, change the Maximum graphing value. To change the scale for the horizontal axis of the graph for this metric, change the value for Display time.

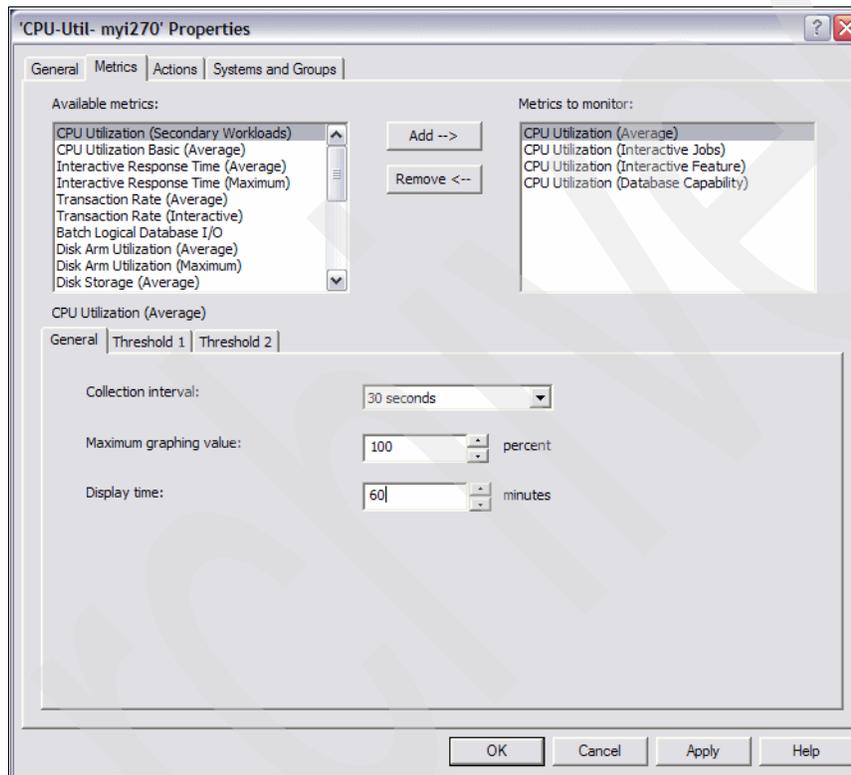


Figure 13-16 New Monitor metric options

4. On the Systems and Groups page, select your systems and groups. You can click Browse to select the endpoint system on which you want to run the monitor.
5. Expand **Management Central** → **Monitors** and click **System**. You should now see the monitor displayed with a status of *Stopped*.
6. To start the monitor, right-click it and select **Start**.

- To view the graph data, double-click the monitor that you created. Click any square in the graph to see a bar graph in the upper right corner that shows the job names sorted by CPU. To find more job details, click any of the bars on the graph as shown in Figure 13-17.

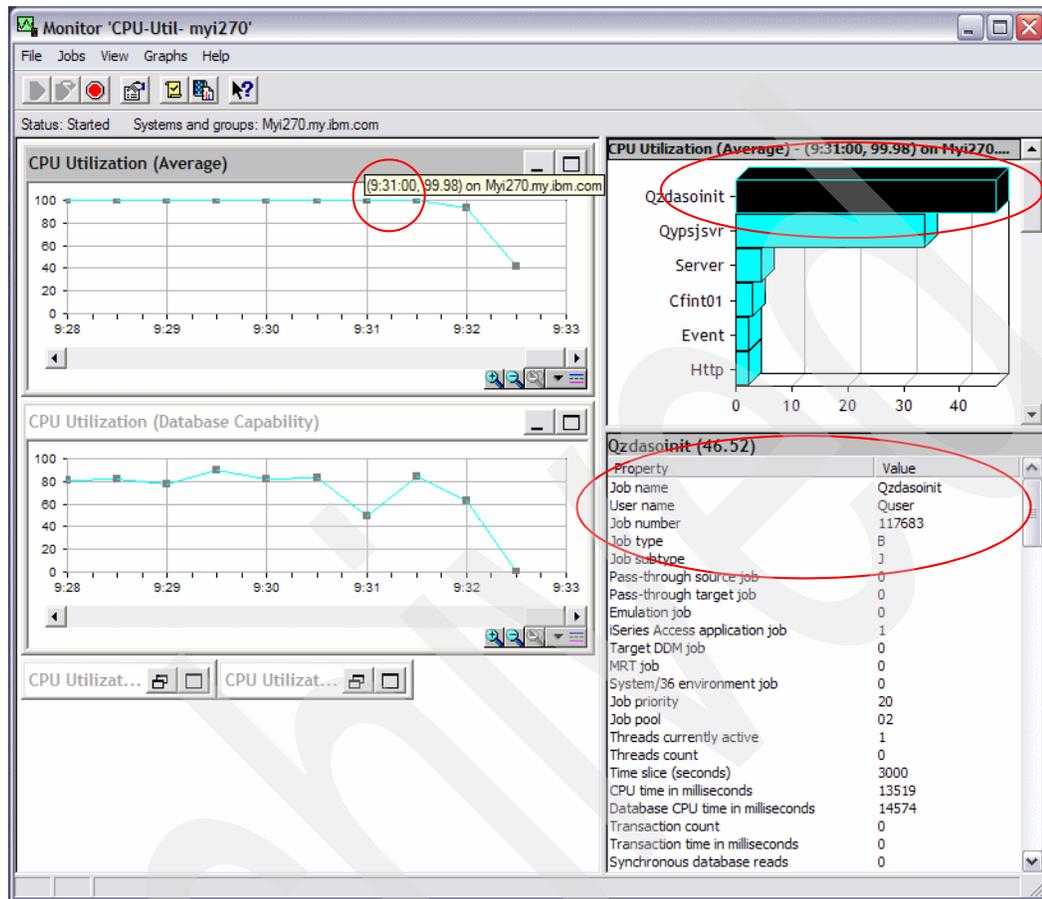


Figure 13-17 System Monitor showing job using CPU

The box below the bar graph shows the fully qualified job detail, consisting of job name, user name, and job number. You can use this information to query a Database Monitor file collecting during this time frame as shown in 13.5, “Using Performance Data of the Database Monitor to find the query that needs optimization” on page 446.

For more information about using Graph History, see *System Management - Performance Version 5 Release 4* on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/rzahx/rzahx.pdf>

13.4 Using Collection Services data to find jobs with high disk I/O counts

You can use Collection Services data to look for jobs with high disk I/O counts. To find the jobs with high disk I/O, you are required to have Performance Tools installed. To start looking for jobs that have a high I/O count, use the Component Report from the PERFORM menu. To access the PERFORM menu, enter the following command:

```
GO PERFORM
```

If you want to interactively view the data, select option 7 (Display performance data). Keep in mind your system resources when you choose this option. This job runs interactively and uses CPU and I/O resources. Or you can choose option 3 (Print performance report), which submits a job to batch.

The following sections explain how to find the jobs with high I/O counts either interactively (option 7) or in batch (option 3).

Option 3: Print performance report

When you select option 3, you must specify the library in which you keep your performance data. In most cases, the library is QMPGDATA. Now perform the following steps:

1. Page up and down until you see a time frame that you want. Type 2 next to the member that you want to use to get a Component Report.

If no members are found and you have started Collection Services, you must create the database files that are needed. See the first bullet point in 13.3.1, “Finding jobs using CPU with the Component Report” on page 429, for information about creating the files.
2. In the next Select Sections for Report display (Figure 13-5 on page 431), you can press **F6** (Print entire report) to select all the sections to be in your report. Since you must often review the storage pool activity and disk activity, we recommend that you use F6. In the display shown in Figure 13-5, you see that we choose option 1 to select Job Workload Activity to find the jobs with high I/O counts.
3. In the Select Categories for Report display (Figure 13-6 on page 432), we type option 1 to select Time interval. Time interval is usually the best option if you do not know any other information. If you want more information, you can press F6 to print the entire report, but it is best to narrow the information down to a time frame in which you are interested.
4. If you selected Time interval on the previous display, you see the Select Time Intervals display (Figure 13-7). You can select intervals that you want to look at based on High Disk Utilization or a time frame.
5. In the Specify Report Options display (Figure 13-8 on page 433), specify any report title that you want. Press **Enter**, and a job is submitted to batch.
6. You then return to the Print Performance Report - Sample data display and see the following message at the bottom of the display:

```
Job 117667/NGSS/PRTCPTTRPT submitted to job queue QBATCH in library QGPL.
```

This message gives you the submitted job for your report. You can find your report by entering the following command:

```
WRKSBMJOB *JOB
```
7. In the Work with Submitted Jobs display, your submitted job is called PRTCPTTRPT. When the PRTCPTTRPT job is in OUTQ status, select option 8 to view the spool file. The report is in spool file QPPTCPTTR. Type 5 on the line that precedes the QPPTCPTTR file.

Figure 13-18 shows a job that has a high I/O count in comparison to other jobs in the report. The disadvantage of using the printed report versus displaying the data interactively is that the printed Component Report is sorted by the job name, not by disk I/O. Displaying the data interactively allows you to sort on disk I/O. Now that we have isolated a job with relatively high disk I/O, we can look at the Database Monitor data that was running at the same time to investigate what the job was doing. 13.5, “Using Performance Data of the Database Monitor to find the query that needs optimization” on page 446, gives examples of how to investigate what SQL, if any, that the job was running based on the job name, job user, and job number.

```

File . . . . . : QPPTCPT
Control . . . . .
Find . . . . .

*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8...+...9...
+...0..

Component Report
Job Workload Activity
CPU Report
Main storage :1024.0MB
Member      :Q247000107 Model/Serial  :270/65-2DF1B
Started:09/04/06 00:01
Library     :QMPGDATA System name    :MYI270 Version/Release:5/ 3.0
Stopped:09/04/06 09:45
Partition ID:000 Feature Code :22A2-2248-1517 Int Threshold :15.60 %
Virtual Processors:1 Processor Units:1.0

Job User Name/ Job T P DB CPU Cpb Tns ----- Disk I/O
-----
Name Thread Number p Pl y Util Util Tns /Hour Rsp Sync Async
Logical
-----
QTOTNTP QNTP 114649 B 02 10 .00 .0 0 0 .000 444 0
0
QTSMTPLTD QTCP 114663 B 02 35 .00 .0 0 0 .000 90 0
0
QTSMTPSRVD QTCP 114650 B 02 35 .00 .0 0 0 .000 6 0
0
QUSRWRK QSYS 114563 M 02 00 .00 .0 0 0 .000 33 0
0
QYPSPFRCOL QSYS 117306 B 02 01 .00 .0 0 0 .000 896 656
0
QZDASOINIT QUSER 117683 C 02 20 .00 79.4 0 0 .000 987 5208
343
QZRCSRVS QUSER 117686 C 02 20 .00 .0 0 0 .000 352 115
0
QZRCSRVS QUSER 117687 C 02 20 .00 .0 0 0 .000 33 1
0
QZRCSRVS QUSER 117689 C 02 20 .00 .0 0 0 .000 157 89
0
Q1PSCH QPM400 114614 B 02 50 .00 .0 0 0 .000 94 3
310
REPLICA QNOTES 115839 B 02 20 .00 .0 0 0 .000 1 0
0

F3=Exit F12=Cancel F19=Left F20=Right F24=More keys

```

Figure 13-18 Example of job having relative high I/O count

Option 7: Display performance data

If you select option 7, you must specify the library in which you keep your performance data. In most cases, the library is QMPGDATA. Then perform the following steps:

1. Select the member that you want to investigate, based on the date and time shown. You do this by typing 1 next to the member.

If no members are found and you started Collection Services, you must create the database files needed. See 13.3.1, “Finding jobs using CPU with the Component Report” on page 429, for information about how to create the files.

2. After you select the member, you can page up and down until you see a time frame you are looking for or until you see a high disk utilization that concerns you. Then you type 1 next to the interval or intervals that you want to examine.

After you select the intervals, you see the Display Performance Data display (Figure 13-10 on page 435). From this display, press **F6** to view all jobs. You can use **F22** to sort by disk I/O as shown in Figure 13-19. If the disk activity is unusually high and Database Monitor data was gathered during this interval, then the Database Monitor data can be queried as explained in 13.5, “Using Performance Data of the Database Monitor to find the query that needs optimization” on page 446, to isolate what the job is doing.

Display Jobs								
Elapsed time . . . : 00:15:00			Member : Q247000107					
			Library : QMPGDATA					
Type options, press Enter.								
5=Display job detail 6=Wait detail								
Option	Job	User	Number	Job Type	CPU Util	Tns Count	Avg Rsp	Disk I/O
	QZDAS0INIT	QUSER	117683	BCH	69.11	0	.0	3459
	CRTPFRTA	QSYS	117645	BCH	.22	0	.0	1459
	QDBSRVXR2	QSYS	114542	SYS	.17	0	.0	1142
	QIJSSCD	QIJS	114623	BCH	.10	0	.0	1085
	ADMIN	QEJBSVR	114948	BCH	.71	0	.0	668
	QIJSCPEM	QIJS	115170	BCH	.09	0	.0	661
								More...
F20=Sort by transactions			F21=Sort by response			F22=Sort by disk I/O		
F24=More keys								

Figure 13-19 Job having a high disk I/O count

You can find additional information about Collection Services data and using the tools in *Systems Management - Performance Tools Reports Version 5 Release 4* in the Information Center on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseres/v5r4/topic/rzahx/rzahxptrpts.pdf>

Also refer to *Systems Management - Performance Version 5 Release 4*, which is available on the Web at:

<http://publib.boulder.ibm.com/infocenter/iseres/v5r4/topic/rzahx/rzahx.pdf>

13.5 Using Performance Data of the Database Monitor to find the query that needs optimization

Working in parallel with Collection Services, you should also collect Database Monitoring data to help in analyzing and pin-pointing which query causes the performance degradation.

In this section, we look at how we can find more information from the Database Monitoring data by performing the following steps:

1. In iSeries Navigator, expand the **system name** → **Database** → **SQL Performance Monitors** and right-click the **Database Monitor data**. Now you have options to Analyze or Show Statements (as shown in Figure 13-20).

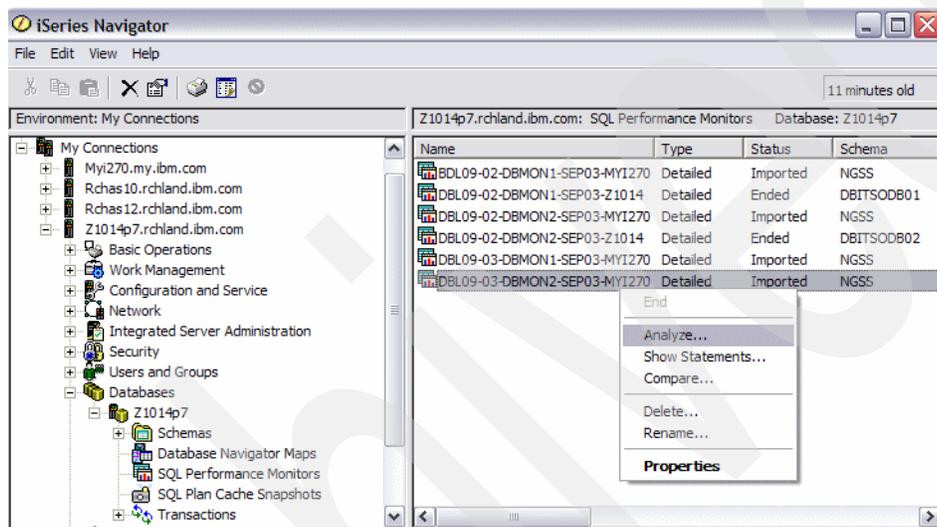


Figure 13-20 Options to Analyze and Show Statements of Database Monitor data

2. When you select **Analyze** option, the Analysis Overview Dashboard is presented as shown in Figure 13-21. The green color checkmark sign indicates if there is further Summary or Statements that you can analyze. For example, you can highlight the **SQL**

statements line and select the **Summary** button to look at the Summary data or select **Statements** button to look at the SQL statements involved.

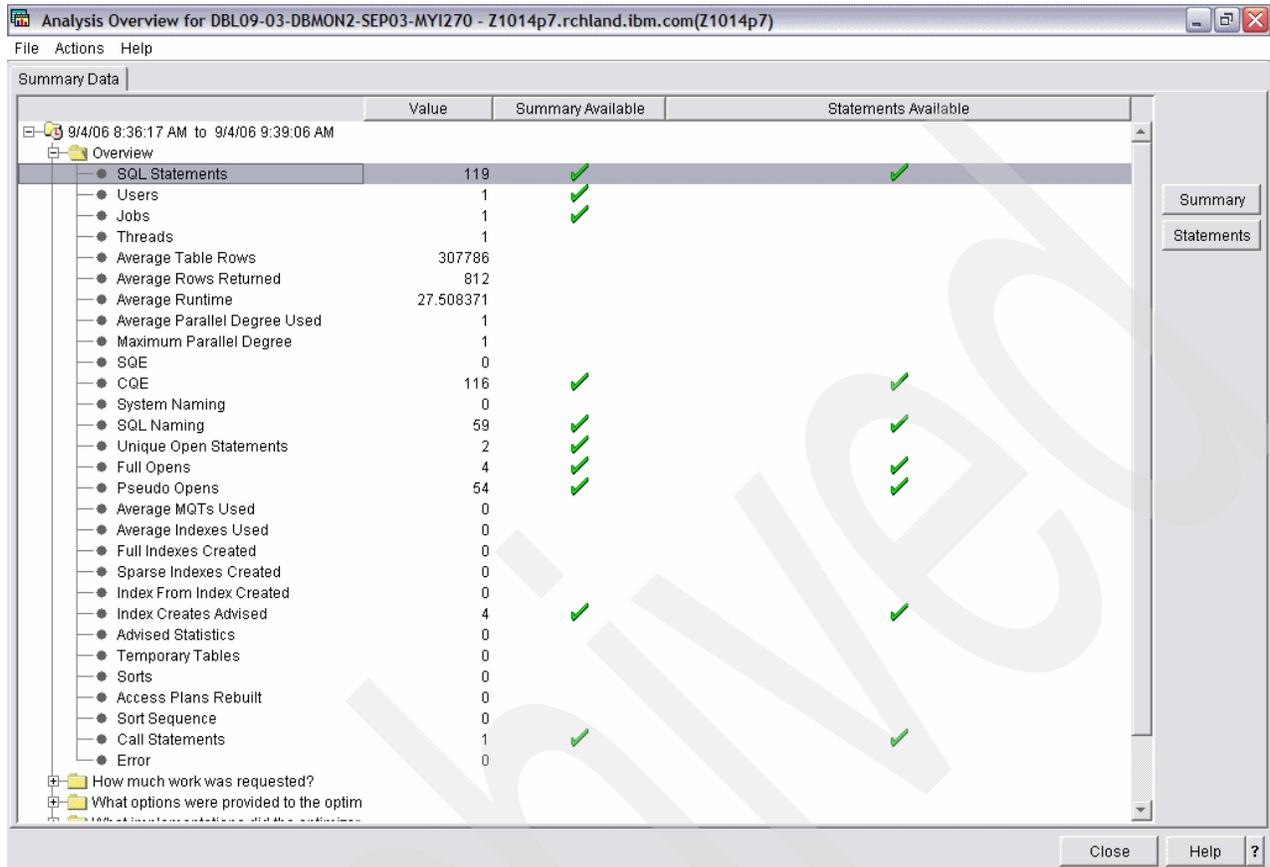


Figure 13-21 The Analysis Overview Dashboard

3. You can also highlight the item line of your interest. Click **Actions** menu from the Dashboard and select **Analysis Queries**. From the Analysis Queries window as shown in

Figure 13-22, you can highlight and right-click the item line of interest and select **View Results** to see the resulting details.

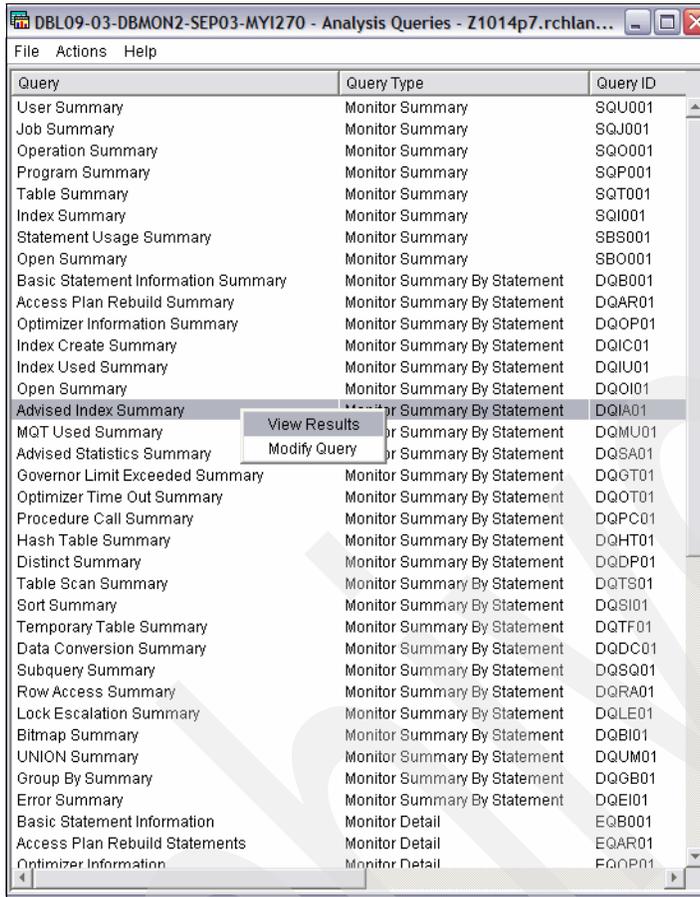


Figure 13-22 Analysis Queries

- You are presented with the Result details window as shown in Figure 13-23. In this case, you will be interested in the line which has the highest Maximum Runtime and proceed to identify the SQL statements involved.

DBLO9-03-DBMON2-SEP03-MY1270 - Advised Index Summary - Z1014p7.rchland.ibm.com(Z1014p7)

Most Expensive Use	Index Creates Advised	Rows In Base Table	Estimated Nonparallel Create Time	Average Indexes Used	Time To Create Index
2006-09-04 09:31:45.950080	54	15000.000		1.000	
2006-09-04 08:51:05.076056	29	15000.000		1.000	
2006-09-04 08:44:17.685080	29	15000.000		1.000	

DBLO9-03-DBMON2-SEP03-MY1270 - Advised Index Summary - Z1014p7.rchland.ibm.com(Z1014p7)

Maximum Runtime	Average Runtime	Minimum Runtime	Total Runtime	Maximum Open Time	Maximum Fetch Time	Maximum Close Time	Maximum Ott
85.352944	57.127851	47.457296	1656.707704			0.000000	8
64.870176	55.751326	51.033064	1616.788480			0.000000	6

DBLO9-03-DBMON2-SEP03-MY1270 - Advised Index Summary - Z1014p7.rchland.ibm.com(Z1014p7)

Statement Text	Variable Values	Full Opens	Pseudo Opens	Table Scans	Temporary Tables	Sorts	Optimizer Time Outs
HARD CLOSE T CURSORS				108	0	0	0
INSERT INTO ORDERS2 SELECT O . * FROM ...		2	27	58	0	0	0
INSERT INTO ORDERS2 SELECT O . * FROM ...		2	27	58	0	0	0

Figure 13-23 Result detail - Advised Index Summary

Alternatively, perform the following steps:

1. Instead of selecting **Analyze** option as shown in Figure 13-20, you can select **Show Statements** option. The Show Statements window is presented as shown in Figure 13-24.

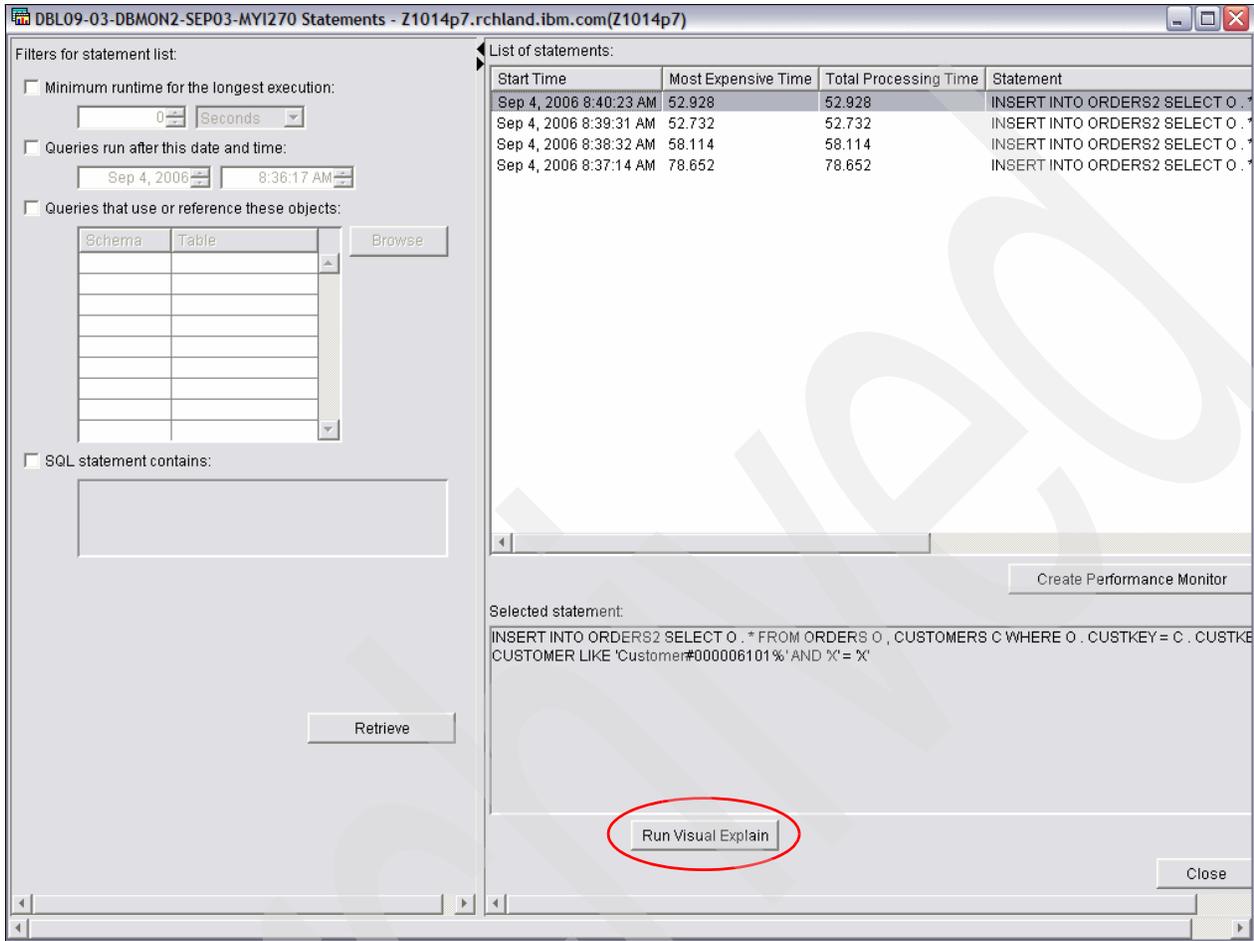


Figure 13-24 Show Statements

2. In this case, you will be interested in the item line with the highest value for Most Expensive Time and Total Processing Time. Highlight the top item line and click **Run**

Visual Explain. The Visual Explain window is presented (Figure 13-25), showing you the graphical representation of the optimization path used by the query optimizer.

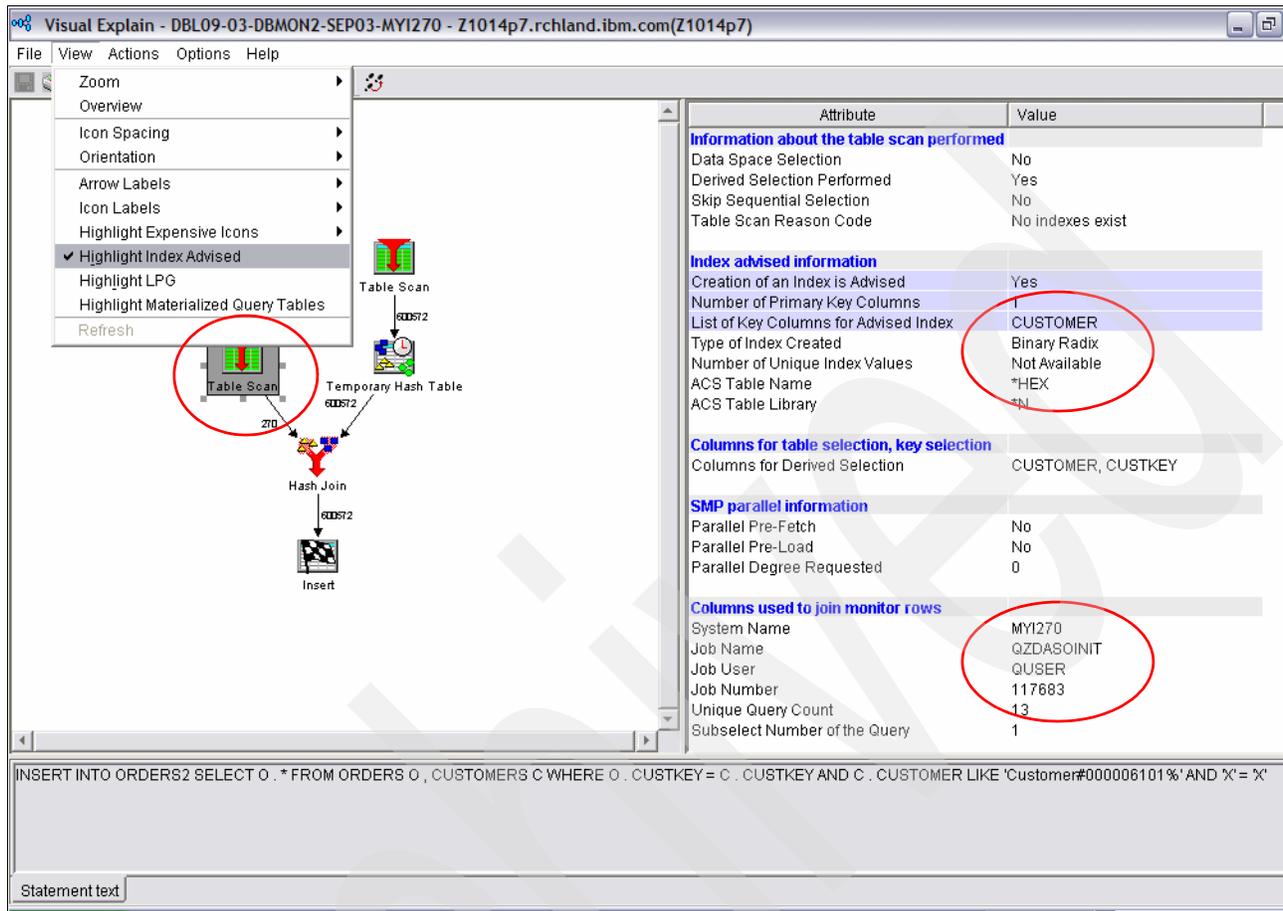


Figure 13-25 Visual Explain

- If you select **View** → **Highlight Index Advised**, this enables the area with index advised to be highlighted. In this case you see Table scan icon is highlighted. You can analyze further on the index advised details presented on the right pane of the Visual Explain window. In this case, you see the CUSTOMER column is advised as the key column. You will also find the Job details on the right pane. In this case, you see QZDASOINIT/QUSER/117683 for the Job-name/Job-user/Job-number, which matches the job that consumed the highest CPU found in the Performance component report (Figure 13-9), the Displaying performance data (Figure 13-11), the Displaying Graph History (Figure 13-14), the Displaying Management center system monitor (Figure 13-17), and the same job that uses the most Disk IO found in Performance component report (Figure 13-18) and the Displaying performance data (Figure 13-19).
- Now if you select the **Actions** menu from the Visual Explain window and select **Advisor**, you are presented with the Index Advisor window (Figure 13-26) which provides you with the details of the index advised. Click the **Create** button to create the advised index on the fly, and optimize the query by setting the optimization goal suggested by the query optimizer.

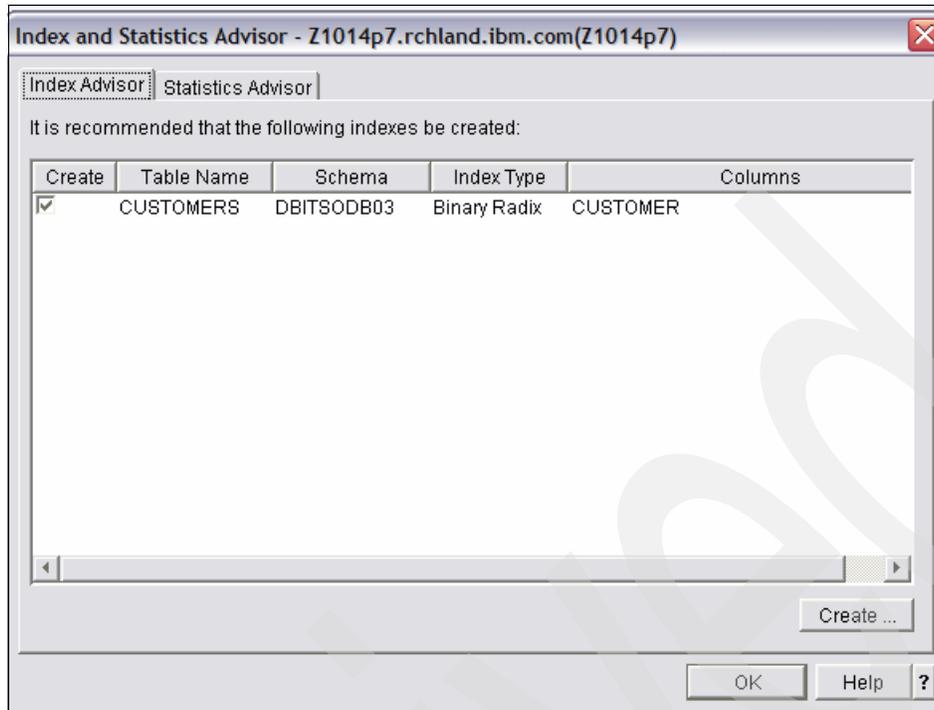


Figure 13-26 Index advisor

Note: For more information about how to analyze the SQL performance data in iSeries Navigator and utilizing the Dashboard, refer to Chapter 5, “Analyzing SQL performance data using iSeries Navigator” on page 117. For more information about querying the database monitor, refer to Chapter 6, “Custom Database Monitor Analysis” on page 173. For more information about how to utilize Visual Explain, refer to Chapter 8, “Analyzing database performance data with Visual Explain” on page 275. For more information about the index advised and Index Advisor, refer to Chapter 9, “Index Advisor” on page 319.

13.6 Using Performance Trace to find object locks

There are many incidences that slow query performance is not caused by insufficient system resources nor unoptimized SQL statements, but rather due to object locks. With this in mind, we discuss using Performance Trace to find object locks in this section.

This OS/400 facility uses trace data to provide information about locks during system operation. With this information you can determine if jobs are being delayed during processing because of unsatisfied lock requests. These conditions are also called *waits*. If they are occurring, you can determine which objects the jobs are waiting for and the length of the wait.

Performance Trace is not part of Performance Tools Collection Services. In such, you must use the Start Performance Trace (STRPFRTRC) and End Performance Trace (ENDPFRTRC) commands to collect the trace information.

Example of Start Performance Trace and End Performance Trace commands:

```
STRPFRTRC  
ENDPFRTRC MBR(TRACE3) LIB(QPFRDATA) TEXT('Lock trace on DBITS0DB08 14SEP06  
4.50PM')
```

Once you have collected the Performance trace, you have to use the Print Lock Report (PRTLCKRPT) to generate the report so that you can perform further analysis. The Lock Report shows information of:

- ▶ File, record, or object contention by time
- ▶ The holding job or object name
- ▶ The requesting job or object name

There are two levels of Lock Report, that is, Summary and Detail. Detail report allows you to have a few options that you can select:

- ▶ *TOD: Provides detail sorted by time of day, followed by a summary
- ▶ *HLD: Includes detail sorted by name of the holding job and time of day, followed by a summary
- ▶ *RQS: This includes detail sorted by name of the requesting job and time of day, followed by a summary
- ▶ OBJ: You are presented with detail sorted by name of the object and time of day, followed by a summary
- ▶ *ALL: If you select this option, there are four reports being produced. The reports include *TOD, *HLD, *RQS, and *OBJ, followed by a summary

Example of Print Lock Report (PRTLCKRPT) commands:

```
PRTLCKRPT MBR(TRACE3) LIB(QPFRDATA) TITLE('SumLckTrc DBITS0DB08 14SEP06 5PM')  
RPTTYPE(*SUM)  
PRTLCKRPT MBR(TRACE3) LIB(QPFRDATA) TITLE('DetailLckTrc DBITS0DB08 14SEP06  
4.50PM') RPTTYPE(*ALL)
```

Let us look at a Lock Report type “*ALL”, starting with the four sections of the report as shown in Figure 13-27.

```

Display Spooled File
File . . . . . : QPPTLCK
Control . . . . .
Find . . . . .
*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8...+...9...+...0...
9/14/06 17:09:16                               Seize/Lock Wait Statistics by Time of Day
ALLLckTrc DBITSODB08 14SEP06 5PM

TOD of Length                                     Object
Wait  of Wait L Requestor's Job Name           Holder's Job Name           Type  Object Name
-----
17.00.51      0  QINTER      00000008  048317 QCMNARB01  00000008  048281 LIB  QSYS
17.01.16     30030 L QPADEV000D NGSS      049110 QPADEV000C NGSS      049098 DS  ORDERS
                                                DBITSODB08 ITEM_FACT
17.01.32      0  QTVDEVICE  QTCP      048397 QINTER      QSYS      048317 DEVD  QPADEV000F
17.01.32      0  QINTER      00000009  048317 QTVDEVICE  00000009  048397 LIB  QSYS
17.02.29      0  MNTASK                                     QPADEV000F NGSS      049111 LIB  QGPL
17.03.12     19  MNTASK                                     QPADEV000F NGSS      049111 LIB  NGSS

9/14/06 17:09:16                               Seize/Lock Wait Statistics by Requesting Job
ALLLckTrc DBITSODB08 14SEP06 5PM

TOD of Length                                     Object
Wait  of Wait L Requestor's Job Name           Holder's Job Name           Type  Object Name
-----
17.02.29      0  MNTASK                                     QPADEV000F NGSS      049111 LIB  QGPL
17.03.12     19  MNTASK                                     QPADEV000F NGSS      049111 LIB  NGSS
17.00.51      0  QINTER      00000008  048317 QCMNARB01  00000008  048281 LIB  QSYS
17.01.32      0  QINTER      00000009  048317 QTVDEVICE  00000009  048397 LIB  QSYS
17.01.16     30030 L QPADEV000D NGSS      049110 QPADEV000C NGSS      049098 DS  ORDERS
                                                DBITSODB08 ITEM_FACT
17.01.32      0  QTVDEVICE  QTCP      048397 QINTER      QSYS      048317 DEVD  QPADEV000F

9/14/06 17:09:16                               Seize/Lock Wait Statistics by Holding Job
ALLLckTrc DBITSODB08 14SEP06 5PM

TOD of Length                                     Object
Wait  of Wait L Requestor's Job Name           Holder's Job Name           Type  Object Name
-----
17.00.51      0  QINTER      00000008  048317 QCMNARB01  00000008  048281 LIB  QSYS
17.01.32      0  QTVDEVICE  QTCP      048397 QINTER      QSYS      048317 DEVD  QPADEV000F
17.01.16     30030 L QPADEV000D NGSS      049110 QPADEV000C NGSS      049098 DS  ORDERS
                                                DBITSODB08 ITEM_FACT
17.02.29      0  MNTASK                                     QPADEV000F NGSS      049111 LIB  QGPL
17.03.12     19  MNTASK                                     QPADEV000F NGSS      049111 LIB  NGSS
17.01.32      0  QINTER      00000009  048317 QTVDEVICE  00000009  048397 LIB  QSYS

9/14/06 17:09:16                               Seize/Lock Wait Statistics by Object
ALLLckTrc DBITSODB08 14SEP06 5PM

TOD of Length                                     Object
Wait  of Wait L Requestor's Job Name           Holder's Job Name           Type  Object Name
-----
17.01.32      0  QTVDEVICE  QTCP      048397 QINTER      QSYS      048317 DEVD  QPADEV000F
17.01.16     30030 L QPADEV000D NGSS      049110 QPADEV000C NGSS      049098 DS  ORDERS
                                                DBITSODB08 ITEM_FACT
17.03.12     19  MNTASK                                     QPADEV000F NGSS      049111 LIB  NGSS
17.02.29      0  MNTASK                                     QPADEV000F NGSS      049111 LIB  QGPL
17.00.51      0  QINTER      00000008  048317 QCMNARB01  00000008  048281 LIB  QSYS
17.01.32      0  QINTER      00000009  048317 QTVDEVICE  00000009  048397 LIB  QSYS

```

Figure 13-27 Four report sections of Lock Report

From the four sections of the Lock Report, you can easily identify which jobs have the long wait, via looking at statistics by Time of day, by Requesting job, by Holding job and by Object. As shown in Figure 13-27, we see that 049110/QPADEV00C/NGSS job has a long wait of 30030ms for member ITEM_FACT of table ORDERS in schema DBITSODB08.

Now let us look at the Summary section of the Lock Report as shown in Figure 13-28.

Requestor's Job Name				Count	Avg Length	Count	Avg Length
-----				-----	-----	-----	-----
MNTASK						26	1
QDBSRVXR2	QSYS	048295				3	84
QINTER	QSYS	048317	00000008			1	
QINTER	QSYS	048317	00000009			1	
QPADEV000D	NGSS	049110		1	30,030		
QTVDEVICE	QTCP	048397				1	
SERVER	QNOTES	048416	0000004E	3	15		

Holder's Job Name				Count	Avg Length	Count	Avg Length
-----				-----	-----	-----	-----
QCMNARB01	QSYS	048281	00000008			1	
QDBSRVXR	QSYS	048291				4	20
QDBSRVXR2	QSYS	048295				2	30
QINTER	QSYS	048317				1	
QPADEV000C	NGSS	049098		1	30,030		
QPADEV000F	NGSS	049111				26	1
QTVDEVICE	QTCP	048397	00000009			1	
SERVER	QNOTES	048419	00000050	3	15		

Object Type	Object Name	Count	Avg Length	Count	Avg Length
-----		-----	-----	-----	-----
DEVD	QPADEV000F			1	
DS	ORDERS DBITSODB08 ITEM_FACT	1	30,030		
DS	QADBIFLD QSYS QADBIFLD			3	103
DS	QADBXREF QSYS QADBXREF			2	40
LIB	NGSS			13	1
LIB	QGPL			13	

Figure 13-28 Summary section of Lock Report

You can easily identify the jobs with the long wait, by examining statistics by Requestors, by Job name, by the Holder's job name and by the Object name. In this case it is 049110/QPADEV00C/NGSS that has a long wait of 30030ms on DBITSODB08/ORDERS(ITEM_FACT).

With both detail and summary statistics provided by Lock Report, you identify the jobs with the long wait. Then you take the necessary action such as reschedule jobs or ensure Commit controls are in their correct places, to reduce or eliminate the object contention. As a result, you can help optimize your query performance.

Archived



Tools to check a performance problem

When a performance problem is occurring, it is important to understand what is happening with the system, even when you know that you are having a problem with SQL queries. There are several commands that you can use to see a high level view of what the system is doing.

Use the tools in this appendix to help you look at the big picture. They will help you determine if SQL queries are causing performance problems if system tuning needs to be done to prevent SQL query performance problems.

WRKACTJOB command

The Work with Active Jobs (WRKACTJOB) command allows you to see what jobs are using system resources. You can sort on any column. Position the cursor over the column that you want to examine and press **F16** to sort it. You can sort by CPU% to find the job or jobs that are using most of the CPU.

Figure A-1 shows the Work with Active Jobs display. You can see that the top three jobs are using the majority of CPU. Refresh this display often to see if the jobs continue to use CPU or if it was a one-time occurrence. It is important to note the function for the jobs that are using CPU. You can find the function by looking in the Function column. See if they are building indexes. If indexes are being built, the function shows *IDX-indexname*, where *indexname* is the name of the index being built.

```
Work with Active Jobs                                     RCHASCLC
                                                         03/04/05 10:13:32
CPU %:   99.8      Elapsed time: 00:00:04      Active jobs: 1300

Type options, press Enter.
  2=Change  3=Hold  4=End  5=Work with  6=Release  7=Display message
  8=Work with spooled files  13=Disconnect ...

Opt  Subsystem/Job  User      Type  CPU %  Function      Status
-----
    QCLNSYSLOG   QPGMR     BCH    22.5  CMD-DLTPRB    RUN
    QPADEV001V   KLD       INT    18.4  CMD-WRKMEDIBRM  RUN
    QPADEV0035   TSWEENEY  INT    17.5  CMD-DSPLICKEY  RUN
    Q1PDR        QPM400    BCH     4.3  PGM-Q1PBATCH  RUN
    PRTCPTTRPT   EILEENPI  BCH     3.7  PGM-QPTBATCH  RUN
    CRTPFRTA     QSYS      BCH     3.7  CMD-CRTPFRDTA  RUN
    QPADEV004W   DHUFFMAN  INT     1.4  MNU-MAIN      RUN
    QPADEV0018   HANS      INT     1.4  CMD-WRKPRB    RUN
    QRWTSRVR     QUSER     BCI     1.3               RUN

More...

Parameters or command
====>
F3=Exit  F5=Refresh  F7=Find  F10=Restart statistics
F11=Display elapsed data  F12=Cancel  F23=More options  F24=More keys
```

Figure A-1 Work with Active Jobs panel display jobs using CPU

The Work with Active Jobs display can also show jobs that are using a large amount of I/O (see Figure A-2). To view the I/O display, enter the WRKACTJOB command and then press **F11**. Then place your cursor in the AuxIO column and press **F16** to sort the column.

In the Work with Active Jobs display, the I/O count shown is only reported after an operation has completed. An example of where the WRKACTJOB I/O count for a job might not match the Work with System Activity (WRKSYSACT) count is when a blocked fetch is done. The WRKSYSACT command shows the actual I/O count, where the WRKACTJOB command does not show the I/O count until the fetch has completed.

In this example, using the WRKACTJOB command, a poor performing SQL statement might appear as though it is performing little to no I/O, but the WRKSYSACT command shows that the job is I/O intensive. For more information about the WRKSYSACT command, refer to “WRKSYSACT command” on page 459.

It is important to press F11 two more times when looking at the I/O with the WRKACTJOB command to reach the display that shows the function for the jobs. If the function is *IDX-indexname*, then the job is building an index. Further investigation must be done to determine why the job is building an index.

```

Work with Active Jobs
RCHASCLC
03/04/05 10:27:54
CPU %: 99.8 Elapsed time: 00:14:25 Active jobs: 1302

Type options, press Enter.
2=Change 3=Hold 4=End 5=Work with 6=Release 7=Display message
8=Work with spooled files 13=Disconnect ...

-----Elapsed-----
Opt Subsystem/Job Type Pool Pty CPU Int Rsp AuxIO CPU %
PRTSYSRPT BCH 2 50 62.3 29230 1.5
QSCSTT0001 BCH 2 25 5743.6 22403 .3
QSCSTT0004 BCH 2 25 6025.5 22346 .3
QPADEV0048 INT 3 20 4.0 109 2.7 15138 .0
QPADEV004W INT 3 20 36282.0 0 .0 6551 .9
QPADEV003N INT 3 20 1.5 37 2.8 3733 .0
QRWTSRVR BCI 2 20 33036.7 3601 .5
QPADEV0018 INT 3 20 2747.9 0 .0 3389 .4
AMQPCSEA BCH 2 35 12.0 2434 .0
More...

Parameters or command
====>
F3=Exit F5=Refresh F7=Find F10=Restart statistics
F11=Display thread data F12=Cancel F23=More options F24=More keys

```

Figure A-2 Work with Active Jobs display showing I/O used

WRKSYSACT command

WRKSYSACT command is provided with the Performance Tools. This command is helpful in finding jobs that use system resources. The advantage of using this command over the WRKACTJOB command is that the WRKSYSACT command shows the Licensed Internal Code (LIC) tasks in addition to the active jobs in the system.

The WRKSYSACT command, by default, sorts on CPU utilization as shows in Figure A-3. It is important to note the elapsed time. To watch for jobs using CPU, press **F10** often to see if the same jobs stay near the top of the list of jobs using CPU. These are the jobs that you want to determine what they are doing.

One way to determine what a job is doing is to look at the function of the job using the WRKACTJOB command as shown in Figure A-1. You can also use the Work with Jobs (WRKJOB) command to see what the job is doing. For more information about the WRKJOB command, refer to “WRKJOB command” on page 463. If it is known that the jobs using CPU

means the job can do other work while waiting for disk I/O to complete. *Synchronous disk I/O* is when a job has to wait for disk I/O work to be done before doing other work.

When you find the job that is using I/O, determine what the job is doing. You can use the WRKACTJOB command to find the job and see what function the job is in at the time as shown in Figure A-1 on page 458. You can also use the WRKJOB command to see what the job is doing. For more information about the WRKJOB command, refer to “WRKJOB command” on page 463.

If SQL is being run, Database Monitor data can help determine the problem. For more information about gathering Database Monitor data, refer to Chapter 4, “Gathering SQL performance data” on page 89.

Work with System Activity						RCHASCLC			
						03/04/05 10:24:50			
Automatic refresh in seconds						5			
Elapsed time		00:00:02		Average CPU util		99.9			
Number of CPUs		4		Maximum CPU util		101.8			
Overall DB CPU util		7.7		Minimum CPU util		98.8			
						Current processing capacity: 3.00			
Type options, press Enter.									
1=Monitor job 5=Work with job									
Opt	Job or Task	User	Number	Thread	Pty	CPU Util	Total Sync I/O	Total Async I/O	DB CPU Util
	SMP00001				0	1.0	0	1786	.0
	GLIDDEN51E	QEJBSVR	620898	0000000C	0	.3	173	0	.0
	LTWAS51ND	QEJBSVR	632143	00002D98	26	.4	144	0	.0
	PRTSYSRPT	EILEENPI	641085	00000089	50	1.3	46	91	.0
	QYPSJSVR	QYPSJSVR	624351	0000002C	0	.2	131	0	.0
	GNETSIZ	AJMAC	639853	00000027	50	1.1	3	124	.0
	NODEAGENT	QEJBSVR	621978	00000015	0	.2	125	0	.0
	MICK51EXP	QEJBSVR	636665	000000D2	0	.2	115	0	.0
More...									
F14=Display jobs only			F15=Display tasks only			F16=Resequence			
F24=More keys									

Figure A-5 Work with System Activity display sorted by I/O

WRKSYSSTS command

The Work with System Status (WRKSYSSTS) command provides a global view of the system. You can get a better view if you press **F21** and select the advanced assistance level. Press **F10** to restart the statistics. It is best to look at the data after a couple of minutes have elapsed. Press **F5** to refresh the display until you see 00:02:00.

Figure A-6 shows a Work with System Status display. When you have a performance problem, check the Work with System Status display to see if your *% CPU used* is higher than you normally run. You can also look at the pools to see if you have abnormally high faulting or paging.

What do you do when *% CPU used* seems high? One item to check is the *% DB Capability*. If the *% DB Capability* is also high, then it is an indication that there is SQL activity is occurring. Use the WRKSYSACT command to display the jobs using CPU. Refer to “WRKSYSACT command” on page 459 for more information.

- ▶ If a high priority job (low number) is using a lot of CPU, greater than 50% for an extended period of time, then the job can cause the entire system to have poor response times. If it is found that one or a few jobs are using the majority of CPU, then ask:
 - Is the priority of the job really appropriate?
 - Is the job running in the correct environment? For example, if the job is interactive, would it be better suited to run in batch?
 - What is the job doing?
- ▶ If the CPU utilization is high, greater than 80%, and all jobs seem to have an equal but small CPU percent, this can mean that there are too many active jobs on the system.

Work with System Status										RCHASCLC
										03/04/05 13:52:55
% CPU used	:	2.7	System ASP	:	1922 G					
% DB capability	:	.0	% system ASP used	:	88.2414					
Elapsed time	:	00:02:00	Total aux stg	:	2055 G					
Jobs in system	:	14182	Current unprotect used	:	6944 M					
% perm addresses	:	.202	Maximum unprotect	:	7476 M					
% temp addresses	:	.910								
Sys Pool	Reserved	Max	----DB----	--Non-DB---	Act-	Wait-	Act-			
Pool	Size M	Size M	Act	Fault Pages	Fault Pages	Wait	Inel	Inel		
1	788.06	227.82	+++++	.0 .0	.9 1.3	18.3	.0	.0		
2	2754.35	2.17	9514	.0 .8	11.4 42.8	2148	.0	.0		
3	4785.78	.00	233	.0 .0	4.1 4.8	32.8	.0	.0		
4	44.91	.00	11	.0 .0	.0 .0	.0	.0	.0		
5	94.20	.00	5	.0 .0	.0 .0	.0	.0	.0		
6	1.25	.00	6	.0 .0	.0 .0	.0	.0	.0		
7	94.20	.00	24	.0 .0	.0 .0	.0	.0	.0		
8	94.20	.00	5	.0 .0	.0 .0	.0	.0	.0		
9	763.33	.17	23	.0 .0	.0 .0	83.5	.0	.0		
										Bottom
===>										
F21=Select assistance level										

Figure A-6 Work with System Status display

For more information about the WRKSYSSTS command and how to view the data via iSeries Navigator, see *Managing OS/400 with Operations Navigator V5R1 Volume 5: Performance Management*, SG24-6565.

WRKOBJLCK command

Using the Work with Object Lock (WRKOBJLCK) command on user profiles can help narrow down clients that are having performance problems. For example, a user is using an ODBC connection and is complaining about having a performance problem. To find the job that the user is running on the iSeries, enter the following command:

```
WRKOBJLCK OBJ(QSYS/userprofile) OBJTYPE(*USRPRF)
```

In this example, *userprofile* refers to the user's iSeries user ID.

A panel is displayed that shows a list of jobs that the user profile has locked. You can work with each job to see if you can isolate the one that is having the problem. You can look at the call stack and objects locked to see a picture of what the job is doing.

If any program beginning with QSQ is found in the call stack or QDBGGETMQO is found, then SQL is being used. Refer to Chapter 3, "Overview of tools to analyze database performance" on page 33, to learn about other tools that you can use to further analyze the job, after the correct job is found.

WRKJOB command

You can use the Work with Job (WRKJOB) command to determine what a job is doing. It includes options to look at the job. Some of the following options might be helpful to check for jobs that have performance problems:

- Use option 1 to find the pool in which the job is running. The subsystem and subsystem pools are shown:

An example is:

```
Subsystem . . . . . : QINTER
Subsystem pool ID . . . . . : 2
```

To find the system pool in which the job is running, enter the Work with Subsystems (WRKSBS) command.

Figure A-7 shows the Work with Subsystems display for our example. As you can see, the *subsystem pool ID* is 2 and the subsystem is QINTER. We go to subsystem QINTER and look under the subsystem pool column of 2, which shows the job is using *system pool 3*.

Work with Subsystems											System:	RCHASCLC
Type options, press Enter.												
4=End subsystem 5=Display subsystem description												
8=Work with subsystem jobs												
Opt	Subsystem	Total Storage (M)	-----Subsystem Pools-----									
			1	2	3	4	5	6	7	8	9	10
	QASE5	.00	2						8			
	QASE51	.00	2	9								
	QBATCH	1.25	2								6	
	QCMN	.00	2									
	QCTL	.00	2									
	QEJBAS51	.00	2									
	QHTTPSVR	.00	2									
	QINTER	.00	2	3	4							5
	QMQM	.00	2									
	QSERVER	.00	2									
More...												
Parameters or command												
====>												
F3=Exit F5=Refresh F11=Display system data F12=Cancel												
F14=Work with system status												

Figure A-7 Work with Subsystems display

- Use option 3 to give the job run attributes, if the job is active. This option is helpful in determining how much CPU and temporary storage are being used in the job. High temporary storage use can be the result of temporary indexes being built.
- Use option 4 to see all the spooled files for the job. Check this to see if any unexpected spooled files are being created.

- ▶ Use option 10 to see the job log for the job. It is important to review the job log. Chapter 3, “Overview of tools to analyze database performance” on page 33, explains how to turn on debug messages to capture optimizer messages when running SQL.
- ▶ Use option 11 to view the call stack for the job. The example in Figure A-8 shows QDBGETMQO at the bottom of the call stack. QDBGETMQO is when the SQL Query Engine (SQE) is getting rows. SQE is discussed in Chapter 2, “DB2 for i5/OS performance basics” on page 11.

If QDBGETM is displayed at the bottom of the call stack, either the Classic Query Engine (CQE) is fetching rows or there is native I/O such as in RPG. It is important to note any user programs that are at the bottom of the call stack. If a user program is found, use the Print SQL Information (PRTSQLINF) command to see if the program contains SQL. For details about PRTSQLINF, refer to Chapter 3, “Overview of tools to analyze database performance” on page 33.

Press **F10** to monitor whether the programs in the call stack change. If any program beginning with QSQ is found in the call stack or QDBGETMQO is found, then SQL is being used.

Display Call Stack				
Job:	QPADEV0027	User:	PEGGYCL	System: RCHASCLC
		Number:	647778	
Thread:	00000046			
Rqs	Program			
Lvl	or			
	Procedure	Library	Statement	Instruction
	QSQIMAIN	QSQL		05CA
	QSQISE	QSQL		0707
	QQUDA	QSYS		03CD
	QQURA	QSYS		0087
	QQURB	QSYS		0677
	QDBGETMQO	QSYS	0000002573	
				Bottom
F3=Exit	F10=Update stack	F11=Display activation group	F12=Cancel	
F16=Job menu	F17=Top	F18=Bottom	F22=Display entire name	

Figure A-8 Display Call Stack display, when using option 11 from the WRKJOB command

iDoctor for iSeries Job Watcher

For a more in depth analysis of a performance problem, you can use the advanced analysis tool called *iDoctor for iSeries Job Watcher*, which we refer to as Job Watcher. Job Watcher is made up of two parts:

- ▶ Tools for collecting data
- ▶ Tools for analyzing and viewing the collected data

A typical situation to use the Job Watcher is for a job that is taking a long time to run but is hardly using any CPU resource and disk I/Os are not particularly excessive. Job Watcher is an excellent tool to help you determine job waits, seizures, and other types of contention. Identifying why a job or multiple jobs or threads are “not doing anything when they should be,” is a primary situation to demonstrate a key set of Job Watcher capabilities.

Job Watcher returns near real-time information about a selected set of jobs, threads, LIC tasks, or all three. It is similar in sampling function to the WRKACTJOB and WRKSYSACT system commands, where each refresh computes delta information for the ending snapshot interval. In Job Watcher, these refreshes can be set to occur automatically, even as frequently as every five seconds. Better yet, Job Watcher harvests the data from the jobs, threads, or tasks being watched in a manner that does not impact other jobs on the system while it is collecting.

Job Watcher collected data includes the following information among other details:

- ▶ Standard WRKSYSACT type information
 - CPU
 - DASD I/O breakdown
 - DASD space consumption
 - For jobs or threads, the user profile under which the job or thread is running
For prestart server jobs that were started under user profile QUSER, you can see the user profile that is currently being serviced by that job/thread, rather than QUSER.
- ▶ Expanded details on types of waits and object lock or seize conditions
- ▶ Last run SQL statements syntax
- ▶ Program or procedure call stack, 1000 levels deep

You can download Job Watcher from the Web at:

http://www.ibm.com/eserver/series/support/i_dir/idoctor.nsf

Select downloads from the left pane and then select the appropriate download option. You can sign up for a 45-day trial to use the product.

For further information about using Job Watcher, refer to the *IBM iDoctor iSeries Job Watcher: Advanced Performance Tool*, SG24-6474.

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 468. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Advanced Functions and Administration on DB2 Universal Database for iSeries*, SG24-4249
- ▶ *IBM iDoctor iSeries Job Watcher: Advanced Performance Tool*, SG24-6474
- ▶ *Stored Procedures, Triggers and User Defined Functions on DB2 Universal Database for iSeries*, SG24-6503
- ▶ *Managing OS/400 with Operations Navigator V5R1 Volume 5: Performance Management*, SG24-6565
- ▶ *Preparing for and Tuning the V5R2 SQL Query Engine on DB2 Universal Database for iSeries*, SG24-6598
- ▶ *Using AS/400 Database Monitor and Visual Explain To Identify and Tune SQL Queries*, REDP-0502

Other publications

These publications are also relevant as further information sources:

- ▶ *DB2 Universal Database for iSeries Database Performance and Query Optimization*
<http://publib.boulder.ibm.com/infocenter/series/v5r3/index.jsp?topic=/rzajq/rzajqmst.htm>
- ▶ *Star Schema Join Support within DB2 UDB for iSeries - Version 3*
http://www-1.ibm.com/servers/enable/site/education/abstracts/16fa_abs.html

Online resources

These Web sites are also relevant as further information sources:

- ▶ Information Center
<http://publib.boulder.ibm.com/infocenter/series/v5r3/index.jsp>
- ▶ iDoctor for iSeries Job Watcher
http://www.ibm.com/eserver/series/support/i_dir/idoctor.nsf

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Symbols

- % CPU used 461
- % DB Capability 461
- *BASIC DBMON parameter 91
- *DETAIL DBMON parameter 91
- *SUMMARY DBMON parameter 91

Numerics

- 1000 Record 174
- 1000 Record (SQL statement summary) 177
- 3000 Record 174, 179
- 3001 Record 174, 181
- 3002 Record 174
- 3002 Record (temporary index created) 182
- 3003 Record 174, 183
- 3004 Record 174, 184
- 3006 Record 174, 185
- 3007 Record 174, 186
- 3008 Record 174
- 3010 Record 174, 187
- 3014 Record 174, 188
- 3015 Record 189
- 3018 Record 174
- 3019 Record 174, 189
- 3020 Record 174
- 3021 Record 175
- 3022 Record 175
- 3023 Record 175
- 3025 Record 175
- 3026 Record 175
- 3027 Record 175
- 3028 Record 175
- 3030 Record 175
- 5002 Record 175
- 5722PT1 425

A

- access methods 408
 - Index probe 408
 - Index scan 408
 - Table probe 408
 - Table scan 408
- access plan 25
 - rebuilt 218
- Alternate Collating Sequence 21
- analysis tools for database performance 33
- ANDing 13
- API support for Memory Resident Database Monitor 55
- asynchronous disk I/O 460
- authentication 6
- automatic summary table 234

B

- binary-radix tree index 12
- bitmap index 13
- bitmap indexing 13
- Boolean arithmetic 13

C

- CFGPFRCOL (Configure Performance Collection) command 428
- Change Query Attributes (CHGQRYA) CL command QRYTIMLMT 85
- CHGQRYA (Change Query Attributes) CL command 85
- CHGSYSVAL QRYTIMLMT 85
- Classic Query Engine (CQE) 11, 16, 464
 - Statistics Manager 22
 - temporary index 182
- Collection Services 425
 - data to identify jobs using system resources 423
 - start 425
- Collector APIs 428
- communications 6
- Component Report 425
- compression algorithm 13
- Configure Performance Collection (CFGPFRCOL) CL command 428
- connection keyword
 - ODBC 102
 - OLE DB 104
- connection properties, OLE DB 103
- correlated subquery 313
- CPU bound 295
- CQE (Classic Query Engine) 11, 16, 464
- CREATE ENCODED VECTOR INDEX statement 13, 15
- Create Logical File (CRTLF) command 21, 211
- Create Performance Data (CRTPFRDTA) command 429
- CRTLF (Create Logical File) command 21, 211
- CRTPFRDTA (Create Performance Data) command 429
- Current SQL for a Job function 34, 78

D

- Data Access Primitives 18
- data conversion problems 97
- Data Definition Language (DDL) 15
- Data Manipulation Language (DML) 15
- data processing 6
- data source name, ODBC 101
- database architecture prior to V5R2 17
- Database Monitor 18, 424–425
 - data organization in a table 176
 - end 94
 - exit program 106
 - global data fields 175
 - JDBC client 105

- ODBC clients 101
- OLE DB client 103
- query examples 196
- record types 174
- start 90
- tips to analyze files 194

Database Monitor record types

- 1000 Record 174, 177
- 3000 Record 174, 179
- 3001 Record 174, 181
- 3002 Record 174, 182
- 3003 Record 174, 183
- 3004 Record 174, 184
- 3006 Record 174
- 3007 Record 174, 186
- 3008 Record 174
- 3010 Record 174, 187
- 3014 Record 174, 188
- 3015 Record 189
- 3018 Record 174
- 3019 Record 174, 189
- 3021 Record 175
- 3022 Record 175
- 3023 Record 175
- 3025 Record 175
- 3026 Record 175
- 3027 Record 175
- 3028 Record 175
- 3030 Record 175
- 5002 Record 175

Database Monitor table

- additional index 195
- SLQ view 195
- subset for faster analysis 194
- database performance analysis tools 33
- Database Performance Monitors 43
- DDL (Data Definition Language) 15
- debug information messages 18
- debug messages 82
- Detailed Database Monitor 90, 114
 - end 94
 - start 90, 97
- Detailed Monitor 44
- disk I/O counts 442
- display performance data 435
- distinct key list 13
- DML (Data Manipulation Language) 15
- dynamic bitmap 13
- dynamic SQL 26
 - EXECUTE IMMEDIATE statement 26
 - SQL PREPARE 26

E

- encoded-vector index (EVI) 12–13
 - recommended use 14
- End Database Monitor (ENDDBMON) command 49, 94
- ENDDBMON 94
- ENDDBMON (End Database Monitor) command 49, 94
- EVI (encoded-vector index) 12–13
 - recommended use 14

- exit program 106
- expert cache 219
- Explain Only 279
- Explain SQL 86
- explainable statement 299
- extended dynamic SQL 26
- external table description
 - Memory Resident Database Monitor 56

F

- filters 97
- full open
 - analysis 203

G

- global field
 - QQ19 175
 - QQJFLD 175
 - QQJNUM 175
 - QQJOB 175
 - QQRID 175
 - QQTIME 175
 - QQUCNT 175
 - QQUSER 175
 - QVC102 175

H

- hash key 316

I

- I/O bound 295
- iDoctor for iSeries Job Watcher 464
- index
 - advised 84, 214
 - over the Database Monitor table 195
 - temporary 211
- Index Advisor 285
 - advise for encoded-vector index 66
 - radix index suggestion 66
- Index Evaluator 67
- index only access (IOA) 410
- index scan-key selection 212
- indexing 9
 - strategy 408
 - tips 410
- IOA (index only access) 410
- iSeries Navigator
 - Create Database Files Now 429
 - Current SQL for a Job 78
 - Graph History 429, 436
 - Visual Explain 61
- isolation level 208

J

- JDBC client, enabling Database Monitor 105
- Job Watcher 464
- jobs using system resources 423

K

key value 12

L

leaf node 12
LIKE predicate 419
List Explainable Statements 303
logical file, avoid use in select statements 411

M

machine interface (MI) 17
Management Central System Monitors 429, 439
management collection object (*MGTCOL) 426
materialized query table (MQT) 234
materialized view 234
Memory Resident Database Monitor 51, 90
 analysis of data conversion problems 97
 API support 55
 QAQQ3000 56
 QAQQ3001 56
 QAQQ3002 56
 QAQQ3003 56
 QAQQ3004 56
 QAQQ3007 56
 QAQQ3008 56
 QAQQ3010 56
 QAQQQRYI 56
 QAQQTEXT 56
 QQQCSDBM 55
 QQQDSDBM 55
 QQQESDBM 55
 QQQSSDBM 55
 QQQSSDBM 55
 external table description 56
Memory-based Database Performance Monitor 44
MI (machine interface) 17
monitor data collection 90
monitor ID 94
MQT (materialized query table) 234
MQT record types 191

N

nonreusable ODP mode 177
nonsensical query 19
numeric data type conversion 415
numeric expression 416

O

object-oriented design 17, 19
ODBC 19
 connection keywords 102
 data source name 101
 Database Monitor 101
ODBC clients, enabled database monitor for 101
ODP (open data path) 300
OLE DB
 client 103

 connection keywords 104
 connection properties 103
OLTP (online transaction processing) 12
online transaction processing (OLTP) 12
Open Data Path 29
open data path 29
open data path (ODP) 83, 300
 reusable 43, 300
open processing time 6–7
optimization record 175
optimization time 6–7
ORing 13

P

perfect index 12
PERFORM menu 425
Performance Management APIs 425
Performance Tools 459
persistent indexes 12
Plan Cache 18
 definition 27
Predictive Query Governor 85
Preparing for and Tuning the SQL Query Engine on DB2
for i5/OS 21
primary key column 212
Print performance report 430
print SQL information 33, 86
Print SQL Information (PRTSQLINF) command 43, 86,
464
PRTSQLINF (Print SQL Information) command 43, 86,
464
pseudo open 203, 206
public monitor 91

Q

QAQQINI option
 IGNORE_DERIVED_INDEX 21
QAQQINI parameter
 MESSAGES_DEBUG 84
 QUERY_TIME_LIMIT 85
QDBFSTCCOL system value 227
QDBGETM 464
QDBGETMQO 464
QPFRADJ system value 5
QQRCD 322
QQRID value 174
QRWTSRVR jobs 6
QSQRPCED API 26
QSQSRVR 6
QSYS/SYSIDXADV 323
QSYS2.SYSIXADV 320
QSYS2/QYQIXADV 320
QSYS2/SYSIXADV 320
query analysis 192
query attributes and values 281
Query Dispatcher 18, 21
query engine 15
query feedback 18
Query Implementation Graph 281

query sorting 223
QUSRMBRD API 67
QZDASOINIT jobs 5

R

radix index 12
 suggestion 66
record types in Database Monitor 174
Redbooks Web site 468
 Contact us xiv
relative record number (RRN) 14, 409
 avoid using to access data 414
RENAME 21
reusable ODP 43, 300
 mode 177
RRN (relative record number) 14, 409
run time 6

S

Satisfactory SQL performance 7
satisfactory SQL performance 7
scalar function, avoid in WHERE clause 420
secondary key column 212
SELECT *, avoid use in select statements 413
select statement
 avoid use of logical files 411
 avoid use of SELECT * 413
SELECT/OMIT DDS keyword 21
select/omit logical file 211
sparse index 136, 211
SQE (SQL Query Engine) 11, 16, 464
SQE Optimizer 18
SQL
 analysis of operation types 202
 elapsed time 199
 embedded 6
 problem causing requests 197
 total time spent in 198
SQL (Standard Query Language) 15
SQL ALIAS 194
SQL package 26, 86
 advantages 27
SQL Performance Monitor 43
 Detailed Monitor 44
 Memory Resident Database Monitor 51
 properties 111
 Summary Monitor 51
 types 90
 Visual Explain 63, 278
SQL Performance Monitor Wizard 96
SQL Query Engine (SQE) 11, 16, 464
 Data Access Primitives 25
 node-based implementation 19
 staged implementation 20
 statistics advised 227
SQL statement optimization 411
SQL statement summary (1000 Record) 177
SQL view for Database Monitor table 195
Standard Query Language (SQL) 15

Start Database Monitor (STRDBMON) command 49
Start Debug (STRDBG) command 43, 82, 85
Start Performance Collection (STRPFRCOL) CL command 425, 428
Start Server Job (STRSRVJOB) CL command 85
static SQL 25
statistics 9
 advised 227
 cardinality of values 24
 frequent values 24
 metadata information 24
 selectivity 24
Statistics and Index Advisor 284
Statistics Manager 17–18, 227
STRDBG (Start Debug) command 43, 82, 85
STRDBMON 90
STRDBMON (Start Database Monitor) command 49
STRPFRCOL (Start Performance Collection) CL command 425, 428
STRSRVJOB (Start Server Job) CL command 85
subsystem pools 463
Summary Database Monitor 90
Summary Monitor 44, 51, 114
Summary Reports
 Detailed Performance Monitor 118
 Memory-Resident 146
symbol table 13
symmetric multiprocessing 20
synchronous disk I/O 461
system pool 463
system resources used by jobs 423
System Wide Index Advised table - QSYS/SYSIDXADV 323

T

table scan 209
temporary index analysis 211
temporary index created (3003 Record) 182
temporary result 84

U

user display I/O 6

V

V5R4 iSeries Information Center URL 174
vector 14
very large database (VLDB) 13
Visual Explain 18, 60
 attributes and values 294
 Explain Only 279
 icons 311
 iSeries Navigator 61
 navigating 281
 non-SQL interface 310
 query environment 293
 Run and Explain 280
 SQL Performance Monitor 63, 278
 toolbar 283

what is 276
VLDB (very large database) 13

W

WHERE clause, avoidance of scalar functions 420
Work with Active Jobs (WRKACTJOB) command 458
Work with Jobs (WRKJOB) command 459
Work with Object Lock (WRKOBJLCK) command 462
Work with Subsystems (WRKSBS) command 463
Work with System Activity (WRKSYSACT) command
458–459
Work with System Status (WRKSYSSTS) command 461
WRKACTJOB (Work with Active Jobs) command 458
WRKJOB (Work with Jobs) command 459
WRKOBJLCK (Work with Object Lock) command 462
WRKSBS (Work with Subsystems) command 463
WRKSYSACT (Work with System Activity) command
458–459
WRKSYSSTS (Work with System Status) command 461

Archived



Redbooks

OnDemand SQL Performance Analysis Simplified on DB2 for i5/OS in V5R4

(1.0" spine)

0.875" x 1.498"

460 <-> 788 pages



OnDemand SQL Performance Analysis Simplified on DB2 for i5/OS in V5R4



Explore and Filter the SQE Plan Cache to enhance Performance Analysis

Navigate the SQL Performance Monitors using the new Dashboard

Optimize your indexing strategy with the new Index Advisor

The goal of database performance tuning is to minimize the response time of your queries. It is also to optimize your server's resources by minimizing network traffic, disk I/O, and CPU time.

This IBM Redbook helps you to understand the basics of identifying and tuning the performance of Structured Query Language (SQL) statements using IBM DB2 for i5/OS. DB2 for i5/OS provides a comprehensive set of tools that help technical analysts tune SQL queries. The SQL Performance Monitors are part of the set of tools that IBM i5/OS provides for assisting in SQL performance analysis since Version 3 Release 6. These monitors help to analyze database performance problems after SQL requests are run. In V5R4 of i5/OS, iSeries Navigator provides a series of new tools for SQL performance analysis that we cover in this book. Capability of visualizing the contents of the SQE Plan Cache, SQE Plan Cache Snapshots, Dashboard, the new reporting tool, On Demand Index Advisor and evaluators such as Index and Materialized Query Tables are among the new tools we cover.

This book also presents tips and techniques based on the SQL Performance Monitors and other tools, such as Visual Explain and all the tools provided in V5R4. You'll find this guidance helpful in gaining the most out of both DB2 for i5/OS and query optimizer when using SQL.

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

SG24-7326-00

ISBN 0738486264