

# Advanced Functions and Administration

## on DB2 Universal Database for iSeries

Learn about referential integrity  
and constraints

See how Database Navigator  
maps your database

Discover the secrets of  
Visual Explain



Hernando Bedoya  
Daniel Lema  
Vijay Marwaha  
Dave Squires  
Mark Walas





International Technical Support Organization

**Advanced Functions and Administration  
on DB2 Universal Database for iSeries**

December 2001

**Take Note!** Before using this information and the product it supports, be sure to read the general information in “Special notices” on page 345.

#### **Fourth Edition (December 2001)**

This edition applies to Version 5, Release 1 of OS/400, Program Number 5722-SS1.

Comments may be addressed to:  
IBM Corporation, International Technical Support Organization  
Dept. JLU Building 107-2  
3605 Highway 52N  
Rochester, Minnesota 55901-7829

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1994, 1997, 2000, 2001. All rights reserved.

Note to U.S Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Contents</b> .....	iii
<b>Preface</b> .....	ix
The team that wrote this redbook .....	ix
Special notice .....	xi
IBM trademarks .....	xi
Comments welcome .....	xii
<b>Part 1. Background</b> .....	1
<b>Chapter 1. Introducing DB2 UDB for iSeries</b> .....	3
1.1 An integrated relational database .....	4
1.2 DB2 UDB for iSeries: An overview .....	4
1.2.1 DB2 UDB for iSeries basics .....	5
1.2.2 DB2 UDB for iSeries advanced functions .....	6
1.3 DB2 Universal Database for iSeries sample schema .....	8
<b>Chapter 2. Using the advanced functions: An Order Entry application</b> .....	11
2.1 Introduction to the Order Entry application .....	12
2.2 Order Entry application overview .....	12
2.3 Order Entry database overview .....	14
2.4 DB2 UDB for iSeries advanced functions in the Order Entry database .....	17
2.4.1 Referential integrity .....	17
2.4.2 Two-phase commit .....	18
<b>Part 2. Advanced functions</b> .....	19
<b>Chapter 3. Referential integrity</b> .....	21
3.1 Introduction .....	22
3.2 Referential integrity concepts .....	22
3.3 Defining a referential integrity relationship .....	24
3.3.1 Constraint prerequisites .....	24
3.3.2 Journaling and commitment control requirements .....	25
3.3.3 Referential integrity and access paths .....	25
3.4 Creating a referential constraint .....	28
3.4.1 Primary key and unique constraints .....	29
3.4.2 Referential constraint .....	30
3.4.3 Another example: Order Entry scenario .....	32
3.4.4 Self-referencing constraints .....	34
3.5 Constraints enforcement .....	35
3.5.1 Locking considerations .....	35
3.5.2 Referential integrity rules ordering .....	36
3.5.3 A CASCADE example .....	39
3.6 Journaling and commitment control .....	43
3.6.1 Referential integrity journal entries .....	45
3.6.2 Applying journal changes and referential integrity .....	49
3.7 Referential integrity application impact .....	50
3.7.1 Referential integrity I/O messages .....	50
3.7.2 Handling referential integrity messages in applications .....	51

3.8	Referential integrity constraint management . . . . .	52
3.8.1	Constraint states . . . . .	52
3.8.2	Check pending . . . . .	53
3.8.3	Constraint commands . . . . .	53
3.8.4	Removing a constraint . . . . .	56
3.8.5	Save and restore considerations . . . . .	59
3.8.6	Restore and journal apply: An example . . . . .	61
3.8.7	Displaying constraint information . . . . .	61
<b>Chapter 4.</b>	<b>Check constraint . . . . .</b>	<b>67</b>
4.1	Introduction . . . . .	68
4.1.1	Domain or table constraints . . . . .	68
4.1.2	Referential integrity constraints . . . . .	68
4.1.3	Assertions . . . . .	68
4.2	DB2 UDB for iSeries check constraints . . . . .	69
4.3	Defining a check constraint . . . . .	70
4.4	General considerations . . . . .	75
4.5	Check constraint integration into applications . . . . .	77
4.5.1	Check constraint I/O messages . . . . .	77
4.5.2	Check constraint application messages . . . . .	78
4.6	Check constraint management . . . . .	79
4.6.1	Check constraint states . . . . .	80
4.6.2	Save and restore considerations . . . . .	81
4.7	Tips and techniques . . . . .	82
<b>Chapter 5.</b>	<b>DRDA and two-phase commitment control . . . . .</b>	<b>83</b>
5.1	Introduction to DRDA . . . . .	84
5.1.1	DRDA architecture . . . . .	84
5.1.2	SQL as a common DRDA database access language . . . . .	84
5.1.3	Application requester and application server . . . . .	84
5.1.4	Unit of work . . . . .	85
5.1.5	Openness . . . . .	86
5.2	Comparing DRDA-1 and DRDA-2 . . . . .	86
5.3	DRDA-2 connection management . . . . .	87
5.3.1	Connection management methods . . . . .	88
5.3.2	Connection states . . . . .	88
5.4	Two-phase commitment control . . . . .	90
5.4.1	Synchronization Point Manager (SPM) . . . . .	90
5.5	DB2 UDB for iSeries SQL support for connection management . . . . .	92
5.5.1	Example of an application flow using DRDA-2 . . . . .	94
5.6	DRDA-1 and DRDA-2 coexistence . . . . .	95
5.7	Recovery from failure . . . . .	96
5.7.1	General considerations . . . . .	96
5.7.2	Automatic recovery . . . . .	96
5.7.3	Manual recovery . . . . .	97
5.8	Application design considerations . . . . .	101
5.8.1	Moving from DRDA-1 to DRDA-2 . . . . .	101
5.9	DRDA-2 program examples . . . . .	102
5.9.1	Order Entry main program . . . . .	102
5.9.2	Deleting an order . . . . .	103
5.9.3	Inserting the detail rows . . . . .	105
5.10	DRDA over TCP/IP . . . . .	108
5.10.1	Configuring DRDA over TCP/IP . . . . .	108

5.10.2	Examples of using DRDA over TCP/IP	112
5.10.3	Troubleshooting DRDA over TCP/IP	117
5.11	DB2 Connect access to an iSeries server via TCP/IP	120
5.11.1	On the iSeries server	120
5.11.2	On the workstation	122
5.11.3	Consideration	124
<b>Chapter 6.</b>	<b>DB2 Import and Export utilities</b>	<b>125</b>
6.1	Introduction	126
6.2	DB2 UDB for iSeries Import utility	126
6.2.1	CPYFRMIMPF	126
6.2.2	Data load example (file definition file)	130
6.2.3	Data load example (Data Definition Language)	133
6.2.4	Parallel data loader	137
6.3	DB2 UDB for iSeries Export utility	138
6.3.1	CPYTOIMPF	138
6.3.2	Creating the import file (TOFILE)	140
6.3.3	Exporting the TOFILE	143
6.3.4	Creating the import file (STMF)	144
6.3.5	Exporting the STMF	148
6.4	Moving data from DB2 UDB 7.2 to DB2 UDB for iSeries	149
6.4.1	First approach: Using the Export and Import utilities	149
6.4.2	Second approach: Using Export and CPYFRMIMPF	152
6.5	Moving data from DB2 UDB for iSeries into DB2 UDB 7.2	152
6.5.1	Using the Import and Export utilities	152
6.5.2	Using the CPYTOIMPF command and the Import utility	153
<b>Part 3.</b>	<b>Database administration</b>	<b>155</b>
<b>Chapter 7.</b>	<b>Database administration</b>	<b>157</b>
7.1	Database overview	158
7.1.1	New in V5R1	159
7.2	DB2 Universal Database for iSeries through Operations Navigator overview	161
7.2.1	Database functions overview	163
7.2.2	Database library functions overview	165
7.2.3	Creating an OS/400 library or collection	166
7.2.4	Library-based functions	168
7.2.5	Object-based functions	181
7.3	Run SQL Scripts	197
7.3.1	ODBC and JDBC connection	202
7.3.2	Running a CL command under SQL script	206
7.3.3	Run SQL Scripts example using a VPN journal	208
7.3.4	Run SQL Scripts Run options	210
7.3.5	DDM/DRDA Run SQL Script configuration summary	216
7.4	Change Query Attributes	217
7.5	Current SQL for a job	218
7.6	SQL Performance Monitors	220
7.6.1	Starting the SQL Performance Monitor	222
7.6.2	Reviewing the SQL Performance Monitor results	226
7.6.3	Importing data collected with Database Monitor	233
<b>Chapter 8.</b>	<b>Database Navigator</b>	<b>239</b>
8.1	Introduction	240
8.1.1	System requirements and planning	240

8.2	Finding Database Navigator	240
8.3	Finding database relationships prior to V5R1M0	242
8.4	Database Navigator maps	244
8.5	The Database Navigator map interface	246
8.5.1	Objects to Display window	253
8.5.2	Database Navigator map display	253
8.6	Available options on each active icon on a map	255
8.6.1	Table options	255
8.6.2	Index options	257
8.6.3	Constraint options	258
8.6.4	View options	259
8.6.5	Journal options	260
8.6.6	Journal receiver options	260
8.7	Creating a Database Navigator map	261
8.7.1	Adding new objects to a map	263
8.7.2	Changing the objects to include in a map	264
8.7.3	Changing object placement and arranging object in a map	265
8.7.4	Creating a user-defined relationship	266
8.8	The Database Navigator map icons	269
<b>Chapter 9. Reverse engineering and Generate SQL</b>		<b>271</b>
9.1	Introduction	272
9.1.1	System requirements and planning	272
9.1.2	Generate SQL	272
9.2	Generating SQL from the library in Operations Navigator	276
9.2.1	Generating SQL to PC and data source files on the iSeries server	281
9.2.2	Generating SQL from the Database Navigator map	289
9.2.3	Generating SQL from DDS	297
<b>Chapter 10. Visual Explain</b>		<b>301</b>
10.1	A brief history of the database and SQL	302
10.2	Database tuning so far	302
10.2.1	Query optimizer debug messages	302
10.2.2	Database Monitor	303
10.2.3	The PRTSQLINF command	303
10.2.4	Iterative approach	303
10.3	Introducing Visual Explain	304
10.3.1	What is Visual Explain	304
10.3.2	Finding Visual Explain	304
10.3.3	Data access methods and operations supported	305
10.4	Using Visual Explain with the SQL Script Center	306
10.4.1	The SQL Script Center	306
10.4.2	Visual Explain Only	307
10.4.3	Run and Explain	307
10.5	Navigating Visual Explain	308
10.5.1	Menu options	310
10.5.2	Action menu items	310
10.5.3	Controlling diagram level of detail	313
10.5.4	Displaying the query environment	314
10.5.5	Visual Explain query attributes and values	315
10.6	Using Visual Explain with Database Monitor data	318
10.7	Non-SQL interface considerations	320
10.7.1	Query/400 and Visual Explain	320



10.7.2 The Visual Explain icons . . . . .	321
10.8 SQL performance analysis using Visual Explain . . . . .	323
10.8.1 Database performance analysis methodology . . . . .	323
<b>Appendix A. Order Entry application: Detailed flow . . . . .</b>	<b>329</b>
Program flow for the Insert Order Header program . . . . .	330
Program description for the Insert Order Header program . . . . .	330
Program flow for the Insert Order Detail program . . . . .	331
Program description for Insert Order Detail program . . . . .	332
Program flow for the Finalize Order program . . . . .	333
Program description for the Finalize Order program . . . . .	334
<b>Appendix B. Referential integrity: Error handling example . . . . .</b>	<b>337</b>
Program code: Order Header entry program – T4249CINS . . . . .	338
<b>Appendix C. Additional material . . . . .</b>	<b>341</b>
Locating the Web material . . . . .	341
Using the Web material . . . . .	341
System requirements for downloading the Web material . . . . .	341
How to use the Web material . . . . .	342
<b>Related publications . . . . .</b>	<b>343</b>
IBM Redbooks . . . . .	343
Other resources . . . . .	343
Referenced Web sites . . . . .	344
How to get IBM Redbooks . . . . .	344
IBM Redbooks collections . . . . .	344
<b>Special notices . . . . .</b>	<b>345</b>
<b>Index . . . . .</b>	<b>347</b>



# Preface

Dive into the details of DB2 Universal Database (UDB) for iSeries advanced functions and database administration. This IBM Redbook aims to equip programmers, analysts, and database administrators with all the skills and tools necessary to take advantage of the powerful features of the DB2 Universal Database for iSeries relational database system. It provides suggestions, guidelines, and practical examples about when and how to effectively use DB2 Universal Database for iSeries.

This redbook contains information that you may not find anywhere else, including programming techniques for the following functions:

- ▶ Referential integrity and check constraints
- ▶ DRDA over SNA, DRDA over TCP/IP, and two-phase commit
- ▶ DB2 Connect
- ▶ Import and Export utilities

This redbook also offers a detailed explanation of the *new* database administration features that are available with Operations Navigator in V5R1. Among the tools, you will find:

- ▶ Database Navigator
- ▶ Reverse engineering and Generate SQL
- ▶ Visual Explain
- ▶ Database administration using Operations Navigator

This redbook is a follow-on from the previous redbook *DB2 UDB for AS/400 Advanced Database Functions*, SG24-4249-02. With the focus on advanced functions and administration in this fourth edition of the book, we moved the information about stored procedures and triggers into a new redbook – *Stored Procedures and Triggers on DB2 Universal Database for iSeries*, SG24-6503.

Prior to reading this redbook, you should have some knowledge of the relational database technology and application development environment on the IBM @server iSeries server.

## 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) Rochester 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 UDB for iSeries. Before joining the ITSO more than one year ago, he worked for IBM Colombia as an AS/400 IT Specialist doing presales support for the Andean countries. He has 16 years experience in the computing field and has taught database classes in Colombian universities. He holds a Masters Degree in computer science from EAFIT, Colombia. His areas of expertise are database technology, application development, and data warehousing.

**Daniel Lema** is an IT Architect at IBM Andean with 13 years of experience. Some of his projects include working with Business Intelligence, ERP implementation, and outsourcing. Previously, he worked as a Sales Specialist for the Midrange Server Product Unit (formerly the AS/400 Product Unit) helping customers and sales people in designing AS/400- and DB2/400-based solutions. He is a lecturer in Information Management and Information

Technology Planning in the Graduate School at EAFIT University and other colombian universities. He is also an Information Systems Engineer and working on his project degree for getting his Applied Mathematics Master Degree at EAFIT University in which he has already finished the academic activities.

**Vijay Marwaha** is an IT Specialist with IBM Global Services - Business Innovation Services, in Cranford, New Jersey. He has 30 years experience in the computing field. He has worked with the System/38, AS/400, and iSeries for the last 16 years. His areas of expertise are database design, application design, and development for performance, data warehousing, and availability. He is also a chemical engineer and holds an MBA from the Indian Institute of Management Calcutta.

**David F Squires** is an IT Specialist at the Technical Support center in the UK. He is a Level 2 Operations Specialist who deals with database and Main Storage issues. He has been working with the AS/400 system since it was announced back in 1988 and continues to work with the iSeries server today. He has more than 15 years experience in the computing field.

**Mark Walas** is the Technical Director of Sierra Training Services Limited in England. Sierra Training is a leading iSeries and AS/400 education provider in the United Kingdom. He is currently responsible for the education strategy and course development of Sierra Training Services. He teaches iSeries and AS/400 courses extensively. He has 23 years of experience in the computing field.

This redbook is based on the projects conducted in 1994, 1997, and 2000 by the ITSO Rochester Center.

The advisor of the *first edition* of this redbook was:

Michele Chilanti  
ITSO, Rochester Center

The authors of the first edition of this redbook in 1994 were:

Thelma Bruzadin, ITEC Brazil  
Teresa Kan, IBM Rochester  
Oh Sun Kang, IBM Korea  
Alex Metzler, IBM Switzerland  
Kent Milligan, IBM Rochester  
Clarice Rosa, IBM Italy

The advisor of the *second* and *third editions* of this redbook was:

Jarek Miszczyk  
ITSO, Rochester Center

The authors of the second edition of this redbook in 1997 were:

Hernando Bedoya, IBM Colombia  
Deepak Pai, IBM India

The authors of the third edition of this redbook in 2000 were:

Christophe Delponte, IBM Belguim  
Roger H. Y. Leung, IBM Hong Kong  
Suparna Murthy, IBM India

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

Mark Anderson  
Christopher Brandt  
Michael Cain  
Jim Cook  
John Eberhard  
Jim Flanagan  
Mietek Konczyk  
Kent Milligan  
Kathy Passe  
Tom Schrieber  
IBM Rochester

Cintia Marques  
IBM Brazil

Simona Pachiarini  
IBM Italy



Andrew Fellows  
IBM UK

## Special notice

This publication is intended to help programmers, analysts, and database administrators to implement DB2 UDB for iSeries. The information in this publication is not intended as the specification of any programming interfaces that are provided by DB2 UDB for iSeries. See the PUBLICATIONS section of the IBM Programming Announcement for DB2 UDB for iSeries for more information about what publications are considered to be product documentation.

## IBM trademarks

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

e (logo)® 	Redbooks™
IBM®	Redbooks Logo 
AFP™	MVS™
AIX®	Operating System/400®
APPN®	OS/2®
AS/400®	OS/390®
CICS®	OS/400®
COBOL/400®	PartnerWorld®
DataPropagator™	Perform™
DB2®	RPG/400®
DB2 Connect™	SAA®
DB2 Universal Database™	S/390®
Distributed Relational Database Architecture™	Sequent®
DPI®	SP™
DRDA®	SP1®
GDDM®	SP2®
Informix™	System/36™
iSeries™	TME®
MORE™	Notes®

## Comments welcome

Your comments are important to us!

We want our IBM 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](http://ibm.com/redbooks)
- ▶ Send your comments in an Internet note to:  
[redbook@us.ibm.com](mailto:redbook@us.ibm.com)
- ▶ Mail your comments to the address on page ii.



# Part 1

# Background

This part introduces the basic concepts of DB2 Universal Database for iSeries. It provides a description of the Order Entry application used to illustrate the use of the advanced features used in DB2 Universal Database for iSeries. Plus, it describes the sample database provided in DB2 Universal Database for iSeries in V5R1.







# Introducing DB2 UDB for iSeries

This chapter includes:

- ▶ A general introduction to DB2 UDB for iSeries
- ▶ An overview on the contents in this redbook
- ▶ A definition of the sample schema provided in V5R1

## 1.1 An integrated relational database

Integration has been one of the major elements of differentiation of the iSeries server platform in the information technology marketplace. The advantages and drawbacks of fully integrated systems have been the subject of endless disputes in the last few years. The success of the iSeries server indicates that integration is still considered one of the premier advantages of this platform. Security, communications, data management, backup and recovery. All of these vital components have been designed in an integrated way on the iSeries server. They work according to a common logic with a common end-user interface. They fit together perfectly, since all of them are part of the same software—the Operating System/400 (OS/400).

The integrated relational database manager has always been one of the most significant facilities that the iSeries server provides to users. Relying on a database manager integrated into the operating system means that virtually all the user data on the iSeries server is stored in a relational database and that the access to the database is implemented by the operating system itself. Some database functions are implemented at a low level in the iSeries server architecture, while some are even performed by the hardware.

A recent survey pointed out that a significant percentage of iSeries server customers do not even know that all of their business data is stored in a relational database. This might sound strange if you think that we consider the integrated database as one of the main technological advantages of the iSeries server platform. On the other hand, this means that thousands of customers use, manage, back up, and restore a relational database every day without even knowing that they have it installed on their system. This level of transparency has been made possible by the integration and by the undisputed ease of use of this platform. These have been key elements of the success of the iSeries server database system in the marketplace.

During the last couple of years, each new release of OS/400 has enhanced DB2 UDB for iSeries with a dramatic set of new functions. As a result of these enhancements, the iSeries server has become one of the most functionally rich relational platforms in the industry.

DB2 UDB for iSeries is a member of the DB2 UDB family of products, which also includes DB2 for OS/390 and DB2 Universal Database. The DB2 UDB family is the IBM proposal in the marketplace of relational database systems and guarantees a high degree of application portability and a sophisticated level of interoperability among the various platforms that are participating in the family.

## 1.2 DB2 UDB for iSeries: An overview

This section provides a quick overview of the major features of DB2 Universal Database for iSeries. A full description of the functions that are mentioned in this section can be found in several IBM manuals, for example:

- ▶ *Database Programming*, SC41-5701
- ▶ *DDS Reference*, SC41-5712
- ▶ *SQL Reference*, SC41-5612

## 1.2.1 DB2 UDB for iSeries basics

As previously mentioned, the major distinguishing characteristic of the iSeries server database manager is that it is part of the operating system. In practice, this means that the large majority of your iSeries server data is stored in the relational database. Although the iSeries server also implements other file systems in its design, the relational database on the iSeries server is the most commonly used by the customers. Your relational data is stored in the database, plus typical non-relational information, such as the source of your application programs.

### Physical files and tables

Data on the iSeries server is stored in objects called *physical files*. Physical files consist of a set of *records* with a predefined layout. Defining the record layout means that you define the data structure of the physical file in terms of the length and the type of *data fields* that participate in that particular layout.

These definitions can be made through the native data definition language of DB2 UDB for iSeries, called *Data Description Specifications* (DDS). If you are familiar with other relational database platforms, you are aware that the most common way to define the structure of a relational database is by using the data definition statements provided by the *Structured Query Language* (SQL). This is also possible on the iSeries server. The SQL terminology can be mapped to the native DB2 UDB for iSeries terminology for relational objects. An *SQL table* is equivalent to a DDS defined physical file. We use both terms interchangeably in this book. Similarly, table *rows* equate to physical file records for DB2 UDB for iSeries, and SQL *columns* are a synonym for *record fields*.

### Logical files, SQL views, and SQL indexes

By using DDS, you can define *logical files* on your physical files or tables. Logical files provide a different view of the physical data, allowing columns subsetting, record selection, joining multiple database files, and so on. They can also provide physical files with an *access path* when you define a *keyed logical file*. Access paths can be used by application programs to access records directly by key or for ensuring uniqueness.

On the SQL side, there are similar concepts. An *SQL view* is almost equivalent to a native logical file. The selection criteria that you can apply in an SQL view is much more sophisticated than in a native logical file. An *SQL index* provides a keyed access path for the physical data exactly the same way as a keyed logical file does. Still, SQL views and indexes are treated differently from native logical files by DB2 UDB for iSeries, and they cannot be considered to exactly coincide.

“Database file” refers to any DB2 UDB for iSeries file, such as a logical or physical file, an SQL table, or view. Any database files can be used by applications for accessing DB2 UDB for iSeries data.

### DB2 UDB for iSeries in a distributed environment

It is becoming more and more common for companies and businesses to organize their computing environment in a distributed way. The need to access remote data is constantly growing. DB2 UDB for iSeries provides several options for operating with remote platforms, both homogeneous and heterogeneous.

The Distributed Data Management (DDM) architecture is the basis for distributed file access. You can create a DDM file on your iSeries server and have it direct your data access requests to a remote database file. This remote file can be another DB2 UDB for iSeries database file or a Customer Information Control System (CICS) managed data file residing on a host platform. Only native data access is allowed for DDM files.

On top of the DDM architecture, IBM has created the Distributed Relational Database Architecture (DRDA). DRDA defines the protocol by which an SQL application can access remote tables and data. DB2 UDB for iSeries participates in this architecture, as do all the products of the DB2 Family. This means that your DB2 UDB for iSeries database can be accessed by any SQL application running on another iSeries server or on DB2 for OS/390, DB2 Universal Database, or DB2 for VM. A DB2 UDB for iSeries application with embedded SQL can access relational data stored in a DB2 for OS/390, DB2 for VM, or on another iSeries server. The DRDA architecture is implemented directly into OS/400.

IBM has also licensed DRDA to many other companies, such as Informix Software Inc., Ingres Corporation, and Oracle Corporation.

The iSeries server provides several other interfaces for client platforms to access DB2 UDB for iSeries data. Client Access for iSeries is a rich product that allows broad interoperability between a PC client and the iSeries server. For database access, Client Access for iSeries provides the PC with:

- ▶ A sophisticated file transfer function that allows subsetting of rows and columns
- ▶ Remote SQL APIs that you can embed in your PC programs to access data stored in DB2 UDB for iSeries tables
- ▶ An Open Database Connectivity (ODBC) interface to DB2 UDB for iSeries data that allows applications that use this protocol to access the iSeries server database

### Terminology

Since the AS/400 system (which today is the iSeries server) was developed before SQL was widely-used, OS/400 uses different terminology than what SQL uses to refer to database objects. The terms and its SQL equivalent are found in Table 1-1. The terms have been interchanged throughout the book.

*Table 1-1 SQL and OS/400 term cross reference*

SQL term	iSeries term
Table	Physical file
View	Non-keyed logical file
Index	Keyed logical file
Column	Field
Row	Record
Schema	Library, collection
Log	Journal
Isolation level	Commitment control level

## 1.2.2 DB2 UDB for iSeries advanced functions

The main purpose of this redbook is to describe, in detail and with practical examples, the rich set of functions that have been implemented in DB2 UDB for iSeries. This section provides a quick overview of the most important advanced features.

## Referential integrity

Referential integrity is a set of mechanisms by which a database manager enforces some common integrity rules among database tables. An example of a referential integrity rule is a customer master table and an invoice table. You do not want invoices related to non-existing customers (every invoice must reference a valid customer). As a consequence of this rule, it makes sense that, if somebody attempts to delete a customer with outstanding invoices, the delete operation is rejected. Without referential integrity implementation, the only way to ensure that these rules are enforced is by writing appropriate routines in the applications.

With referential integrity, this kind of rule can be implemented directly into the database. Once the rules are defined, DB2 UDB for iSeries automatically enforces them for the users.

## Check constraint

Check constraints are validity checks that can be placed on fields of database physical files and columns of SQL tables. It ensures that the value being entered in a column of a table belongs to the set of valid values defined for that column. For example, you may specify that the “legal” values for an employee evaluation field are defined as an integer, such as 2, 3, 4, or 5. Without the check constraint, a user can enter any integer value into such a column. To ensure that the actual value entered is as specified, you must use a trigger or code the rule in your application.

## DRDA and two-phase commit

DRDA is the architecture that meets the needs of application programs requiring access to distributed relational data. This access requires connectivity to and among relational database managers. The database managers can run in like or unlike operating environments and communicate and operate with each other in a way that allows them to execute SQL statements on another computer system. There are several degrees of distribution of database management system functions. DB2 UDB for iSeries currently supports the following levels of distribution:

### ► Remote unit of work

With a remote unit of work, an application program executing in one system can access data at a remote system using SQL. A remote unit of work supports access to one database system within a unit of work (transaction). If the application needs to interact with another remote database, it has to commit or rollback the current transaction, stop the current connection, and start a new connection.

### ► Distributed unit of work

With a distributed unit of work, within one unit of work, an application executing in one system can direct SQL requests to multiple remote database systems. When the application is ready to commit the work, it initiates the commit, and commitment coordination is provided by a synchronization point manager. Whether an application can update multiple databases in one unit of work depends on the two-phase commit protocol support between the application's location and the remote systems.

## Procedures and triggers

Stored procedures and triggers used to be part of the original book. Due to their importance we have decided to move them to the new redbook *Stored Procedures and Triggers on DB2 Universal Database for iSeries*, SG24-6503.

The first part of this book discusses, in detail, referential integrity, check constraints, and DRDA and two-phase commit.

## 1.3 DB2 Universal Database for iSeries sample schema

Within the code of OS/400 V5R1M0, there is a stored procedure that creates a fully functioning database. This database contains tables, indexes, views, aliases, and constraints. It also contains data within these objects. This database is used in this book to illustrate the new Database Navigator functions announced with Operations Navigator V5R1M0.

This database also helps with problem determination since the program is shipped with the OS/400 V5R1M0 code. By calling a simple program, you can create a duplicate of this database on any system running V5R1M0. This enables customers and support staff to work on the same database for problem determination.

This database can also be used as a learning tool to explain the various functions available at V5R1M0 with Database Navigator. Furthermore, it provides a method for teaching applications programmers or new database administrators how relationships can be built on the iSeries server between tables, schemas, indexes, etc.

Working on the same database provides the ability for customers around the world to see the new functionality at V5R1M0. It also simplifies the setup environment for the workshops that are created in the future for use by customers.

You create the database by issuing the following SQL statement:

```
CALL QSYS.CREATE_SQL_SAMPLE('SAMPLEDBXX')
```

This statement can be found in the pull-down box of the Run SQL Script window example shown in Figure 1-1.

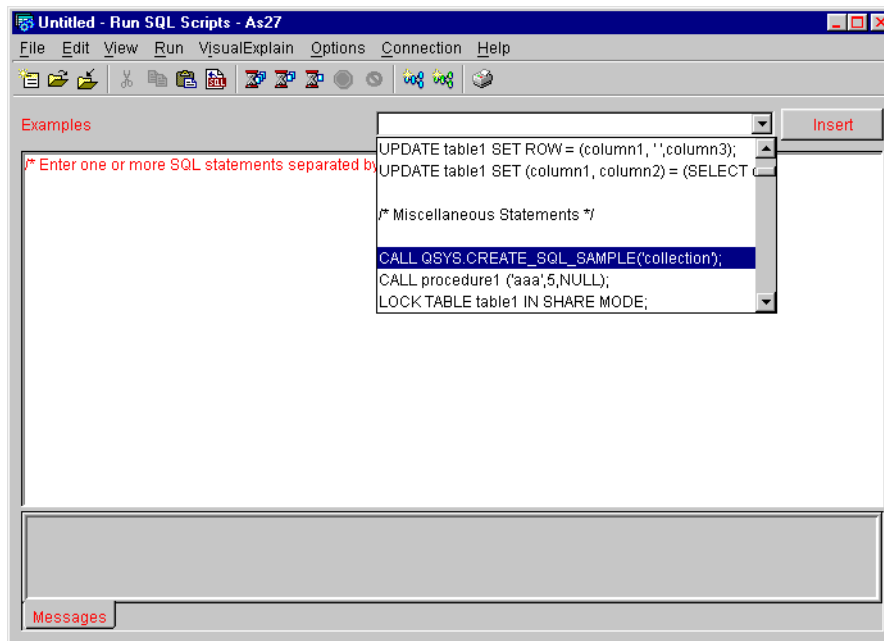


Figure 1-1 Example display showing the schema CREATE statement

**Note:** The schema name needs to be in uppercase. This sample schema will also be used in future DB2 Universal Database for iSeries documentation.

As a group, the tables include information that describes employees, departments, projects, and activities. This information makes up a sample database demonstrating some of the features of DB2 Universal Database for iSeries. An entity-relationship (ER) diagram of the database is shown in Figure 1-2.

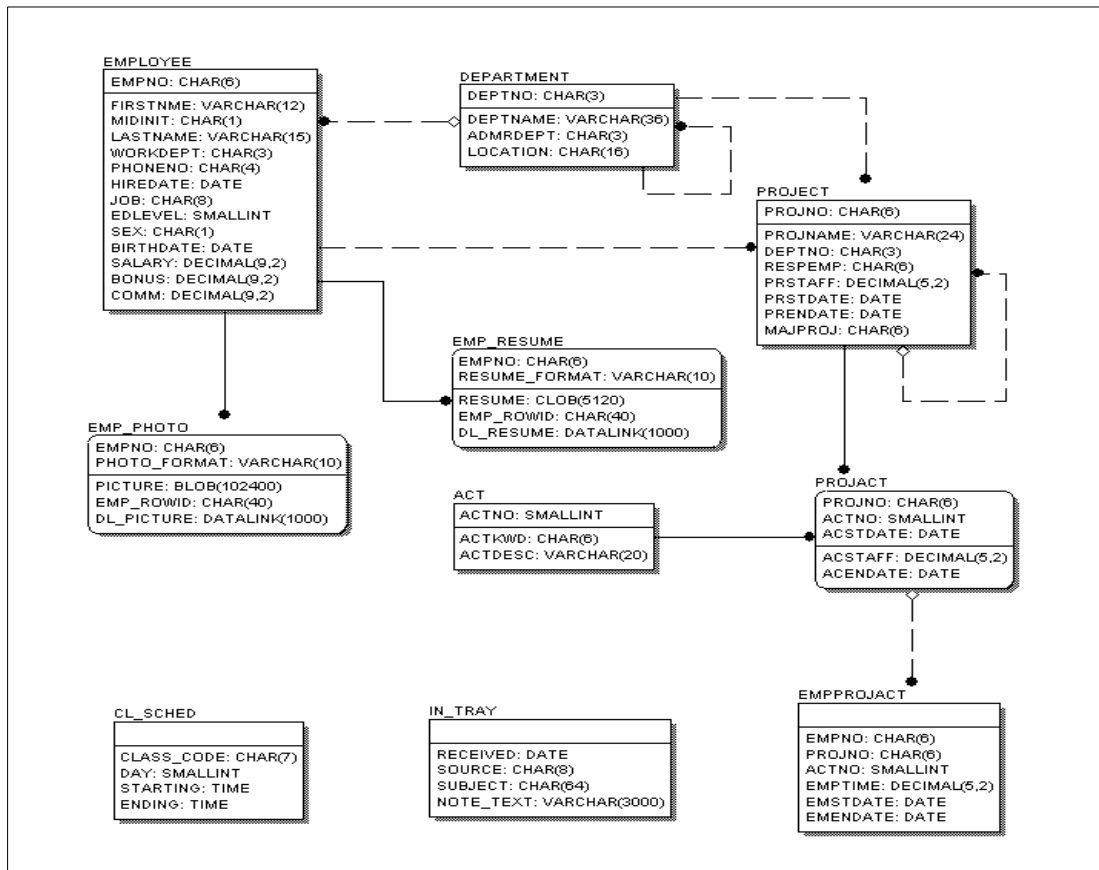


Figure 1-2 Sample schema: Entity-relationship diagram

The tables are:

- ▶ Department Table (DEPARTMENT)
- ▶ Employee Table (EMPLOYEE)
- ▶ Employee Photo Table (EMP\_PHOTO)
- ▶ Employee Resume Table (EMP\_RESUME)
- ▶ Employee to Project Activity Table (EMPPROJACT)
- ▶ Project Table (PROJECT)
- ▶ Project Activity Table (PROJACT)
- ▶ Activity Table (ACT)
- ▶ Class Schedule Table (CL\_SCHED)
- ▶ In Tray Table (IN\_TRAY)

Indexes, aliases, and views are created for many of these tables. The view definitions are not included here. There are three other tables created that are not related to the first set:

- ▶ Organization Table (ORG)
- ▶ Staff Table (STAFF)
- ▶ Sales Table (SALES)

**Note:** Some of the examples in this book use the sample database that was just described.







## **Using the advanced functions: An Order Entry application**

This chapter provides:

- ▶ A description of the Order Entry scenario
- ▶ The database structure
- ▶ The logical flow of each application component
- ▶ A highlight of the advanced functions used in this application

## 2.1 Introduction to the Order Entry application

This chapter describes how a simple Order Entry application can take advantage of the advanced functions that are available with DB2 UDB for iSeries. It provides a description of the complete application, in terms of logical flow and database structure. The actual implementation of this application can be found in the specific chapters where we exploit this application scenario to show you how to use the DB2 UDB for iSeries functions.

By presenting an application scenario, we intend to show how the advanced DB2 UDB for iSeries functions can be applied to a real-life environment and the technical implications of using those functions. For this reason, the application may appear simplistic in some respects (for example, the user interface or some design choices). We present a simple, easy-to-understand scenario that includes most of the aspects we discuss throughout this redbook.

We chose to develop the various components of the application using different programming languages to show how the various languages can interact with the DB2 UDB for iSeries functions.

As mentioned previously, the stored procedures and triggers moved to a separate redbook called *Stored Procedures and Triggers on DB2 Universal Database for iSeries*, SG24-6503.

## 2.2 Order Entry application overview

The Order Entry application shown in Figure 2-1 represents a simple solution for an office stationery wholesaler.

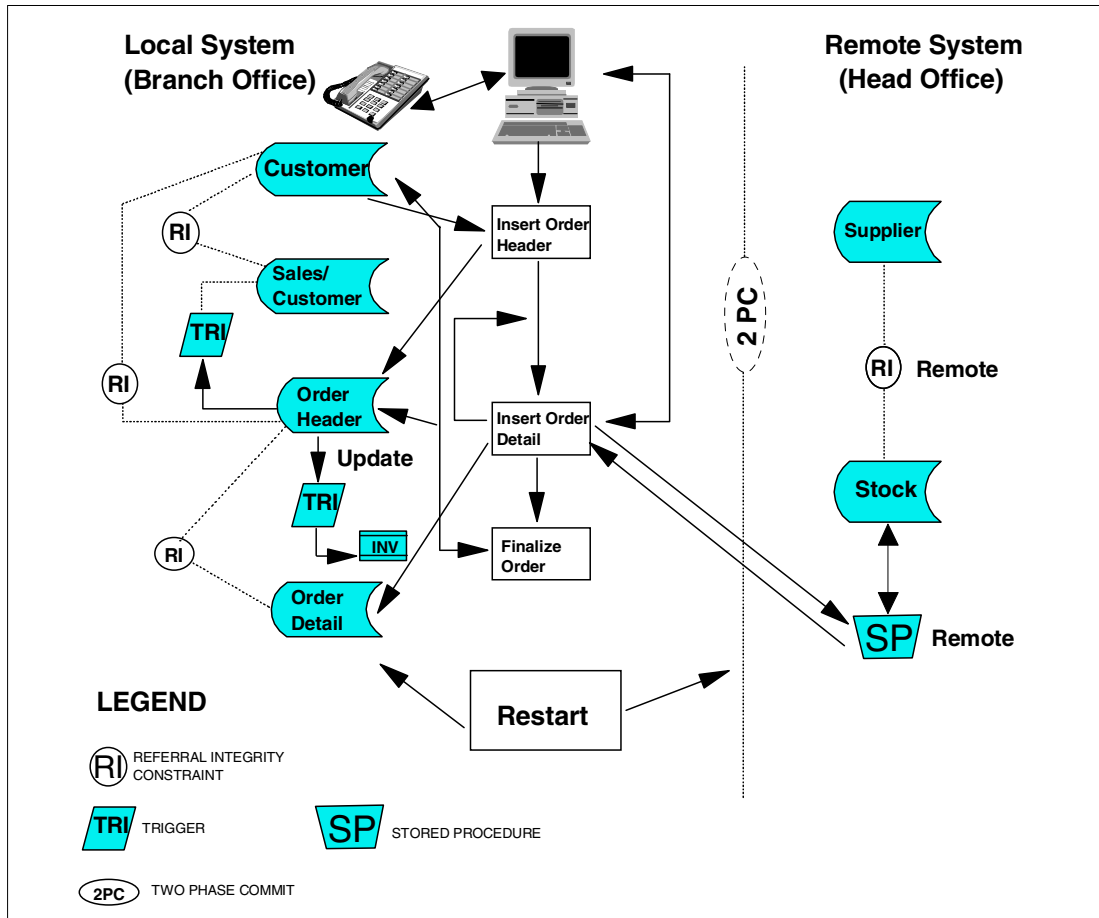


Figure 2-1 Application overview: Interaction of DB2 UDB for iSeries functions

This application has the following characteristics:

- ▶ The wholesale company runs a main office and several branch offices.
- ▶ A requirement of the branch offices is their autonomy and independence from the main office.
- ▶ Data is, therefore, stored in a distributed relational database. Information about customers and orders is stored at the branch office, where the central system keeps information about the stock and suppliers.
- ▶ A main requirement of this company is the logical consistency of the database. All orders, for example, must be related to a customer, and all the products in the inventory must be related to a supplier. This is why we need to use referential integrity in this database. See 3.4.3, “Another example: Order Entry scenario” on page 32, which describes how referential integrity can be configured for this particular scenario.
- ▶ The sales representative contacts the customer over the telephone. Each sales representative is assigned a pool of customers. According to the policy of the sales division of this company, a sales representative is allowed to place orders only for a customer of their pool. This policy is needed to guarantee a fair distribution of the commissions on the sales representative’s turnover. This requirement can be effectively enforced by means of a trigger program that automatically checks the relationship between a customer and the sales representative when the order is placed. This is addressed in *Stored Procedures and Triggers on DB2 Universal Database for iSeries*, SG24-6503.

- ▶ In placing an order, the sales representative first introduces some general data, such as the order date, the customer code, and so on. This process generates a row in the Order Header table.
- ▶ The sales representative then inserts one or more items for that specific order. If the specific item is out of stock, we want the application to look in the inventory for an alternative article. The inventory is organized in categories of products and, on this basis, the application performs a search. Since the inventory table is located remotely, we use a DRDA(\*) connection between the systems. In addition, since the process of searching the inventory may involve many accesses to the remote database, a stored procedure is called to carry out this task.
- ▶ When the item or a replacement has been found, the inventory is updated, and a row is inserted in the local order detail table.
- ▶ At this point, we want to release the inventory row to allow other people to place a new order for the same product. We commit the transaction at this time. DB2 UDB for iSeries ensures the consistency of the local and remote databases, thanks to the *two-phase commitment control* support.
- ▶ When all order items have been entered, the order is finished and a finalizing order program is called. This program can:
  - Add the total amount of the order to the Customer table to reflect the customers' turnover
  - Update the total revenue produced by the sales representative on this customer
  - Update the total amount of the order in the Order Header table
- ▶ An update event of the Order Header table starts another trigger program that writes the invoice immediately at the branch office.
- ▶ As we mentioned, the “atomic” logical transaction is completed when a single item in the order has been inserted to reduce the locking exposures. If the system or the job fails, we must be able to detect incomplete orders. This can be done when the user restarts the application. A simple restart procedure will check for orders having the total equal to zero (not “finalized”). These orders are deleted and the stock quantity of all the items is increased by the amount that we had reserved during the order placement. We can also present a choice menu to the user, asking whether the incomplete orders should be finalized.

## 2.3 Order Entry database overview

The Order Entry application is based on a distributed database. Each branch office location keeps all the data related to its own customers in its local database. The information concerning the items available in the warehouse is stored in the remote database at the head office.

The local database consists of these physical files/tables:

- ▶ **CUSTOMER** table: Contains the information related to the customers
- ▶ **ORDERHDR** table: With the data related to where the Order items are stored
- ▶ **ORDERDTL** table: Where each row represents a Detail of an Order
- ▶ **SALESCUS** table: Keeps the relationship between a sales representative and the customers for whom that sales representative is authorized to place orders

The central database consists of two tables:

- ▶ **STOCK** table: Contains information about the contents of the warehouse
- ▶ **SUPPLIER** table: Contains information related to the suppliers

Table 2-1 through Table 2-7 on page 16 show the record layouts for the files of both local and central databases.

Table 2-1 CUSTOMER table

Field name	Alias	Type	Description
CUSTOMER_NUMBER	CUSBR	CHAR(20)	Customer number
CUSTOMER_NAME	CUSNAM	CHAR(20)	Customer name
CUSTOMER_TELEPHONE	CUSTEL	CHAR(15)	Customer phone number
CUSTOMER_FAX	CUSFAX	CHAR(15)	Customer fax number
CUSTOMER_ADDRESS	CUSADR	CHAR(20)	Customer address
CUSTOMER_CITY	CUSCTY	CHAR(20)	Customer city
CUSTOMER_ZIP	CUSZIP	CHAR(5)	Customer ZIP code
CUSTOMER_CRED_LIM	CUSCRD	DEC(11,2)	Customer credit limit
CUSTOMER_TOT_AMT	CUSTOT	DEC(11,2)	Customer total amount

Table 2-2 ORDERHDR table

Field name	Alias	Type	Description
ORDER_NUMBER	ORHNBR	CHAR(5)	Order number
CUSTOMER_NUMBER	CUSBR	CHAR(5)	Customer number
ORDER_DATE	ORHTE	DATE	Order date
ORDER_DELIVERY	ORHDLY	DATE	Order delivery date
ORDER_TOTAL	ORHTOT	DEC(11,2)	Order total
ORDER_SALESREP	SRNBR	CHAR(10)	Sales Rep. number

Table 2-3 ORDERHDR table

Field name	Alias	Type	Description
ORDER_NUMBER	ORHNBR	CHAR(5)	Order number
PRODUCT_NUMBER	PRDNBR	CHAR(5)	Product number
ORDERDTL_QUANTITY	ORDQTY	DEC(5,0)	Order detail quantity
ORDERDTL_TOTAL	ORDTOT	DEC(9,2)	Order detail total

Table 2-4 SALESCUS table

Field name	Alias	Type	Description
SALESREP_NUMBER	SRNBR	CHAR(10)	Sales Rep. number
CUSTOMER_NUMBER	CUSBR	CHAR(5)	Customer number
SALES_AMOUNT	SRAMT	DEC(11,2)	Sales rep. total amount for this customer

Table 2-5 SUPPLIER table

Field name	Alias	Type	Description
SUPPLIER_NUMBER	SPLNBR	CHAR(5)	Supplier number
SUPPLIER_NAME	SPLNAM	CHAR(20)	Supplier name
SUPPLIER_TELEPHONE	SPLTEL	CHAR(15)	Supplier phone number
SUPPLIER FAX	SPLFAX	CHAR(15)	Supplier fax number
SUPPLIER ADDRESS	SPLADR	CHAR(20)	Supplier address
SUPPLIER_CITY	SPLCTY	CHAR(20)	Supplier city
SUPPLIER_ZIP	SPLZIP	CHAR(5)	Supplier ZIP code

Table 2-6 STOCK table

Field name	Alias	Type	Description
PRODUCT_NUMBER	PRDNBR	CHAR(5)	Product number
PRODUCT_DESC	PRDDDES	CHAR(20)	Product description
PRODUCT_PRICE	PRDPRC	DEC(7,2)	Product unit price
PRODUCT_AVAIL_QTY	PRDPRC	DEC(5,0)	Product available quantity
SUPPLIER_NUMBER	SPLNBR	CHAR(4)	Supplier number
PRODUCT_CATEGORY	PRDCAT	CHAR(4)	Product category
PROD_MIN_STOCK_QTY	PRDQTM	DEC(5,0)	Product minimum stock quantity

Table 2-7 STOCKPIC table

Field name	Alias	Type	Description
PRODUCT_NUMBER	PRDNBR	CHAR(5)	Product number
PRODUCT_PICTURE	PRDPIC	BLOB	Product picture

## 2.4 DB2 UDB for iSeries advanced functions in the Order Entry database

Figure 2-2 shows the Order Entry database structure and how the advanced database functions have been implemented.

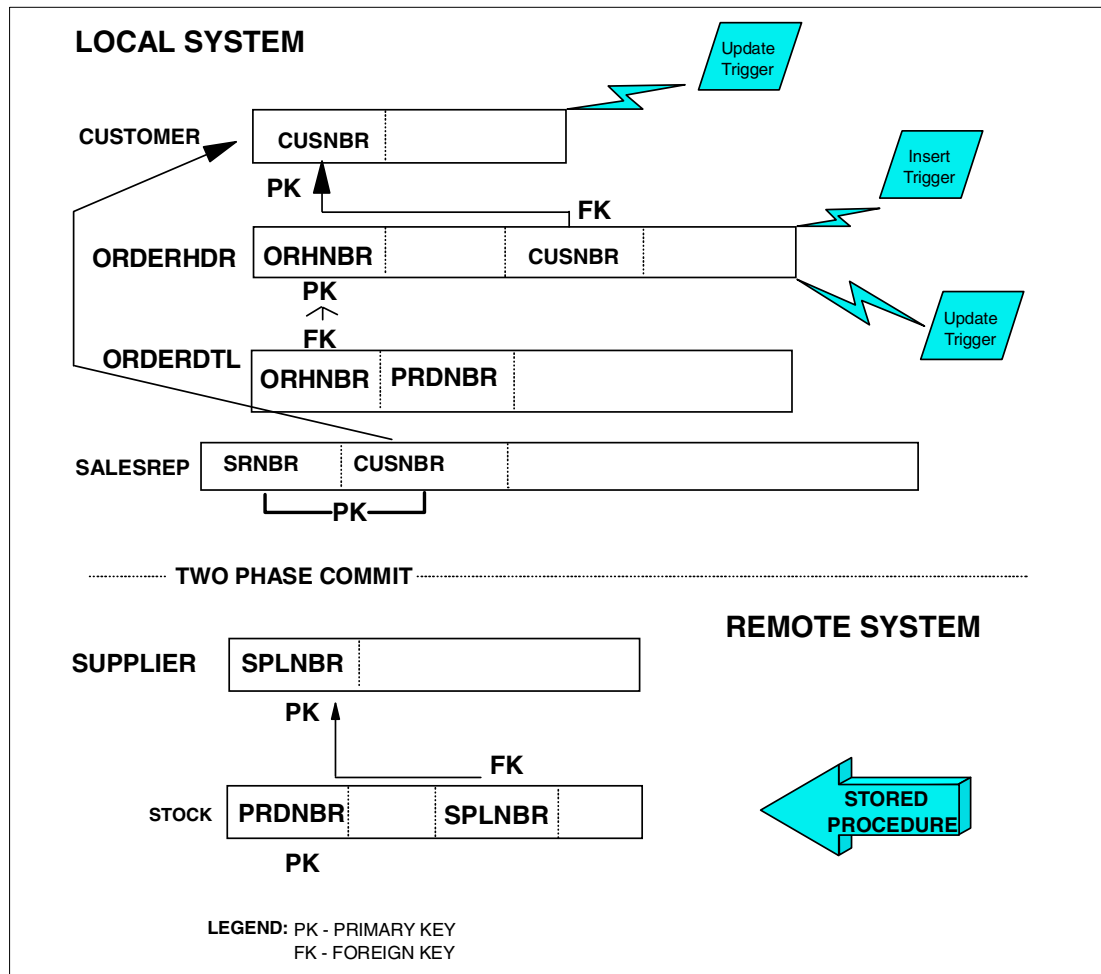


Figure 2-2 Order Entry application database structure

As stated in the overview of this chapter, the main objective of presenting this application scenario along with this specific database design is to show how the functions provided by DB2 UDB for iSeries can be used and how they can work together in a single application. Let's analyze Figure 2-2 from each function standpoint.

### 2.4.1 Referential integrity

On both the local and the remote system, the physical files/tables previously described represent entities tied to each other by logic and business relationships:

- Relationships among the CUSTOMER, ORDERHDR, and SALESREP tables:

We want every order to refer to an existing customer, and we want to prevent anybody from deleting a customer that has related orders. Similarly, each sales representative must be in charge of existing customers, so that each sales representative in the SALESREP file must be associated to a customer code that exists in the CUSTOMER table.

These two relationships are described in Figure 2-2, where the referential integrity network for our database is explained.

► Relationship between the ORDERHDR and ORDERDTL tables:

We require that every detail item in the Order Detail table be related to an existing header in the Order Header table. Additionally, when an order has to be removed, we want the detail information to be deleted as well. This business rule is translated into the arrow linking the ORHNBR column in ORDERDTL to the same column in the ORDERHDR table.

► Relationship between the STOCK and SUPPLIER tables:

At the remote side, we have a business relationship between the STOCK and SUPPLIER tables. We need to know who provides us with each of our products, so we do not want to keep an item in the STOCK table if its supplier is not present in the SUPPLIER table. For the same reason, we cannot allow the deletion of a supplier as long as we have a product provided by that supplier stored in the STOCK table. This business rule is represented by the arrow linking the SPLNBR column in the STOCK file to the same one in the SUPPLIER table.

We want these relationships to be enforced at any time, even when data is changed through interfaces, such as Interactive SQL or Data File Utility (DFU). For this reason, this scenario provides a good example of referential integrity implementation.

As described in 3.2, “Referential integrity concepts” on page 22, these relationships can easily be translated into a proper *referential integrity constraint*. Once these constraints are defined, DB2 UDB for iSeries automatically keeps our data consistent with the business rules, regardless of the kind of interface is used to change the contents of the database. Application programmers do not need to implement any integrity checking in their applications, which provides benefits in terms of ease of development and maintenance.

## 2.4.2 Two-phase commit

The company database is distributed between a central site, where the STOCK and SUPPLIER table are located, and several remote branch offices. The warehouse is located at the central site and is centrally managed there.

On the other hand, the information related to the customers and orders is independently managed at each branch office.

Consequently, our application will access both the local and the remote database. In a single unit of work, the application updates both the STOCK table on the remote side and the ORDERDTL table at the local side.

DRDA-2 and two-phase commit guarantee the consistency of the entire database, even after system failures. See Chapter 5, “DRDA and two-phase commitment control” on page 83, for a complete discussion of DRDA-2 and two-phase commit.





## Part 2

# Advanced functions

This part describes, in detail and with practical examples, the rich set of functions that have been implemented in DB2 UDB for iSeries. Among the most important advanced features, you will find:

- ▶ Referential integrity
- ▶ Check constraints
- ▶ DRDA and two-phase commit
- ▶ Import and Export utilities





# Referential integrity

This chapter discusses:

- ▶ Referential integrity concepts
- ▶ Defining referential integrity relationships
- ▶ Creating referential integrity constraints
- ▶ Constraint enforcement
- ▶ Application impacts of referential integrity
- ▶ Constraint management

## 3.1 Introduction

Referential integrity deals with relationships between data values in a relational database. These data relationships are usually closely tied with business rules. For example, every part order must reference a valid customer. In DB2 UDB for iSeries, once these relationships and rules are defined to the database manager, the system automatically ensures that they are enforced at all times, regardless of the interface used to change the data (an application, the Data File Utility, Interactive SQL, and so on).

For example, in the Order Entry database described in 2.3, “Order Entry database overview” on page 14, all the records in the Order file should have a customer number matching an existing customer in the Customer file. Moreover, a customer should not be removed when it has existing orders in the database. These are the types of data relationships that benefit from referential integrity.

In a referential integrity environment, such relationships or business rules are defined to the DB2 UDB for iSeries with *referential constraints*. DB2 UDB for iSeries supports both a native and an SQL interface for associating constraints with your database files.

Before referential integrity was available in DB2 UDB for iSeries, application programmers were responsible for enforcing these types of relationships in their programs. This programming effort is no longer needed now that the referential integrity support has been implemented in DB2 UDB for iSeries. Database management system (DBMS)-supported referential integrity provides greater application development productivity since programmers now have less code to write, test, and maintain. The integrity enforcement is done automatically by DB2 UDB for iSeries.

Application integrity enforcement also had no protection against data changes made through other interfaces, such as an interactive PC user. The constraints are now enforced in all environments resulting in greater data integrity and consistency, leaving less room for user error.

Referential integrity may also improve your application performance because the integrity checks performed by DB2 UDB for iSeries are more efficient than those done in an application program. The DBMS can use more efficient methods for enforcing these relationships at a lower level in the system that eliminates a majority of the overhead associated with application-level enforcement.

## 3.2 Referential integrity concepts

In DB2 UDB for iSeries, the following table (physical file) constraints have been introduced:

- ▶ Unique constraints
- ▶ Primary key constraints
- ▶ Referential constraints
- ▶ Check constraints

You can find a detailed definition of these constraints in *Database Programming*, SC41-5701, and *SQL Reference*, SC41-5612. This section provides the concepts and the basic definitions that are necessary for referential integrity. The terms table and physical file constraints refer to the same database definitions.

### ***Unique constraint***

A unique constraint is the rule that identifies a *unique key* in a database file. A unique key is a field or a set of fields in a physical file that must be unique, ascending, and can contain null-capable fields.

### ***Primary key constraint***

A primary key constraint identifies a *primary key* in a database file. A primary key is a field or a set of fields in a physical file that must be unique, ascending, and cannot contain null-capable fields.

**Note:** A new function was added in V4R2M0 that allows a primary key constraint to be defined where one or more columns in the key allow NULL values. When this condition is detected, a check constraint is implicitly added to the file to ensure that the column will not contain NULL values. This means that this check constraint will prevent any NULL values from being inserted into columns defined for the primary key.

### ***Parent key***

A parent key is a field or a set of fields in a physical file that must be unique, ascending, and contain null values. Both a primary and unique key can be the parent key in a referential constraint.

### ***Foreign key***

A foreign key is a field or a set of fields in a physical file whose value, if not null, must match a value of the parent key in the related referential constraint. The value of a foreign key is null if at least one of the key fields is null.

### ***Referential constraint***

A referential constraint is the file attribute that causes the database to enforce referential integrity for the defined relationship.

### ***Referential integrity***

Referential integrity is the state of a database in which the values of the foreign keys are valid. That is, each non-null foreign key value has a matching parent key value.

### ***Parent file***

The parent file contains the parent key in a referential constraint.

### ***Dependent file***

The dependent file contains the foreign key in a referential constraint.

### ***Referential constraint rules***

The referential constraint definitions also include delete and update rules that define which actions should be taken by the DBMS when a parent key value is updated or deleted.

#### **► Delete rule**

A delete rule is applied when a row in the parent file is deleted. A record is deleted from the parent file. Its parent key has matching foreign key values in the dependent file with:

- A CASCADE rule: The system also deletes all of the matching records in the dependent file.
- A SET NULL rule: The system sets all null-capable fields in the matching foreign keys to null. The foreign key fields that are not null-capable are not updated.

- A SET DEFAULT rule: The system sets the matching foreign key values to their corresponding default value. This default foreign key value must also have a matching parent key value.
- A RESTRICT rule: If at least one dependent record exists, the system prevents the parent key deletion. An exception is returned.
- A NO ACTION rule: This is similar to the restrict rule. However, enforcement is delayed until the logical end of the operation. If the operation results in a violation, the system prevents the parent key deletion and returns an exception to the user.

► **Update rule**

An update rule is applied when a parent key is updated. An update is issued for a parent key value that is matching some foreign keys in the dependent file with:

- A RESTRICT rule: If at least one dependent record exists, the system prevents the parent key update. An exception is returned.
- A NO ACTION rule: This is the same as the Restrict rule. However, enforcement is delayed until the logical end of the operation. If the operation results in a violation, the system prevents the parent key update and returns an exception to the user.

► **Check constraint**

Check constraint ensures that users authorized to change a column's value use only values that are valid for that column.

***Referential cycle or cyclic constraints***

A set of referential constraints forms a referential cycle if any file in the chain depends on itself. A simple example of a referential cycle is given by *self-referencing constraints* (referential constraints that have a primary and foreign key in the same file). See 3.4.4, “Self-referencing constraints” on page 34, for further discussion and an example.

***Check pending***

This is the state of a referential constraint when potential mismatches exist between foreign and parent keys for a constraint relationship.

## 3.3 Defining a referential integrity relationship

This section describes the considerations that you should take into account when setting up a referential integrity relationship:

- Prerequisites for a referential integrity constraint
- Journaling and commitment control requirements for referential integrity constraints
- Referential integrity access path considerations
- Verifying the current integrity of your database

### 3.3.1 Constraint prerequisites

You can find a full description of the prerequisites and limitations on the database files and the constraints themselves in *Database Programming*, SC41-5701. The basic requirement is that your parent key and foreign key must have matching field attributes and definitions. This section also points out some other considerations.

When defining a referential constraint, the foreign key and parent key null attributes do not have to exactly match. When a foreign key contains null-capable fields, DB2 UDB for iSeries treats the entire foreign key value as null whenever any of the foreign key fields is null. This behavior is defined in the standards as a *match option* of *no match*. Currently, this is the only match option supported by DB2 UDB for iSeries. The null foreign key behavior is important because referential integrity only ensures that non-null foreign keys have a matching parent key value.

You will experience better performance when your foreign key fields and parent key fields have identical null attributes. In fact, the non-null field attributes deliver the best performance.

Ideally, your parent and foreign key fields should be fairly stable, something similar to a person's social security number. This is due to the fact that, to guarantee integrity, the system must verify referential integrity each time your parent and foreign key values change. Therefore, the less your foreign and parent keys change, the less time the DBMS spends verifying referential integrity.

### 3.3.2 Journaling and commitment control requirements

When a referential constraint is defined with a delete or update rule other than RESTRICT, the system has to perform some actions on the corresponding foreign keys each time a delete or an update of the parent key takes place. For a delete case, for example, it deletes the matching dependent records when the delete rule is CASCADE. The DBMS must ensure that the parent key record and all matching dependent records are deleted. All of these record deletions must be considered as *one* logical operation.

To ensure the atomicity of this operation, the system requires journaling and commitment control in some cases. If the delete or update rule is other than RESTRICT, both the parent and the dependent files *must be journaled*. In addition, the parent and dependent file must be journaled to the same journal receiver. See 3.6, "Journaling and commitment control" on page 43, for further discussion.

Since the restrict and no action rules cause similar rule enforcement, the restrict rule provides better performance since journaling and commit are not required.

### 3.3.3 Referential integrity and access paths

DB2 UDB for iSeries uses access paths (or indexes) to perform the referential constraint enforcement as efficiently as possible. The DBMS, however, does not require its own access path for this enforcement. When a constraint is added to a physical file, the system first tries to share an existing path. If one cannot be shared, a new access path is created. This sharing is similar to the sharing performed for logical files today.

When a constraint is added to a physical file and an access path matching the constraint criteria exists, this access path is shared, and the ownership of the access path itself is transferred to the physical file. Similarly, if a logical file access path is shared, access path ownership is transferred from the logical file to the physical file. If an existing access path cannot be shared, a new one is created and owned by the physical file. The user does not have direct access to this newly created access path.

Similarly, when a logical file or an SQL index is created on a physical file with existing constraints, the system tries to share the constraint access paths. See *Database Programming*, SC41-5701, for detailed information about access path sharing.

If the existing access path has more than one key field, the constraint only shares that access path if they are defined with the same key fields in the same sequence. Partial sharing is not allowed. If the existing access path has been created over FLD1, FLD2, and FLD3, when you create a constraint, that access path is shared only if the key of the constraint exactly matches FLD1, FLD2, and FLD3. If, for example, the constraint is defined over just FLD1 and FLD2, the system has to build a new access path.

When an SQL index or logical file is deleted and the associated access path is shared by a constraint, the actual access path is left, and ownership remains with the associated physical file. Similarly, when a file constraint is removed and the access path is being shared, ownership is transferred back to the corresponding logical file or SQL index. If the constraint is not sharing an access path, both the constraint and the associated access path are removed.

Physical file constraints are not separate objects, such as logical files and SQL indexes. Referential integrity constraints and their associated access paths are part of the file description. In fact, when a physical file is saved, the system also saves all the constraints and their associated access paths that have been defined for that file.

On the contrary, when you save a physical file that has related logical files, the user is responsible for saving these logical files. For this reason, when a unique keyed access path is required, define a unique constraint instead of a logical file or an SQL index.

Since they provide a keyed access path, physical file constraints are similar to logical files. If you run an SQL query on a file with constraints defined over it, the query optimizer evaluates *all* the access paths available: logical files, SQL indexes, and constraint access paths (see the example in Figure 3-1).

For example, consider the ORDERDTL file in the Order Entry database. This file has a primary key constraint defined on the ORHNBR and PRDNBR fields, and a referential constraint with foreign key ORHNBR and parent key ORHNBR in the ORDERHDR file. We may create an SQL index ORDDTLIDX with the key fields ORHNBR and PRDNBR:

```
CREATE INDEX ORDENTL/ORDDTLIDX
      ON ORDENTL/ORDERDTL
      (ORDER_NUMBER, PRODUCT_NUMBER)
```

In this case, we find the following message in the job log:

```
CPI3210: File ORDDTLIDX in ORDENTL shares access path.
```

The second-level text specifies that the logical owner of the access path is member ORDERDTL in the ORDENTL/ORDERDTL file.

On the other hand, you may create the ORDERDTL file without a primary key constraint and create a unique logical file over ORDER\_NUMBER and PRODUCT\_NUMBER. Afterwards, if you add a referential constraint over the same fields (see 3.4, "Creating a referential constraint" on page 28), you receive the following message:

```
CPI3210: File ORDERDTL in ORDENTL shares access path.
```

The second-level text specifies that the logical owner of the access path is member ORDERDTL in the ORDENTL/ORDERDTL file. The system shares the existing access path built when the logical file was created, but the ownership of the access path itself is transferred to the physical file ORDERDTL.

If you update a record in ORDERDTL, the message shown in Figure 3-1 appears in the job log.



```

                                Display All Messages
                                System:  SYSTEM03
Job . . . :  P23KRZ75D      User . . . :  ITSCID07      Number . . . :  003869

4 > DSPJOB
   ODP created.
   Blocking used for query.
   All access paths were considered for file ORDERDTL.
   Additional access path reason codes were used.
   Arrival sequence access was used for file ORDERDTL.
   ODP created.
   ODP deleted.
   1 rows updated in ORDERDTL in ORDENTL.

                                                                More...

Press Enter to continue.

F3=Exit  F5=Refresh  F12=Cancel  F17=Top  F18=Bottom

```

Figure 3-1 SQL optimizer uses constraint access paths

The second-level text for the message (shown in bold) is shown in Figure 3-2.

```

                                Additional Message Information
Message ID . . . . . :  CPI432C      Severity . . . . . :  00
Message type . . . . . :  Information
Date sent . . . . . :  05/19/01      Time sent . . . . . :  19:20:08

Message . . . . . :  All access paths were considered for file ORDERDTL.
Cause . . . . . :  The OS/400 Query optimizer considered all access paths
built over member ORDERDTL of file ORDERDTL in library ORDENTL.
The list below shows the access paths considered. If file ORDERDTL in
library ORDENTL is a logical file then the access paths specified are actually
built over member ORDERDTL of physical file ORDERDTL in library ORDENTL.
Following each access path name in the list is a reason code which
explains why the access path was not used. A reason code of 0 indicates
that the access path was used to implement the query.
ORDENTL/ORDERDTL 4, ORDENTL/ORDDTL_HORD
The reason codes and their meanings follow:
1 - Access path was not in a valid state. The system invalidated the
access path.
2 - Access path was not in a valid state. The user requested that the
access path be rebuilt.
3 - Access path is a temporary access path (resides in library QTEMP) and
was not specified as the file to be queried.
4 - The cost to use this access path, as determined by the optimizer, was
higher than the cost associated with the chosen access method.

                                                                More...

Press Enter to continue.

F3=Exit  F6=Print  F9=Display message details  F12=Cancel
F21=Select assistance level

```

Figure 3-2 Physical file constraints evaluated by the optimizer

As highlighted in bold in Figure 3-2, ORDENTL/ORDERDTL is the access path shared by the primary key and the SQL index ORDDTLIDX. ORDENTL/ORDDTL\_HORD is the access path created for the referential constraint ORDDTL\_HORD.

### File availability

When adding a referential constraint, the DBMS exclusively locks the file and access paths involved. The system must then verify that every foreign key value is valid. This add and verification process can take as little as several seconds or minutes to complete. When the existing files contain a large number of records (hundreds of millions), this process can possibly run for hours. The add process is much quicker when the constraint access paths are shared instead of building them from scratch. Consider the impact on file availability before you create a constraint during normal system activity.

### Referential integrity verification queries

Before you create referential constraints over existing files, you may want to check if any mismatches exist between your candidate parent and foreign keys. Unmatched (or orphan) foreign key values can be determined with one of the following queries. In these queries, DEPFIL is the dependent file with a foreign key consisting of FKEYFLD, while PARFILE and PKEYFLD are the parent file and parent key:

```
SELECT * FROM mylib/DEPFIL
        WHERE FKEYFLD NOT IN
        (SELECT PKEYFLD FROM mylib/PARFILE)
```

or

```
OPNQRYF FILE((mylib/DEPFIL) (mylib/PARFILE))
        FORMAT(MYLIB/DEPFIL)
        JFLD((DEPFIL/FKEYFLD PARFILE/PKEYFLD))
        JDFTVAL(*ONLYDFT)
```

In most cases, the queries take longer to run than the system verification process performed during the execution of the Add Physical File Constraint (ADDPFCST) or Change Physical File Constraint (CHGPFCST) commands. Be careful with the verification queries on large files. This may be a good place for using DB2 UDB for iSeries Query Governor (CHGQRYA).

## 3.4 Creating a referential constraint

This section discusses the interfaces and commands that you can use to add a referential constraint and create a *referential integrity network*. A referential integrity network is a set of physical files linked by referential constraints. We also use the term *cascade network* to indicate a referential integrity network where the constraints are linked by delete cascade rules.

Two interfaces are available for creating physical file (or table) constraints:

- ▶ The native interface that supplies the new CL command, Add Physical File Constraint (ADDPFCST), to add a physical file constraint to a database file
- ▶ The SQL interface that provides:
  - CREATE TABLE statement: Has been enhanced with the CONSTRAINT clause that allows a table constraint to be added when creating a table.
  - ALTER TABLE statement: Allows a table constraint to be added to an existing table with the ADD clause.

Either interface can be used to define a constraint over physical files or SQL tables. However, only SQL supports a constraint definition at table creation time.

**Note:** In DB2 UDB for iSeries, the SQL interfaces allow you to specify a column-name longer than 10 characters. If the column-name is longer than 10 characters, a system-column name is automatically generated. The SQL constraint interface supports both the column-name and the system-column-name. In contrast, only the system-column-name can be specified when using the native interface for constraint processing. See the *SQL Reference*, SC41-5612, for further information.

### 3.4.1 Primary key and unique constraints

The first step in creating a referential constraint is to identify the parent key. You can use a unique or primary key constraint to identify the parent key.

Only one primary key constraint can be associated with a physical file. However, you can define multiple unique constraints over the same file. When a primary key constraint is added to a physical file, the associated access path becomes the primary access path of the file (for example, the access path used to access the file when the OPNDBF command is issued).

If you want to define a primary key or a unique constraint over your CUSTOMER file with customer number (CUSNBR) as the parent key, you have several options from which you can choose.

At creation time, you can define the primary key or unique constraint on the SQL CREATE TABLE statement:

```
CREATE TABLE mylib/CUSTOMER
  (CUSTOMER_NUMBER FOR COLUMN CUSNBR CHAR (5) NOT NULL ,
  ..... other fields ..... ,
  CONSTRAINT customer_key
  PRIMARY KEY (CUSNBR))
```

Or, similarly, you can define:

```
CREATE TABLE mylib/CUSTOMER
  (CUSTOMER_NUMBER FOR COLUMN CUSNBR CHAR (5) NOT NULL ,
  ..... other fields ..... ,
  CONSTRAINT customer_key
  UNIQUE (CUSNBR))
```

You can also easily add constraints to existing files. In this case, the existing records must not contain any duplicate values for the unique or primary key fields. If the system finds duplicate values, the constraint is not added, and an error message is returned.

With the native interface, you must issue the ADDPFCST command with the following parameters:

```
ADDPFCST FILE(mylib/CUSTOMER)           or   ADDPFCST FILE(mylib/CUSTOMER)
      TYPE(*PRIKEY)                         TYPE(*UNQ CST)
      KEY(CUSNBR)                           KEY(CUSNBR)
      CST(customer_key)                       CST(customer_key)
```

With SQL, the equivalent action takes place with the following two ALTER TABLE statements:

```
ALTER TABLE mylib/CUSTOMER           or   ALTER TABLE mylib/CUSTOMER
      ADD CONSTRAINT customer_key         ADD CONSTRAINT customer_key
      PRIMARY KEY (CUSNBR)               UNIQUE (CUSNBR)
```

If the physical file that was created is uniquely-keyed (with DDS), the associated access path is the primary access path and a potential parent key. In this case, a primary key constraint can be created over this file only when its fields match those of the file's primary access path. A unique constraint can be defined over any set of fields in the file capable of being a unique key.

If a physical file was not created as a unique-keyed file, a user cannot add any primary key constraint to the file. Only unique constraints can be added.

If the parent file does not have an existing keyed access path that can be shared for the primary key or unique constraint, the system creates one constraint.

### 3.4.2 Referential constraint

Now consider the Order Entry Database structure and the business rule existing between CUSTOMER and ORDERHDR files as described in 2.4.1, "Referential integrity" on page 17. In that scenario, note these points:

- ▶ A user does not want anyone to create an order for a customer that does not exist in the database. This means that we want to prevent anyone from inserting a new record in the ORDERHDR file if its corresponding Customer Number (CUSNBR) is not in the CUSTOMER file. This rule can be translated into a referential integrity constraint between CUSTOMER (parent file) and ORDERHDR (dependent file), where CUSNBR in CUSTOMER is the parent key and CUSNBR in ORDERHDR the foreign key.
- ▶ In addition, the user wants to prevent updates or removals of a customer in the CUSTOMER file when outstanding orders exist in the ORDERHDR file for this customer. To ensure this data relationship, the delete and update rule should be set to RESTRICT or NOACTION. Both the delete and update rules use RESTRICT for this particular example.

Using SQL, the constraint can be defined when we create the ORDERHDR table:

```
CREATE TABLE mylib/ORDERHDR
  (ORDER_NUMBER FOR COLUMN ORHNBR CHAR (5) NOT NULL ,
  CUSTOMER_NUMBER FOR COLUMN CUSNBR CHAR (5) NOT NULL ,
  ..... other fields ..... ,
  CONSTRAINT orderhdr_cnbr
    FOREIGN KEY (CUSNBR)
    REFERENCES mylib/CUSTOMER (CUSNBR)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
```

Otherwise, if the ORDERHDR file already exists, use a native interface:

```
ADDPFCST FILE(mylib/ORDERHDR)
TYPE(*REFCST)
KEY(CUSNBR)
CST(orderhdr_cnbr)
PRNFILE(mylib/CUSTOMER)
PRNKEY(CUSNBR)
DLTRULE(*RESTRICT)
UPDRULE(*RESTRICT)
```

Or, use the SQL interface:

```
ALTER TABLE mylib/ORDERHDR
ADD CONSTRAINT orderhdr_cnbr
FOREIGN KEY (CUSNBR)
REFERENCES mylib/CUSTOMER (CUSNBR)
ON DELETE RESTRICT
ON UPDATE RESTRICT
```

During the creation of this referential constraint, the DBMS first tries to share an existing access path for the foreign key. If one cannot be shared, the DBMS creates an access path. Once the foreign key access path is identified, DB2 UDB for iSeries then verifies that every non-null foreign key value has a matching parent key.

If the system finds invalid foreign key values during the creation of the referential constraint, the constraint is still added to the file. The DBMS also automatically disables the referential constraint and marks the relationship as check pending. However, if invalid foreign key values are found during constraint creation through the SQL interface, the constraint is not added to the file.

### Implicit creation of a primary key constraint

DB2 UDB for iSeries allows you, in some cases, to define a referential constraint on a dependent file even if there is no primary or unique key constraint defined on the parent file. In these cases, a primary key constraint with a system-generated name is implicitly added to the parent file. A requirement for the implicit creation of the primary key constraint is that the fields of the parent file, chosen as parent key fields, satisfy the conditions for parent keys: unique and not null-capable. They must also exactly match the attributes of the foreign key.

Figure 3-3 shows a situation where an implicit primary key constraint is being created.

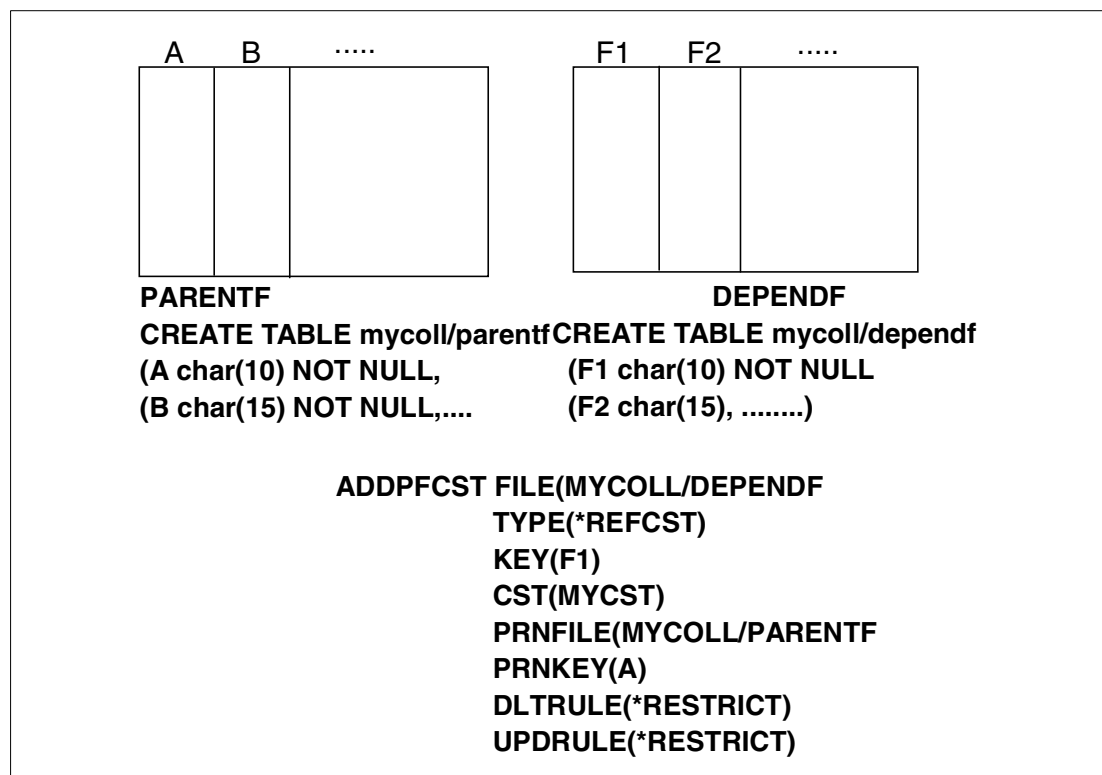


Figure 3-3 Implicit creation of a primary key constraint

In the scenario previously described, the ADDPFCST statement generates two constraints:

- ▶ MYCST, which is the referential constraint for the DEPENDF file
- ▶ A system-generated constraint that is a primary key constraint on the PARENTF file

This option is available only by using the ADDPFCST command. No implicit primary key is ever created by a CREATE TABLE or ALTER TABLE specifying a FOREIGN KEY constraint.

## Multiple constraints

You can add multiple constraints to a physical file in a single step by using the SQL CREATE TABLE statement, for example:

```
CREATE TABLE mylib/ORDERHDR
  (ORDER_NUMBER    FOR COLUMN ORHNBR CHAR (5) NOT NULL ,
   CUSTOMER_NUMBER FOR COLUMN CUSNBR CHAR (5) NOT NULL ,
   ..... other fields ..... ,
   CONSTRAINT orderhdr_key
     PRIMARY KEY (ORHNBR)
   CONSTRAINT orderhdr_cnbr
     FOREIGN KEY (CUSNBR)
     REFERENCES mylib/CUSTOMER (CUSNBR)
     ON DELETE RESTRICT
     ON UPDATE RESTRICT)
```

This statement creates an ORDERHDR file with an ORHNBR field as the primary key and CUSNBR as the foreign key in a referential constraint having CUSNBR in the CUSTOMER file as a parent key and both the delete and the update rules set to RESTRICT.

### 3.4.3 Another example: Order Entry scenario

You now set up the referential integrity network for the Order Entry database. All the business rules described in 2.4.1, "Referential integrity" on page 17, can be translated into physical file constraints.

► Key fields definition:

- Customer\_Number (CUSNBR) must be unique in the CUSTOMER file.
- Order\_Number (ORHNBR) must be unique in the ORDERHDR file.
- Order\_Number plus Product\_Number (ORHNBR plus PRDNBR) must be unique in the ORDERDTL file.
- SalesRep\_Number plus Customer\_Number (SRNBR plus CUSNBR) must be unique in the SALESCUS file.
- Supplier\_Number (SPLNBR) must be unique in the SUPPLIER file.
- Product\_Number (PRDNBR) must be unique in the STOCK file.

Each of these identifies the primary access path and can potentially be defined as a parent key.

- An order should not be inserted into the ORDERHDR file unless it references an existing customer in the CUSTOMER file. This relationship identifies a referential constraint between ORDERHDR and the CUSTOMER file. A customer should not be deleted or have their customer number changed when outstanding orders for this customer exist in the ORDERHDR file. This relationship can be enforced with the delete and update rules set to RESTRICT.
- An order detail entry should not be inserted into the ORDERDTL file without referencing a valid order number in the ORDERHDR file. This relationship identifies a referential constraint between the ORDERDTL and ORDERHDR files. When an order is deleted, all of its order detail rows have to be deleted. An order number should not be updated when it has existing detail rows in the ORDERDTL file. This leads to choosing a delete rule of CASCADE and an update rule of RESTRICT.
- A sales representative should not be inserted in the SALESCUS file until the associated customer exists in the CUSTOMER file. This identifies a referential constraint between the SALESCUS and CUSTOMER files. When a customer is removed, the corresponding sales representative information should be removed. Again, the customer number cannot

be changed if it is referenced by records in the SALESCUS. Therefore, the update rule should be RESTRICT, and the delete rule should be CASCADE.

Let's focus on the local database (the CUSTOMER, ORDERHDR, ORDERDTL, and SALESREP files) as shown in Figure 3-4.

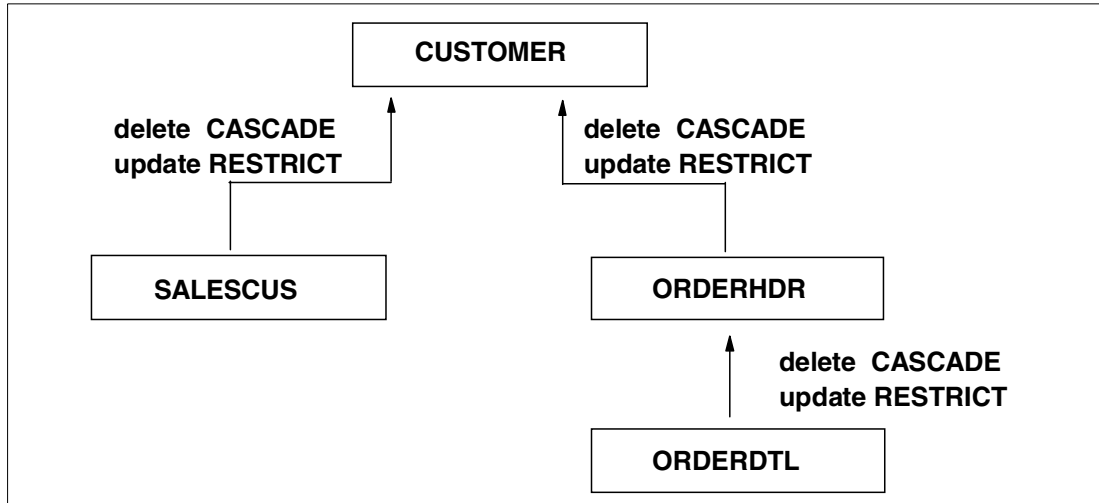


Figure 3-4 Order Entry referential integrity network

To define referential constraints, the parent key has to exist before creating the referential constraint. Follow this process:

1. Create the CUSTOMER file with a primary constraint on CUSNBR:

```

ADDPFCST FILE(mylib/CUSTOMER) TYPE(*PRIKEY)
  KEY(CUSNBR)
  CST(CustKey)
  
```

2. Create the SALESCUS file with:

- A unique constraint on SRNBR, plus CUSNBR:

```

ADDPFCST FILE(mylib/SALECUS) TYPE(*UNQCST)
  KEY((CUSNBR SRNBR))
  CST(SalesCusKey)
  
```

- A referential constraint with CUSNBR as a foreign key and CUSTOMER as a parent file:

```

ADDPFCST FILE(mylib/SALECUS) TYPE(*REFCST)
  KEY(CUSNBR)CST(SalesCusCNbr)
  PRNFILE(mylib/CUSTOMER)
  PRNKEY(*PRIKEY)
  DLTRULE(*CASCADE)
  UPDRULE(*RESTRICT)
  
```

3. Create the ORDERHDR file with:

- A primary constraint on ORHNBR:

```

ADDPFCST FILE(mylib/ORDERHDR) TYPE(*PRIKEY)
  KEY(ORHNBR)
  CST(OrderHKey)
  
```

- A referential constraint with CUSNBR as a foreign key and CUSTOMER as a parent file:

```

ADDPFCST FILE(mylib/ORDERHDR) TYPE(*REFCST)
  KEY(CUSNBR) CST(OrderHdrCNbr)
  
```

```
PRNFILE(mylib/CUSTOMER) PRNKEY(*PRIKEY)
DLTRULE(*RESTRICT) UPDRULE(*RESTRICT)
```

#### 4. Create ORDERDTL file with:

A referential constraint with ORHNBR as a foreign key and ORDERHDR as a parent file:

```
ADDPFCST FILE(mylib/ORDERDTL) TYPE(*REFCST)
KEY(ORHNBR)CST(OrderHdrNum)
PRNFILE(mylib/ORDERHDR) PRNKEY(ORHNBR)
DLTRULE(*CASCADE) UPDRULE(*RESTRICT)
```

Here is an example of the SALESCUS file using the SQL Create Table interface:

```
CREATE TABLE ordent1/SALESCUST
(SALESREP_NUMBER FOR COLUMN SRNBR CHAR(10) NOT NULL,
CUSTOMER_NUMBER FOR COLUMN CUSNBR CHAR(5) NOT NULL,
SALES_AMOUNT FOR COLUMN SRAMT DEC(11,2) NOT NULL WITH DEFAULT,
CONSTRAINT salescus_key PRIMARY KEY (SRNBR, CUSNBR),
CONSTRAINT salescus_cnbr FOREIGN KEY (CUSNBR)
REFERENCES ordent1/CUSTOMER (CUSNBR)
ON DELETE CASCADE ON UPDATE RESTRICT)
```

### 3.4.4 Self-referencing constraints

A self-referencing constraint is a referential constraint that have a primary and foreign key in the same physical file. You can use these constraints when you want to enforce a hierarchical structure on your data because a self-referential constraint implements a tree-relationship among the records of your file where the root of the tree has a null foreign key value.

When adding data to a file with a self-referential constraint, you have to follow a precise sequence. You need to start by inserting the “root” value.

For example, in a company, the EMPLOYEE file contains all the employees; Employee\_Number (EMPNO) is the primary key of the file. On the other hand, the manager of an employee must also be an employee and has to be the parent key of their associated employee records in the same file.

In this case, you need to define a referential constraint with MGRID as a foreign key and EMPNO as a parent key. EMPLOYEE is both a parent and a dependent file:

```
CREATE TABLE TEST/EMPLOYEE (EMPID      INT      NOT NULL WITH DEFAULT ,
                             NAME       CHAR(30) NOT NULL WITH DEFAULT ,
                             MGRID      INT
                             ,
                             DEPTNO     INT
                             ,
                             POSITION    CHAR(30) NOT NULL WITH DEFAULT ,
CONSTRAINT employee_key
PRIMARY KEY (EMPID),
CONSTRAINT employee_mgr
FOREIGN KEY (MGRID)
REFERENCES test/EMPLOYEE (EMPID)
ON DELETE SET NULL
ON UPDATE RESTRICT)
```

In the EMPLOYEE file example, you can only insert an employee record if the corresponding manager has already been inserted. Therefore, the first record to insert is for the Chief Executive Officer. This record's foreign key value is NULL. Afterwards, your insertions follow each branch of the hierarchy down to the lowest level.



## 3.5 Constraints enforcement

The enforcement of referential constraints is performed during any update or deletion of parent records and any time a dependent record is updated or deleted.

### 3.5.1 Locking considerations

The DBMS uses different locks on the parent and dependent rows when enforcing referential constraints. The lock type depends on the type of enforcement being performed and the delete and update rules.

#### Foreign key enforcement

The sequence for inserting a dependent row or updating a foreign key value to a non-null value is:

- ▶ A shared lock (\*SHRUPD) is obtained on the dependent file and file member.
- ▶ An update lock is obtained on the dependent record being inserted or updated.
- ▶ A read lock is established on the matching record of the parent file, if it exists.

If a matching parent key value does not exist, a referential constraint violation is signaled (CPF502D), and the requested operation is rolled back. All locks are released at the end of the operation.

#### NOACTION and RESTRICT rule enforcement

NOACTION and RESTRICT rule enforcement do not require any data changes to the matching dependent records. The DBMS immediately performs RESTRICT enforcement. NOACTION enforcement is delayed until the logical end of the operation (see the example in Figure 3-7 on page 38).

The sequence for updating or deleting a parent key value with a NOACTION or RESTRICT rule is:

1. A shared lock (\*SHRUPD) is obtained on the parent file and file member.
2. An update lock is obtained on the parent record being deleted or updated.
3. A read lock is established on the first matching record in the dependent file, if any.

If a matching foreign key value exists, a referential constraint violation is signaled (CPF503A), and the requested operation is rolled back. All locks are released at the end of the operation.

#### CASCADE, SET NULL, and SET DEFAULT rules

When the delete rule is CASCADE, SET NULL, or SET DEFAULT, deleting a parent record that has matching rows in the dependent file causes delete or update operations on the matching dependent rows.

The sequence for deleting a parent key value with CASCADE, SET NULL, or SET DEFAULT rules is:

- ▶ A shared lock (\*SHRUPD) is obtained on the parent file and file member.
- ▶ An update lock is obtained on the parent record being deleted.
- ▶ A shared lock (\*SHRUPD) is obtained only on the dependent file member. The system also logically opens and allocates the dependent file at this time.
- ▶ All matching dependent records (if any) are allocated exclusively with an update lock, and the corresponding update or delete operation is executed.

These locks are released at the end of the logical operation or the next explicit user commit. The DBMS does not logically close and de-allocate the dependent file until the parent file is closed. Therefore, other system functions, such as CLRPFM, that need exclusive access to a file cannot work on the dependent file until the parent file is closed.

If the system is unable to obtain the required locks, constraint enforcement cannot be performed and the requested operation is not allowed. This may happen, for example, when you have just deleted a parent row with a parent key value of "X" and DB2 UDB for iSeries is trying to "cascade" that deletion to the dependent file. However, another job is actually updating the dependent row that has a foreign key value of "X" at the same time. Therefore, the DBMS cannot obtain the required locks for a cascade rule. The parent row delete request is not allowed, and the following error message is returned indicating that constraint enforcement cannot be performed:

```
CPF502E: Referential constraints could not be validated for member...
```

See 3.7.1, "Referential integrity I/O messages" on page 50, for further discussion on the new CPF messages associated with referential integrity.

### 3.5.2 Referential integrity rules ordering

When a physical file is the parent file for more than one referential constraint and these constraints have different delete rules, the DBMS sequences the rules as follows:

1. The RESTRICT rule is applied first. Therefore, if at least one of the constraints has the delete rule RESTRICT, the deletion is prevented, and none of the dependent records are updated or deleted.
2. CASCADE rule
3. SET NULL rule
4. SET DEFAULT rule
5. NOACTION rule

If you have a cascade network, deleting a record in the parent file causes the deletion of all the matching records in the dependent file. If the dependent file is itself a parent file in another referential constraint, the deletion might propagate actions to the lower level and so on. If any failure occurs, the system rolls back all the changes.

On the other hand, when you mix different delete rules in your referential integrity network, you can delete a parent record. This is true only if no dependent file involved is a parent in a referential constraint that has the delete rule RESTRICT or NOACTION, or none of the records being deleted has any dependent record. Figure 3-5 shows this situation.

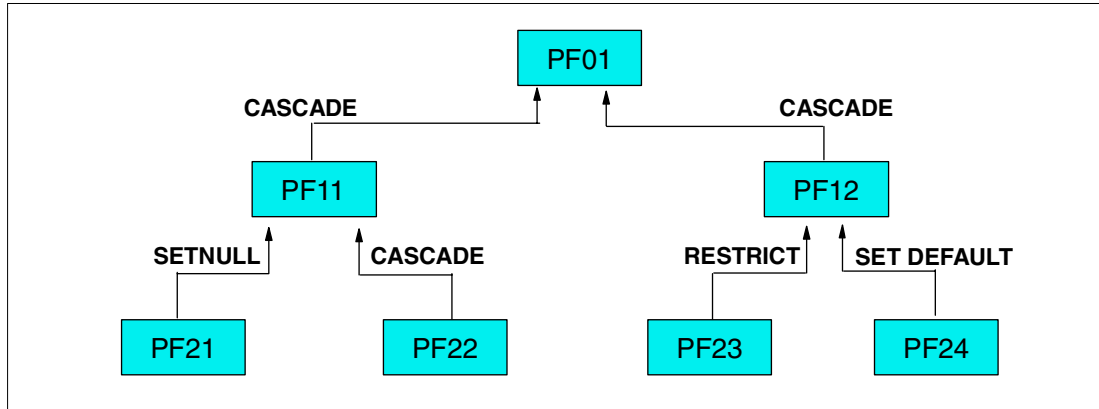


Figure 3-5 Delete propagation in a referential integrity network

In the referential integrity network shown in Figure 3-5, a delete operation on PF01 causes:

- ▶ A deletion of the dependent records in PF11. Each of these deletions, in turn, issues:
  - Updating the related dependent records in PF21, and setting the foreign key values to NULL.
  - Deleting all of the related dependent records in the PF22 file.
- ▶ By deleting the dependent records in PF12, each of these deletions, in turn, causes:
  - Updating the related dependent records in PF24, and setting the foreign key values to their default values.
  - But, for the constraint existing between PF12 and PF23, the delete rule is RESTRICT. Therefore, if the records that are about to be deleted in PF12 have dependent records in PF23, their deletion is prevented. In turn, since it cannot delete all the records in the PF12 file, the system prevents even the deletion of the original record in PF01.

In this example, note these points:

- ▶ The delete operation from PF01 is executed.
- ▶ The cascaded deletions to PF11 and PF12 are performed. As soon as the records are deleted from PF12, the RESTRICT rule is enforced.
- ▶ The system issues an error message and rolls back the previous deletions, ending the implicit commitment control cycle.

On the contrary, if you do not have a RESTRICT or NOACTION rule when the user or an application issues a delete on PF01, as shown in Figure 3-6, the following actions occur:

- ▶ The delete from PF01 is executed.
- ▶ The cascaded deletions to PF11 and PF12 are performed.
- ▶ The cascaded deletions to PF22 are performed.
- ▶ The SET NULL rule on PF21 is executed.
- ▶ The SET DEFAULT rule on PF24 is handled.
- ▶ If any failure occurs, the system rolls back all the changes.

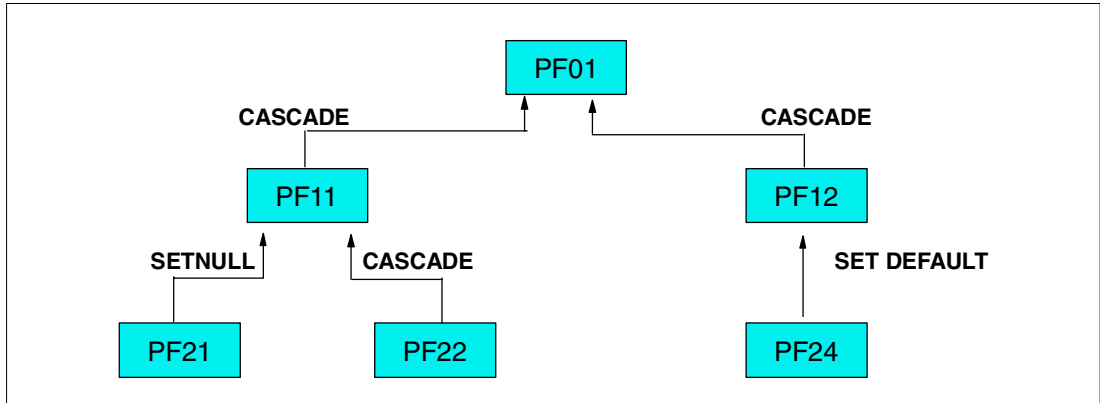


Figure 3-6 Delete propagation in a referential integrity network

Whether you use a RESTRICT rule or a NOACTION rule broadly depends on your application environment needs and whether you intend to add database triggers to your database. Even if you do not intend to define any trigger on your database, you may still want to differentiate between RESTRICT and NOACTION, especially if the parent key in the referential integrity relationship is subject to operations that affect multiple rows, such as an SQL UPDATE statement.

**Note:** For a discussion on how DB2 UDB for iSeries sequences the referential integrity rules and the execution of trigger programs, refer to *Stored Procedures and Triggers on DB2 Universal Database for iSeries*, SG24-6503.

Consider the example shown in Figure 3-7.

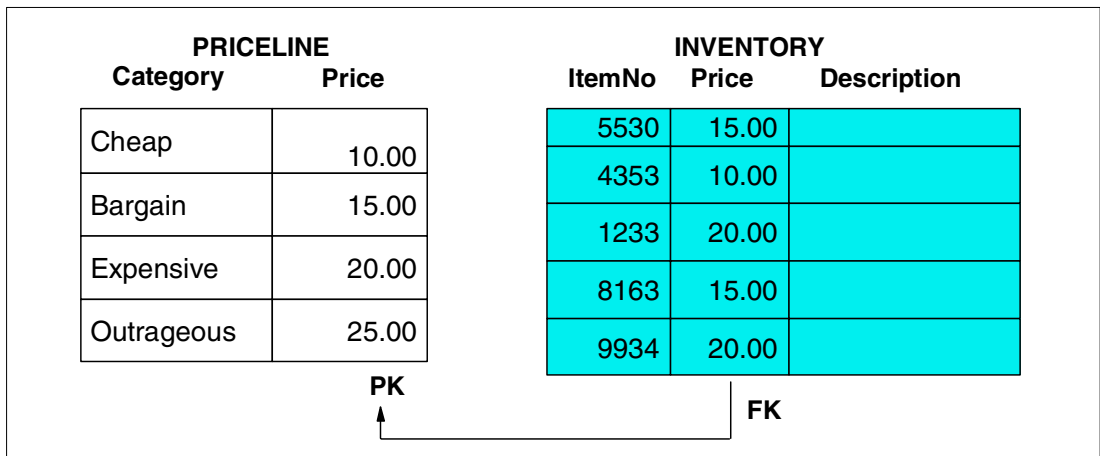


Figure 3-7 Impact of RESTRICT versus NOACTION

If you want to update your PRICETABLE and lower all the prices by five dollars, you may use the following SQL statement:

```
UPDATE PRICETABLE SET PRICE = PRICE - 5.00
```

The statement runs successfully if the referential integrity rule for the constraint shown in Figure 3-7 is NOACTION. After the first record is updated, a parent key value of \$10.00 no longer exists for the INVENTORY file. However, a NOACTION rule allows the enforcement to be delayed until after all the rows have been updated. At this point, a parent key value of \$10.00 exists and no constraint violation is signaled.

### 3.5.3 A CASCADE example

A database contains the following files:

- ▶ ORDERH: Contains the Order Headers
- ▶ DETAIL: Contains the items of any order
- ▶ FEATURE: Contains all the features associated with the products in the DETAIL file

In this case, a record cannot be inserted in FEATURE if the related product is not in the DETAIL file. Likewise, you cannot insert an order item in DETAIL if the related order header is not in ORDERH.

On the other hand, when you delete an order, you should remove all the related items and all the corresponding features from the database. For this reason, you need to define two referential constraints:

- ▶ The first one between FEATURE and DETAIL
- ▶ The second one between DETAIL and ORDERH

For both constraints, the delete rule must be CASCADE. The update rule can be either RESTRICT or NOACTION.

Now, create the tables that were previously described:

```
CREATE TABLE TEST/ORDERH
  (ORDER_NUMBER      FOR COLUMN ORHNBR CHAR (5) NOT NULL,
   CUSTOMER_NUMBER  FOR COLUMN CUSNBR CHAR (5) NOT NULL,
   ORDER_INS_DATE   FOR COLUMN ORHDTE DATE NOT NULL,
   ORDER_DELIV_DATE FOR COLUMN ORHDLY DATE NOT NULL,
   ORDER_TOTAL      FOR COLUMN ORHTOT DEC(11,2) NOT NULL WITH DEFAULT 0,
   CONSTRAINT ORDERH_KEY PRIMARY KEY (ORHNBR))
```

```
CREATE TABLE TEST/DETAIL
  (ORDER_NUMBER      FOR COLUMN ORHNBR CHAR (5) NOT NULL,
   PRODUCT_NUMBER    FOR COLUMN PRDNBR CHAR (5) NOT NULL,
   PRODUCT_QUANTITY  FOR COLUMN PRDQTY DEC (5, 0) NOT NULL,
   PRODUCT_TOTAL     FOR COLUMN PRDTOT DEC (9, 2) NOT NULL,
   CONSTRAINT DETAIL_KEY PRIMARY KEY (ORHNBR, PRDNBR),
   CONSTRAINT DETAIL_ORD FOREIGN KEY (ORHNBR)
     REFERENCES TEST/ORDERH (ORHNBR)
     ON DELETE CASCADE ON UPDATE RESTRICT)
```

```
CREATE TABLE TEST/FEATURE
  (ORDER_NUMBER      FOR COLUMN ORHNBR CHAR (5) NOT NULL,
   PRODUCT_NUMBER    FOR COLUMN PRDNBR CHAR (5) NOT NULL,
   FEATURE_NUMBER    FOR COLUMN FTRNBR CHAR (5) NOT NULL,
   FEATURE_QUANTITY  FOR COLUMN FTRQTY DEC(5,0) NOT NULL,
   FEATURE_TOTAL     FOR COLUMN FTRTOT DEC(9,2) NOT NULL,
   CONSTRAINT FTR_KEY PRIMARY KEY (ORHNBR, PRDNBR, FTRNBR),
   CONSTRAINT FTR_PRD FOREIGN KEY (ORHNBR, PRDNBR)
     REFERENCES TEST/DETAIL (ORHNBR,PRDNBR)
     ON DELETE CASCADE ON UPDATE RESTRICT)
```

If TEST is not an SQL collection, you must explicitly start journaling the files to the same journal. The following commands create the journal and journal receiver and start journaling for ORDERH, DETAIL, and FEATURE:

```

CRTJRNRCV JRNRCV(mylib/JRNRCV)

CRTJRN JRN(mylib/JRN) JRNRCV(mylib/JRNRCV)
MNGRCV(*SYSTEM) DLTRCV(*YES)

STRJRNP FILE(TEST/ORDERH
          TEST/DETAIL
          TEST/FEATURE)
          JRN(mylib/JRN)

```

You can insert a complete order interactively or through an application according to the following logic sequence:

1. Insert the order data into ORDERH.
2. Insert a product into DETAIL. If this item has features, insert the related features into FEATURE.
3. Repeat this point down to the last order item.

If any error occurs during this process, issue a ROLLBACK. If all the operations end successfully, you may COMMIT the inserts.

For example, you may insert the order data shown in Figure 3-10 on page 43.

If you try to insert a dependent record before you insert the related parent record, the system cannot perform the insertion, and an error message is issued. In our example, the following insert statement may be performed before you insert the corresponding order header data in ORDERH:

```
INSERT INTO TEST/DETAIL VALUES ('77120', '00200', 5, 500)
```

In this case, the system issues the message:

```
CPF502D: Referential constraint violation on member DETAIL.
```

The second-level text explains that you cannot insert that record because it does not match any parent key (Figure 3-8).

```

Additional Message Information

Message ID . . . . . : CPF502D      Severity . . . . . : 30
Message type . . . . . : Notify
Date sent . . . . . : 06/05/01     Time sent . . . . . : 18:11:17

Message . . . . . : Referential constraint violation on member DETAIL.
Cause . . . . . : The operation being performed on member DETAIL file
DETAIL in library TEST failed. Constraint DETAIL_ORD prevents record number
0 from being inserted or updated in member DETAIL of dependent file DETAIL
in library TEST because a matching key value was not found in member ORDERH
of parent file ORDERH in library TEST. If the record number is zero, then
the error occurred on an insert operation. The constraint rule is 1. The
constraint rules are:
  1 -- *RESTRICT
  2 -- *NOACTION
Recovery . . . . . : Either specify a different file, change the file, or
change the program. Then try your request again.

More...

Press Enter to continue.

F3=Exit  F6=Print  F9=Display message details  F12=Cancel
F21=Select assistance level

```

*Figure 3-8 Inserting a foreign key that does not match any parent key value*

Likewise, you may try to update a row in DETAIL having matching records in FEATURE, for example:

```

UPDATE TEST/DETAIL SET PRDNBR = '99999'
WHERE PRDNBR = '00420'

```

In this case, the system issues the following message:

CPF503A: Referential constraint violation on member DETAIL.

The second-level text explains that you cannot update that product number because it has depending features (Figure 3-9).

```

Additional Message Information

Message ID . . . . . : CPF503A      Severity . . . . . : 30
Message type . . . . . : Sender copy
Date sent . . . . . : 06/05/01      Time sent . . . . . : 18:27:26

Message . . . . . : Referential constraint violation on member DETAIL.
Cause . . . . . : The operation being performed on member DETAIL file
DETAIL in library TEST failed. Constraint FTR_PRD prevents record number 3
from being deleted or updated in member DETAIL of parent file DETAIL in
library TEST because a matching key value exists in member FEATURE of
dependent file FEATURE in library TEST. The constraint rule is 1. The
constraint rules are:
  1 -- *RESTRICT
  2 -- *NOACTION
Recovery . . . . . : Either specify a different file, change the file, or
change the program. Then try your request again.
Possible choices for replying to message . . . . . :
                                                    More...

Press Enter to continue.

F3=Exit  F6=Print  F9=Display message details  F12=Cancel
F21=Select assistance level

```

*Figure 3-9 Updating a parent key that has matching foreign keys*

Figure 3-10 shows how deleting from one of the files propagates to the dependent files. For example, the deletion of product number 00420 from DETAIL issues the deletion of three records in FEATURE. Deleting order number 77120 causes the deletion of three records in DETAIL. Each of these propagates the deletion to its matching records in FEATURE. With a single statement, all the matching rows in the cascade network are deleted:

```
DELETE FROM TEST/ORDERH
```

Here, ORHNBR = '77120'.



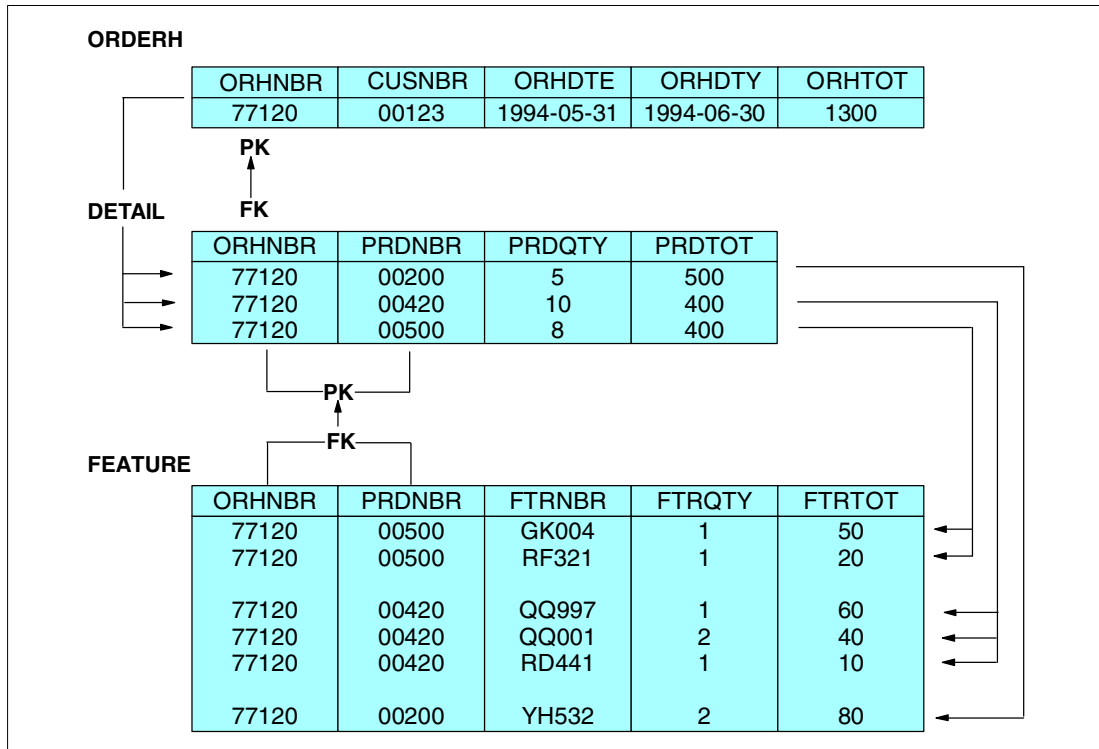


Figure 3-10 Example of a cascade network

In a cascade network with multiple levels, DB2 UDB for iSeries implements what is called the *breadth cascade* as opposed to the *depth cascade* implemented elsewhere. In the scenario described in Figure 3-10, DB2 UDB for iSeries deletes the record from the Order Header file first. Then, it deletes all the records from the DETAIL file, and then, all the records from the FEATURE file.

### 3.6 Journaling and commitment control

As stated in 3.3.2, “Journaling and commitment control requirements” on page 25, if a referential integrity network has update and delete rules other than RESTRICT, the DBMS requires journaling and commitment control. Again, this requirement helps DB2 UDB for iSeries ensure the atomicity of operations that change or delete multiple records due to referential constraints. Either *all* or *none* of the record operations must complete.

For example, you may delete a record that activates a chain of cascaded deletes. If some failure occurs during the cascade process before the DBMS can delete all the dependent records, all the records deleted so far are undeleted and the parent and dependent files are returned to their previous state. Journaling and commitment control enable the DBMS to ensure this type of transaction atomicity.

Both the parent and dependent files must be journaled and journaled to the same receiver. Technically, only the parent file needs to be journaled for NO ACTION rules. In addition, the user is responsible for starting the journaling of their physical files.

However, the user can use system change journal management when setting up the journaling environment to offload journal management responsibilities to the system. If MNGRCV(\*SYSTEM) and DLTRCV(\*YES) are specified on the CRTJRN or CHGJRN commands, the system automatically manages the attachment of a new journal receiver and deletes the old receiver after the new one has been attached. Therefore, the user can choose to start journaling and let the system take care of the management work.

In contrast, the system implicitly starts a commitment control cycle for the user if the delete or update rule requires commitment control whenever the current application or user is running with no commitment control.

This implicit commitment control cycle is transparent to the user and application program. If any failure occurs before the update or delete operation has been carried out by the system, all the changes related to the database operation are rolled back automatically. Other changes previously made by the application are not affected by this automatic rollback.

Let's consider the example shown in Figure 3-11, where the application working on those files is not using commitment control.

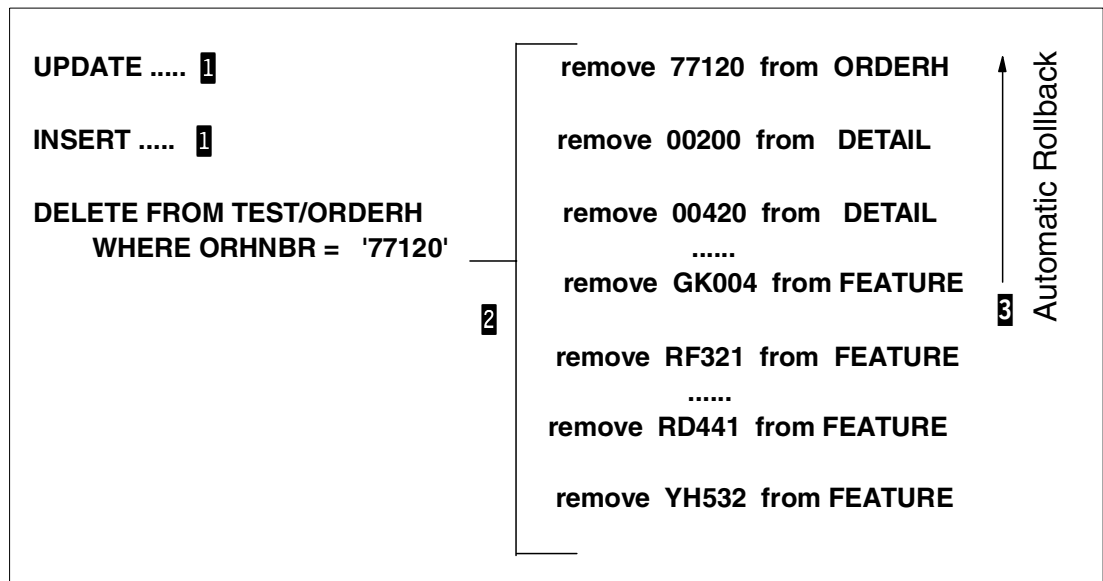


Figure 3-11 System-started commitment control cycle

When the DELETE operation is performed, DB2 UDB for iSeries activates an implicit commitment control cycle **2**. If a failure occurs in **3**, the records that were removed are placed back into the files. Any changes in **1** are not affected by an automatic rollback.

Figure 3-12 shows the same scenario as previously described, but with a native RPG ILE program handling the delete cascade.

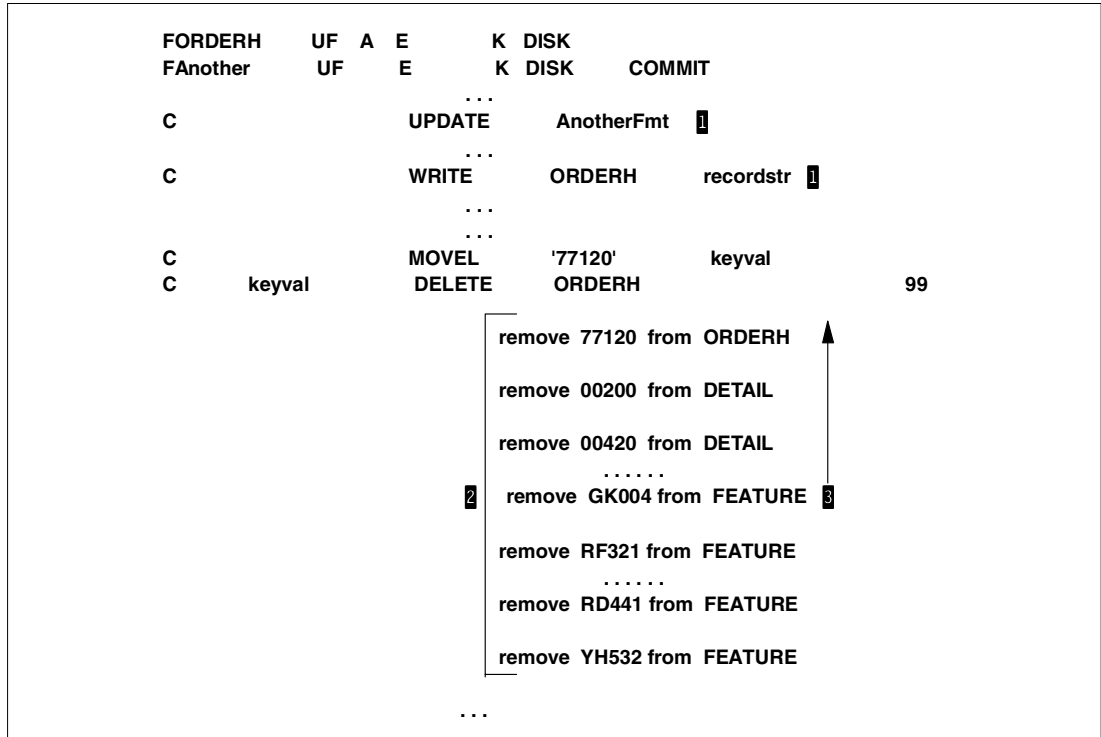


Figure 3-12 A native application and a delete cascade

### 3.6.1 Referential integrity journal entries

A new attribute has been added to the journal entries to identify which journal entries were created as a result of referential constraint enforcement. The term *side-effect journal entries* is used in this discussion to refer to these new entries.

This side-effect information is identified by the new *Ref Constraint (Yes/No)* parameter in the Display Journal Entry Details display. If a record is deleted from a dependent file directly, the change is recorded into the journal with an entry specifying Ref Constraint is No. If the same record is deleted by DB2 UDB for iSeries as the result of enforcing a Delete CASCADE rule, the system records a side-effect journal entry having Ref Constraint set to Yes.

If you consider the example in Figure 3-10 on page 43, when you delete a record from ORDERH, the system automatically removes all the related products and, for each product, all the corresponding features. When you remove order 77120, the system logs the journal entries shown in Figure 3-13:

```
DELETE FROM TEST/ORDERH
```

Here, ORHNBR = '77120'.

Display Journal Entries							
Journal . . . . . : QSQJRN				Library . . . . . : TEST			
Type options, press Enter.							
5=Display entire entry							
Opt	Sequence	Code	Type	Object	Library	Job	Time
	2304	F	OP	ORDERH	TEST	P23KRZ75D	22:37:02
	2305	C	BC			P23KRZ75D	22:37:03
	2306	C	SC			P23KRZ75D	22:37:03
	2309	R	DL	ORDERH	TEST	P23KRZ75D	22:37:03
	2311	R	DL	DETAIL	TEST	P23KRZ75D	22:37:04
	2312	R	DL	DETAIL	TEST	P23KRZ75D	22:37:04
	2313	R	DL	DETAIL	TEST	P23KRZ75D	22:37:04
	2315	R	DL	FEATURE	TEST	P23KRZ75D	22:37:05
	2316	R	DL	FEATURE	TEST	P23KRZ75D	22:37:05
	2317	R	DL	FEATURE	TEST	P23KRZ75D	22:37:05
	2318	R	DL	FEATURE	TEST	P23KRZ75D	22:37:05
	2319	R	DL	FEATURE	TEST	P23KRZ75D	22:37:05
	2320	R	DL	FEATURE	TEST	P23KRZ75D	22:37:05
	2322	F	CL	ORDERH	TEST	P23KRZ75D	22:37:05
	2323	C	EC			P23KRZ75D	22:37:06

Figure 3-13 Journal entries after deleting a parent record

Referring to Figure 3-13, OP means Open Member, CL is Close Member, and DL means Delete Record. The BC entry corresponds to a Start Commitment Control operation, and the SC entry is a Start of Commit cycle (the delete action was performed with Commitment Control level \*CHG).

After the parent file is opened and the commitment control cycle is started, an application first deletes the parent record (Entry# - 2309). The DBMS then gains control and enforces the associated delete CASCADE rules, causing all the matching rows in the dependent file (all the products) and eventually all the features related to the products to be deleted. Side-effect journal entries (2311 through 2320) are logged as a result of the constraint enforcement performed by DB2 UDB for iSeries.

**Note:** Until the parent file is closed (entry 2322) in this delete cascade operation, you *cannot* run the Change Journal (CHGJRN) command on this journal. This is due to the fact that the system requires all of the files involved in this logical transaction to be closed so that a synchronization point can be established for this journal. After this synchronization point is established, the system de-allocates the journal, making it available to any system function.

If you use option 5 on the DL entry for the ORDERH file, the complete entry for the explicit parent key delete is shown in Figure 3-14.

```

                                Display Journal Entry

Object . . . . . : ORDERH          Library . . . . . : TEST
Member . . . . . : ORDERH          Sequence . . . . . : 2309
Code . . . . .   : R - Operation on specific record
Type . . . . .   : DL - Record deleted

                                Entry specific data
Column   *...+....1....+....2....+....3....+....4....+....5
00001    '77120001232001-05-312001-06-30      '

                                                                 Bottom

Press Enter to continue.

F3=Exit  F6=Display only entry specific data
F10=Display only entry details  F12=Cancel  F24=More keys

```

Figure 3-14 Journal entry information for a delete cascade operation

The corresponding entry details are shown in Figure 3-15.

```

                                Display Journal Entry Details

Journal . . . . . : QSQJRN          Library . . . . . : TEST
Sequence . . . . . : 2309

Code . . . . .   : R - Operation on specific record
Type . . . . .   : DL - Record deleted

Object . . . . . : ORDERT          Library . . . . . : TEST
Member . . . . . : ORDERT          Flag . . . . .   : 1
Date . . . . .   : 06/07/01        Time . . . . .   : 22:37:03
Count/RRN . . . . : 2              Program . . . . . : QCMD

Job . . . . .    : 005547/ITSCID07/P23KRZ75D
User profile . . . : USERID07      Ref Constraint . . . : No
Commit cycle ID . . : 2306         Trigger . . . . . : No

Press Enter to continue.

F3=Exit  F10=Display entry  F12=Cancel  F14=Display previous entry
F15=Display only entry specific data

```

Figure 3-15 Application-related journal entry

As shown in bold in Figure 3-15, the system reports that this delete operation was not the result of referential constraint enforcement.

In contrast, the side-effect entry details all specify Ref Constraint as yes. For example, the complete entry 2311 is shown in Figure 3-16.

```

                                Display Journal Entry

Object . . . . . :  DETAIL          Library . . . . . :  TEST
Member . . . . . :  DETAIL          Sequence . . . . . :  2311
Code . . . . .   :  R - Operation on specific record
Type . . . . .   :  DL - Record deleted

                                Entry specific data
Column  *...+....1....+....2....+....3....+....4....+....5
00001   '7712000420          '

                                                                Bottom

Press Enter to continue.

F3=Exit  F6=Display only entry specific data
F10=Display only entry details  F12=Cancel  F24=More keys

```

Figure 3-16 Journal entry information for a dependent record

This deletes product 00420. The corresponding detailed information is shown in Figure 3-17.

```

                                Display Journal Entry Details

Journal . . . . . :  QSQJRN          Library . . . . . :  TEST
Sequence . . . . . :  2311

Code . . . . .   :  R - Operation on specific record
Type . . . . .   :  DL - Record deleted

Object . . . . . :  DETAIL          Library . . . . . :  TEST
Member . . . . . :  DETAIL          Flag . . . . .   :  1
Date . . . . .   :  06/07/01        Time . . . . .   :  22:37:04
Count/RRN . . . . :  3              Program . . . . . :  QCMD

Job . . . . .    :  005547/ITSCID07/P23KRZ75D
User profile . . . :  USERID07      Ref Constraint . . :  Yes
Commit cycle ID . . :  2306         Trigger . . . . . :  No

Press Enter to continue.

F3=Exit  F10=Display entry  F12=Cancel  F14=Display previous entry
F15=Display only entry specific data

```

Figure 3-17 Journal entry details for a referential integrity side-effect journal entry

Notice that the field marked in bold in Figure 3-17 means that this delete operation was performed by the DBMS due to referential constraint enforcement.

### 3.6.2 Applying journal changes and referential integrity

When you apply or remove journal changes, DB2 UDB for iSeries does not allow referential constraints to prevent the recovery of your database files. Although each apply or remove change is allowed, the associated referential constraints are constantly verified to prevent you from violating the referential integrity of your database. If the journal change violates referential integrity, the constraint is marked as check pending, and the system continues on to the next journal entry. See the check pending discussion in 3.8.1, “Constraint states” on page 52.

Moreover, during the process of applying or removing journal changes, update and delete rules are ignored. If you have a cascade delete rule, for example, removing a record from the parent file does not remove any of the dependent records. This is because the dependent record changes are also recorded in your journal with the side-effect journal entries discussed in 3.6.1, “Referential integrity journal entries” on page 45. These entries can be applied as well.

This design allows you to use the journal entries to recover your database files to a known state without violating the integrity of your database.

To avoid check pending situations, you must apply or remove journal changes on all files in your referential integrity network to ensure that your related parent and dependent files are recovered to the same data level.

Consider the example in Figure 3-10 on page 43. If you experience a data loss, you may need to restore all the files in the referential integrity network. When you apply the journal changes, include all the files involved in the referential integrity network:

```
APYJRNCHG JRN(TEST/QSQJRN)
          FILE((*ALL))
          CMTBDY(*YES)
```

This way, you are protected from check pending conditions and from data inconsistencies.

On the other hand, if you apply the journal entries only to ORDERH, order 77120 is deleted, but all the related products are still in the database. The system allows you to apply the journal changes with the following command:

```
APYJRNCHG JRN(TEST/QSQJRN)
          FILE((TEST/ORDERH))
          CMTBDY(*YES)
```

The `DETAIL_ORD` constraint (between `ORDERH` and `DETAIL`) is found in the established or enabled state, with a check pending status of `YES`. To bring the two files back to the same data level, you may also apply the journal changes to the other files in the network. Consider our example, `DETAIL` and `FEATURE`:

```
APYJRNCHG JRN(TEST/QSQJRN)
          FILE((TEST/DETAIL) (TEST/FEATURE))
          CMTBDY(*YES)
```

At this point, you have to re-enable the constraints so that the system can re-verify this relationship.

Figure 3-18 summarizes the database changes that can cause a check pending condition (marked with CP) when they are applied through an Apply Journal Changes (APYJRNCHG) command only to a parent or only to a dependent file and, similarly, when they are removed from some, but not all, of the network files.

On dependent files			On parent files		
	APY	RMV		APY	RMV
Insert	CP	--	Insert	--	CP
Update	CP	CP	Update	CP	CP
Delete	--	CP	Delete	CP	--

Figure 3-18 Check pending after APYJRNCHG

Always apply or remove journal entries within commit boundaries, starting from the beginning of a logical unit of work down to the end of a logical unit of work, because the system guarantees the data consistency within the commit boundaries. Therefore, when you apply journal changes, set the CMTBDY value to \*YES in the APYJRNCHG command.

## 3.7 Referential integrity application impact

Before referential integrity is implemented, referential integrity validations must be performed by the application program. Now you can let DB2 UDB for iSeries ensure your data integrity through the referential integrity constraints.

As mentioned earlier, using referential integrity may improve your application performance. The integrity checks are much more efficient and quicker when performed at the operating system level rather than by an application.

However, once a programmer has defined referential constraints to the DBMS, the existing integrity checks should be removed from the application program. Otherwise, the application performance will degrade because the same checking is being performed twice (at the application level and at the system level).

The application programmer must also consider the fact that, once the referential integrity constraints are defined to the DBMS, referential integrity enforcement is performed at all times on all interfaces. If you have applications that only need the data to be consistent at specific points in time or applications where the inconsistency is accepted because another program will correct it, DBMS referential constraints may prevent these applications from running smoothly. A programmer must verify that the DBMS-supported referential integrity matches the integrity and business rules currently enforced by their applications.

### 3.7.1 Referential integrity I/O messages

Several new error messages have been defined to handle the errors occurring during referential integrity enforcement. Instead of coding integrity checks into your application programs, coding is now needed to handle the new referential integrity error conditions that can be raised by DB2 UDB for iSeries during referential constraint enforcement.



## Notify messages

There are three new notify messages for referential integrity errors:

- ▶ **CPF502D**: Referential constraint violation member <member name>  
This message is issued when the user or the application tries to insert or update a foreign key, and a matching parent key value does not exist.
- ▶ **CPF502E**: Referential constraints could not be validated for member <member name>  
This message is issued when the system cannot validate a referential constraint because of a record or a file lock.
- ▶ **CPF503A**: Referential constraint violation on member <member name>  
This message is issued when the delete rule is NOACTION or RESTRICT and the user or the application tries to delete or update a parent key having matching foreign key values.

These messages have a severity level of 30, and the default reply is “Cancel”.

## Escape messages

There are two new escape messages for referential integrity errors:

- ▶ **CPF523B**: Referential constraint error processing member <member name>  
This message is issued when the system cannot enforce a referential constraint.
- ▶ **CPF523C**: Referential constraints journal error  
This message is issued when the system cannot enforce a referential constraint because the corresponding parent and dependent files are not journaled, or they are not journaled to the same journal.

Both messages have a severity level of 30 and fall into the range of escape messages that are unrecoverable.

## 3.7.2 Handling referential integrity messages in applications

To handle these messages, new file status codes have been provided for ILE languages. In the original program model (OPM) environment, any message due to errors in referential integrity enforcement maps to the existing I/O error status codes “01299” for RPG/400 and “90” for COBOL/400.

### Referential integrity messages in ILE RPG programs

You can check the new status “01222” if you want to handle the CPF502E message. There is also a corresponding inquiry message RNQ1222 and a corresponding escape message RNX1222. Both of them have severity level 99 and the following text:

```
Unable to allocate a record in file &7 due to referential  
constraint error (R C G D F).
```

Status “01022” handles CPF502D and CPF503A. There is also a corresponding inquiry message RNQ1022 and a corresponding escape message RNX1022. Both of them have severity level 99 and the following text:

```
Referential constraint error on file &7.
```

The existing status code “01299” and the corresponding inquiry message RNQ1299 and escape message RNX1299 are used to handle escape messages CPF523C and CPF523B.

## Referential integrity messages in ILE COBOL programs

Status “9R” handles all the notify messages previously listed for referential integrity exceptions. Both escape messages are handled by the status code “90” set for the exceptions in the CPF5200 range.

## Referential integrity messages in ILE C programs

ILE/C maps these messages to the existing error number values.

## SQLCODE values mapping referential integrity messages

The SQLCODE values are:

- ▶ SQLCODE 530 handles the notify message CPF502D.
- ▶ SQLCODE 531 indicates that you are updating a parent key with matching dependent records.
- ▶ SQLCODE 532 indicates that you are deleting a parent key with matching dependent records.

See Appendix B, “Referential integrity: Error handling example” on page 337, for a coding example about error handling when using referential integrity.

## 3.8 Referential integrity constraint management

This section describes:

- ▶ Constraint states
- ▶ Check pending condition
- ▶ Commands you can use to manage referential integrity constraints
- ▶ Save and restore
- ▶ How to obtain information about referential integrity constraints

### 3.8.1 Constraint states

A referential constraint can be in one of the following states:

- ▶ **DEFINED state:** The constraint definition exists at the file level, but the constraint is not enforced. Defined constraints are purely by definition and not by function. The file members do not have to exist for the constraint to be defined.
  - *Defined/enabled:* A constraint that remains enabled when it is moved to the established state
  - *Defined/disabled:* A constraint that remains disabled when it is moved to the established state
- ▶ **ESTABLISHED state:** A referential constraint is established when the foreign key attributes match those of the parent key and both files contain a member. The constraint has now been formally created in the DBMS. In this state, the constraint can be:
  - *Established/enabled:* DB2 UDB for iSeries enforces referential integrity for this constraint.
  - *Established/disabled:* DB2 UDB for iSeries does not enforce referential integrity for a constraint in this state. However, the access paths associated with the constraint are still maintained.

See *Database Programming*, SC41-5701, for a complete discussion of constraint states.

## 3.8.2 Check pending

A referential constraint is placed in check pending status if the DBMS determines that mismatches may exist between the parent and foreign keys. The check pending status only applies to referential constraints in the *established/enabled* state.

There are several operations that can cause a check pending condition:

- ▶ Adding referential constraints to existing files with invalid data
- ▶ Abnormal system failures
- ▶ Save/restore operations
- ▶ Apply/remove journal changes

When a referential constraint relationship has been marked as check pending, the associated parent and dependent files can be opened, but the system imposes some restrictions on the I/O operations to those files:

- ▶ Only read and insert operations are allowed on the parent file.
- ▶ No I/O operations are allowed on the dependent file.

The system imposes these restrictions to ensure that applications and users are not accessing and changing records that are possibly inconsistent and, therefore, violating referential integrity.

To move a constraint relationship out of check pending, you must use disable (CHGPFCST) to disable the constraint that allows any I/O operations to be performed on the parent and dependent file. You can then correct your parent and foreign key values so that they again meet referential integrity. Once the data corrections are completed, you can enable the constraint that causes DB2 UDB for iSeries to process and verify that every non-null foreign key value is valid. If this verification finds mismatches, the relationship is again marked as check pending and the process repeats itself.

The check pending status of a file can be determined with the Work with Physical File Constraints (WRKPFCST) command (refer to Figure 3-20 on page 57) and the Display Physical File Description (DSPFD) command (refer to Figure 3-23 on page 62).

## 3.8.3 Constraint commands

The commands provided to manage referential integrity constraints are:

- ▶ Change Physical File Constraint (CHGPFCST)
- ▶ Display Check Pending Constraint (DSPCPCST)
- ▶ Work with Physical File Constraints (WRKPFCST)
- ▶ Edit Check Pending Constraint (EDTCPCST)
- ▶ Remove Physical File Constraint (RMVPFCST)

### CHGPFCST command

The Change Physical File Constraint (CHGPFCST) command provides a way to:

- ▶ *Enable* a referential constraint:

Enable causes the system to verify the data integrity of the specified constraint (for example, every non-null foreign key value has a matching parent key). If the verification is successful, the referential constraint is enforced by DB2 UDB for iSeries. Remember that this enable process may not be a short-running operation when the associated files contain a large number of records.

► *Disable* a referential constraint:

Disabling a constraint essentially turns off referential integrity for that constraint relationship. Although the constraint is still defined in the DBMS, the DBMS no longer enforces referential integrity for the disabled constraint relationship. Any I/O operation is allowed on the parent and dependent file, even if that operation violates referential integrity.

As mentioned in the check pending section, the disable option is used with check pending constraints so that users can clean up their parent and foreign key data before having the system re-verify the constraint.

Disabling a constraint can allow faster file I/O operations in performance-critical situations. However, you must consider the trade-off in this situation. While the constraint is disabled, the data can violate referential integrity, and you are unaware of the violation until the constraint is re-enabled. In addition, you must wait for the system to re-verify all of your foreign key values on the re-enable.

To limit your data integrity exposure when a constraint is disabled, first use the Allocate Object (ALCOBJ) command to exclusively lock the files associated with the constraint to be disabled. This allocation prevents other users from changing the file data while the constraint is disabled. Then, use the De-allocate Object (DLCOBJ) command to free the files once the referential constraint has been re-enabled.

Before enabling or disabling a constraint, the system obtains:

- Exclusive allow-read locks on the parent file, member, and access paths
- Exclusive no-read locks on the dependent file, member, and access paths

These locks are released at the end of the CHGPFCST command.

### DSPCPCST command

The Display Check Pending Status (DSPCPCST) command can be used on referential constraints that are in a disabled state to display which records in the dependent file do not have matching parent key values, thereby causing the check pending condition.

The following example shows how the DSPCPCST output can be used to fix a constraint that is currently marked as check pending. In the Order Entry database, we define a referential constraint ORDERHDR\_CNBR (Parent Key and foreign key is the Customer\_Number field in both files) between *existing* CUSTOMER and ORDERHDR files having the contents listed in Table 3-1 and Table 3-2.

Table 3-1 CUSTOMER table

CUSTOMER_NUMBER	CUSTOMER_NAME	...
10509	Benson Mary	
15006	Smith Steven	...
14030	Peterson Robert	...
13007	Robinson Richard	...
21603	White Paul	...

Table 3-2 ORDERHDR table

ORDER_NUMBER	...	CUSTOMER_NUMBER	ORDER_DATE
00010	...	10509	05/08/01
00020	...	10509	05/09/01
02020	...	12312	02/03/01
02021	...	12312	04/13/01
02022	...	12312	04/25/01

The constraint is marked as check pending because ORDERHDR contains records related to Customer 12312, which does not exist in the CUSTOMER file.

In this case, follow these steps:

1. Lock up your referential integrity network with the ALCOBJ command while you are correcting your parent and foreign key data:

```
ALCOBJ OBJ((CUSTOMER *FILE *EXCL *FIRST)
           (ORDERHDR *FILE *EXCL *FIRST))
```

2. If the constraint is not yet disabled, disable the constraint so that the DSPCPCST command can read the dependent file:

```
CHGPFCST FILE(ORDERHDR)
          CST(ORDERHDR_CNBR)
          STATE(*DISABLED)
```

3. Display which records in ORDERHDR have a customer number that does not exist in the CUSTOMER file:

```
DSPCPCST FILE(ORDENTL/ORDERHDR)
          CST(ORDERHDR_CNBR)
```

The output of this command is shown in Figure 3-19.

Display Report			
Width . . .:	142		
Column . .:	1		
Control . . .:			
Line . . .:	1	2	3
ORDER_NUMBER	CUSTOMER_NUMBER	ORDER_DATE	
000001	02020	12312	02/03/2001
000002	02021	12312	04/13/2001 . . .
000003	02022	12312	04/25/2001
***** * * * * E N D O F D A T A * * * * *			

Figure 3-19 DPSCPCST output

4. According to the DSPCPCST output, clean up your foreign and parent keys value. In this case, it appears that Customer 12312 needs to be added to the CUSTOMER file.
5. Once the data is corrected, enable the constraint so that the DBMS can verify that your parent and foreign key data is now in sync:

```
CHGPFCST FILE(ORDERHDR)
          CST(ORDERHDR_CNBR)
          STATE(*ENABLED)
```

6. Now that the constraint has been successfully enabled, release the locks on your referential integrity network with the DLCOBJ command:

```
DLCOBJ OBJ((CUSTOMER *FILE *EXCL *FIRST)
           (ORDERHDR *FILE *EXCL *FIRST))
```

### 3.8.4 Removing a constraint

This section shows how to remove physical file (or table) constraints. Both the native and SQL interfaces can be used to remove file constraints:

- ▶ The native interface provides the Remove Physical File Constraint (RMVPCST) command.
- ▶ The SQL interface allows you to remove an existing constraint from a file through the DROP clause of the ALTER TABLE statement.

The SQL interface supports the removal of one constraint at a time. The following statement removes the `customer_key` constraint from the CUSTOMER file:

```
ALTER TABLE mylib/CUSTOMER
           DROP CONSTRAINT customer_key
```

The following statements remove (respectively) the primary key, the `constraint_name` unique constraint, and the `constraint_name` referential constraint from the CUSTOMER file:

```
ALTER TABLE mylib/CUSTOMER
           DROP PRIMARY KEY
```

```
ALTER TABLE mylib/CUSTOMER
           DROP UNIQUE constraint_name
```

```
ALTER TABLE mylib/CUSTOMER
           DROP FOREIGN KEY constraint_name
```

In contrast, the native interface allows you to remove more than one constraint at a time. In addition, you can sub-select the physical file constraints you want to remove by specifying the option that only referential constraints, marked as check pending, should be removed.

Let's examine the impact of the RMVPCST command according to the different values of its parameters. The following statement removes the `constraint_name` constraint from CUSTOMER file:

```
RMVPCST FILE(mylib/CUSTOMER)
        CST(constraint_name)
        TYPE(constraint_type)
```

If `CST(*CHKPND)` is specified, all the referential constraints in the check pending condition are removed, regardless of the value of the TYPE parameter. The following statement removes all the `constraint_type` constraints from the CUSTOMER file in `mylib`:

```
RMVPCST FILE(mylib/CUSTOMER)
        CST(*ALL)
        TYPE(constraint_type)
```

In this case, the system removes the unique or referential constraints following the sequence in which they have been created:

```
RMVPCST FILE(mylib/CUSTOMER)
        CST(*ALL)
```

The RMVPCST statement removes all the constraints defined over the CUSTOMER file in *mylib*, including the damaged constraints since the TYPE default value is \*ALL.

In this case, the system removes the primary key constraint first, then all of the unique constraints (in their creation sequence), and finally, all of the referential constraints (in their creation sequence).

### WRKPCST command

The Work with Physical File Constraints (WRKPCST) command is similar to the other Control Language *Work* commands. With this command, you can gain access to most of the constraint operations from a single display. The WRKPCST command lets you see one or all the physical file constraints defined over one or more files, depending on the values you set for the WRKPCST parameters. Figure 3-20 displays the sample output from the WRKPCST command.

Work with Physical File Constraints						
Type options, press Enter.						
2=Change 4=Remove 6=Display records in check pending						
Opt	Constraint	File	Library	Type	State	Check Pending
	CUSTOMER_K	> CUSTOMER	ORDENTL	*PRIKEY		
	ORDDTL_KEY	ORDERDTL	ORDENTL	*PRIKEY		
	ORDDTL_HOR	> ORDERDTL	ORDENTL	*REFCST	EST/ENB	NO
	ORDERHDR_K	> ORDERHDR	ORDENTL	*PRIKEY		
	ORDERHDR_C	> ORDERHDR	ORDENTL	*REFCST	EST/ENB	YES
	SALESREP_K	> SALESUS	ORDENTL	*PRIKEY		
	SALESREP_C	> SALESUS	ORDENTL	*REFCST	EST/ENB	NO
	STOCK_KEY	STOCK	ORDENTR	*PRIKEY		
	STOCK_SNBR	STOCK	ORDENTR	*REFCST	EST/ENB	NO
	SUPPLIER_K	> SUPPLIER	ORDENTR	*PRIKEY		
Parameters for options 2, 4, 6 or command						Bottom
===>						
F3=Exit		F4=Prompt		F5=Refresh		F12=Cancel
F16=Repeat position to		F17=Position to		F15=Sort by		
F22=Display constraint name						

Figure 3-20 Work with Physical File Constraints display

On this display, you can:

- ▶ Change the state of constraints (option 2):  
This option invokes the CHGPCST command (see “CHGPCST command” on page 53).
- ▶ Remove a constraint (option 4):  
This option invokes the RMVPCST command (see 3.8.4, “Removing a constraint” on page 56, for more details).
- ▶ Display constraints in check pending status (option 6):  
This option executes the DSPCPCST command (see “DSPCPCST command” on page 54).

The state column lists the status of the referential constraints: defined or established and enabled or disabled. The check pending status column displays which constraints are currently in check pending. Disabled constraints are always shown as being in check pending condition although check pending does not apply to disabled constraints.

## EDTCCPST command

The Edit Check Pending Constraints (EDTCCPST) command allows you to manage the verification of referential constraints that have been marked as check pending. The system displays the constraints marked as check pending and the estimated time it takes the system to verify the constraint once the parent and foreign key data have been corrected.

From our previous example (Figure 3-20), the corresponding EDTCCPST display output is shown in Figure 3-21 with the ORDERHDR\_CNBR constraint that was placed in check pending status.

Edit Check Pending Constraints					SYSTEM03	
					05/14/01 18:39:36	
Type sequence, press Enter.						
Sequence: 1-99, *HLD						
-----Constraints-----						
Seq	Status	Cst	File	Library	Verify Time	Elapsed Time
1	RUN	STOCK >	STOCK	ORDENTR	00:10:00	00:02:40
2	READY	SALES >	SALESCUS	ORDENTL	00:01:48	00:00:00
*HLD	CHKPND	ORDER >	ORDERHDR	ORDENTL	00:00:01	00:00:00

Bottom

F3=Exit   F5=Refresh   F12=Cancel   F13=Repeat all   F15=Sort by  
F16=Repeat position to   F17=Position to   F22=Display constraint name

Figure 3-21 Edit Check Pending Constraints display

From this display, you can set a sequence for the constraints verification. You can also delay the verify process to a later time, specifying \*HLD on the sequence field. DB2 UDB for iSeries starts verifying the constraints right after you specify the sequence. The elapsed time since the beginning of the process is also displayed. During this process, the constraint status is set to RUN. Other constraints waiting for verification are marked with READY.

### Verifying at IPL time

The Edit Check Pending Constraints display (Figure 3-22) is shown during a manual mode IPL if there are constraints in check pending condition.



```

                                Edit Check Pending Constraints
                                SYSTEM03
                                05/24/01 11:14:25
IPL threshold . . . . . 50 0-99

Type sequence, press Enter.
Sequence: 1-99, *HLD

-----Constraints----- Verify   Elapsed
Seq  Status      Cst   File      Library  Time     Time
*HLD  CHKPND      ORDER > ORDERHDR  ORDENTL  00:45:30 00:05:15
*HLD  CHKPND      SALES > SALESCUS  ORDENTL  00:01:43 00:00:36
*HLD  CHKPND      STOCK > STOCK    ORDENTR  00:00:25 00:00:05

                                                                Bottom
F5=Refresh      F13=Repeat all  F15=Sort by   F16=Repeat position to
F17=Position to F22=Display constraint name

```

Figure 3-22 Editing Check Pending Constraint display at IPL time

On this display, you have three alternatives:

- ▶ If you want the system to suspend the IPL and verify a constraint at this time, for that constraint, you have to type a Sequence value less than or equal to the IPL threshold number.
- ▶ If you need the system to verify a constraint after the IPL, you have to use a sequence value greater than the threshold. The IPL then continues, and at the IPL completion, the system automatically starts verifying that constraint.
- ▶ If you want to handle the check pending condition by yourself during the normal activity, hold the constraint verification by leaving the Sequence value set to \*HLD.

If several constraints must be verified at the same time, either during IPL or at the end, you can specify an ordering sequence for them by inserting ordered values into the Sequence field.

### 3.8.5 Save and restore considerations

As mentioned in 3.3.3, “Referential integrity and access paths” on page 25, when a set of database files is saved, all the physical file constraints and associated access paths are saved as well. At restore time, the system attempts to re-establish the constraints for the user.

During the restore operation, the system determines whether the parent and dependent files associated with the referential constraints are at the same data level (in other words, at the same integrity level according to their constraints). If the system determines that the related files and constraints are not at the same level, the constraint relationship is marked as check pending. The system does not spend time verifying every foreign key value during the restore. It only checks the data level of the associated files. This data level verification is much quicker than the DBMS verification of every foreign key value and still preserves referential integrity.

Other DBMS automatically either place the constraints in check pending or verify every foreign key value when you load backup copies of your database files onto the system. DB2 UDB for iSeries gives you the benefit of the doubt when restoring database backups. For example, you always save both your parent and dependent files every Monday night. A system failure on Thursday necessitates that you load the backup tape copies of your dependent and parent file. DB2 UDB for iSeries then quickly verifies that the dependent and parent files being restored are at the same data level (which is true since they were backed up together) and leaves the referential integrity constraint in a valid state. This allows you to move your backup onto the system as quickly as possible while still guaranteeing referential integrity.

Here's an example of DB2 UDB for iSeries protecting your data integrity. You restore a version of the dependent file without restoring the corresponding version of the parent file. This only leads to a check pending condition when some parent records have changed since the save operation took place, which now causes your parent and newly restored dependent files to be at different data levels. For this example, we assume that some parent records have changed since the save operation. The associated referential constraint is marked as check pending since data inconsistencies may exist due to the different data levels detected by the DBMS. You are responsible for cleaning up this check pending situation before users and applications can fully access these files.

To avoid check pending and the associated recovery work, always save your referential integrity network in the same save request. This keeps the associated parent and dependent files at the same level so that you can restore the network with one request.

When your referential integrity network is split across different libraries, you cannot save and restore the network with a single request. In this case, you need to prevent other jobs from changing your file data levels during your multiple request save or restore operation by using the Allocate Object (ALCOBJ) command to lock up your referential integrity network.

Here's an example of the steps to follow in this situation:

- ▶ When saving your referential integrity network:
  - a. Allocate the files you have to save with the **ALCOBJ** command, and set Lock State to **\*EXCL**.
  - b. Save your network.
  - c. Release the locks on the files by using the De-allocate Object (**DLCOBJ**) command.
- ▶ When restoring your referential integrity network:
  - a. Allocate the libraries your files are restored into by using the **ALCOBJ** command and setting Lock State to **\*EXCLRD**.
  - b. Restore your files in any sequence.
  - c. Release the locks previously established on the libraries using the **DLCOBJ** command.

When a dependent file is restored and the parent file is still missing, the constraint is left in a defined/enabled state. As soon as the parent file is restored, the constraint is established and the data levels immediately are verified. Therefore, the parent and dependent files can be restored in any sequence while still avoiding check pending.

When you restore files belonging to a referential integrity network, the system can determine whether the files are at different data levels for every single constraint. Restoring files at different data levels may result in a mix of check pending and non-check pending constraints. Only the constraints potentially affected by the database changes that caused the data level mismatch are put into check pending.

If you restore a database file over an existing one, the existing constraints are preserved. For example, if you remove some constraints from the file currently on the system, the additional constraints saved on the media are not restored.

### 3.8.6 Restore and journal apply: An example

Consider the example described in 3.5.3, “A CASCADE example” on page 39. You may want to save this referential integrity network. Since all the files are in the same library, issue a single save request:

```
SAVOBJ OBJ(ORDERH DETAIL FEATURE)
      LIB(TEST) DEV(device)
      OBJTYPE(*FILE)
```

Consider the example where a system failure has caused you to lose the DETAIL file, and you now need to recover your referential integrity network. Follow these steps:

1. Allocate the involved files to avoid changes by other jobs. Use the **ALCOBJ** command with Lock Type set to **\*EXCL** to prevent other users from reading inconsistent data.
2. Restore all of the referential integrity network:

```
RSTOBJ OBJ(ORDERH DETAIL FEATURE)
      SAVLIB(TEST) DEV(device) OBJTYPE(*FILE)
```

3. Apply journal changes to all of the involved files:

```
APYJRNCHG JRN(TEST/QSQJRN)
      FILE((TEST/ORDERH) (TEST/DETAIL) (TEST/FEATURE))
      CMTBDY(*YES)
```

4. De-allocate the ORDERH, DETAIL, and FEATURE files.

For details on journaling, commitment control, and applying journal entries, see *Backup and Recovery Guide - Advanced*, SC41-3305.

### 3.8.7 Displaying constraint information

You can display or output the constraints and their related attributes and states for a file in the following ways:

- ▶ Run the Display Physical File Description (DSPFD) command
- ▶ Run the Display Database Relations (DSPDBR) command
- ▶ Query the system catalog tables

#### DSPFD and DSPDBR commands

The DSPFD command also provides a complete description of all the constraints defined for a file. You can select this specific information by specifying:

```
DSPFD FILE(ORDENTL/ORDERHDR) TYPE(*CST)
```

This command shows which constraints are defined for the ORDERHDR file and their description as shown in Figure 3-23.

```

                                Display Spooled File
File . . . . . : QPDSPFD                                Page/Line  1/1
Control . . . . .                               Columns  1 - 78
Find . . . . .
*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...
5/16/01                                Display File Description
DSPFD Command Input
File . . . . . : FILE                                ORDERHDR
Library . . . . . :                                ORDENTL
Type of information . . . . . : TYPE
File attributes . . . . . : FILEATR                *ALL
System . . . . . : SYSTEM                        *LCL
File Description Header
File . . . . . : FILE                                ORDERHDR
Library . . . . . :                                ORDENTL
Type of file . . . . . :                               Physical
File type . . . . . : FILETYPE                    *DATA
Auxiliary storage pool ID . . . . . :                01
Constraint Description
  Primary Key Constraint
    Constraint . . . . . : CST                        ORDERHKEY
    Type . . . . . : TYPE                            *PRIMARY
    Key . . . . . : KEY                              ORHNBR
    Number of fields in key . . . . . :                1
    Key length . . . . . :                            5
  Referential Constraint
    Constraint . . . . . : CST                        ORDERHRCNBR
    Type . . . . . : TYPE                            *REFCST
    Check pending . . . . . :                          NO
    Constraint state . . . . . : STATE                ESTABLISHED
    *ENABLED
  Parent File Description
    File . . . . . : PRNFILE                        CUSTOMER
    Library . . . . . : LIB                          ORDENTL
    Parent key . . . . . : PRNKEY                    CUSNBR
    Foreign key . . . . . : FRNKEY                    CUSNBR
    Delete rule . . . . . : DLTRULE                  *RESTRICT
    Update rule . . . . . : UPDRULE                  *RESTRICT

```

Figure 3-23 Physical file constraints from DSPFD

In the Constraint Description section (highlighted in bold) in Figure 3-23, all of the parameter values set through the ADDPFCST command or ALTER TABLE/CREATE TABLE statements for each constraint are listed.

The DSPFD command issued for a given file shows a referential constraint definition only for the parent file. To determine which referential constraints refer to this file as a parent file, you must use the DSPDBR command. This command lists these constraints in the Dependent Files section, where some new information has been added to differentiate among referential constraints, logical files, SQL indexes, or SQL views.

Figure 3-24 shows this information for the ORDERHDR file.

```

                                Display Spooled File
File . . . . . : QPDSPDBR                                Page/Line  1/1
Control . . . . .                               Columns  1 - 78
Find . . . . .
*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...
  5/16/01                                Display Data Base Relations
DSPDBR Command Input
  File . . . . . : FILE                                ORDERHDR
  Library . . . . . :                                ORDENTL
  Member . . . . . : MBR                                *NONE
  Record format . . . . . : RCDFMT                    *NONE
  Output . . . . . : OUTPUT                            *
Specifications
  Type of file . . . . . :                               Physical
  File . . . . . :                                ORDERHDR
  Library . . . . . :                                ORDENTL
  Member . . . . . :                                *NONE
  Record format . . . . . :                            *NONE
  Number of dependent files . . . . . :
Files Dependent On Specified File
  Dependent File      Library      Dependency  JREF      Constraint
  SALE                ORDENTL   Data
  TOTALSALE           ORDENTL   Data
  YEARSALE            ORDENTL   Data
  ORDERDTL           ORDENTL   Constraint          ORDERHDRNUM

                                                                 Bottom
F3=Exit  F12=Cancel  F19=Left  F20=Right  F24=More keys

```

Figure 3-24 Referential constraints from DSPDBR on the parent file

As you can see by comparing the Constraint Description line (in bold) from Figure 3-23 and the last line in bold in Figure 3-24, the DSPFD and DSPDBR commands provide complete information about the constraints involving the physical files in question.

### Catalog inquiry

DB2 UDB for iSeries provides a system-wide catalog. The SQL catalog is a set of views in the QSYS2 library built over the cross-reference files where DB2 UDB for iSeries maintains all information related to the structure and the contents of all database files. The catalog also keeps information related to the physical file constraints. You can retrieve any information you need about the constraints defined over your database files using the system views provided in the QSYS2 library:

- ▶ **SYSCST**: General information about constraints. The underlying catalog tables are QADBFCST and QADBXRREF.
- ▶ **SYSCSTCOL**: Information about the columns referenced in a constraint. This is a view defined over the QADBCCST and QADBIFLD catalog tables.
- ▶ **SYSCSTDEP**: Information about the constraint dependencies on tables. The catalog tables involved are QADBFCST and QADBXRREF.
- ▶ **SYSKEYCST**: Information about the primary, unique, and foreign keys. The underlying catalog tables are QADBCCST, QADBIFLD, and QADBFCST.
- ▶ **SYSREFCST**: Information about referential constraints from the cross-reference file table QADBFCST.

Consider this example:

```
SELECT *
FROM SYSCST
WHERE TABLE_NAME = 'ORDERDTL' AND TABLE_SCHEMA = 'ORDENTL'
```

This query returns information at the constraint level about the constraints defined over the ORDERDTL file in the ORDENTL library. The most significant details are shown in Figure 3-25.

	CONSTRAINT_SCHEMA	CONSTRAINT_NAME	CONSTRAINT_TYPE
1	ORDENTL	ORDERDTL_KEYS	PRIMARY KEY
2	ORDENTL	ORDERHDRNUM	FOREIGN KEY

Figure 3-25 Constraint information

To see which fields constitute the key of the constraint, you have to query SYSCSTCOL. Use the previous example:

```
SELECT *
FROM SYSCSTCOL
WHERE TABLE_NAME = 'ORDERDTL' AND TABLE_SCHEMA = 'ORDENTL'
ORDER BY CONSTRAINT_NAME
```

This query returns the names of the fields forming the various constraint keys of ORDERDTL file. See Figure 3-26.

	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	CONSTRAINT_SCHEMA	CONSTRAINT_NAME	SYSTEM_COLUMN_NAME
1	ORDENTL	ORDERDTL	ORDER_NUMBER	ORDENTL	ORDERDTL_KEYS	ORHNBR
2	ORDENTL	ORDERDTL	PRODUCT_NUMBER	ORDENTL	ORDERDTL_KEYS	PRDNBR
3	ORDENTL	ORDERDTL	ORDER_NUMBER	ORDENTL	ORDERHDRNUM	ORHNBR

Figure 3-26 Constraint column information

The catalog table, SYSKEYCST, keeps more detailed information regarding key fields in a physical file constraint, such as the ordinal position of the field, in the key, and this position in the table layout:

```
SELECT *FROM SYSKEYCST
WHERE CONSTRAINT_SCHEMA = 'ORDENTL'
AND CONSTRAINT_NAME = 'ORDERDTL_KEYS'
AND TABLE_SCHEMA = 'ORDENTL'
AND TABLE_NAME = 'ORDERDTL'
```

This statement returns the information shown in Figure 3-27.

	CONSTRAINT_NAME	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	COLUMN_POSITION
1	ORDERDTL_KEYS	ORDENTL	ORDERDTL	ORDER_NUMBER	1	1
2	ORDERDTL_KEYS	ORDENTL	ORDERDTL	PRODUCT_NUMBER	2	2

Figure 3-27 Detailed constraint key information

For a referential constraint, detailed information can be selected from SYSREFCST (in the ORDERDTL case, for example):

```

SELECT * FROM SYSREFCST
WHERE CONSTRAINT_NAME = 'ORDERHDRNUM'
AND CONSTRAINT_SCHEMA = 'ORDENTL'

```

This statement returns the information shown in Figure 3-28.

CONSTRAINT_SCHEMA	CONSTRAINT_NAME	UNIQUE_CONSTRAINT_NAME	UPDATE_RULE	DELETE_RULE
1	ORDENTL	ORDERHDRNUM	RESTRICT	RESTRICT

Figure 3-28 Referential constraint information

To determine the complete definition of a referential constraint through catalog views, you need to perform a join:

- ▶ From SYSREFCST, you can retrieve the name of the unique or primary key constraint identifying the parent key.
- ▶ By using the name of the constraint, SYSCST provides the name and library of the corresponding parent file and the type of the constraint itself (primary key or unique constraint).
- ▶ SYSCSTCOL gives the parent key (unique or primary key) fields.

These actions can be expressed through the following query:

```

SELECT C.UNIQUE_CONSTRAINT_SCHEMA
      , C.UNIQUE_CONSTRAINT_NAME
      , A.CONSTRAINT_TYPE
      , A.TABLE_SCHEMA
      , A.TABLE_NAME
      , C.UPDATE_RULE
      , C.DELETE_RULE
      , B.COLUMN_NAME
FROM   SYSCST A, SYSCSTCOL B, SYSREFCST C
WHERE  C.CONSTRAINT_SCHEMA = 'ORDENTL'
      AND C.CONSTRAINT_NAME = 'ORDERHDRNUM'
      AND B.CONSTRAINT_SCHEMA = C.UNIQUE_CONSTRAINT_SCHEMA
      AND B.CONSTRAINT_NAME = C.UNIQUE_CONSTRAINT_NAME
      AND A.CONSTRAINT_SCHEMA = C.UNIQUE_CONSTRAINT_SCHEMA
      AND A.CONSTRAINT_NAME = C.UNIQUE_CONSTRAINT_NAME
GROUP BY C.UNIQUE_CONSTRAINT_SCHEMA
        , C.UNIQUE_CONSTRAINT_NAME
        , A.CONSTRAINT_TYPE
        , A.TABLE_SCHEMA
        , A.TABLE_NAME
        , C.UPDATE_RULE
        , C.DELETE_RULE
        , B.COLUMN_NAME

```

The output of the previous query consists of as many rows as the parent key fields. In the example of the ORDDTL\_HORD constraint (see 3.4.3, “Another example: Order Entry scenario” on page 32), the query returns the output shown in Figure 3-29.

CONSTRAINT_SCHEMA	CONSTRAINT_NAME	UNIQUE_CONSTRAINT_NAME	UPDATE_RULE	DELETE_RULE
1	ORDENTL	ORDERHDRNUM	RESTRICT	RESTRICT

Figure 3-29 Parent key information







# Check constraint

This chapter explains:

- ▶ DB2 UDB for iSeries check constraints
- ▶ Defining a check constraint
- ▶ General considerations
- ▶ Application impacts of check constraint
- ▶ Check constraint management
- ▶ Tips and techniques

## 4.1 Introduction

One of the main contributions of the SQL-92 standard is the specification of a rich collection of integrity constraints. The constraints in SQL-92 can be classified into three categories:

- ▶ Domain or table constraints
- ▶ Referential integrity constraints
- ▶ General assertions

Each of these constraints are explained in the following sections.

### 4.1.1 Domain or table constraints

Table or domain constraints in SQL-92 are used to enforce restrictions on the data allowed in particular columns of particular tables. Any column in a table may be declared as NOT NULL. This indicates that null values are not permissible for that column. In addition, a set of one or more columns may be declared as UNIQUE. This indicates that two rows may not have the same values for certain columns, which are those that form the key for the table. Each table also can have, at most, one designated PRIMARY KEY consisting of a set of one or more columns. Primary keys must be both unique and not null.

**Note:** New function was added in V4R2M0 to allow a primary key constraint to be defined where one or more columns in the key allow NULL values. When this condition is detected, a check constraint is implicitly added to the file to ensure that the column will not contain NULL values. This means that this check constraint will prevent any NULL values from being inserted into columns defined for the primary key.

The permissible values of a column may also be restricted by means of a CHECK constraint. A CHECK clause specifies a condition that involves the column whose values are restricted. Semantically, a CHECK constraint is valid if the condition evaluates to *true* or *unknown* for every row in the table.

### 4.1.2 Referential integrity constraints

A referential integrity constraint involves two tables called the *parent table* and the *dependent table*. Intuitively, every row in the referencing table must be a “child” of some row in the referenced table. Referential integrity disallows “orphans” that are created by insertions (of child rows), updates (of child or parent rows), or deletions (of parent rows). Referential integrity can be violated by insertions or updates to the referencing table or by updates or deletions to the referenced table.

### 4.1.3 Assertions

Assertions in SQL-92 constraints provide the ability for expressing general constraints that may involve multiple tables. As in CHECK constraints, the condition that is evaluated can be an arbitrary SQL predicate. The assertion is satisfied if the condition evaluates to *true* or *unknown*.

This chapter describes how DB2 UDB for iSeries supports the CHECK constraint that is part of the table constraints of SQL-92.

## 4.2 DB2 UDB for iSeries check constraints

Check constraints in DB2 UDB for iSeries let you ensure that users authorized to change a column's value use only values that are valid for that column. It ensures that the value being entered in a column of a table belongs to the set of valid values defined for that field. For example, you may specify that the "legal" values for an employee evaluation field defined as an integer might be 2, 3, 4, or 5. Without the check constraint, users can enter any integer value into such a field. To ensure that the actual value entered is 2, 3, 4, or 5, you must use a trigger or code the rule in your application program.

A check constraint increases the data integrity because the constraints are validated against every interface (RPG, Data File Utility, ODBC client programs, Interactive SQL, etc.) that updates or inserts database records. The operating system enforces the rules, not the application program. For this reason, there is no way to bypass any control, and the integrity is assured. The programmer no longer has to add this verification code to every application program that updates or inserts a database record.

A check constraint is associated with a file and contains check conditions that are enforced against every row in the file. Whenever a row is inserted or updated, the database manager evaluates the check condition against the new or changed row to guarantee that all new field values are valid. If invalid values are found, the database manager rejects the insert or update operation.

Here are some examples:

- ▶ **Range checking:** The field value must be between 1 and 50
- ▶ **Domain or value checking:** The field can be one of the following values: 1, 3, 5, 7, or 9
- ▶ **Field comparisons:** *total\_sales < credit\_limit*

Remember that the check constraint is valid if the condition evaluates to *true* or *unknown*.

Some of the current alternatives to the check constraint are to:

- ▶ Code the constraints in the application programs. This may give more flexibility in coding the business rules, but the rules are not enforced in all of the iSeries interfaces (for example, DFU or ODBC client programs).
- ▶ Use the DDS keywords (COMP, RANGE, VALUES) in the display and logical files. The problem with this approach is that the rules are only enforced in green-screen applications.
- ▶ Use before triggers. In this case, the rule is enforced on all interfaces, but it is not a part of a database table. It is not a declarative approach.

There are some obvious advantages for using the CHECK constraint option in the iSeries server:

- ▶ There is much less coding to do if the business rules are defined only once in the database.
- ▶ The administration is much easier because the business rules become part of the database.
- ▶ The data integrity of the database is improved because the rules are enforced on all interfaces.
- ▶ Since the database manager is performing the validation, the enforcement is more efficient than the application level enforcement.

## 4.3 Defining a check constraint

This section discusses the interfaces and commands that you can use to add a check constraint. We refer to the native interface and the SQL interface. Let's start with the native CL command `ADDPFCST`. In the following example, we define a check constraint in the `CUSTOMER` file, where the `customer_total` (`CUSTOT`) cannot be greater than the `customer_credit_limit` (`CUSCRD`).

Enter the `ADDPFCST` command, and press F4. The display shown in Figure 4-1 appears.

```
                                Add PF Constraint (ADDPFCST)

Type choices, press Enter.

File . . . . . > CUSTOMER      Name
Library . . . . . >  ORDAPPLIB  Name, *LIBL, *CURLIB
Constraint type . . . . . > *CHKCST  *REFCST, *UNQCST, *PRIK
Constraint name . . . . .      CUSCRD_LIMIT_CUSTOT

F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

Figure 4-1 Prompt for the `ADDPFCST` command

Note that the type of constraint is `*CHKCST`. The name of the constraint must be unique in the library where it is being created. The display shown in Figure 4-2 prompts you for the check condition.

```
                                Add PF Constraint (ADDPFCST)

Type choices, press Enter.

Check constraint . . . . . > 'CUSTOT <= CUSCRD'

F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

Figure 4-2 Check condition for a check constraint

The condition clause of a check constraint can be up to 2000 bytes long.

**Important:** The condition clause of a check constraint is a restricted form of the search-condition of the WHERE and HAVING clauses of the SQL statements. You do not need the DB2 Query Manager and SQL Development Kit for iSeries product to define the condition through the native interface.

When you add a constraint to an existing file, the existing records must not violate the constraint condition. If the system finds records violating the constraint, a diagnostic message is issued for the first 25 rows that failed the check, and the constraint is set to the check pending condition. The display in Figure 4-3 shows the diagnostic message issued by the system.

```
Additional Message Information
Message ID . . . . . : CPD32D3      Severity . . . . . : 20
Message type . . . . . : Diagnostic
Date sent . . . . . : 10/16/01     Time sent . . . . . : 11:20

Message . . . . . : Field values are not valid for check constraint.
Cause . . . . . : Check constraint CUSCRD_LIMIT_CUSTOT for file
CUSTOMER in library ORDAPPLIB is in check pending.
    The constraint is in check pending because record 1 in the file has a
    field value that conflicts with the check constraint expression.
    If the record number for the file is 0, then the record either cannot
    be identified or does not apply to the check pending status.
Recovery . . . . . :
    Use the CHGPFPCST command for the file to disable the constraint.
    Use the DSPPCPCST command for the file to display the records that are
    causing the constraint to be in check pending.
    Update the file to make sure each field value does not conflict with
    the check constraint expression.

Press Enter to continue.

F3=Exit  F6=Print  F9=Display message details
F10=Display messages in job log  F12=Cancel  F21=Select assistance level
```

Figure 4-3 Detailed message for CPD32D3

You can use the Relative Record Number (RRN) scalar function to identify the record that is violating the constraint. This is accomplished by typing the following command in an Interactive SQL session:

```
SELECT rrn(customer), cusnbr FROM customer
```

Figure 4-4 shows the results of this query. In this case, the record with the customer number equal to 100 violates the constraint since its relative record number happens to be 1.

```

                                Display Data
                                Data width . . . . . :
Position to line . . . . .      Shift to column . . . . .
....+....1....+....2....
RRN ( CUSTOMER )  CUSNBR
                   1  00100
                   2  00001
                   3  00003
                   5  00009
                   6  00990
                   7  00008
                   8  00500
                   9  00007
                  11  55555
                  12  00400
                  13  00201
                  14  00101
                  15  00102
                  16  00103
                  17  00045
More
F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split

```

Figure 4-4 Query result

Now let's see how to create a check constraint using the SQL interface. The SQL interface provides two ways to create a check constraint:

- ▶ CREATE TABLE statement, which has the constraint clause
- ▶ ALTER TABLE statement, which allows a table constraint to be added to an existing table with the ADD constraint option

At creation time, you can define the check constraint with the SQL CREATE TABLE statement:

```

CREATE TABLE ORDAPPLIB/CUSTOMER
(CUSTOMER_NUMBER FOR COLUMN CUSNBR CHAR(5) NOT NULL WITH DEFAULT,
CUSTOMER_NAME FOR COLUMN CUSNAM CHAR (20) NOT NULL WITH DEFAULT,
.....
.....
.....
CONSTRAINT CUSCRD_LIMIT_CUSTOT
CHECK (CUSTOT <= CUSCRD ))

```

The CREATE TABLE statement also allows you to define a check constraint at the column level. The restriction is that a column-level constraint cannot reference any other column. Here is an example:

```

CREATE TABLE ORDAPPLIB/EMPLOYEE (
EmpID# CHAR(2),
EmpName CHAR(30),
Salary INTEGER CONSTRAINT salarychk
CHECK(Salary > 0 AND Salary < 10000),
Bonus INTEGER CHECK(Bonus >=0),
CONSTRAINT BonusSalaryChk
CHECK (bonus<= salary))

```

There is an advantage that SQL CREATE TABLE has over CRTPF when you define constraints. CREATE TABLE allows both the DB file object and associated constraints to be created on a single command. CRTPF is always a two-step process:

1. Use **CRTPF** to create the DB object.
2. Use **ADDPFCST** to create your constraints.

You can also add check constraints to existing files. This is illustrated in the following example:

```
ALTER TABLE ORDAPPLIB/CUSTOMER
      ADD CONSTRAINT CUSCRD_LIMIT_CUSTOT
      CHECK(CUSTOT <= CUSCRD)
```

When you are adding a constraint to an existing file, the existing records must not violate the constraint. If the system finds records violating the constraint, an error is issued, similar to the one shown in Figure 4-5, and the constraint is *not* created.

```
Additional Message Information

Message ID . . . . . :  SQL0544      Severity . . . . . :  30
Message type . . . . . :  Diagnostic

Message . . . :  CHECK constraint CUSCRD_LIMIT_CUSTOT cannot be added
Cause . . . :  Existing data in the table violates the CHECK constraint
              rule in constraint CUSCRD_LIMIT_CUSTOT. The constraint cannot be added
Recovery . . :  Change the data in the table so that it follows the
                constraint specified in CUSCRD_LIMIT_CUSTOT. Try the request again.

Press Enter to continue.

F3=Exit  F6=Print  F9=Display message details
F10=Display messages in job log  F12=Cancel  F21=Select assistance level
```

Figure 4-5 Additional message for SQL0544

In this case, you must correct the records that are violating the constraint before you try to create it again.

**Important:** The behavior of the **ADDPFCST** command is different than the **ALTER TABLE** SQL statement when they encounter violating records during the creation of the check constraint. In the first case, the constraint is added, while in the second case, it is not added. This is also true for the **CREATE TABLE** statement. The SQL interface complies with the SQL-92 standard, while the native interface follows the traditional OS/400 approach.

After the constraint is successfully created, you can see its definition by using the **DSPFD ORDAPPLIB/CUSTOMER** command. Press the Page Down key to see the display shown in Figure 4-6.

```

                                Display Spooled File
File . . . . . : QPDSPFD                               Page/Line  2/21
Control . . . . .                               Columns   1 - 78
Find . . . . .
*...+...1...+...2...+...3...+...4...+...5...+...6...+...7..
Constraint Description
  Primary Key Constraint
    Constraint . . . . . : CST          QSYS_CUSTOMER_
    Type . . . . . : TYPE          *PRIMARY
    Key . . . . . : KEY          CUSNBR
    Number of fields in key . . . . . :          1
    Key length . . . . . :          5
  Check Constraint
    Constraint . . . . . : CST          CUSTOT_LIMITED
    Type . . . . . : TYPE          *CHKCST
    Check pending . . . . . :          NO
    Constraint state . . . . . : STATE  ESTABLISHED
    Check constraint expression . . . . . : CHKCST  CUSTOT <= CUSCRD
F3=Exit  F12=Cancel  F19=Left  F20=Right  F24=More keys

```

Figure 4-6 Spooled file for the DSPFD command

You can also see constraints associated with the file by entering the **WRKPFCS** command. The display shown in Figure 4-7 appears.

```

                                Work with Physical File Constraints
Type options, press Enter.
  2=Change  4=Remove  6=Display records in check pending

Opt  Constraint      File      Library   Type      State
----  -
QSYS_CUSTO >  CUSTOMER  ORDAPPLIB *PRIKEY
CUSTOT_LIM >  CUSTOMER  ORDAPPLIB *CHKCST  EST/ENB

Parameters for options 2, 4, 6 or command
===>
F3=Exit  F4=Prompt  F5=Refresh  F9=Retrieve  F12=Cancel  F15=Sort
F16=Repeat position to  F17=Position to  F22=Display constraint name

```

Figure 4-7 Result of the WRKPFCS command

Let's look at some examples:

```

ALTER TABLE ORDAPPLIB/STOCK
ADD CONSTRAINT PRODUCT_PRICE_MIN
CHECK(PRODUCT_PRICE > 0 AND PRODUCT_AVAILABLE_QTY >= 0)

```



```

ALTER TABLE ORDAPPLIB/CUSTOMER           2
  ADD CONSTRAINT CUSTOMER_TYPE
  CHECK(CUSTYP IN ('01', '02', '03', '04', '05', '08', '10'))

ALTER TABLE ORDAPPLIB/EMPLOYEE           3
  ADD CONSTRAINT SALARY_RANGE
  CHECK(EMPSAL BETWEEN 1000 AND 300000)

ALTER TABLE ORDAPPLIB/EMPLOYEE_TRANSAC  4
  ADD CONSTRAINT HOURS_LABORED
  CHECK(ORDINARY_HOURS + EXTRA_HOURS < 168)

```

#### Explanation:

- 1 This check constraint in the STOCK file checks that every price of a product has a price greater than 0 and, at the same time, that the quantity available of a product is greater than or equal to 0.
- 2 This check constraint in the CUSTOMER file checks that each customer is associated with one of the enumerated types.
- 3 This check constraint in the EMPLOYEE file checks that the salary of an employee is in the range of \$1,000 and \$300,000.
- 4 This check constraint in the EMPLOYEE\_TRANSAC file checks that an employee cannot work more than 168 hours in a week. Note the calculations involving two fields of the same row.

## 4.4 General considerations

This section highlights some considerations that you must take into account when you define CHECK constraints. Let's start with the condition clause of the check constraint.

The condition clause of a check constraint can contain any expression or functions allowed on an SQL WHERE clause with the following exceptions:

- ▶ You cannot reference columns of a different table.
- ▶ You cannot reference rows of the same table, which means you cannot use the following column functions:
  - SUM
  - AVERAGE
  - MIN
  - MAX
  - COUNT
- ▶ Subqueries are not allowed.
- ▶ Host variables are not allowed.
- ▶ Parameter markers are not allowed.
- ▶ The following special registers cannot be used:
  - CURRENT TIMEZONE
  - CURRENT SERVER
  - USER

The condition clause of a check constraint can reference more than one column of the same record of the file.

DB2 UDB for iSeries does not prevent conflicting constraints from being defined. Suppose you just created the CUSTOMER file and then you define the following two CHECK constraints before you enter the first record:

```
ALTER TABLE ORDAPPLIB/CUSTOMER
  ADD CONSTRAINT IMPAIR_TYPE
  CHECK(CUSTYP IN ('01', '03', '05', '07','09'))

ALTER TABLE ORDAPPLIB/CUSTOMER
  ADD CONSTRAINT PAIR_TYPE
  CHECK(CUSTYP IN ('02', '04', '06', '08','10'))
```

In the preceding example, the two constraints that are defined prevent the insertion of any record to the CUSTOMER file. If one check condition is valid, the other one is not valid.

Let's try to insert the following record into the CUSTOMER file:

```
INSERT INTO ORDAPPLIB/CUSTOMER
  (CUSNBR, CUSTYP) VALUES('00001', '01')
```

The message shown in Figure 4-8 is displayed.

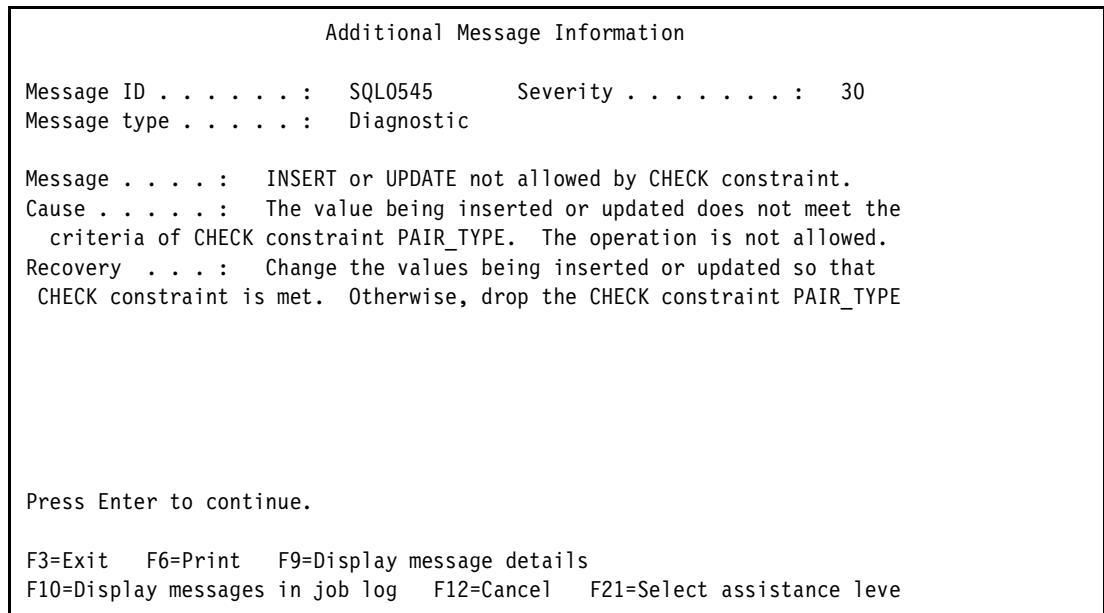


Figure 4-8 Additional message for SQL0545

If we change the CUSTYP value to “02”, the other constraint is violated.

**Important:** It is the developer's responsibility to ensure that check constraints are not mutually exclusive.

Other considerations of which you should be aware are:

- ▶ The constraint name has to be unique across all constraint types that exist in the file's library.
- ▶ A table or file has a limit of 300 combined constraints (referential constraints, primary, unique, and check constraints).
- ▶ Only single member files are supported.

- ▶ When you add a check constraint, DB2 UDB for iSeries makes an exclusive lock on the table for the verification of the condition clause.

**Note:** The verification for adding a check constraint to large files can take some time. In our test environment, it took about 10 minutes to verify a 5 million row table on a 50S machine.

## 4.5 Check constraint integration into applications

Before check constraint support was implemented in DB2 UDB for iSeries, check constraint validations had to be performed by the application program. Now you can let DB2 UDB for iSeries ensure your data integrity both through the referential integrity and check constraint definitions.

Using the check constraint definitions may improve your application's performance. The domain checks are much more efficient and quicker when performed at the operating system level rather than by an application code.

However, once a programmer has defined check constraints and referential constraints to the DBMS, the existing integrity checks should be removed from the application program. Otherwise, the application performance will degrade since the same checking is being performed twice (at the application level and at the system level).

### 4.5.1 Check constraint I/O messages

The enforcement of the check constraint definitions is done when:

- ▶ An insert is being done to the table with check constraints.
- ▶ An update is being done to the table with check constraints.
- ▶ A delete is being done on a parent table that has a referential integrity constraint defined with their dependent tables and a SET DEFAULT or SET NULL is specified.

A new message has been defined to handle the error occurring during a check constraint enforcement. Instead of coding domain checks in the application programs, coding is needed for handling check constraint error conditions. The text of the message is shown in Figure 4-9.

```

                                Display Formatted Message Text
                                System:  SY
Message ID . . . . . : CPF502F
Message file . . . . . : QCPFMSG
  Library . . . . . :   QSYS

Message . . . . . : Check constraint violation on member CUSTOMER.
Cause . . . . . : The operation being performed on member CUSTOMER
  file CUSTOMER in library ORDAPPLIB failed. Constraint PAIR_TYPE
  prevents record number 2 from being inserted or updated because the
  field value conflicts with the check constraint.
If the record number is zero, then the error occurred on a
insert operation. The reason code is 01. The reason codes and their
meanings are as follows:
  01 - Violation due to insert or update operation.
  02 - Violation caused by a referential constraint.
Recovery . . . . . : Either specify a different file, change the file, or
  change the program. Then try your request again.
Possible choices for replying to message . . . . . :
  C -- The request is canceled.

Press Enter to continue.

F3=Exit  F11=Display unformatted message text  F12=Cancel

```

Figure 4-9 Detailed message for CPF502F

## 4.5.2 Check constraint application messages

To handle these messages in SQL procedures or in embedded SQL statements, new SQL codes have been provided for this purpose. In Figure 4-10, you can see the new messages added to the SQL run time.

```

                                Display Message Descriptions
                                System:  SYS
Message file:  QSQLMSG      Library:  QSYS

Position to . . . . .      Message ID

Type options, press Enter.
  5=Display details  6=Print
Op Message ID Severity Message Text
  SQL0543      30  Constraint &1 conflicts with SET NULL or SET Default
  SQL0544      30  CHECK constraint &1 cannot be added.
  5 SQL0545      30  INSERT or UPDATE not allowed by CHECK constraint.
  SQL0546      30  CHECK condition of constraint &1 not valid.
  SQL0551      30  Not authorized to object &1 in &2 type *&3.
  SQL0552      30  Not authorized to &1.
  SQL0557      30  Privilege not valid for table or view &1 in &2
  SQL0569      10  Not all requested privileges revoked from object
  SQL0570      10  Not all requested privileges to object &1 in &2
  SQL0573      30  Table &1 in &2 does not have a matching parent

F3=Exit  F5=Refresh  F12=Cancel

```

Figure 4-10 SQL messages for the check constraint

The SQL message, SQL0545, is the most important for the application programmers. The detailed description is shown in Figure 4-11.

```
Display Formatted Message Text
System:  SYSTEM1
Message ID . . . . . :  SQL0545
Message file . . . . . :  QSQLMSG
Library . . . . . :  QSYS

Message . . . . . :  INSERT or UPDATE not allowed by CHECK constraint.
Cause . . . . . :  The value being inserted or updated does not meet the
criteria of CHECK constraint &1. The operation is not allowed.
Recovery . . . . . :  Change the values being inserted or updated so that
the CHECK constraint is met. Otherwise, drop the CHECK constraint &1.

Press Enter to continue.

F3=Exit  F11=Display unformatted message text  F12=Cancel
```

Figure 4-11 Detailed message for SQL0545

SQL0545 has an SQLSTATE of “23513”, which is useful for a condition or handler declarations in SQL procedures. Refer to *Stored Procedures and Triggers on DB2 Universal Database for iSeries*, SG24-6503, for a detailed discussion on SQL procedures.

Now let's see how the errors are reported in ILE programs:

- ▶ In ILE RPG, you can check the new status 1022, which handles the CPF502F message.
- ▶ In ILE COBOL, the file status “9W” handles the check constraint violation.
- ▶ ILE C maps these messages to the existing error codes.
- ▶ OPM programs map these messages to the existing generic error codes.

## 4.6 Check constraint management

This section discusses the management considerations of the check constraints. The managing part of a check constraint is the same as a referential integrity constraint. The commands to manage the constraints are exactly the same as for referential integrity. These commands are:

- ▶ Change Physical File Constraint (CHGPPFCST)
- ▶ Display Check Pending Constraint (DSPCPCST)
- ▶ Work with Physical File Constraint (WRKPPFCST)
- ▶ Edit Check Pending Constraint (EDTCPCST)
- ▶ Remove Physical File Constraint (RMVPPFCST)

For a complete description of these commands, refer to 3.8, “Referential integrity constraint management” on page 52.

## 4.6.1 Check constraint states

A check constraint is the same as a referential constraint in terms of its possible states. The four states of a check constraint are:

- ▶ Defined and enabled
- ▶ Defined and disabled
- ▶ Established and enabled
- ▶ Established and disabled

Each term is explained in the following list:

- Defined** The constraint definition has been added to the file, but not all the pieces of the file are there for enforcement. For example, the file's member does not exist.
- Established** The constraint definition has been added to the file, and all the pieces of the file are there for enforcement.
- Enabled** The check constraint is enforced if the constraint is also established. If the constraint is defined, the file member does not exist for enforcement.
- Disabled** The constraint definition is *not* enforced regardless of whether the constraint is established or defined.

Use the **WRKPF CST** command to see the constraints defined for the CUSTOMER file.

```
Work with Physical File Constraints

Type options, press Enter.
2=Change 4=Remove 6=Display records in check pending

Opt Constraint      File      Library  Type      State      Check
                  >        >        >        >        >        Pending
QSYS_CUSTO         > CUSTOMER  ORDAPPLIB *PRIKEY
CUSTOT_LIM         > CUSTOMER  ORDAPPLIB *CHKCST  EST/ENB  NO
CUSCRD_VS_         > CUSTOMER  ORDAPPLIB *CHKCST  EST/DSB  YES

Parameters for options 2, 4, 6 or command
===>
F3=Exit  F4=Prompt  F5=Refresh  F9=Retrieve  F12=Cancel  F15=Sort
F16=Repeat position to  F17=Position to  F22=Display constraint name
```

Figure 4-12 Physical file constraints

There are two check constraint definitions for this file. One is established and enabled, and the other one is established and disabled. At the same time, the constraint that is disabled has some records in a check pending condition. If you want to see the records that are causing the check condition, type option **6** next to the constraint with the check pending status.

Figure 4-13 shows the job log of the job in which the **ADDPFCST** command was executed and caused the check pending condition.

```

                                Display All Messages
                                System:   RC
Job . . . : QPADEV0004   User . . . : HERNANDO   Number . . . : 00

CHECK constraint CUSTOT_LIMIT_CUSCRD cannot be added.
3 > ADDPFCST FILE(ORDAPPLIB/CUSTOMER) TYPE(*CHKCST) CST(CUSCRD_VS_CUSTOT
ST('CUSCRD < CUSTOT'))
Field values are not valid for check constraint. 1
Field values are not valid for check constraint.
Field values are not valid for check constraint.
Field values are not valid for check constraint.
Field values are not valid for check constraint.
Field values are not valid for check constraint.
Field values are not valid for check constraint.
Field values are not valid for check constraint.
Field values are not valid for check constraint.
Constraint is in check pending. 2
Constraint was added. 3
1 constraint(s) added to file CUSTOMER but constraint(s) in error.

Press Enter to continue.

F3=Exit  F5=Refresh  F12=Cancel  F17=Top  F18=Bottom

```

Figure 4-13 Check constraint messages

**Notes:**

- 1 The condition is checked for every single record in the file. In this case, there are several records that do not meet the condition.
- 2 Since there are records that violate the condition, a check pending status is set for the constraint.
- 3 Since the native interface has been used, the constraint is added to the file.

## 4.6.2 Save and restore considerations

When a table is saved, all of its check constraints are saved. For restores, if the check constraints are on the save media and the saved file does not exist, the check constraint is added to the file. However, if the file exists in the system, any check constraint on the SAVE media is ignored.

If you are restoring data and the data results in a check constraint violation, the data is restored and the constraint goes into a check pending condition. The state of the constraint stays the same.

When you run the CRTDUPOBJ command from a file with check constraints, the check constraints are propagated from the original file to the new file. Both the constraint state and the check pending status are replicated from the original to the destination file.

## 4.7 Tips and techniques

We complete this chapter with some tips for those of you who are responsible for moving the business rules to the database. This applies to both check constraints and referential integrity constraints.

Start identifying and isolating the application code that is responsible for the referential integrity and check constraint checks. These pieces of code that are duplicated in several different programs should be rewritten in ILE procedures that can be bound and reused by several application programs. These procedures can then be defined as trigger programs to make the concerning checks.

At the same time, start to clean up your data. This can be done by queries that highlight records that are violating the constraints, or you can natively define the constraints and use the WRKPF CST command to see the records that are in check pending. At this step, carefully schedule the time for these queries so you do not interrupt the normal operation of the business.

Once your data is clean, it is time to enable all the constraints and remove the trigger programs that are no longer needed.

Before you decide on your check constraint naming convention, consider this tip: It might make it easier to turn system constraint errors into meaningful user feedback by defining your constraint name to be the message number that you really want displayed to the end user.

When you define the check constraints, a question may arise: Is it better to have *one large* check constraint or multiple check constraints? When there is more than one check constraint for a file, the system implicitly ANDs the result of each constraint during the enforcement phase. From the performance point of view, *one large* constraint performs slightly better than multiple check constraints because the implicit AND processing is eliminated. On the other hand, it is easier to manage and identify multiple but simpler check constraints. It is easier to identify problems when the system detects violations of the constraint. It is up to the application programmer or the DBA to decide which approach is better. Let's look at an example:

```
ALTER TABLE ORDAPPLIB/STOCK
  ADD CONSTRAINT PRODUCT_PRICE_MIN
  CHECK(PRODUCT_PRICE > 0)

ALTER TABLE ORDAPPLIB/STOCK
  ADD CONSTRAINT PRODUCT_AVAIL_MIN
  CHECK(PRODUCT_AVAILABLE_QTY >= 0)

ALTER TABLE ORDAPPLIB/STOCK
  ADD CONSTRAINT PRODUCT_MIN_STOCK
  CHECK(PRODUCT_MIN_STOCK_QTY >= 0)
```

These three check constraints can be combined in one constraint as shown in the following example:

```
ALTER TABLE ORDAPPLIB/STOCK
  ADD CONSTRAINT STOCK_CONSTRAINTS
  CHECK(PRODUCT_PRICE > 0 AND PRODUCT_AVAILABLE_QTY >= 0
        AND PRODUCT_MIN_STOCK_QTY >= 0)
```

**Note:** Keep in mind that, if you have multiple check constraints that are violated on an insert operation, only a single error message is returned. The system stops enforcement and signals the error on the first check constraint violation it finds.





## DRDA and two-phase commitment control

This chapter presents:

- ▶ DRDA evolution from DRDA-1 to DRDA-2
- ▶ DRDA-2 connection management
- ▶ Two-phase commitment control
- ▶ SQL support for DRDA-2
- ▶ Coexistence between DRDA-1 and DRDA-2
- ▶ Recovering from failures
- ▶ Application design considerations
- ▶ A DRDA-2 program example
- ▶ DRDA over TCP/IP
- ▶ DB2 Connect setup over TCP/IP

## 5.1 Introduction to DRDA

The Distributed Relational Database Architecture (DRDA) represents IBM's proposal in the arena of distributed database access. This architecture defines the rules, the protocols, and the semantics for writing programs implementing distributed data access. All the platforms participating in this architecture must comply with these rules and definitions.

This chapter does not discuss, in detail, every component of DRDA. The purpose is to provide you with a brief outlook on DRDA evolution and to describe the implementation of DRDA in the DB2 UDB for iSeries environment.

### 5.1.1 DRDA architecture

Distributed Relational Database Architecture allows you to access data in a distributed relational database environment by using SQL statements in your applications. The architecture has been designed to allow distributed data access for systems in *like* and *unlike* operating environments. This means that your applications can access data residing on homogeneous or heterogeneous platforms.

DRDA is based on these IBM and non-IBM architectures:

- ▶ SNA Logical Unit Type 6.2 (LU 6.2)
- ▶ TCP/IP Socket Interface
- ▶ Distributed Data Management (DDM) architecture
- ▶ Formatted Data Object Content Architecture (FD:OCA)
- ▶ Character Data Representation Architecture (CDRA)

On the iSeries server, DRDA is part of DB2 UDB for iSeries, which is part of the OS/400 operating system.

### 5.1.2 SQL as a common DRDA database access language

SQL has become the most common data access language for relational databases in the industry. SQL was chosen as part of DRDA because of its high degree of standardization and portability.

In a distributed environment, where you want to access data at remote locations, the SQL requests are routed to the remote systems and they are executed remotely. Prior to sending the remote SQL request, a DRDA application must establish a *connection* with the remote relational database where the data is located. This is the purpose of the CONNECT SQL statement provided by DRDA.

### 5.1.3 Application requester and application server

In a distributed relational database environment, the system running the application and sending the SQL requests across the network is called an *application requester* (AR). Any remote system that executes SQL requests coming from the application requester is also known as an *application server* (AS). Some platforms can participate in a distributed database environment as both an application requester and an application server. The diagram in Figure 5-1 shows the current application requester and application server capabilities of different database management systems.

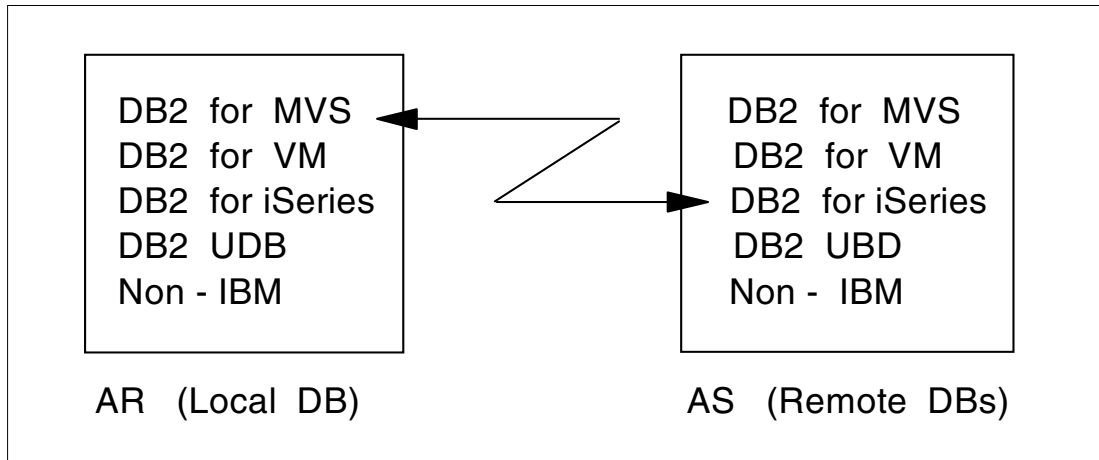


Figure 5-1 Current support for application requester (AR) and application server (AS)

**Note:** Currently, the DB2 Universal Database and DB2 Connect offer different levels of DRDA implementation depending on the OS platform. The support level equivalent to that of OS/400 is available for AIX, Windows NT, HP-UX, and OS/2. Consult the appropriate documentation for the latest additions.

## 5.1.4 Unit of work

Unit of work (UoW), unit of recovery (UR), or logical transaction are different ways to refer to the same concept. The DRDA terminology prefers the term *unit of work*. Unit of work refers to a sequence of database requests that carry out a particular task, such as in a banking application when you transfer money from your savings account to your checking account. This task has its logical independence and should be treated “atomically”, which means that either all its components are executed or none of them are. You do not want your savings balance to be updated without your checking balance being updated too. A unit of work is generally terminated by a commit operation if the entire task completes successfully. For more information about UoW, refer to *Distributed Database Programming*, SC41-5702.

DRDA defines the following levels of service regarding UoW:

- ▶ Level 0, Remote Request (RR):
  - One request within one UoW to one DBMS. Remember that DB2 UDB for iSeries provides one DBMS. DB2 for OS/390 or DB2 Universal Database can provide multiple DBMSs on the same system.
  - Remote request was available before DRDA, thanks to DDM support.
- ▶ Level 1, Remote Unit of Work (RUW):
  - One or more SQL requests within one UoW to a single DBMS.
  - Switching to a different location is possible, but a new UoW must be started and the previous one must be completed.
  - Remote Unit of Work is supported by both the SNA and TCP/IP implementations of DRDA.
- ▶ Level 2, Distributed Unit of Work (DUW):
  - Many SQL requests within one UoW to several DBMS.
  - Two-phase commit is required.
  - A single request may reference objects residing on the same DBMS.

- The Distributed Unit of Work is currently supported only by the SNA implementation of DRDA.
- ▶ Level 3, Distributed Request (DR):
  - In addition to the services provided by Distributed UoW, DR allows a single SQL request to include references to multiple DBMSs, such as joining tables stored at different locations.
  - This is an architected level and will be available in the future.

The diagram in Figure 5-2 may be helpful in understanding the levels of DRDA.

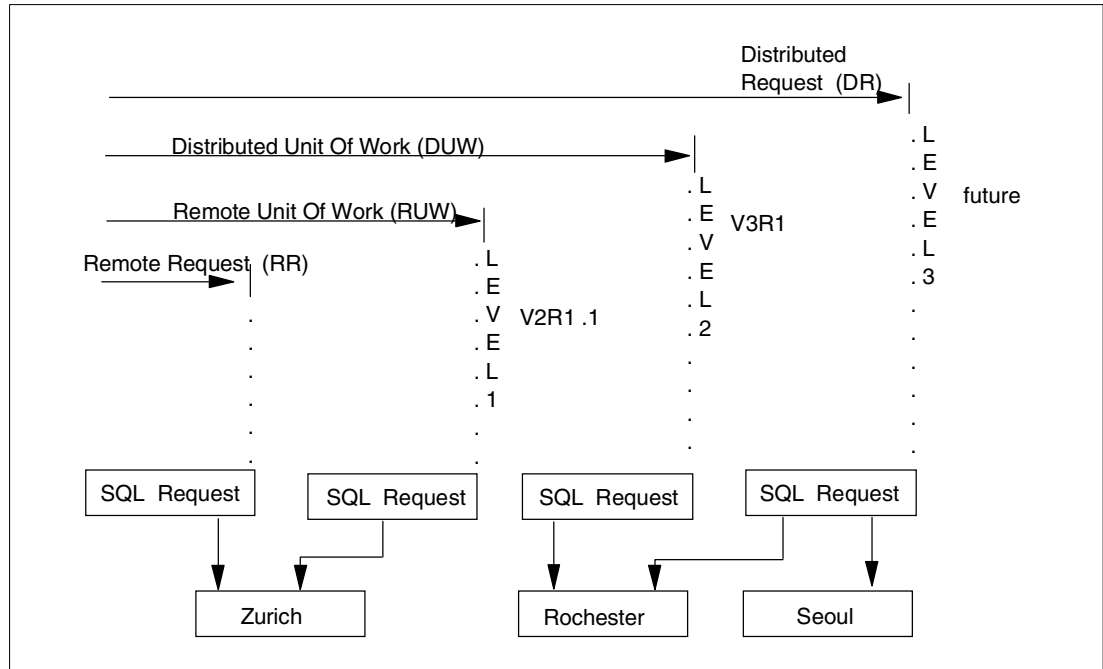


Figure 5-2 Architected service levels of DRDA

### 5.1.5 Openness

Many non-IBM relational database providers (for example, Informix, Oracle Sybase, XDB Systems, and others) implement different levels of DRDA support in their products. DRDA offers the ability to access and exchange data in *like* and *unlike* system environments, therefore, contributing to the openness of IBM platforms in regard to interoperability.

## 5.2 Comparing DRDA-1 and DRDA-2

The difference between DRDA-1 and DRDA-2 from an application point of view is illustrated in Figure 5-3. DRDA-2 introduces:

- ▶ Two-phase commit protocol to keep multiple databases in synchronization
- ▶ *Synchronization Point Manager* (SPM) to manage the two-phase commit
- ▶ A new *connection management*
- ▶ New SQL statements to manage multiple connections

DRDA-1 cannot maintain multiple connections in one unit of work. To connect to a different application server, the application must be in a connectable state, which is achieved by ending the unit of work with a COMMIT or ROLLBACK statement.

DRDA-2 can connect to multiple servers without losing the existing connections. A single unit of work can span multiple servers. Keep in mind that a single SQL statement still cannot address more than one server at a time. For example, it is still not possible to join two files residing on different systems.

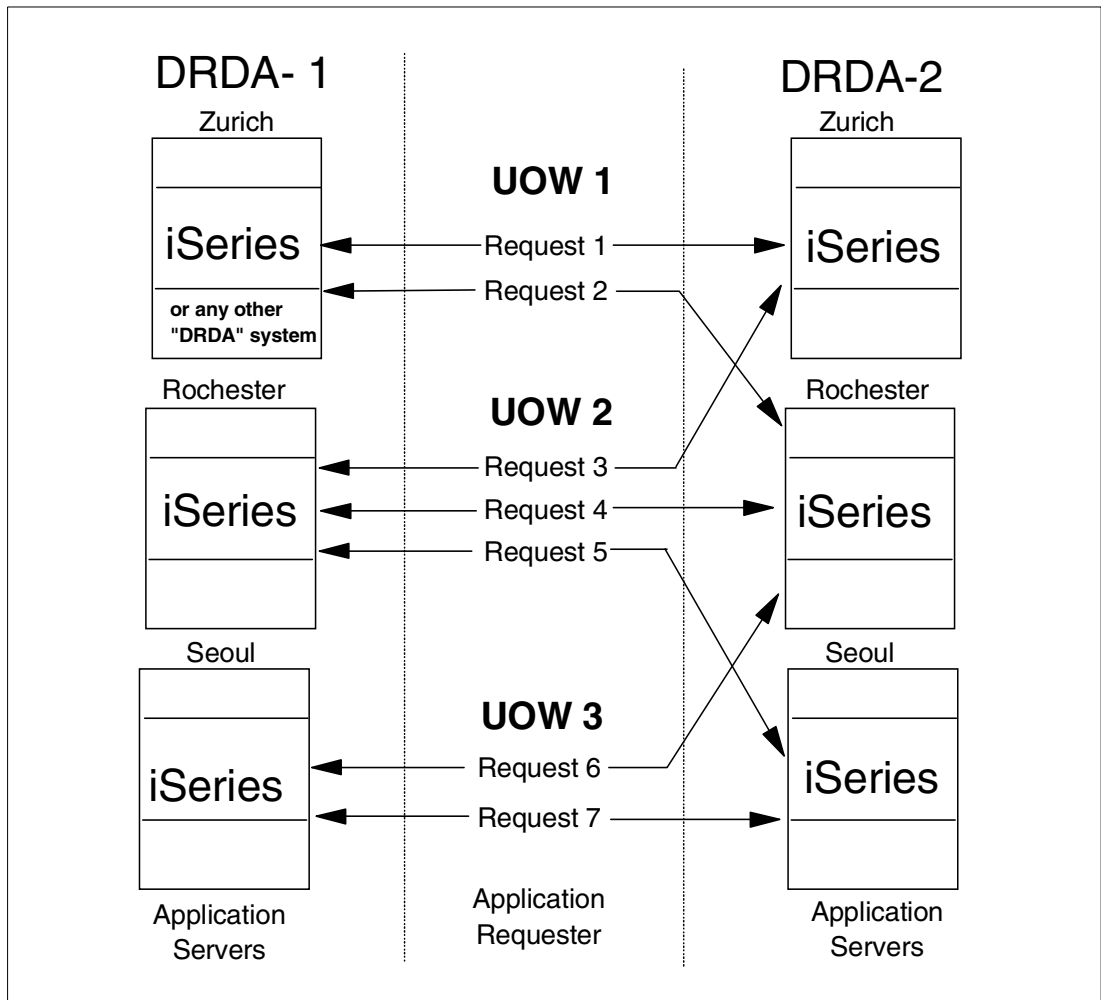


Figure 5-3 Remote Unit of Work (DRDA-1) versus Distributed Unit of Work (DRDA-2)

Figure 5-3 shows three units of work. The arrows pointing to the left indicate the only possible way to access data in a DRDA-1 application. The arrows pointing to the right show the new flexibility of a DRDA-2 application accessing multiple systems in the same UoW. Let's consider, for example, the Rochester system on the right-hand side. It issues three requests: Request 2, Request 4, and Request 6. Each of these requests belong to a different unit of work. The Rochester system on the left-hand side also issues three requests (Request 3, Request 4, and Request 5), each targeting a different application server within one UoW.

### 5.3 DRDA-2 connection management

Connection management refers to the set of mechanisms by which you can direct your database requests in a distributed database network. DRDA-2 has enhanced connection management, which allows an application program to keep alive the existing connections and perform I/O operations on multiple relational databases within the same unit of work. Currently, this architected level of DRDA is available only over SNA. It will also be available in

future releases over the TCP/IP implementation of DRDA. There are also some changes to the way the CONNECT statement behaves in DRDA-2 if you compare it with the DRDA-1 CONNECT behavior. In DRDA-1, the current connection is destroyed when a new CONNECT statement is issued. In DRDA-2, another CONNECT statement does not destroy the existing connections. A new one is created instead and becomes the current connection.

Also, if you issue a CONNECT statement toward an existing connection, you receive negative SQLCODE if your programs use DRDA-2 connection management and the current connection does not change. In a DRDA-1 program, this operation is legitimate and does not return an error.

### 5.3.1 Connection management methods

As we previously indicated, DB2 UDB for iSeries allows you to use both the DRDA-1 connection management method and the DRDA-2 connection manager. When you create your SQL program or module, you specify which connection method you want to use:

- ▶ DRDA-1 connection management is set on the CRTSQLxxx command by specifying RDBCNNMTH(\*RUW).
- ▶ DRDA-2 connection management is specified on the CRTSQLnnn command by specifying RDBCNNMTH(\*DUW), which is also the default for the creation commands.

Because the connection method changes the semantics of the CONNECT statement, you must be aware of this parameter when you are recompiling existing applications, because they can behave in a different way if compiled with the \*DUW option. For more details, see 5.6, “DRDA-1 and DRDA-2 coexistence” on page 95.

DB2 UDB for iSeries also allows you to specify that an implicit connection must take place when the program is started. This is the purpose of the RDB parameter on the precompiler commands. This implicit CONNECT will be of the type specified in the RDBCNNMTH parameter.

If an RDB name is specified, the connection to this remote database is established automatically at program start time. Therefore, the first re-connection statement in the program has to be SET CONNECTION. If CONNECT is initiated to this application server by the program logic, the SQL0842 message (SQLCODE = -842 - “Connection to relational database xxx already exists”) is sent to the application. Check for this SQLCODE explicitly after every CONNECT statement. In general, this SQLCODE can be ignored by the application. Remember that a CONNECT statement followed by SQLCODE = -842 does not change the current connection.

### 5.3.2 Connection states

DRDA-2 introduces new connection states. A connection may be either held or released and current or dormant. For clarification, see Figure 5-4.

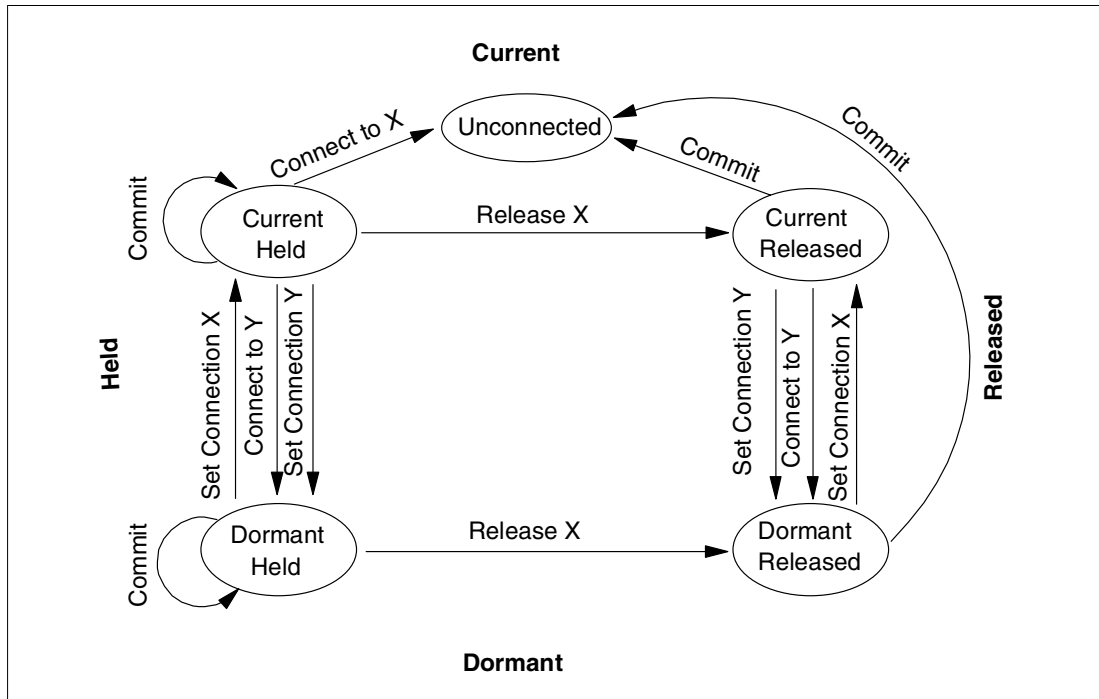


Figure 5-4 Connection states

Figure 5-4 shows a general picture of the architecture. The application goes into an UNCONNECTED state when the current connection is destroyed. The application ends in an unconnected state if all the connections are released and a commit is performed. The CONNECT and RELEASE statements allow the application to change a connection state from held to released:

- ▶ **Released state:** Means that a disconnect will occur at the next successful commit operation (a rollback has no affect on connections). Therefore, a released state can be thought of as a pending disconnect.
- ▶ **Held state:** Means that a connection will not be lost at the next commit operation. A connection in the released state cannot be put back into a held state. This means that a connection may remain in a released state across unit of work boundaries if a ROLLBACK is issued.

Regardless of whether a connection is in a held or released state, a connection can also be in a current or dormant state:

- ▶ **Current state:** Means that the connection is the one used for SQL statements that are executed.
- ▶ **Dormant state:** Means that the connection is suspended. While in this state, no SQL statements can use this connection. Nevertheless, SQL statements can always be executed against the current connection.

If you want a dormant connection to become the current connection, use the SET CONNECTION SQL statement. The existing current connection will become dormant. In fact, there can be only one connection in the current state at a time. All other connections are in a dormant state. You cannot use the CONNECT statement to make a dormant connection current in a DB2 UDB for iSeries application. The semantic of the CONNECT statement is different in DB2 UDB for iSeries and DB2 for OS/390, where a CONNECT to an existing connection equates to a SET CONNECTION statement.

When a connection goes to the dormant state, all the open cursors, locks, and prepared statements are preserved. When this connection becomes current again in the same unit of work, all locks, cursors, and prepared statements are restored to their previous values.

In a network where systems at different levels coexist, you may have connections to DRDA-2 servers and to DRDA-1 servers at the same time. Connection to DRDA-1 servers must be dropped by using the DISCONNECT statement.

Once disconnected, an application must connect to the database again before it can direct SQL statements to it.

## 5.4 Two-phase commitment control

Synchronizing multiple databases requires additional effort compared to the process of keeping data consistent on a single system. Because multiple physical locations are involved, the synchronization process is split into *two phases* to ensure data consistency across multiple locations. The database managers involved in the distributed unit of work must make sure that either all of them commit their changes or roll all the changes back consistently.

The protocol by which multiple database managers can keep their data in sync is called *two-phase commitment control*.

In an application using two-phase commit, the COMMIT statement generates a rather complex sequence of operations that allows the various agents in the network to keep their data in a consistent state. Also, a two-phase commit protects your applications against network or system failures that may occur during the transaction. In these cases, the database managers involved in the unit of work automatically roll back their changes.

As mentioned already, current DRDA-2 implementation is based on LU 6.2 architecture. When an LU 6.2 conversation supports a two-phase commitment control data flow, we say that it is a *protected conversation*. Some new verbs have been added to the APPC protocol to support protected conversations. You have direct access to this support on the iSeries server by using ICF files or CPI-C functions in your applications.

Any LU 6.2 conversation not capable of a two-phase commitment control flow is called *unprotected conversation*. DRDA-1 supports only unprotected conversations.

### 5.4.1 Synchronization Point Manager (SPM)

To control the two-phase commit flow, DB2 UDB for iSeries implements a component called *Synchronization Point Manager*. The SPM also controls rollback among the various protected resources. Either all changes are committed or they are rolled back.

With distributed updates, sync point managers on different systems cooperate to ensure that resources reach a consistent state. The example in Figure 5-5 shows the type of information that flows between the application requesters and application servers to commit work on protected conversations.



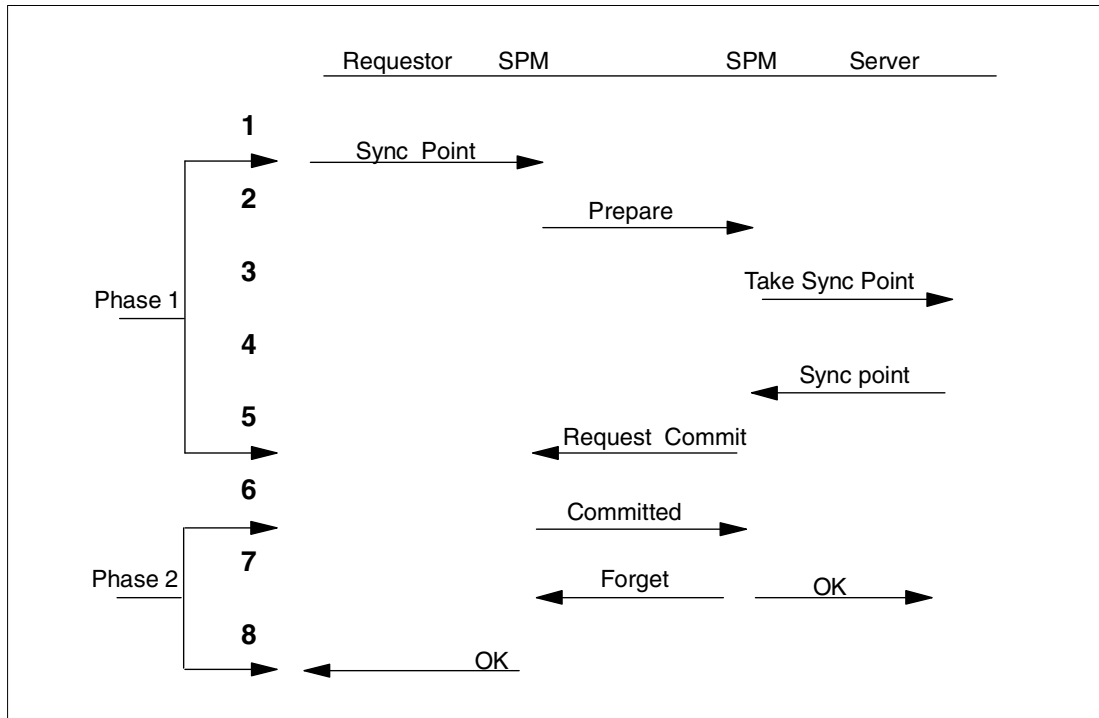


Figure 5-5 Technical view of two-phase commit flow

Each application requester and each application server have a sync point manager attached. DBMS participating in a Distributed Unit of Work has to cooperate to ensure the synchronization. *Phase one* consists of this process:

1. The application requester issues a COMMIT. All participating application servers and the application requester must be synchronized. The requester must wait now until it receives the OK from its SPM.
2. The SPM of the application requester sends a *prepare for commit* request to the SPMs of the servers.
3. All the SPMs at the server systems initiate the process of logging all the database changes and reaching the sync point.
4. The servers send a completion message to their SPMs.
5. The SPM requests a commit from the application servers SPM.

For *phase two*, proceed with the following actions:

1. Once the SPMs have received the responses and logged them, they return a *committed* message.
2. The server SPMs send *Forget* to the application requester SPM and *OK* to their application servers. Everything has been synchronized now.
3. The application requester receives the OK from its SPM and can continue.

Figure 5-6 shows the application view of the two-phase commit flow.

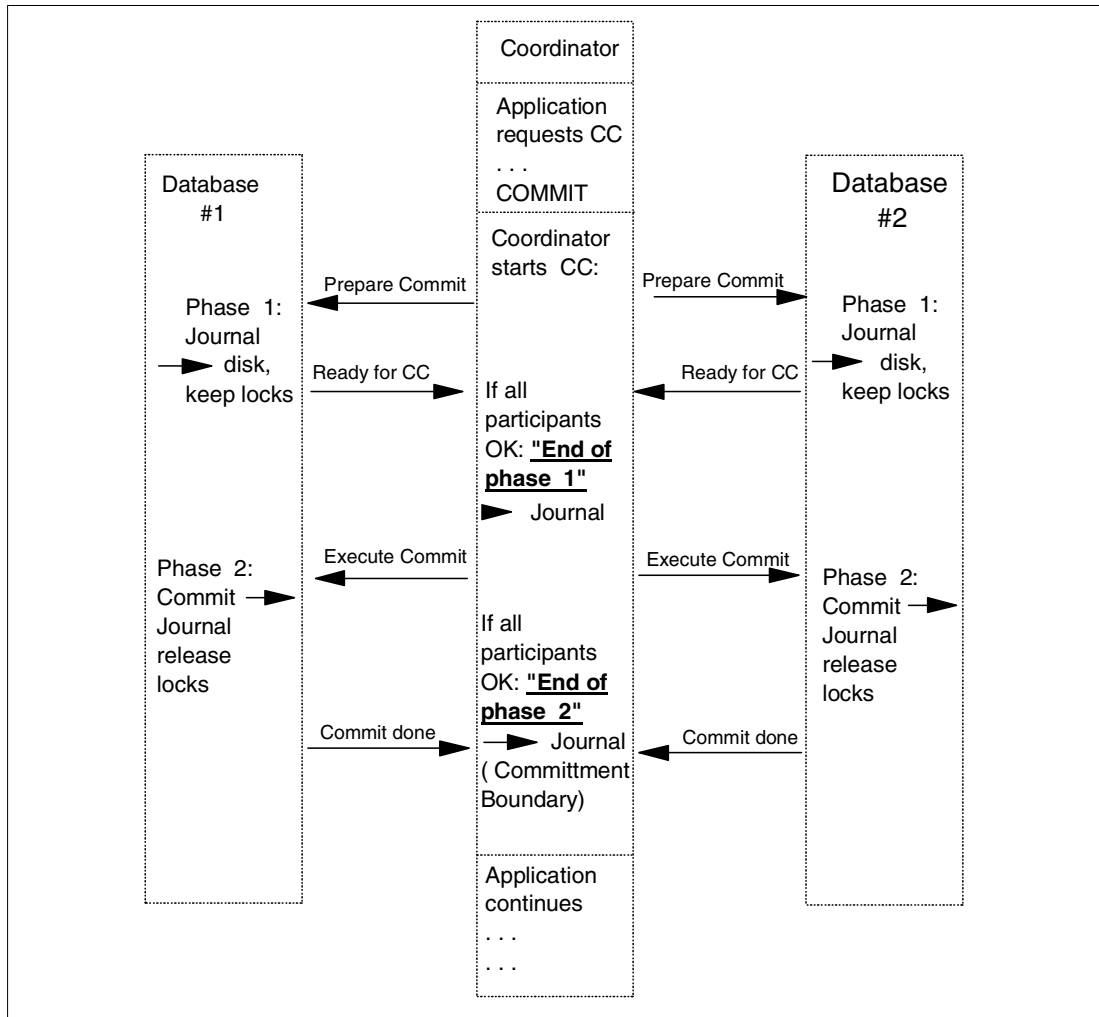


Figure 5-6 Application view of two-phase commit flow

**Note:** The more database management systems (DBMS) that are involved in one Distributed Unit of Work, the more messages have to flow. This requires more resources and causes additional communication line traffic.

## 5.5 DB2 UDB for iSeries SQL support for connection management

With DRDA-2, some SQL connection statements were added or have changed. They are listed here in alphabetical order. For more details, refer to *Distributed Database Programming*, SC41-5702.

### CONNECT (Type 1)

The CONNECT (Type 1) statement connects an activation group within an application process to the identified application server, using the rules for the Remote Unit of Work. The term *activation group* refers to a substructure of a job that contains the resources necessary to run programs. For more details on activation groups, see *ILE Concepts*, SC41-5606.

If your application runs multiple activation groups, each connection is private to the activation group that issued it. The connection terminates if the activation group ends. This termination occurs whether the application is using the DRDA-1 or DRDA-2 connection methods.

A program compiled with the RDBCNNMTH parameter of the CRTSQLpgm command set to \*RUW runs with the CONNECT Type 1 connection method.

Consecutive CONNECT statements can be executed successfully because CONNECT does not remove the activation group from the connectable state. A connect to the application server to which the activation group is currently connected is executed similar to any other CONNECT statement. CONNECT cannot execute successfully when it is preceded by any SQL statement other than CONNECT, COMMIT, or ROLLBACK. To avoid an error, execute a COMMIT or ROLLBACK operation before the CONNECT.

## CONNECT (Type 2)

The CONNECT (Type 2) statement connects an activation group within an application process to the identified application server using the rules for the Distributed Unit of Work. This server is then the current server for the process.

A program runs with this connection management method if it is compiled with the RDBCNNMTH parameter of the CRTSQLpgm command set to \*DUW.

**Note:** CONNECT (Type 2) cannot be issued a second time to the same server as long as the connection is still alive. Use SET CONNECTION to activate another server.

## DISCONNECT

The DISCONNECT statement destroys one or more connections for unprotected connections (connections without two-phase commit). See 5.4, “Two-phase commitment control” on page 90.

You cannot issue a DISCONNECT statement toward a protected conversation or toward any connection that sent SQL statements during the current unit of work.

## RELEASE

The RELEASE statement places one or more connections in the released state. This statement is allowed only for protected conversations.

If the statement is successful, each identified connection is placed in the released state and is, therefore, destroyed during the execution of the next COMMIT operation. Keep in mind that a ROLLBACK will not end the connection.

**Note:** Creating and maintaining active connections requires some effort on behalf of the systems involved in the database network. This is why your applications should drop active connections if they are not going to be reused.

If the current connection is in the released state when a commit operation is executed, destroying that connection places the activation group in the unconnected state. In this case, the next SQL statement to this application server must be CONNECT. If SET CONNECTION is used to this application server, an error (SQL0843, “Connection to relational database xxx does not exist.”) is encountered. SET CONNECTION is only possible to an application server other than the previously released one.

## SET CONNECTION

SET CONNECTION activates an already connected server so that all SQL statements, from now on, are directed to this server until another SET CONNECTION is issued, a CONNECT to a new server is executed, or the connection is ended. The SET CONNECTION statement brings the state of the connection from dormant to current.

After the activation group is reconnected to the server, it finds that its environment is in the same status as when it left this connection. The connection reflects its last use by the activation group with regard to the status of locks, cursors, and prepared statements.

### 5.5.1 Example of an application flow using DRDA-2

In Figure 5-7, you can find a high-level description of a DRDA-2 application flow. Notice that the environment includes the coexistence of both the DRDA-2 and DRDA-1 systems.

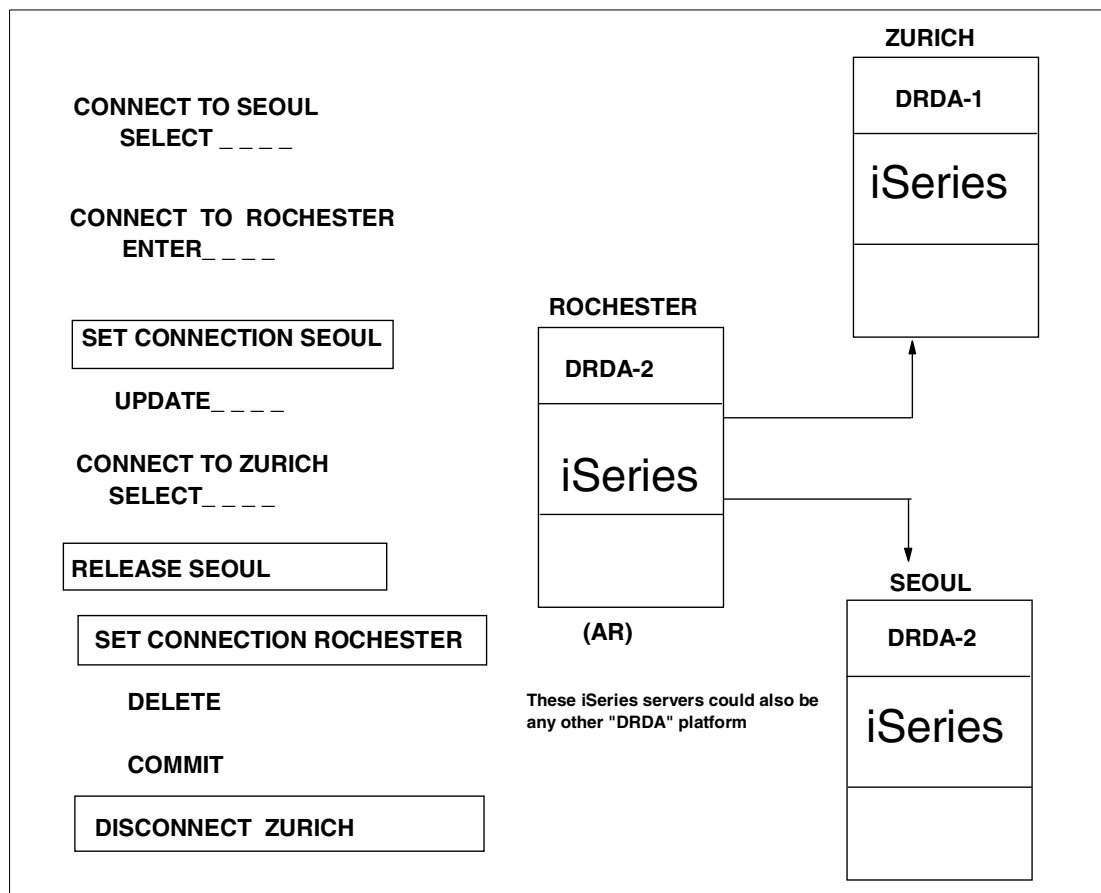


Figure 5-7 Application flow example using the new connection management

The systems located in Rochester and Seoul support DRDA-2, where the system in Zurich is running at DRDA-1. The AR in Rochester connects to Seoul, does some work, connects back to the local database without COMMIT, and reconnects to Seoul through the new SET CONNECTION statement. Then Rochester connects to Zurich, Seoul is released, and Rochester reconnects to the local database. Before we finish our unit of work, we release and disconnect Zurich. After finishing the unit of work, no more remote connections are active because we released Seoul and disconnected Zurich. Finally, we are connected to our local database, Rochester.

## 5.6 DRDA-1 and DRDA-2 coexistence

As Figure 5-7 shows, DRDA-2 and DRDA-1 systems can coexist in the same network, and a DRDA-2 application can access both types of application servers during its execution. Since DRDA-1 application servers do not support a protected conversation, some limitations may apply as to which systems can be accessed in update mode. An application requester determines, at the initial connect time, whether a DRDA-1 application server can be updated. A DRDA-2 application requester connection to a DRDA-1 application server can be used to perform updates when:

- ▶ The program was compiled with an isolation level other than \*NONE.
- ▶ There are no other connections or they all are DRDA-1 read-only connections.

**Note:** COMMIT(\*NONE) on DB2 UDB for iSeries means that no transaction isolation is done and no logs are written. It can only be used in a DB2 UDB for iSeries-like environment.

At connect time, the DB2 UDB for iSeries application requester chooses whether a sync point manager is used and, thus, whether the application server can be updated. Depending on this decision, different DRDA and commitment control protocols are used. Table 5-1 shows how the different flows can be mixed together.

Table 5-1 *Mixing DRDA levels*

Application server requesters	Application servers		
	DRDA-1	DRDA-2 1PC	DRDA-2 2PC
DRDA-1	DRDA-1	DRDA-1	DRDA-1
DRDA-2 1PC	DRDA-1	DRDA-2 1PC	DRDA-2 1PC
DRDA-2 2PC	DRDA-1	DRDA-2 1PC	DRDA-2 2PC
<b>Notes:</b> 1PC = Single phase commit 2PC = Two-phase commit			

Table 5-1 indicates:

- ▶ Application requester is at DRDA-1:
  - All application servers use the DRDA-1 flow. This implies a single-phase commit and unprotected conversations.
- ▶ Application requester is at DRDA-2 supporting a single-phase commit (1PC):
  - When the application server is at DRDA-1, DRDA-1 protocol is used. All others use a DRDA-2 flow with single-phase commit.
- ▶ Application requester is at DRDA-2 with a two-phase commit:
  - When the application server is at DRDA-1, DRDA-1 flows are used. When the application server is at DRDA-2 with single-phase commit capability, DRDA-2 single-phase commit flow is used. When the application server is at DRDA-2 with two-phase commit capability, DRDA-2 two-phase commit flow is used.

In a heterogeneous environment, the protocol used depends on the application requester according to Table 5-1.

## 5.7 Recovery from failure

As we mentioned earlier, in most cases the recovery is totally automatic. When the systems detect a failure, the current transaction is automatically rolled back on all the systems. Still, there is a narrow window in the two-phase commit cycle where a network failure or a system failure may leave the transaction in a pending state because the application requester cannot determine which action to take. This window is located right before the last step of the two-phase commit process, when the application server may already have committed a transaction, but for some reason cannot send the final acknowledgment. When the transaction hangs, all the locks are preserved and the application receives an I/O error.

This section describes how the system or the users can recover after a network or a system failure in a two-phase commit environment.

### 5.7.1 General considerations

To control the synchronization over multiple systems, DB2 UDB for iSeries uses a *Logical Unit of Work ID*. This identifier is the same on all systems involved, whether they are application requesters or application servers with like or unlike platforms.

On DB2 UDB for iSeries, the unit of work ID looks similar to the following example:

```
APPNET.ROCHESTER.X'F2DEB3D611CA'.00001
```

**Note:** This ID is composed of four parts, where:

- ▶ *APPNET* is the APPN Net-ID
- ▶ *ROCHESTER* is the application requester system name
- ▶ *X'F2DE....* is related to the job, running a protected conversation
- ▶ *000...* relates to the program call within the job

On the iSeries server, this identifier should actually be called *activation group ID* because the number remains the same over the life of an activation group. The program or activation group can start a large number of units of work.

Ending the program and calling it again changes the last part of the identification number, which then looks similar to this example:

```
APPNET.ROCHESTER.X'F2DEB3D611CA'.00003
```

If you start a new job on the iSeries server and run the same application, the identifier will change its third component:

```
APPNET.ROCHESTER.X'F2E1B36111CB'.00001
```

**Note:** A new job changes the *last two parts* of the identification number.

### 5.7.2 Automatic recovery

DB2 UDB for iSeries with DRDA-2 and two-phase commitment control provides a comprehensive recovery mechanism after system, network, or job failures.

Automatic recovery was tested with programs from the Order Entry Application example described in Chapter 2, “Using the advanced functions: An Order Entry application” on page 11, particularly with the Insert Order Detail program documented in Appendix A, “Order Entry application: Detailed flow” on page 329. This program (INSDET) calls a stored

procedure (STORID) on a remote iSeries server. The stored procedure updates a STOCK table on the remote system. Then, the calling program inserts an order detail record in a ORDERDTL table on the local system. After doing this, the Distributed Unit of Work (DUW) is completed.

**Note:** ROCHESTER is the local system (AR). ZURICH is the remote system (AS).

In this test scenario, the stored procedure program on the remote system was abruptly terminated, cancelling the job before the database changes on both systems were committed.

In this case, the remote system (application server) rolled back the one database change automatically and provided information in the job log. At the application requester, information provided in the program ended, but not before rolling back the local database change, which was a record insert. The rollback operation is needed since the calling program received SQL error return code -918, which corresponds to message SQL0918. The details are shown in Figure 5-8.

```

                                Display Formatted Message Text
                                System:  ROCHESTER
Message ID . . . . . :  SQL0918
Message file . . . . . :  QSQLMSG
  Library . . . . . :  QSYS

Message . . . . . :  ROLLBACK is required.
Cause . . . . . :  The activation group requires a ROLLBACK to be performed
  prior to running any other SQL statements.
Recovery . . . . . :  Issue a ROLLBACK CL command or an SQL ROLLBACK statement
  and then continue.

```

Figure 5-8 Message SQL0918

The job log of the remote system (ZURICH) reported the following information:

```

.....
CPI9152 Information Target DDM job started by source system.
CPI3E01 Information Local relational database accessed by ROCHESTER.
CPC1125 Completion Job ../ITSCID06/ROCHESTER was ended by user ITSCID03.
CPD83DD Diagnostic Conversation terminated; reason 02.
                                02 -- The conversation was issued a Deallocate Type
                                (Abend) to force the remote location to roll back.
CPF4059 Diagnostic System abnormally ended the transaction with device ROCHESTER.
CPI8369 Information 1 pending changes rolled back; reason 01.
                                01 -- The commitment definition is in a state of Reset.
CPF83E4 Diagnostic Commitment control ended with resources not committed.
.....

```

### 5.7.3 Manual recovery

The Work with Commitment Definition (WRKCMDFN) command allows users to manage commitment definitions for any job on the system. This command becomes particularly useful when a system or line failure causes transactions to hang while waiting for synchronization.

A commitment definition reports information about a job commitment control status after commitment control has been started with either the Start Commitment Control (STRCMTCTL) command or by a program containing embedded SQL commitment statements.

## Using Work with Commitment Definitions

This command provides detailed information about the commitment control status of an activation group. The main display may look similar to the example in Figure 5-9, where only one active commitment definition is shown.

```

Work with Commitment Definitions
System: ROCHESTER

Type options, press Enter.
 5=Display status  12=Work with job  14=Forced commit
16=Forced rollback ...

      Commitment
Opt  Definition Job      User      Number  Resync In
5    *DFACTGRP  P23KXC48E  ITSCID06  004590  NO

Bottom

3=Exit  F5=Refresh  F9=Command line  F11=Display logical unit of work
12=Cancel  F16=Sort by logical unit of work ID  F24=More keys

```

Figure 5-9 Work with Commitment Definition command display

If more activation groups are involved, more commitment definitions are listed. When you choose option 5 (Display status), three more displays with further details of the commitment definition shown in Figure 5-10 through Figure 5-12 appear.

```

Display Commitment Definition Status
ROCHESTER
05/25/01 23:03:42
Job: P23KXC48E User: ITSCID06 Number: 004590

Commitment definition . . . . . : *DFACTGRP
Activation group . . . . . : 2

Logical Unit of Work ID . . . . . : APPNET.ROCHESTER.X'F32D995711EE'.00003
Job active . . . . . : YES
Server job . . . . . :
Resource location . . . . . : REMOTE
Default lock level . . . . . : *CHG
Role . . . . . :
State . . . . . : RESET
Date/time stamp . . . . . :
Resync in progress . . . . . : NO
Number of commits . . . . . : 2
Number of rollbacks . . . . . : 0

More...

Press Enter to continue.

F3=Exit F5=Refresh F6=Display resource status F9=Command line
F12=Cancel

```

Figure 5-10 Display Commitment Definition Status (Part 1 of 3)



```

                                Display Commitment Definition Status
                                05/25/01 23:03:42
                                ROCHESTER
Job:  P23KXC48E      User:  ITSCID06      Number:  004590

Commitment definition . . . . . :  *DFACTGRP
Activation group . . . . . :  2

Heuristic operation . . . . . :
Default journal . . . . . :
  Library . . . . . :
Notify object . . . . . :  *NONE
  Library . . . . . :
  Object type . . . . . :
  Member . . . . . :

More...

F3=Exit  F5=Refresh  F6=Display resource status  F9=Command line
F12=Cancel

```

Figure 5-11 Display Commitment Definition Status (Part 2 of 3)

```

                                Display Commitment Definition Status
                                05/25/01 23:03:42
                                ROCHESTER
Job:  P23KXC48E      User:  ITSCID06      Number:  004590

Commitment definition . . . . . :  *DFACTGRP
Activation group . . . . . :  2

Commitment options:
  Wait for outcome . . . . . :  WAIT
  Action if problems . . . . . :  ROLLBACK
  Vote read-only permitted . . . :  NO
  Action if End . . . . . :  WAIT

Bottom

F3=Exit  F5=Refresh  F6=Display resource status  F9=Command line
F12=Cancel

```

Figure 5-12 Display Commitment Definition Status (Part 3 of 3)

Press F6 to look more into the details of a commitment definition. A window is displayed about the status of the single resources that are protected by commitment control (Figure 5-13).



Showing an example of manual recovery is not an easy task, since there is really little chance that the transaction will be interrupted when it is in an undecided state. The two-phase commitment control critical window is very narrow.

In these rare situations, the WRKCMDFN command provides a way to complete the transaction and release all the locks. It is up to the user to determine whether to use a commit or a rollback and force the transaction boundary by using either of the following methods on the display shown in Figure 5-9 on page 98:

- ▶ Option 14 (Forced commit)
- ▶ Option 16 (Forced rollback)

The manual recovery process may violate the alignment of the various databases in the network. Avoid this procedure if the automatic resynchronization is still possible by restoring the communication among the systems. Force the end of the transaction if the environment where the transaction was running before the failure cannot be restored, such as in the case of data loss or other serious system or network outages.

## 5.8 Application design considerations

This section describes how application developers should design their programs to fully exploit the flexibility provided by DRDA-2 in a distributed environment.

### 5.8.1 Moving from DRDA-1 to DRDA-2

The essential advantage of the Distributed Unit of Work (DRDA-2) over Remote Unit of Work (DRDA-1) is represented by the ability to access different locations within the same transaction, allowing much more flexibility to database and application design.

The flexibility offered by DRDA-2 introduces more complexity in regard to handling the connections within your applications. Multiple connections may be active at the same time, and you may need to determine whether your application is already connected to a specific location. To obtain this information, check the SQLCODE after issuing a CONNECT statement directed to that particular location. If you receive SQLCODE = -842, this means that the connection is already active and that you may need to perform a SET CONNECTION to establish that location as the current connection. If you receive SQLCODE = 0, the connection has just been activated and becomes the current connection.

#### Performance in a DRDA-2 environment

The higher the number is of the connections concurrently active, the higher the impact is on the application and system performance. Design your applications by trying to find the right balance between keeping your connections active, so that you do not need to restart them when you need them, and releasing the idle connections to reduce the system overhead.

The behavior of the initial connection depends on the programming model used by your application:

▶ **OPM programs:**

In a DRDA-1 program, each initial call of a program implicitly connects to the database specified in the RDB parameter of the CRTSQLxxx command. When the program terminates its execution, the connection is destroyed. If the same program is called several times within a job, the implicit connection is established each time. In a DRDA-1 program, you can count on this behavior and avoid coding the initial connection.

► **ILE programs:**

If ILE programs are created using the default parameters, the initial connection to the location specified in the RDB parameter will occur once in the life of an activation group. The connection will last as long as the activation group exists. In general, this behavior depends on the value of the CLOSQLCSR parameter, which defaults to \*ENDACTGRP. If your program runs in the default activation group or in a named activation group, you may need to check for existing connections.

## 5.9 DRDA-2 program examples

This section gives three examples of programs using DRDA-2 connection management and two-phase commitment control. The programs are taken from the Order Entry scenario.

### 5.9.1 Order Entry main program

The main program of our application only has the purpose of establishing the connections to the local and remote databases and of calling the various subprograms. The design choice of establishing all the necessary connections at the beginning allows the developers of the subprograms to rely on the existing connections. In the subprograms, there are only SET CONNECTION statements.

The following code listing shows a COBOL version of the main program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. T4249MAIN.
*
* This is the main program of the order entry application.
* The program establishes all the connections, so that the
* various sub-programs will need to issue only SET CONNECTION
* statements. At the end of the cycle, this program will
* release all the connections and commit all the changes.
*
ENVIRONMENT DIVISION.
*
DATA DIVISION.
*
WORKING-STORAGE SECTION.
*
* The error flag parameter is used by the various sub-programs
* to communicate a failure.
* No Errors: ERRFLG = 0
* Failure : ERRFLG = 1
*
01 ERR-FLG      PIC X(1).
01 TOTAMT      PIC S9(11) PACKED-DECIMAL.
01 CUSNBR      PIC X(5).
01 ORDNBR      PIC X(5).
*
EXEC SQL INCLUDE SQLCA END-EXEC.
PROCEDURE DIVISION.
*
EXEC SQL CONNECT TO RCHASM02
END-EXEC.
*
* Establish connections and check for successfull connect
*
IF SQLCODE NOT = 0 AND SQLCODE NOT = -842 THEN
```

```

        DISPLAY "Error connecting to RCHASM02"
    END-IF.
    EXEC SQL CONNECT TO RCHASM03
    END-EXEC
    IF SQLCODE NOT = 0 AND SQLCODE NOT = -842 THEN
        DISPLAY "Error connecting to RCHASM03"
    END-IF.
* Calling the restart procedure, that checks for
* incomplete orders and deletes them.
    CALL "T4249RSTR" USING ERR-FLG.
    IF ERR-FLG = 0 THEN
* Calling the insert order header program
    CALL "T4249CINS" USING CUSNBR, ORDNBR, ERR-FLG
    IF ERR-FLG = 0 THEN
* Calling the insert detail rows program
    CALL "T4249RIDT" USING CUSNBR, ORDNBR, ERR-FLG
    IF ERR-FLG = 0 THEN
* Calling the finalize order program
    CALL "T4249FNLO" USING CUSNBR,
                                ORDNBR,
                                TOTAMT,
                                ERR-FLG

        IF ERR-FLG = 0 THEN
            STOP RUN
        END-IF
    END-IF
    END-IF
    END-IF.
* In case of errors, perform a ROLLBACK
    EXEC SQL ROLLBACK
    END-EXEC.
    STOP RUN.

```

## 5.9.2 Deleting an order

This program may be invoked either at the beginning of the application execution (if some incomplete orders are found for the user) or at the end of it (if the user requests a cancellation of the order). The program scans the order detail rows and, for each item, it updates the quantity in the stock file at the remote site. At the end, the program deletes the order header. This operation causes all of the detail rows to go away as well because of the CASCADE rule that we implemented. If no errors are encountered in the process, the program commits the entire transaction.

The following code listing shows a COBOL implementation of this procedure:

```

IDENTIFICATION DIVISION.
PROGRAM-ID. T4249CORD.
*
* This program scans all the details referring to the input
* order number; it updates the available quantity of each
* detail in the remote STOCK file.
*
ENVIRONMENT DIVISION.
*
DATA DIVISION.
*
WORKING-STORAGE SECTION.
*
01 H-ORDQTY    PIC S9(5) PACKED-DECIMAL.
01 H-PRDQTA   PIC S9(5) PACKED-DECIMAL.

```

```

01 H-PRDNBR    PIC X(5).
01 H-ORHNBR    PIC X(5).
*
EXEC SQL INCLUDE SQLCA END-EXEC.
*
EXEC SQL DECLARE DETAIL CURSOR FOR
SELECT PRDNBR
      ,ORDQTY
FROM ORDENTL/ORDERDTL
WHERE ORHNBR = :h-ORHNBR
END-EXEC.
*
LINKAGE SECTION.
*
01 WK-ORHNBR    PIC X(5).
01 ERR-FLG      PIC X(1).
*
PROCEDURE DIVISION USING WK-ORHNBR
                        ERR-FLG.
*
MOVE WK-ORHNBR TO H-ORHNBR.
MOVE "0"       TO ERR-FLG.
*
EXEC SQL SET CONNECTION RCHASMO3 END-EXEC.
*
IF SQLCODE = 0 THEN
EXEC SQL OPEN DETAIL END-EXEC
ELSE
MOVE "1" TO ERR-FLG
END-IF.
*
* Read each detail's ordered quantity and update STOCK
*
PERFORM UNTIL SQLCODE NOT = 0 OR ERR-FLG NOT = 0
*
EXEC SQL FETCH DETAIL INTO :h-PRDNBR
                        ,:h-ORDQTY
END-EXEC
*
IF SQLCODE = 0 THEN
*
EXEC SQL SET CONNECTION RCHASMO2 END-EXEC
*
IF SQLCODE = 0 THEN
*
EXEC SQL UPDATE ORDENTR/STOCK
SET PRDQTA = PRDQTA + :h-ORDQTY
WHERE PRDNBR = :h-PRDNBR
END-EXEC
IF SQLCODE NOT = 0 THEN
MOVE "1" TO ERR-FLG
END-IF
*
ELSE
MOVE "1" TO ERR-FLG
END-IF
*
EXEC SQL SET CONNECTION RCHASMO3 END-EXEC
*
IF SQLCODE NOT = 0 THEN

```

```

        MOVE "1" TO ERR-FLG
        END-IF
*
        END-IF
        END-PERFORM.
*
        IF SQLCODE < 0 AND ERR-FLG = 0 THEN
            MOVE "1" TO ERR-FLG
        END-IF.
*
        IF ERR-FLG = 0 THEN
            EXEC SQL DELETE FROM ORDENTL/ORDERHDR
                WHERE ORHNBR = :h-ORHNBR
            END-EXEC
*
            IF SQLCODE NOT = 0 THEN
                MOVE "1" TO ERR-FLG
            END-IF
        END-IF.
*
        IF ERR-FLG = "0" THEN
            EXEC SQL COMMIT END-EXEC
        ELSE
            EXEC SQL ROLLBACK END-EXEC
        END-IF.
*
        GOBACK.

```

### 5.9.3 Inserting the detail rows

The following example is an excerpt of the Insert Detail program. This fragment of code shows only the statements that are relevant to a DRDA-2 connection and two-phase commitment control. The full implementation of this program (INSDET) can be found in *Stored Procedures and Triggers on DB2 Universal Database for iSeries, SG24-6503*, since this program activates a remote stored procedure.

This program inserts the detail order item in the ORDERDTL file at the local database and updates the quantity in the inventory by calling a remote stored procedure, which accesses the STOCK file at the remote system:

```

        This program excerpt (from program INSDET)
        =====
        shows the vital statements for
        -- DRDA-2 connection management and
        -- two-phase commitment control ...
        .....
        .....
        ROCHESTER is local database system
        ZURICH is remote database system
        .....
        .....
        .....
*
*****
* The following Two-Phase Commit for local and remote system *
* is only executed, if the conditions are met ... *
* The first COMMIT after one DUW therefore is done only after *
* the ordered quantity has been deducted from STOCK file on *
* the remote system and the first order record has been *

```

```

* inserted correctly in the ORDERDTL file on the local system.*
* Every item record for an order is committed, because      *
* of releasing the record lock on STOCK file.                *
* Note: SQL COMMIT in the program starts commitment control  *
* for the activation group automatically:                      *
*****
*
C/EXEC SQL
C+ COMMIT
C/END-EXEC
.....
.....
.....
*
*****
* -- Connection to the REMOTE database: --                    *
* At start of the program DRDA connection mgmt. establishes *
* connection automatically to the remote sys. as according to *
* the relational database specified in the compil. parameter *
* in command CRTSQLxxx RDB(...). Therefore this program      *
* is connected to remote database ZURICH already.              *
* For further remote re-connections SET CONNECTION is used:   *
*****
*
.....
.....
C/EXEC SQL
C+ SET CONNECTION ZURICH                                     After 1.connect
C/END-EXEC                                                  remote
.....
.....
.....
*
*****
* The CALL of the stored procedure (at the remote system)    *
* is prepared and executed.                                  *
* It updates the STOCK file, and searches for alternatives,  *
* if necessary.                                              *
*****
Code for stored procedure, see chapter "Stored Procedures".
.....
.....
.....
.....
*****
* -- Connection to the LOCAL database: --                    *
* At this point, connection to the local database is estab- *
* lished. For the first time in the execution of the program *
* the CONNECT statement has to be executed.                  *
* The connection to the local database then goes to dormant *
* state, after connecting to the remote DB (above) again.    *
* For further local re-connections SET CONNECTION is used:   *
*****
*
C          *IN51      IFEQ '0'                               1st connect
C/EXEC SQL                                     --      local
C+ CONNECT TO ROCHESTER
C/END-EXEC
C          MOVE '1'      *IN51                               After 1.connect
C          ELSE                                                local

```



```

C/EXEC SQL
C+ SET CONNECTION ROCHESTER
C/END-EXEC
C          END
*
*****
* An order detail record is inserted in the local database, *
* if referential integrity rules are not violated, i.e.      *
* the primary key of ORDERDTL file must be unique, and/or a *
* corresponding order number must exist in the ORDERHDR     *
* parent file. Otherwise an SQL error message is sent from *
* database management:                                     *
*****
*
C/EXEC SQL
C+ INSERT INTO ORDENTL/ORDERDTL (ORHNBR, PRDNBR, ORDQTY, ORDTOT)
C+ VALUES(:ORDNBR, :DPRDNR, :DQUANT, :DITTOT)
C/END-EXEC
.....
.....
.....
C          SQLCOD      IFEQ -530          RI Constraint
.....
.....
*
*****
* If ORDERHDR parent file does not have corresponding      *
* order number (RI rule violated),                          *
* update of order quantity in STOCK file on remote system *
* is rolled back by two-phase commitment control management: *
*****
*
C/EXEC SQL
C+ ROLLBACK
C/END-EXEC
.....
.....
.....
C          GOTO BEGIN
.....
.....
.....
*
*****
* If PF3 is pressed, order entry has finished. All        *
* connections are released in order to save on resources:  *
*****
*
C/EXEC SQL
C+ RELEASE ALL
C/END-EXEC
*
*****
* The following COMMIT statement activates previous RELEASE: *
*****
*
C/EXEC SQL
C+ COMMIT

```

C/END-EXEC  
.....  
.....  
.....

## 5.10 DRDA over TCP/IP

So far, we have dealt with either DRDA over SNA or with the DRDA implementation on the iSeries server in general. This section discusses the iSeries server implementation of DRDA over TCP/IP. The requirement for DRDA over TCP/IP stems from the explosive growth in usage of this protocol and the fact that many large accounts are already running TCP/IP or are moving all new applications to TCP/IP. The support for DRDA over TCP/IP on the iSeries server has been made available with OS/400 version V4R2M0. The implementation of DRDA over TCP/IP up to V4R5 supports DRDA level 1. This satisfies the UNIX or Windows NT client and DataPropagator needs. It is in V5R1 that the implementation of DRDA over TCP/IP supports DRDA level 2.

The DRDA application server is based on multiple connection-oriented server jobs running in the QSYSWRK subsystem. A DRDA background program (listener) listens for TCP connect requests on well-known DRDA port 446. The DRDA server jobs are defined by prestart job entries. Once the application requester connects to the listener at the AS, the listener issues a request to wake up a prestarted server job. The listener then passes the socket descriptor to the server job, and any further communication occurs directly between the client application and server job.

If you use the SNA implementation of DRDA, you need to configure the controller that governs the communication between the local and the remote system. Then, you need to refer to this controller in the device description parameter of the ADDRDBDIRE command. If you use the TCP/IP implementation of DRDA, you can refer to the IP address of the remote server on the Remote Location Name parameter of the ADDRDBDIRE command, and specify the port if it is other than the default DRDA port of 446. The iSeries server always uses the default port. This eliminates all of the complexity of configuring the communications between the two servers.

### 5.10.1 Configuring DRDA over TCP/IP

This section covers the configuration process for DRDA over TCP/IP between two iSeries servers. The system located in Rochester takes the role of the application server, and the system in Zurich accesses the database located on the Rochester machine as an application requester.

The configuration process for this simple scenario consists of these phases:

- ▶ **Setting up the application server:** This phase involves the following configuration activities:
  - a. Configuring TCP/IP on the AS system
  - b. Setting the attributes for the DDM server job
  - c. Starting the DDM server job
- ▶ **Setting up the application requester:** This phase involves the following configuration activities:
  - a. Configuring TCP/IP on the AR system
  - b. Adding the AS system to the Relational Database Directory
  - c. Defining the user profile under which the connect to the AS is being done

## Configuring TCP/IP on the application server

First you have to make sure that there is an appropriate host table entry for the local ROCHESTER machine. At this point, you may also want to add a host name for the AR machine located in Zurich. An alternative approach is to specify a remote domain name server in your TCP/IP configuration for automatic host name resolution.

In our example, we outline the steps required for defining the TCP/IP host table entry:

1. At the CL command line, enter the **GO CFGTCP** command.
2. The TCP/IP configuration menu is shown. Here, select option **10**.
3. The Work with TCP/IP Host Table Entries display is shown. Check that there is a valid entry for the local host.
4. Choose the **ADD** option, and enter the Internet address of the remote AR in the column named Internet Address (Figure 5-15).

```

                                Work with TCP/IP Host Table Entries
                                                System:  ROCHESTER
Type options, press Enter.
  1=Add  2=Change  4=Remove  5=Display  7=Rename

      Internet      Host
Opt  Address      Name
  1  10.10.10.2
  -  10.10.10.1   ROCHESTER
  -  127.0.0.1   LOOPBACK
                        LOCALHOST
  
```

Figure 5-15 Working with the host table entries

5. This invokes the Add TCP/IP Host Table Entry (ADDTCPHTE) command. On this command, you can define the name and the alias names of the remote host (Figure 5-16).

```

                                Add TCP/IP Host Table Entry (ADDTCPHTE)

Type choices, press Enter.

Internet address . . . . . > '10.10.10.2'
Host names:
  Name . . . . . ZURICH_____
_____
_____
_____
+ for more values _
Text 'description' . . . . . HOST TABLE ENTRY FOR AS/400 SYSTEM AT
ZURICH_____
  
```

Figure 5-16 Adding the host table entries

## Setting the attributes for the DDM server job

Before you start the server job on the AS system, you can change the job's attributes by using the Change DDM TCP/IP Attributes (CHGDDMTCPA) command. There are two attributes that can be changed with this command:

- ▶ **AUTOSTART**: Specifies whether to automatically start the DDM server when TCP/IP is started. This parameter takes effect the next time the STRTCP command is run.

- **PWDRQD**: Specifies whether client systems are required to have a password in addition to a user ID on incoming connection requests to this system as a server. This parameter takes effect on the next DRDA or DDM connect request over TCP/IP.

Now, follow these steps:

1. On the CL command line, enter the **CHGDDMTCPA** command, and press PF4 so you can see the current settings.
2. Change the DDM server job attributes to the values shown in Figure 5-17.

```

Change DDM TCP/IP Attributes (CHGDDMTCPA)

Type choices, press Enter.

Autostart server . . . . . AUTOSTART      *YES
Password required . . . . . PWDRQD       *YES

```

Figure 5-17 Changing the DDM server job attributes

**Note:** The value of the PWDRQD attribute has some implications for the Change Relational Database Directory Entry (CHGRDBDIRE) and Remove Relational Database Directory Entry (RMVRDBDIRE) commands. A bit in the \*LOCAL RDB directory entry is used to store if a password is required to access this iSeries server by an AR. An inquiry message CPA3E01 is issued if the local entry is changed to a non-local entry or if the \*LOCAL entry is deleted. The following text is associated with this message:

Removing the \*LOCAL directory entry may cause loss of configuration data.  
(C G)

We strongly recommend that you record the current setting before you proceed.

### Starting the DDM server job

Use the **STRTCPSVR SERVER(\*DDM)** command to start the DDM server job. Now you can find a new job, QRWTLSTN, running in the QSYSWRK subsystem. This is the listener job waiting for connect requests on port 446.

### Configuring TCP/IP on the application requester

Configuring TCP/IP on the application requester involves exactly the same steps as for the application server. Refer to “Configuring TCP/IP on the application server” on page 109.

### Adding the AS system to the relational database directory

Probably the most important step in DRDA over TCP/IP configuration is adding the relational database entry for the remote database to which you want to connect. The relational database entry defines the location of the remote server and the method of connection.

1. Use the **ADDRDBDIRE** command to add the RDB entry for the application server located in Rochester (Figure 5-18).

```

Add RDB Directory Entry (ADDRDBDIRE)

Type choices, press Enter.

Relational database . . . . . ROCHESTER_____
Remote location:
  Name or address . . . . . 10.10.10.1_____
_____
_____
Type . . . . . *IP_          *SNA, *IP
Text . . . . . RDB ENTRY FOR THE AS/400 SYSTEM IN RO
CHESTER

```

Figure 5-18 Adding the relational database entry

2. On the Relational Database parameter of the ADDRDBDIRE command, specify the name of the database on the remote server.
3. On the Remote Location parameter of the ADDRDBDIRE command, specify the *Internet address* of the remote application server. If you already specified the Internet address of the remote server in TCP/IP host table entry, you can use the host name in place of the Internet address. In our example, we specified ROCHESTER rather than 10.10.10.1. This allows you to have some flexibility if, for some reason, you change the Internet address of the remote server.
4. On the Type parameter of the ADDRDBDIRE command, specify the value **\*IP**. This signifies to the local system that you are using the TCP/IP implementation of DRDA to connect to the remote server. Note that **\*SNA** is the default setting for this parameter.
5. On the PORT parameter, specify the default value **\*DRDA**. Some servers, such as DB2 Universal Database, use a different port. You need to find out which port to specify from the documentation for the specific server product and set that port number.

### Defining a user profile for DRDA over the TCP/IP connection

You can use the Add Server Authentication Entry (ADDSVRAUTE) command to add authentication information for a given user under which the connect is being done. The user ID and password are associated with the user profile and remote application server. This information flows to the AS each time the AR issues a connect request.

1. Make sure that you have **\*SECADM** special authority, as well as **\*OBJMGT** and **\*USE** authorities, to the user profile to which the server authentication entry is being added.
2. Check whether the retain server security data (QRETSVRSEC) system value is set to 1. If the value is 0 (do not retain data), the password is not saved in the entry.
3. Type the **ADDSVRAUTE** command and press F4.
4. Add the authentication entry shown in Figure 5-19 and Figure 5-20.



## Interactive SQL example

Probably the easiest way to take advantage of a DRDA connection to a remote database is to use Interactive SQL. The following simple SQL session documents all major points to remember while running DRDA over TCP/IP:

1. Start Interactive SQL with the Start SQL (**STRSQL**) command. Make sure that the commitment control level you are running at is at least **\*CHG**. At the SQL prompt, press the F13 key, and then select option **1**. Change the Commitment Control Attribute to **\*CHG**. Return to the SQL session by pressing F3. Now you are ready to test the SQL statements shown in Figure 5-21.

```
Enter SQL Statements

Type SQL statement, press Enter.
Session was saved and started again.
STRSQL parameters were ignored.
Current connection is to relational database ZURICH. 1
> release all 2
RELEASE of all relational databases completed.
> commit 3
Commit completed.
> connect to rochester
Current connection is to relational database ROCHESTER. 4
> select * from ordapplib/customer
SELECT statement run complete.
> update ordapplib/customer set cuscrd = cuscrd * 1.1 5
where cusnbr = '99995'
1 rows updated in CUSTOMER in ORDAPPLIB.
> select * from ordapplib/customer
SELECT statement run complete.
> commit 6
Commit completed.
> call caseproc ('99995',0) 7
CALL statement complete.
> select * from ordapplib/customer
SELECT statement run complete.
> commit
Commit completed.
===>

Bottom
```

Figure 5-21 Using DRDA over TCP/IP with SQL

The following list explains the SQL statements that are numbered in Figure 5-21:

- 1 When you start your Interactive SQL session, you are connected, by default, to your local database.
- 2 The initial connection to the local system is protected by two-phase commit protocols. If a subsequent connection is made to a system that has only RUW capability, that connection is read-only. Therefore, you cannot perform any committable transactions, including automatically creating an SQL package for the Interactive SQL program, if the connection is to a non-iSeries server and this is the first time the connection is attempted. The solution to this is to drop the connection to the local database before you connect to the remote server. You may use the SQL statement **RELEASE ALL** to accomplish this task. When you execute this command, the resources held by any database in the system are released and any pending transaction is rolled back.

- 3 The COMMIT statement is required to move the connection from a released to an unconnected state. See the discussion in 5.3.2, “Connection states” on page 88, for more information.
- 4 After you establish the connection to the remote AS, make sure that the connection type is 1. Place the cursor on the connection message, and press F1. A message display with the information shown in Figure 5-22 appears.

```

Additional Message Information
Message ID . . . . . : SQL7971      Severity . . . . . : 00
Message Type . . . . : Information

Message . . . : Current connection is to relational database ROCHESTER.
Cause . . . . : The product identification is QSQ04020, the server class
name is QAS, and the user ID is JAREK. The connection method used is *DUW
The connection type is 1. A list of the connection types follows:
-- Type 1 indicates that committable updates can be performed and either
the connection uses an unprotected conversation, is a connection to an
application requester driver program using *RUW connection method, or is
local connection using *RUW connection method.
-- Type 2 indicates that the conversation is unprotected and no
committable updates can be performed.
-- Type 3 indicates that the conversation is protected and it is unknown
if committable updates can be performed.
-- Type 4 indicates that the conversation is unprotected and it is
unknown

More...
```

Figure 5-22 Connection type for DRDA over TCP/IP

- 5 Because this is the first \*RUW connection, committable updates can be performed.
  - 6 Changes to the remote database should be committed before the connection is dropped.
  - 7 One of the most powerful features of the DRDA architecture is the ability to run remote stored procedures. The stored procedure in this step performs exactly the same action as the update statement in 5. For a detailed discussion on DB2 UDB for iSeries stored procedure implementation, refer to *Stored Procedures and Triggers on DB2 Universal Database for iSeries*, SG24-6503.
2. To finish the SQL session, press F3. You may want to save your current session by using option 1.

### ILE C example

The way you code your application program, which accesses remote AS with DRDA over TCP/IP support, is similar to the procedure described for the Interactive SQL session. The following example code highlights the most important considerations:

1. Compile your program with an isolation level of \*CHG or above. If you have a connection to the remote AS at compile time, you may create an appropriate SQL package on the target system. The following Create C-embedded SQL (CRTSQLCI) command creates the program object on the local system and the SQL package at the remote server:

```

CRTSQLCI OBJ(ORDAPPLIB/SQLCNCT) SRCMBR(SQLCNCT) RDB(ROCHESTER)
OBJTYPE(*PGM) OUTPUT(*PRINT) RBCNMTH(*RUW)
```

2. The ILE C code is listed here:



```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <decimal.h>

EXEC SQL BEGIN DECLARE SECTION;
    char          Chr_CusNbr[ 5 ];
    char          Chr_CusNam[ 20 ];
    char          Chr_CusTel[ 15 ];
    char          Chr_CusFax[ 15 ];
    char          Chr_CusAdr[ 20 ];
    char          Chr_CusCty[ 20 ];
    char          Chr_CusZip[ 5 ];
    decimal( 11,2 ) Nmpd_CusCrd;
    decimal( 11,2 ) Nmpd_CusTot;
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;

void main()
{
    char Chr_Commit;

    printf( "Please enter the value for Customer Number      :\n" );
    gets( Chr_CusNbr );
    printf( "Please enter the value for Customer Name        :\n" );
    gets( Chr_CusNam );
    printf( "Please enter the value for Customer Tel         :\n" );
    gets( Chr_CusTel );
    printf( "Please enter the value for Customer Fax         :\n" );
    gets( Chr_CusFax );
    printf( "Please enter the value for Customer Address       :\n" );
    gets( Chr_CusAdr );
    printf( "Please enter the value for Customer City           :\n" );
    gets( Chr_CusCty );
    printf( "Please enter the value for Customer Zip             :\n" );
    gets( Chr_CusZip );
    printf( "Please enter the value for Customer Credit Limit :\n" );
    scanf( "%D(11,2)", &Nmpd_CusCrd );

    EXEC SQL
    release all; 1
    if ( sqlca.sqlcode != 0 )
    {
        printf( "Error occurred in the release of databases\n" );
        printf( "The SQLCODE is %d\n", sqlca.sqlcode );
        printf( "The Error Message :\n" );
        printf( "%s\n", sqlca.sqlerrmc );
        exit( -1 );
    }
    printf( "Released all Databases...\n" );
    EXEC SQL
    commit; 2
    if ( sqlca.sqlcode != 0 )
    {
        printf( "Error occurred in commit release of database\n" );
        printf( "The SQLCODE is %d\n", sqlca.sqlcode );
        printf( "The Error Message :\n" );
        printf( "%s\n", sqlca.sqlerrmc );
        exit( -1 );
    }
}

```

```

}

EXEC SQL
connect to ROCHESTER; 3
if ( sqlca.sqlcode != 0 )
{
    printf( "Error occured in Connecting to Database\n" );
    printf( "The SQLCODE is %d\n", sqlca.sqlcode );
    printf( "The Error Message :\n" );
    printf( "%s\n", sqlca.sqlerrmc );
    exit( -1 );
}
printf( "Successfully connected to ROCHESTER..\n" );

EXEC SQL
commit;
if ( sqlca.sqlcode != 0 )
{
    printf( "Error occured in commit of connection\n" );
    printf( "The SQLCODE is %d\n", sqlca.sqlcode );
    printf( "The Error Message :\n" );
    printf( "%s\n", sqlca.sqlerrmc );
    exit( -1 );
}
printf( "Committed the Connection...\n" );

EXEC SQL
call 4
ordapplib/inscst(
:Chr_CusNbr,
:Chr_CusNam,
:Chr_CusTel,
:Chr_CusFax,
:Chr_CusAdr,
:Chr_CusCty,
:Chr_CusZip,
:Nmpd_CusCrd
);
if ( sqlca.sqlcode != 0 )
{
    printf( "Error occured in calling stored procedure\n" );
    printf( "The SQLCODE is %d\n", sqlca.sqlcode );
    printf( "The Error Message :\n" );
    printf( "%s\n", sqlca.sqlerrmc );
    EXEC SQL
    rollback;
    if ( sqlca.sqlcode != 0 )
    {
        printf( "Error occured in rollback\n" );
        printf( "The SQLCODE is %d\n", sqlca.sqlcode );
        printf( "The Error Message :\n" );
        printf( "%s\n", sqlca.sqlerrmc );
        exit( -1 );
    }
    printf( "Rollback Complete...\n" );
}
else
{
    EXEC SQL
    commit;
}

```

```

        if ( sqlca.sqlcode != 0 )
        {
            printf( "Error occured in commit\n" );
            printf( "The SQLCODE is %d\n", sqlca.sqlcode );
            printf( "The Error Message :\n" );
            printf( "%s\n", sqlca.sqlerrmc );
            exit( -1 );
        }
        printf( "Commit Complete...\n" );
    }
    exit(0);
}

```

#### Notes:

- ▶ Disconnect from the local database since it is protected by a two-phase commit. If you have a connection to a \*DUW capable database, all subsequent connections to \*RUW capable databases are read-only.
- ▶ The COMMIT statement is needed to change the local database status from released to unconnected.
- ▶ Connect to the remote AS using DRDA over TCP/IP. The connection method is \*RUW, and committable updates are permitted.
- ▶ Call the remote stored procedure. This procedure inserts a new record into the customer file.

### 5.10.3 Troubleshooting DRDA over TCP/IP

DRDA over TCP/IP works fine until someone changes something. While handling the problems, you need to be single-minded about isolating them. First, ask yourself these simple questions:

- ▶ Is the server job running on the application server?

Make sure that QRWTLSTN is running in the QSYSWRK subsystem. Start the **NETSTAT** command, and select option 3 to check whether the listener job listens on the well-known port 446.

- ▶ Are you authorized to use the connection?

Does the server require a password along with the user ID on the connection request? If a password is needed, add your profile by using the **ADDSVRAUTE** command to the server authorization entry list.

- ▶ Does your connection permit committable updates?

Use Interactive SQL to check the connection type to the remote application server. Refer to “Interactive SQL example” on page 113, for a detailed discussion on this subject.

If you went over this simple checklist and still encounter problems with your DRDA over TCP/IP connection, it is time to take a more systematic approach:

1. On the AS system, find the prestart job that is servicing your requests. When you start the listener job with the STRTCPSVR command, one or more prestart jobs are started in the QSYSWRK subsystem. The name of this prestart job is QRWTSRVR, and the user profile under which the job runs initially is QUSER. When your request to start a connection is accepted by the prestart job, it swaps QUSER to your user profile. The easiest way to identify the fully-qualified name for the prestart job servicing your requests is to look into the history log. There should be a log entry pertaining to your user ID (Figure 5-23).

```

Display History Log Contents

Job 034557/QUSER/QRWTSRVR started on 12/17/01 at 11:44:41 in subsystem QSYSWRK
Job 034558/QUSER/QRWTSRVR started on 12/17/01 at 11:44:41 in subsystem QSYSWRK
DDM job 034509/QUSER/QRWTSRVR servicing user JAREK on 12/17/01 at 11:44:42.

```

Figure 5-23 Identifying a server job servicing your profile

An alternative method is to use the **WRKACTJOB** command.

2. Once you identify your prestart job, you can start a service job with the **STRSRVJOB** command (Figure 5-24).

```

Start Service Job (STRSRVJOB)

Type choices, press Enter.

Job name . . . . . QRWTSRVR Name
User . . . . . QUSER Name
Number . . . . . 034509 000000-999999

```

Figure 5-24 Starting a service job

3. Enter the **STRDBG** command, and press F4. Change the Update production Files parameter to **\*YES** (Figure 5-25).

```

Start Debug (STRDBG)

Type choices, press Enter.

Program . . . . . PGM *NONE
Library . . . . .
+ for more values

Default program . . . . . DFTPGM *PGM
Maximum trace statements . . . . . MAXTRC 200
Trace full . . . . . TRCFULL *STOPTRC
Update production files . . . . . UPDPROD *YES

```

Figure 5-25 Starting debug for the prestart job

4. If the connection to the AS is still active, you can check the AR-AS interaction by looking at the job log of the prestarted job that is servicing your requests. Use the following command on the AS to display the job log:

```
DSPJOBLOG JOB(034509/QUSER/QRWTSRVR)
```

```

                                Display All Messages
                                System: ROCHESTER
Job . . . : QRWTSRVR  User . . . : QUSER          Number . . . : 034509

Job 034557/QUSER/QRWTSRVR started on 12/17/01 at 11:44:41 in subsystem
QSYSWRK in QSYS. Job entered system on 12/17/01 at 11:44:41.
Target job assigned to handle DDM connection started by source system ove
TCP/IP.
ACGDTA for 034557/QUSER/QRWTSRVR not journaled; reason 1.
Local relational database accessed by ZURICH.
Number of parameters on CALL not valid for procedure INSCST in ORDAPPLIB.

```

Figure 5-26 Job log of the prestart job

Looking at the job log entries (Figure 5-26), you can now see that you were trying to call the INSCST stored procedure with an incorrect number of parameters.

5. Close your connection to the AS system. Since the prestart job was being serviced, the job log associated with this job is saved in a spooled file. This spooled file is stored with your user ID.

**Note:** The job log is also saved when the system detects that a serious error occurred in processing the request that ended the connection.

6. Use the Work with Spooled File (**WRKSPLF**) command to display the content of the spooled file (Figure 5-27).

```

5769SS1 V4R2M0 980228          Display Job Log          ROCHESTER 12/17/01 14:28:56
Page 1
Job name . . . . . : QRWTSRVR      User . . . . . : QUSER          Number . . . . . :
034558
Job description . . . . . : QUSER      Library . . . . . : QGPL
MSGID   TYPE          SEV   DATE       TIME      FROM PGM   LIBRARY    INST    TO PGM
LIBRARY  INST
CPF1124 Information      00   12/17/01  11:44:41  QWTP1IPP   QSYS       0599      *EXT
*N
                                Message . . . . . : Job 034558/QUSER/QRWTSRVR started on 12/17/01 at 11:44:41
                                in subsystem QSYSWRK in QSYS

SQL0440 Diagnostic      30   12/17/01  14:26:56  QSQCUTE   QSYS       1B9A      QSQCUTE   QSYS
1B9A
                                Message . . . . . : Number of parameters on CALL not valid for procedure
                                INSCST in ORDAPPLIB.
                                Cause . . . . . : The number of parameters specified on a CALL statement is
                                not the same as the number of parameters declared for procedure INSCST in
                                ORDAPPLIB. Recovery . . . : Specify the same number of parameters on the
                                CALL as on the procedure definition. Try the request again.

```

Figure 5-27 Spooled file content

7. Stop debugging with the End Debug (**ENDDBG**) command, and stop the service job with the End Server Job (**ENDSRVJOB**) command.

## 5.11 DB2 Connect access to an iSeries server via TCP/IP

Since OS/400 V4R2, it is possible to connect to an iSeries server from DB2 Connect on a Windows machine using TCP/IP. The following example employs OS/400 V4R5 and the DB2 UDB for Windows NT Version 7.1.

### 5.11.1 On the iSeries server

The following process lists the necessary steps to be performed on the iSeries server:

1. Verify that the TCP/IP stack is working correctly. To do this, obtain the IP address of the iSeries server or hostname, and ping the iSeries server from the DB2 Connect machine. To find the IP address, go to the Configure TCP/IP menu. Enter **CFGTCP**, and choose **Work with TCP/IP interface**. The IP address should be displayed as shown in Figure 5-28.

```
Work with TCP/IP Interfaces                                System:  AS23
Type options, press Enter.
  1=Add  2=Change  4=Remove  5=Display  9=Start  10=End

  Internet      Subnet      Line      Line
Opt Address      Mask        Description Type
-----
      10.10.10.10  255.255.255.0  TRNLINE  *TRLAN

Bottom

F3=Exit      F5=Refresh  F6=Print list  F11=Display interface status
F12=Cancel   F17=Top     F18=Bottom
```

Figure 5-28 Work with TCP/IP Interfaces

2. To find the hostname, go to the Configure TCP/IP menu, and choose **Work with TCP/IP host table entries**. You should find the hostname that has been assigned to the IP address.
3. You need a relational database (RDB) name for the iSeries server. If it has already been created, you can display it by using the **DSPRDBDIRE** command. The RDB with a location of \*LOCAL is the one you need, as shown in Figure 5-29. If it has not been created, use the **ADDRDBDIRE** command to add the RDB entry. For example, the following command would add an RDB entry named DALLASDB:

```
ADDRDBDIRE RDB(DALLASDB) RMTLOCNAME(*LOCAL)
```

```

Display Relational Database Directory Entries

Position to . . . . .

Type options, press Enter.
  5=Display details  6=Print details

Option  Relational      Remote
        Database      Location              Text
        AS23           *LOCAL               DB entry for local AS23
        AS24           10.10.10.20         RBD Entry for AS24

F3=Exit  F5=Refresh  F6=Print list  F12=Cancel
(C) COPYRIGHT IBM CORP. 1980, 2000.
Bottom

```

Figure 5-29 Display Relational Database Directory Entries

4. You must create a collection called NULLID. The reason for this is that the utilities shipped with DB2 Connect and DB2 UDB store their packages in the NULLID collection. Since it does not exist by default in the iSeries server, you must create it using the following command:

```
CRTLIB LIB(NULLID)
```

5. Products that support DRDA automatically perform any necessary code page conversions at the receiving system. For this to happen, both systems need a translation table from their code page to the partner code page. The default Coded Character Set Identifier (CCSID) on the iSeries server is 65535. Since DB2 Connect does not have a translation table for this code page, you need to change the individual user profiles to contain a page. You need to change the individual user profiles to contain a CCSID that can be converted properly by DB2 Connect. For US English, this is 037. For other languages, see *DB2 Connect Personal Edition Quick Beginning*, GC09-2967. The following command changes the CCSID for an individual user profile to 037:

```
CHGUSRPRF userid CCSID(037)
```

6. Verify that you are using the default port 446 for DRDA service. To do this, go to the Configure TCP/IP menu (CFGTCP), select **Configure Related Tables**, and then select **Work with service table entries**. Verify that the DRDA service is set for port 446, as shown in Figure 5-30.

```

Work with Service Table Entries
System: AS23

Type options, press Enter.
  1=Add  4=Remove  5=Display

Opt  Service                Port  Protocol

     drda                    446  udp
     echo                     7   tcp
     echo                     7   udp
     exec                     512  tcp
     finger                    79  tcp
     finger                    79  udp
     ftp-control               21  tcp
     ftp-control               21  udp
     ftp-data                  20  tcp
     ftp-data                  20  udp
     gopher                    70  tcp

More...

Parameters for options 1 and 4 or command
===>
F3=Exit  F4=Prompt  F5=Refresh  F6=Print list  F9=Retrieve  F12=Cancel
F17=Top  F18=Bottom

```

Figure 5-30 Work with Service Table Entries

7. The Distributed Data Management (DDM) job must be started for DRDA to work. If you want the DDM job to be automatically started whenever TCP/IP is started, you can change the attributes of the DDM job using the **CHGDDMTCPA** command and set the Autostart server parameter to **\*YES**.
8. If you choose not to autostart the server, issue the following command to start the DDM server job:
 

```
STRTCPSVR(*DDM)
```
9. Make sure you have user IDs defined on the iSeries server for the users that will be connecting.

### 5.11.2 On the workstation

The following steps are required on DB2 UDB:

1. Launch Client Configuration Assistant (**db2cca** from the command prompt).
2. Click the **Add** button to add a new data source.
3. On the **Source** tab, choose **Manual configuration**, and click **Next**.
4. On the **Protocol** tab, choose **TCP/IP** for protocol, and select the item **The database physical residence on a host or AS/400**. Then, select the option **Connect directly to the server**, as shown in Figure 5-31. Click **Next**.



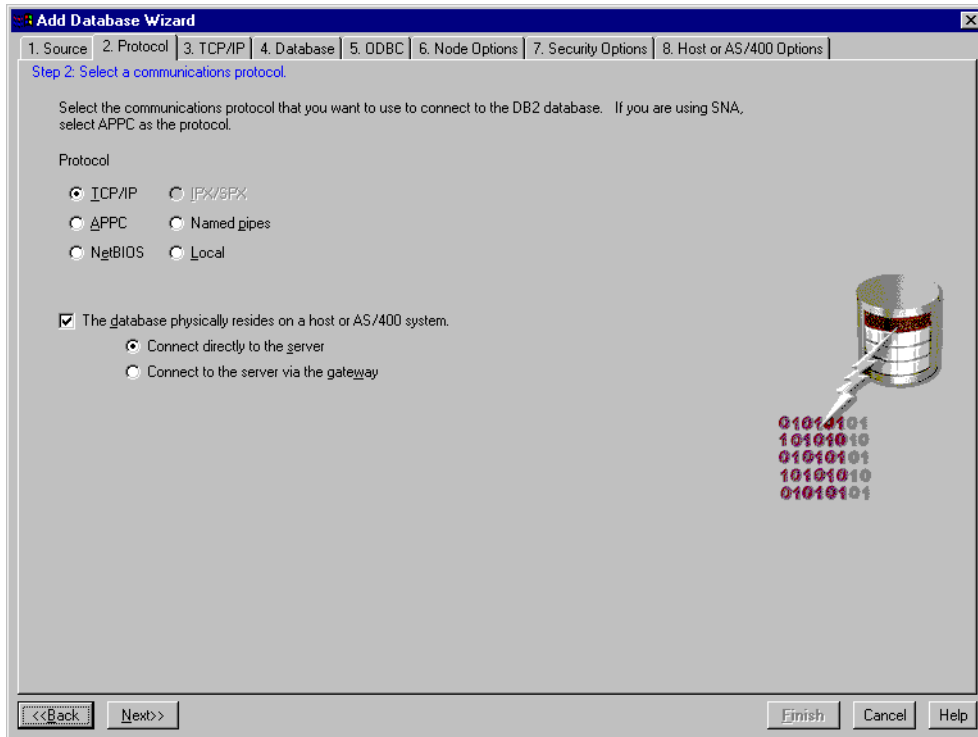


Figure 5-31 Protocol tab of the workstation configuration

5. On the **TCP/IP** tab, fill in the host name of the iSeries server. The port number should be 446. Click **Next**.
6. On the **Database** tab, fill in the relational database name, and click **Next**.
7. If you plan to use the ODBC applications, click the **ODBC** tab and select **Register this database for ODBC as a system data source**.
8. Click **Finish**.
9. Click **Test Connection** to verify that the connection works. You are prompted for an iSeries server user ID and password, as shown in Figure 5-32.

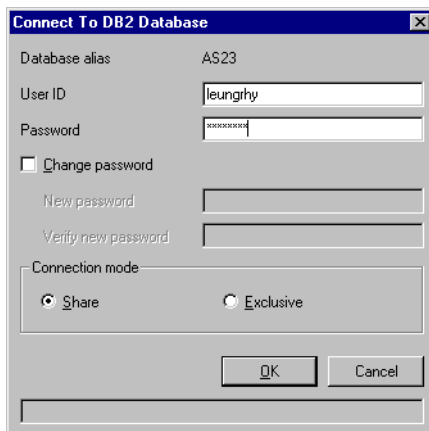


Figure 5-32 Prompt for iSeries server user ID and password

10. Enter your user ID and password, and then click **OK**. If the connection test passed, a successful message box appears, as shown in Figure 5-33.



Figure 5-33 Message box for successful connection

### 5.11.3 Consideration

Once you complete the configuration in the previous section, you should be able to access iSeries server data. However, you may notice that there are some areas that work differently in the iSeries server and other platforms. These areas are discussed in the following sections.

#### Administrative interface

In the current implementation, you cannot perform administrative functions for the iSeries server database through the UDB control center. The best tool for the iSeries server database is Operations Navigator. The differences in the administrative interface are due to the variation of administrative requirements and the operation of the underlying operating system. Several administrative functions are not available by DB2 Universal Database for iSeries because the database manager and operating system automatically handle the tasks. For example, DB2 Universal Database for iSeries doesn't provide a RUNSTATS utility for optimizer statistics because its database manager keeps these statistics current at all times. Likewise, there is no concept of table spaces in DB2 Universal Database for iSeries. DB2 Universal Database for iSeries does not support the notion of independent, isolated databases on the iSeries server. Instead, DB2 Universal Database for iSeries is implemented as a single system-wide database.

#### Journaling in DB2 Universal Database for iSeries

DB2 Universal Database for iSeries is so reliable that database administrators may not journal all their tables. However, if you are connecting to an iSeries server database through DB2 Connect, tables must be journaled before the database can be accessed for update. Otherwise, you only have read-only access to the table. If you attempt to update the table without journaling, you would see an error message such as this example:

```
----- Command entered -----
insert into result values ('Insert','Insert from UDB/NT command center');
-----
DB21034E The command was processed as an SQL statement because it was not a
valid Command Line Processor command. During SQL processing it returned:
SQL7008N REXX variable "RESULT " contains inconsistent data.
SQLSTATE=55019
```



## DB2 Import and Export utilities

This chapter covers the following topics:

- ▶ Explains the CPYFRMIMPF command (Import utility)
- ▶ Explains the CPYTOIMPF command (Export utility)
- ▶ Explains how to use the Import and Export utilities from DB2 UDB V7.2

## 6.1 Introduction

A data loader utility enables the loading of data exported from other database servers into DB2 UDB for iSeries.

Two commands in OS/400 are available for users to import (load), and export (unload) data to and from the iSeries server:

- ▶ **Copy From Import File (CPYFRMIMPF):** Loads the imported data into the DB2 UDB for iSeries table
- ▶ **Copy To Import File (CPYTOIMPF):** Prepares the DB2 UDB for iSeries table data for export from the iSeries server

## 6.2 DB2 UDB for iSeries Import utility

Database tables from heterogenous databases, such as Oracle, Sybase, Microsoft SQL Server, etc., can be ported to DB2 UDB for iSeries tables using this utility.

### 6.2.1 CPYFRMIMPF

The Copy From Import File (CPYFRMIMPF) command is used to load data to DB2 UDB for iSeries after the file is copied to a source file (FROMFILE) and then into a DB2 UDB for iSeries table (TOFILE). This is shown in Figure 6-1.

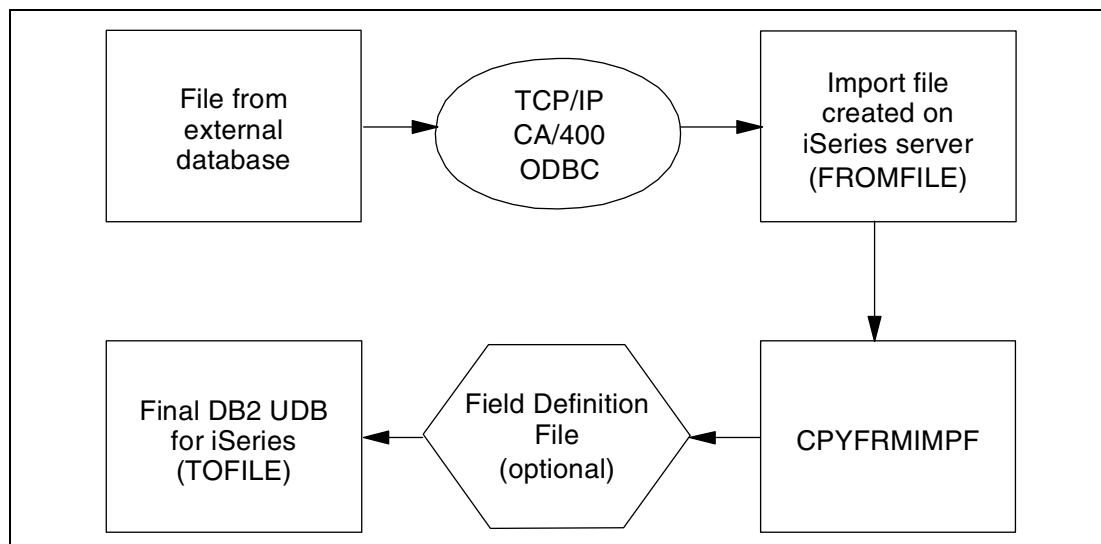


Figure 6-1 Data load flow

The following steps summarize a data load from a database table:

1. Create an import file for the data that will be copied to DB2 UDB for iSeries. The format for this data can be in delimited format or fixed format.
2. Send the data to the import file (typically with FTP or Client Access).
3. Create a DB2 UDB for iSeries externally described database file(table) or DDM file that contains the resulting data (target file) of the import file.
4. Create the Field Definition File if a \*FIXED data format is used.

5. Use the **CPYFRMIMPF** command to copy (translate or parse the records) from the import file to the target file.

### The source file (FROMFILE)

The source file (FROMFILE) can be any one of the following file types:

- ▶ Stream file
- ▶ DDM file
- ▶ Tape file
- ▶ Source physical file
- ▶ Distributed physical file
- ▶ Program described physical file
- ▶ Single format logical file
- ▶ Externally described physical file with one field (of non-numeric data type)

**Note:** If an externally described physical file has one field, the data type must be CHARACTER, IGC OPEN, IGC EITHER, IGC ONLY, GRAPHIC, or variable length.

The file can be copied or imported to the iSeries server using several methods, including:

- ▶ TCP/IP file transfer (text transfer)
- ▶ CA/400 support (file transfer, ODBC)
- ▶ Copy From Tape (CPYFRMTAP) command

Sending the data into the import file causes the necessary ASCII to EBCDIC data conversions to occur.

### The target file (TOFILE)

The source file is copied to the database target file, also referred to as the TOFILE. The target file can be any one of the following file types:

- ▶ Source file
- ▶ DDM file
- ▶ Distributed physical file
- ▶ Program described file
- ▶ Externally described physical file

### Data format

The data contained in the imported file can be in either the delimiter format or the fixed format:

- ▶ **Character delimited:** A delimiter format import file has a series of characters (delimiters) that are used to define where fields begin and end. The parameters of the command defines what characters are used for delimiters.
- ▶ **Fixed format:** A fixed format import file uses the user-defined Field Definition File (FDF) that defines the format of the import file. The Field Definition File is used to define where fields begin, end, and are null. The record format of import file (DTAFMT) parameter determines if the source file is delimited (\*DLM) or fixed (\*FIXED).

### Field definition file

The field definition file to describe fixed formatted files must use the format shown in Table 6-1.

Table 6-1 Field definition format

Field name	Starting position	Ending position	Null character position
Field1	1	12	13
Field2	14	24	0
Field3	25	55	56
*END			

In reference to Table 6-1, note the following statements:

- ▶ Field name is the name of the field in the TOFILE. FDF is case sensitive.
- ▶ The Starting position is the byte in the FROMFILE from where the data is copied.
- ▶ The Ending position is the byte in the FROMFILE from where the data is copied.
- ▶ The Null character position is the byte in the FROMFILE that indicates if the field is null. A value of “Y” means the field is null. A value of “N” means the field is not null. If this value is “0”, no null character is provided.
- ▶ \*END is the indicator for the end of the field definition file and must be included.

### Delimited format import file

The import file’s data is interpreted by the following characters and data types for a delimited format import file:

#### ▶ Blanks

- All leading and trailing blanks are discarded for character fields unless enclosed by string delimiters.
- A field of all blanks is interpreted as a null field for character data.
- Blanks cannot be embedded within a numeric field.
- A blank cannot be selected as a delimiter.

#### ▶ Null fields

A null field is defined as:

- Two adjacent field delimiters (no data in between)
- A field delimiter followed by a record delimiter (no data in between), an empty string
- A field of all blanks

If the field is null, the following statement is true:

*If the output field is not nullable and the import is a null field, the record is not copied, and an error is signaled.*

#### ▶ Delimiters

- A delimiter cannot be a blank.
- A string delimiter cannot be the same as a field delimiter, record delimiter, date separator, or time separator.
- A string delimiter can enclose all non-numeric fields (character, date, time, and so on). The string delimiter character should not be contained within the character string.
- A field and record delimiter can be the same character.
- The defaults for delimiters are as follows:

- **String:** Double quote (")
  - **Field:** Comma (,)
  - **Decimal point:** Period (.)
  - **Record:** End of record (\*EOR)
- If the data type of the *from* parameter is CHARACTER, OPEN, EITHER, or ONLY, all double-byte data must be contained within string delimiters or shift characters (for OPEN, EITHER, or ONLY data types).
- ▶ **Numeric field**
- Numeric fields can be imported in decimal or exponential form.
  - Data to the right of the decimal point may be truncated depending on the output data format.
  - Decimal points are either a period or a comma (command option).
  - Signed numeric fields are supported, + or -.
- ▶ **Character or Varcharacter fields**
- Fields too large to fit in the output fields are truncated (right), and a diagnostic message is sent.
  - An empty string is defined as two string delimiters with no data between them.
  - For a character to be recognized as a starting string delimiter, it must be the first non-blank character in the field. For example, 'abc' with ' as the delimiter is the same as abc for input.
  - Data after an ending string delimiter and before a field or record delimiter is discarded.
- ▶ **IGC or VarIGC fields**
- The data from the FROMFILE is copied to the TOFILE, and if any invalid data is received, a mapping error is generated.
  - Data located between the Shift Out and Shift In characters is treated as double byte data and is not parsed for delimiters. The Shift characters, in this case, become “string delimiters”.
- ▶ **Graphic, VarGraphic fields**
- The data from the FROMFILE is copied to the TOFILE.
- ▶ **CCSIDs (coded character set identifiers)**
- The data from the FROMFILE is read into a buffer by the CCSID of the FROMFILE. The data in the buffer is checked and written to the TOFILE. The CCSID of the open TOFILE is set to the value of the FROMFILE, unless a TOFILE CCSID is used. If a TOFILE CCSID is used, the data is converted to that CCSID. If the FROMFILE is a tape file, and the FROMCCSID(\*FILE) parameter is specified, the job CCSID is used, or the FROMFILE CCSID is requested by the user.
  - The character data (delimiters) passed in on the command is converted to the CCSID of the FROMFILE. This allows the character data of the FROMFILE and command parameters to be compatible.
- ▶ **Date field**
- All date formats supported by the iSeries server can be imported, including: \*ISO, \*USA, \*EUR, \*JIS, \*MDY, \*DMY,\*YMD, \*JUL, and \*YYMD.
  - A date field can be copied to a timestamp field.

- ▶ **Time field**
  - All time formats supported by the iSeries server can be imported, including: \*ISO, \*USA, \*EUR, \*JIS, and \*HMS.
  - A time field can be copied to a timestamp field.
- ▶ **Date and time separators**

All valid separators are supported for date and time fields.
- ▶ **Timestamp field**

Timestamp import fields must be 26 bytes. The import ensures that periods exist in the time portion and a dash exists between the date and time portions of the timestamp.
- ▶ **Number of fields mismatch**

If the FROMFILE or TOFILE do not have the same number of fields, the data is either truncated to the smaller TOFILE size, or the extra TOFILE fields receives a null value. If the fields are not null capable, an error message is issued.

## 6.2.2 Data load example (file definition file)

A source file, IMPF\_TEST, was created using the Create Physical File (CRTPF) command specifying a record length of 258 bytes. The customer data was then transferred to the iSeries server source file using FTP. A sample of the data is shown in Figure 6-2.

```

Display Physical File Member
File . . . . . : IMPF_TEST      Library . . . . . : TPSTAR
Member . . . . . : IMPF_TEST   Record . . . . . : 1
Control . . . . .           Column . . . . . : 1
Find . . . . .
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
1      ,"Customer#000000001    ",15 Monroe Ave, Chicago IL 60601
2      ,"Customer#000000002    ",Stewartville MN 55976
3      ,"Customer#000000003    ",389 Dexter Pl, Fargo ND
4      ,"Customer#000000004    ",10 N Main, Wausau WI
5      ,"Customer#000000005    ",Bailey Bldg, Bedford Falls
6      ,"Customer#000000006    ",101 Superior St, Duluth MN
7      ,"Customer#000000007    ",1921 N 5th St, St Louis MO
8      ,"Customer#000000008    ",32891 Park Ave, New York NY
9      ,"Customer#000000009    ",1032 S Broadway, Littleton CO
10     ,"Customer#000000010    ",5672 Cobb Pkwy, Bldg 3, Atlanta GA
11     ,"Customer#000000011    ",8192 River Rd, Aurora IL
12     ,"Customer#000000012    ",County Rd 9, Pine Island MN
13     ,"Customer#000000013    ",25th and Main, Appleton WI
14     ,"Customer#000000014    ",2342 Center St, Earlville IL
15     ,"Customer#000000015    ",444 Michigan Ave, Chicago

```

Figure 6-2 Customer data sample

A target file, or TOFILE, was created using Data Definition Specification (DDS) called CUST\_IMPF. At this time, the CPYFRMIMPF command can be used to format the delimited file as shown in Figure 6-3.



```

Copy From Import File (CPYFRMIMPF)

Type choices, press Enter.

From stream file . . . . .

From file:
  File . . . . . IMPF_TEST      Name
  Library . . . . . *LIBL      Name, *LIBL, *CURLIB
  Member . . . . . *FIRST     Name, *FIRST
To data base file:
  File . . . . . CUST_IMPFB    Name
  Library . . . . . *LIBL      Name, *LIBL, *CURLIB
  Member . . . . . *FIRST     Name, *FIRST
Replace or add records . . . . *ADD      *ADD, *REPLACE, *UPDADD
Stream file record length . . *TOFILE  Number, *TOFILE
From CCSID . . . . . *FILE    1-65533, *FILE
Record delimiter . . . . . *EOR   Character value, *ALL...
Record format of import file . *DLM     *DLM, *FIXED
String delimiter . . . . . ' '    Character value, *NONE
                                          More...

```

Figure 6-3 CPYFRMIMPF example

A fixed format using a field definition file (FDF) can also be used to convert the data. The example in Figure 6-4 of the FDF, CUST.FDF, was created in the Screen Edit Utility (SEU) as a TEXT file.

```

LEVEL 0 SCREEN CHECK YOUR LEVEL This is screen.Columns . . . : 1 71      Browse
V2KEA45/QTXTSRC
SEU==>                                          CUST.FDF
FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
***** Beginning of data *****
001.00 CUSTKEY          1      12    0
002.00 CUSTOMER        14      40    0
003.00 ADDRESS          42      83    0
004.00 PHONE           85     101    0
005.00 MKTSEGMENT     103     114    0
006.00 COUNTRY        116     142    0
007.00 CONTINENT      144     170    0
008.00 REGION         172     198    0
009.00 TERRITORY      200     226    0
010.00 SALES00001     228     254    0
011.00 DUMMYKEY       256     258    0
012.00 *END
***** End of data *****

```

Figure 6-4 Field definition file

The CPYFRMIMPF command for the \*FIXED format is shown in Figure 6-5 and Figure 6-6.

```

Copy From Import File (CPYFRMIMPF)

Type choices, press Enter.

From stream file . . . . .

From file:
  File . . . . . > IMPF_TEST      Name
    Library . . . . .      *LIBL      Name, *LIBL, *CURLIB
    Member . . . . .      *FIRST      Name, *FIRST
To data base file:
  File . . . . . > CUST_IMP      Name
    Library . . . . .      *LIBL      Name, *LIBL, *CURLIB
    Member . . . . .      *FIRST      Name, *FIRST
Replace or add records . . . . . *ADD      *ADD, *REPLACE, *UPDADD
Stream file record length . . . *TOFILE      Number, *TOFILE
From CCSID . . . . .      *FILE      1-65533, *FILE
Record delimiter . . . . . > *ALL      Character value, *ALL...
Record format of import file . . > *FIXED      *DLM, *FIXED
String delimiter . . . . .      ''      Character value, *NONE
                                                    More...

```

Figure 6-5 CPYFRMIMPF Command (Part 1 of 3)

```

Copy From Import File (CPYFRMIMPF)

Type choices, press Enter.

Remove leading blanks . . . . . *LEADING      *LEADING, *NONE
Field delimiter . . . . .      ','      Character value, *TAB
Field definition file:
  File . . . . . > QTXSRC      Name
    Library . . . . . > TPSTAR      Name, *LIBL, *CURLIB
    Member . . . . . > CUST_FDF      Name, *FIRST
Decimal point . . . . .      *PERIOD      *PERIOD, *COMMA
Date format . . . . .      *ISO      *ISO, *USA, *EUR, *JIS...
Date separator . . . . .      '/'      /, -, ., ,, *BLANK
Time format . . . . .      *ISO      *ISO, *USA, *EUR, *JIS, *HMS
Time separator . . . . .      ':'      :, ,, *BLANK
Copy from record number:
  Copy from record number . . . *FIRST      Number, *FIRST
  Number of records to copy . . *END      Number, *END
Errors allowed . . . . .      *NOMAX      Number, *NOMAX
                                                    More...

```

Figure 6-6 CPYFRMIMPF command (Part 2 of 3)

```

Copy From Import File (CPYFRMIMPF)

Type choices, press Enter.

Error record file:
  File . . . . . *NONE          Name, *NONE
  Library . . . . .           Name, *LIBL, *CURLIB
  Member . . . . .           Name, *FIRST
  Replace or add records . . . . *ADD      *ADD, *REPLACE
  Replace null values . . . . . *NO      *NO, *FLDDFT

                                                                    Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys

```

Figure 6-7 CPYFRMIMPF Command (Part 3 of 3)

If a field in the source file is not included in the target file, omit the field in the FDF file.

Enhancements have been made to the CPYFRMIMPF Import utility by adding the following parameters:

- ▶ **Remove leading blanks (RMVBLANK)**
  - If \*LEADING is specified along with STRDLM(\*NONE), then DB2 UDB for iSeries strips leading blanks from a character string before placing the resulting string in the specified character column.
  - With \*NONE, all leading blanks are included in the result string that is copied into the specified target character column.
- ▶ **Replace null values (RPLNULLVAL)**
  - When \*FLDFT is specified, if the data being imported (for example, blanks in a numeric field) causes DB2 UDB for iSeries to place a null value in a target column that does not allow nulls, then DB2 UDB for iSeries will assign the default value to the target column instead.
  - When the default value \*NO is specified, no replacement of null values is performed.

### 6.2.3 Data load example (Data Definition Language)

The Data Definition Language (DDL) source file STAFF.ddl and the Database extract file STAFFA.csv in comma-separated variable (CSV) format reside on the source system. The database extract file has to be exported to DB2 UDB for iSeries on the target iSeries server AS23. A TCP/IP connection exists between the two systems.

To transfer these two files to the iSeries server, follow these steps:

1. FTP the ddl file and csv file to the iSeries server as shown here:

```
C:\>ftp as23 1
Connected to AS23.
220-QTCP at rchasm23.rchland.ibm.com.
220 Connection will close if idle more than 5 minutes.
User (AS23:(none)): vijay 2
331 Enter password.
Password: 3
230 VIJAY logged on.
ftp> put c:\vijay\staff.ddl vijay/sqlsrc.staff 4
200 PORT subcommand request successful.
150 Sending file to member STAFF in file SQLSRC in library VIJAY.
250 File transfer completed successfully.
ftp: 317 bytes sent in 0.00Seconds 317000.00Kbytes/sec.
ftp> put c:\vijay\staffa.csv vijay/staffa.staffa 5
200 PORT subcommand request successful.
150 Sending file to member STAFFA in file STAFFA in library VIJAY.
250 File transfer completed successfully.
ftp: 2205 bytes sent in 0.01Seconds 220.50Kbytes/sec.
ftp>quit 6
```

#### Notes:

- 1 From a command line, type **FTP** to the iSeries server AS23.
- 2 Enter your user ID and press Enter.
- 3 Type your password and press Enter.
- 4 Type the **PUT** sub-command to copy the *staff.ddl* file in the *vijay* directory to member *STAFF* in source physical file *SQLSRC* in the library *VIJAY*. Note the use of the forward slash (/) and the period (.) in the target file name (library/file.member) format.
- 5 Type the **PUT** sub-command to copy the extracted database file *staffa.csv* in the *vijay* directory to the single field physical file member *STAFFA* in the physical file *STAFFA* in the library *VIJAY*. Note the use of the forward slash (/) and the period (.) in the target file name (library/file.member) format.
- 6 Type **QUIT** and press Enter to exit the FTP session.

Figure 6-8 shows the DDL source imported to create the table *STAFFI* on the iSeries server.

```

Columns . . . :   1 71           Browse           VIJAY/SQLSRC
SEU==>                                     STAFF
FMT **   ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
          ***** Beginning of data *****
0001.00
0002.00 CREATE TABLE VIJAY.STAFFI (
0003.00 ID          SMALLINT NOT NULL ,
0004.00 NAME       VARCHAR(9) CCSID 37 DEFAULT NULL ,
0005.00 DEPT       SMALLINT DEFAULT NULL ,
0006.00 JOB        CHAR(5) CCSID 37 DEFAULT NULL ,
0007.00 "YEARS"    SMALLINT DEFAULT NULL ,
0008.00 SALARY     DECIMAL(7, 2) DEFAULT NULL ,
0009.00 COMM       DECIMAL(7, 2) DEFAULT NULL
0010.00 );
0011.00
          ***** End of data *****

F3=Exit   F5=Refresh   F9=Retrieve   F10=Cursor   F11=Toggle   F12=Cancel
F16=Repeat find   F24=More keys

(C) COPYRIGHT IBM CORP. 1981, 2000.

```

Figure 6-8 Imported DDL for the STAFFA table

Figure 6-9 shows the STAFFA data file imported in the CSV data format to the VIJAY library.

*...+...1...+...2...+...3...+...4...+...5...+...6.						
10	,"Sanders "	,20	,"Mgr "	,7	,18357.50	,300.00
20	,"Pernal "	,20	,"Sales",	,8	,18171.25	,1112.45
30	,"Marenghi "	,38	,"Mgr "	,5	,17506.75	,500.00
40	,"O'Brien "	,38	,"Sales",	,6	,18006.00	,846.55
50	,"Hanes "	,15	,"Mgr "	,10	,20659.80	,.00
60	,"Quigley "	,38	,"Sales",	,0	,16808.30	,650.25
70	,"Rothman "	,15	,"Sales",	,7	,16502.83	,1152.00
80	,"James "	,20	,"Clerk",	,0	,13504.60	,128.20
90	,"Koonitz "	,42	,"Sales",	,6	,18001.75	,1386.70
100	,"Plotz "	,42	,"Mgr "	,7	,18352.80	,.00
110	,"Ngan "	,15	,"Clerk",	,5	,12508.20	,206.60
120	,"Naughton "	,38	,"Clerk",	,0	,12954.75	,180.00
130	,"Yamaguchi",	,42	,"Clerk",	,6	,10505.90	,75.60
140	,"Fraye "	,51	,"Mgr "	,6	,21150.00	,.00
150	,"Williams "	,51	,"Sales",	,6	,19456.50	,637.65
160	,"Molinare "	,10	,"Mgr "	,7	,22959.20	,.00
170	,"Kermisch "	,15	,"Clerk",	,4	,12258.50	,110.10
180	,"Abrahams "	,38	,"Clerk",	,3	,12009.75	,236.50
190	,"Sneider "	,20	,"Clerk",	,8	,14252.75	,126.50
200	,"Scoutten "	,42	,"Clerk",	,0	,11508.60	,84.20
210	,"Lu "	,10	,"Mgr "	,10	,20010.00	,.00
220	,"Smith "	,51	,"Sales",	,7	,17654.50	,992.80
230	,"Lundquist",	,51	,"Clerk",	,3	,13369.80	,189.65
240	,"Daniels "	,10	,"Mgr "	,5	,19260.25	,.00
250	,"Wheeler "	,51	,"Clerk",	,6	,14460.00	,513.30
260	,"Jones "	,10	,"Mgr "	,12	,21234.00	,.00
270	,"Lea "	,66	,"Mgr "	,9	,18555.50	,.00
280	,"Wilson "	,66	,"Sales",	,9	,18674.50	,811.50
290	,"Quill "	,84	,"Mgr "	,10	,19818.00	,.00
300	,"Davis "	,84	,"Sales",	,5	,15454.50	,806.10
310	,"Graham "	,66	,"Sales",	,13	,21000.00	,200.30
320	,"Gonzales "	,66	,"Sales",	,4	,16858.20	,844.00
330	,"Burke "	,66	,"Clerk",	,1	,10988.00	,55.50
340	,"Edwards "	,84	,"Sales",	,7	,17844.00	,1285.00
350	,"Gafney "	,84	,"Clerk",	,5	,13030.50	,188.00

Figure 6-9 Imported STAFFA.csv data file

- Use the Run SQL Statement (RUNSQLSTM) command from the iSeries server command line to use the DDL source to create the *STAFFI* table in library *VIJAY*:

```
RUNSQLSTM SRCFILE(VIJAY/SQLSRC) SRCMBR(STAFF) COMMIT(*NONE) NAMING(*SQL)
```

This command has the \*SQL naming convention specified as used in the DDL in Figure 6-8. Also the COMMIT parameter is specified with the value \*NONE as because just want to create the table in library VIJAY and do not plan to use commitment control.

- The last step imports the STAFFA file in the CSV data format using the CPYFRMIMPF command. It also includes populating the *STAFFI* table in the library *VIJAY*. Type the following command from a command line and press Enter to accept the defaults for string a delimiter and field separator:

```
CPYFRMIMPF FROMFILE(VIJAY/STAFFA) TOFILE(VIJAY/STAFFI) MBROPT(*REPLACE)
```

Member STAFFI file STAFFI in VIJAY cleared.

35 records copied from member STAFFA.

Use the **RUNQRY** command to look at the data in table *STAFFI*:

```
RUNQRY *N VIJAY/STAFFI
```

This command produces the report shown in Figure 6-10.

Display Report							
Position to line . . . . .							Report width . . . . . : 64
Line . . . . .							Shift to column . . . . .
	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
000001	10	Sanders	20	Mgr	7	18,357.50	300.00
000002	20	Pernal	20	Sales	8	18,171.25	1,112.45
000003	30	Marenghi	38	Mgr	5	17,506.75	500.00
000004	40	O'Brien	38	Sales	6	18,006.00	846.55
000005	50	Hanes	15	Mgr	10	20,659.80	.00
000006	60	Quigley	38	Sales	0	16,808.30	650.25
000007	70	Rothman	15	Sales	7	16,502.83	1,152.00
000008	80	James	20	Clerk	0	13,504.60	128.20
000009	90	Koonitz	42	Sales	6	18,001.75	1,386.70
000010	100	Plotz	42	Mgr	7	18,352.80	.00
000011	110	Ngan	15	Clerk	5	12,508.20	206.60
000012	120	Naughton	38	Clerk	0	12,954.75	180.00
000013	130	Yamaguchi	42	Clerk	6	10,505.90	75.60
000014	140	Fraye	51	Mgr	6	21,150.00	.00
000015	150	Williams	51	Sales	6	19,456.50	637.65
000016	160	Molinare	10	Mgr	7	22,959.20	.00

More...

F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split

Figure 6-10 RUNQRY report for the STAFF1 table

## 6.2.4 Parallel data loader

When using the CPYFRMIMPF command, you can take advantage of loading data into DB2 Universal Database for iSeries in parallel when the DB2 UDB for iSeries Symmetric Multiprocessing (SMP) licensed feature of OS/400 is installed and activated on the iSeries server to activate parallelism.

DB2 UDB for iSeries uses multiple tasks to load the large file. The command breaks the file into blocks and submits the blocks in parallel; the entire file is processed at the same time. The number of tasks used during the copy is determined by DEGREE(\*NBRTASKS) on the Change Query Attributes (CHGQRYA) command.

Table 6-2, based on the results of performance testing by the Teraplex Center using a 12-processor iSeries server, shows the advantages of parallel processing. The test used an import file containing 350 million rows with 100 GB of data.

Table 6-2 Parallel data load

Load time	Degree of parallel processing
47+ hours	1
4+ hours	12

## 6.3 DB2 UDB for iSeries Export utility

DB2 UDB for iSeries tables can be exported into a flat file with the CPYTOIMPF CL command (single threaded only).

### 6.3.1 CPYTOIMPF

The Copy To Import File (CPYTOIMPF) command is used to export data from a DB2 UDB for iSeries table to either a source physical file or a stream file. The command copies an externally defined file to an *import file*; the term import file is used to describe a file created for the purpose of copying data between heterogenous databases. The import file (TOSTMF or TOFILE parameter of the command) can be sent to an external system using FTP or Client Access. The export data flow is shown in Figure 6-11.

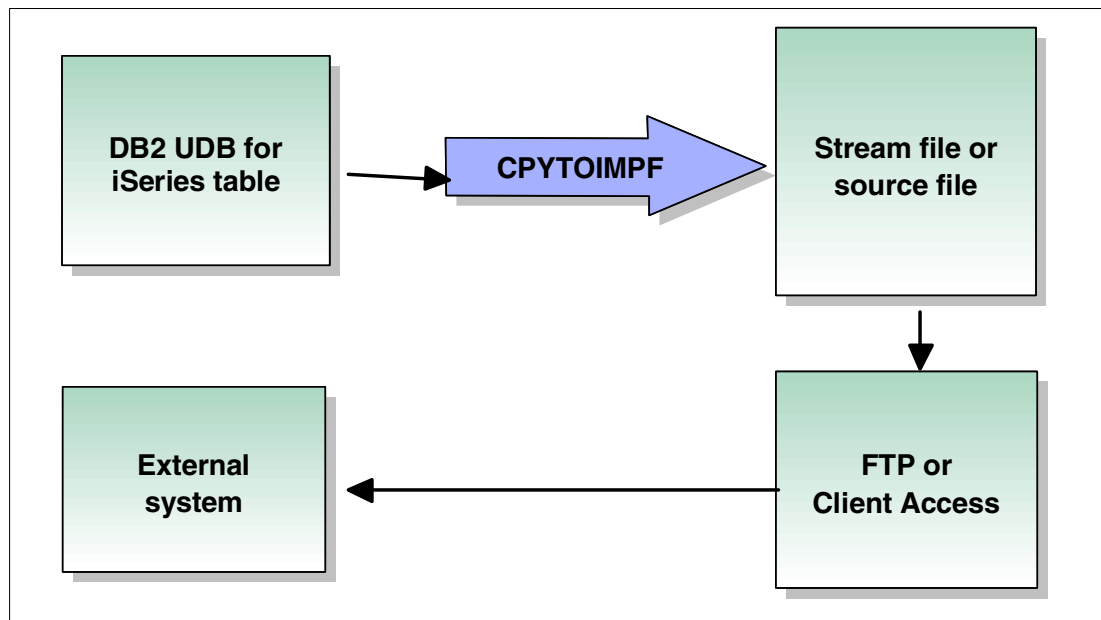


Figure 6-11 Data export flow

The following steps summarize a data load from a database file:

1. Create an import file (TOFILE) for the data that will be copied to the external system. The format for this data can be in delimited format or fixed format.
2. Use the **CPYTOIMPF** command to copy (translate or parse the records) from the source DB2 UDB for iSeries file (FROMFILE) to the import file (TOFILE).
3. Send the data in the import file (typically with FTP or Client Access) to the external system.

#### The source file (FROMFILE)

The source file (FROMFILE) can be any one of the following file types:

- ▶ Source physical file
- ▶ Distributed physical file
- ▶ Single format logical file
- ▶ Externally described physical file with one field (of non-numeric data type)

**Note:** If an externally described physical file has one field, the data type must be CHARACTER, IGC OPEN, IGC EITHER, IGC ONLY, GRAPHIC, or variable length.



The file can be copied or exported from the iSeries server using several methods, including:

- ▶ TCP/IP file transfer (text transfer)
- ▶ CA/400 support (file transfer, ODBC)
- ▶ Copy To Tape (CPYTOTAP) command

Sending the data into the import file causes the necessary EBCDIC to ASCII data conversions to occur.

### The target file (TOFILE)

The source file (FROMFILE) is copied to the import file, also referred to as the TOFILE or TOSTMF. The import file can be any one of the following file types:

#### ▶ Parameter TOFILE

- Source physical file
- If the file is not a source file, the file can have only one field; the field of the file cannot be a numeric data type
- Program described physical file
- Externally described physical file that can have only one field; the field of the file cannot be a numeric data type

#### ▶ Parameter TOSTMF

Specifies the path name of the output stream file to which the source file is copied.

### Data format (DTAFMT)

The data can be copied to the TOFILE as either delimiter format or fixed format:

#### ▶ Delimited format (\*DLM): A series of characters as delimiters to define where strings, fields, and records begin and end.

- Delimiters cannot be blank.
- A period cannot be a string delimiter.
- A string delimiter cannot be the same as a field or record delimiter.
- The defaults for delimiters are:
  - String: Double quote (“)
  - Field: Comma (,)
  - Record: End of record (\*EOR)

#### ▶ Fixed format (\*FIXED): Each field of the file is copied without delimiters

- The NULLS parameter can only have the value \*YES if DTAFMT(\*FIXED) is used. This places either a “Y” or “N” after field data indicating if the field is null or not null.
- The NULLS parameter can also have the default value \*NO. This does not place a “Y” or “N” after field data.
- A field definition file is not needed.

Additional function has been added to the following parameters of the CPYTOIMPF command:

#### ▶ Stream file code page (STMCODPAG)

Allows you to specify the code page of the target stream file. In the past, you would use another tool or command to first create the stream file with the desired code page to override the default behavior of the command.

► **Replace or add records (MBROPT)**

If the CPYTOIMPF command is given an empty database table to copy, then DB2 UDB for iSeries now clears the target stream file when MBROPT(\*REPLACE) is specified.

### 6.3.2 Creating the import file (TOFILE)

This section shows the creation of the TOFILE using the \*FIXED and \*DLM data formats. The DB2 UDB for iSeries STAFF table in the library VIJAY is exported to another database server along with the DDL source for the file from the member STAFF in the SQLSRC source physical file in the library VIJAY:

1. Use the CRTPF command to create a single field physical file with a record length of 72 bytes:

```
CRTPF FILE(VIJAY/PF72) RCDLEN(72) MAXMBRS(*NOMAX)
```

2. Use the RMVM command to remove the PF72 member from the PF72 file:

```
RMVM FILE(VIJAY/PF72) MBR(PF72)
```

More members are added to the file PF72 as we use the Export utility to create the import file (TOFILE).

3. Use the CPYTOIMPF command to copy the STAFF file and add the STAFFNLN member to the file PF72; the command specifies the DTAFMT(\*FIXED) and NULLS(\*NO) parameters as shown here:

```
CPYTOIMPF FROMFILE(VIJAY/STAFF) TOFILE(VIJAY/PF72 STAFFNLN) MBROPT(*REPLACE)
DTAFMT(*FIXED) NULLIND(*NO)
```

Figure 6-12 shows a partial list of the resulting member STAFFNLN.

Display Physical File Member						
File . . . . .	PF72	Library . . . . .	VIJAY			
Member . . . . .	STAFFNLN	Record . . . . .	1			
Control . . . . .		Column . . . . .	1			
Find . . . . .						
* . . . + . . . . 1 . . . + . . . . 2 . . . + . . . . 3 . . . + . . . . 4 . . . + . . . . 5 . . . + . . . . 6 . . . + . . . . 7 . .						
10	Sanders	20	Mgr 7	18357.50	300.00	
20	Pernal	20	Sales8	18171.25	1412.45	
30	Marenghi	38	Mgr 5	17506.75	500.00	
40	O'Brien	38	Sales6	18006.00	846.55	
50	Hanes	15	Mgr 10	20659.80	0.0	
60	Quigley	38	Sales0	16808.30	650.25	
70	Rothman	15	Sales7	16502.83	1152.00	
80	James	20	Clerk0	13504.60	128.20	
90	Koonitz	42	Sales6	18001.75	1386.70	
100	Plotz	42	Mgr 7	18352.80	0.0	
110	Ngan	15	Clerk5	12508.20	206.60	
120	Naughton	38	Clerk0	12954.75	180.00	
130	Yamaguchi	42	Clerk6	10505.90	75.60	
140	Fraye	51	Mgr 6	21150.00	0.0	
150	Williams	51	Sales6	19456.50	637.65	
						More . . .
F3=Exit	F12=Cancel	F19=Left	F20=Right	F24=More keys		

Figure 6-12 TOFILE with DTAFMT(\*FIXED) NULLS(\*NO)

- Use the CPYTOIMPF command to copy the *STAFF* file and add the *STAFFNLY* member to the file *PF72*; the command specifies the DTAFMT(\*FIXED) and NULLS(\*YES) parameters as shown here:

```
CPYTOIMPF FROMFILE(VIJAY/STAFF) TOFILE(VIJAY/PF72 STAFFNLY) MBROPT(*REPLACE)
DTAFMT(*FIXED) NULLIND(*YES)
```

Figure 6-13 shows a partial list of the resulting member *STAFFNLY*; notice the “Y” after each field that has a null value and “N” after each field that does not have a null value.

Display Physical File Member						
File . . . . .	PF72	Library . . . . .	VIJAY			
Member . . . . .	STAFFNLY	Record . . . . .	1			
Control . . . . .		Column . . . . .	1			
Find . . . . .						
* . . . + . . . . 1 . . . + . . . . 2 . . . + . . . . 3 . . . + . . . . 4 . . . + . . . . 5 . . . + . . . . 6 . . . + . . . . 7 . .						
10	NSanders	N20	NMgr N7	N18357.50	N300.00	N
20	NPernal	N20	NSalesN8	N18171.25	N1412.45	N
30	NMarengi	N38	NMgr N5	N17506.75	N500.00	N
40	NO'Brien	N38	NSalesN6	N18006.00	N846.55	N
50	NHanes	N15	NMgr N10	N20659.80	N0.0	Y
60	NQuigley	N38	NSalesN0	Y16808.30	N650.25	N
70	NRothman	N15	NSalesN7	N16502.83	N1152.00	N
80	NJames	N20	NClerkN0	Y13504.60	N128.20	N
90	NKoonitz	N42	NSalesN6	N18001.75	N1386.70	N
100	NPlotz	N42	NMgr N7	N18352.80	N0.0	Y
110	NNgan	N15	NClerkN5	N12508.20	N206.60	N
120	NNaughton	N38	NClerkN0	Y12954.75	N180.00	N
130	NYamaguchi	N42	NClerkN6	N10505.90	N75.60	N
140	NFraye	N51	NMgr N6	N21150.00	N0.0	Y
150	NWilliams	N51	NSalesN6	N19456.50	N637.65	N
						More...
F3=Exit	F12=Cancel	F19=Left	F20=Right	F24=More keys		

Figure 6-13 TOFILE with DTAFMT(\*FIXED) NULLS(\*YES)

- Use the CPYTOIMPF command to copy the *STAFF* file and add the *STAFFDLM* member to the file *PF72*; the command specifies the DTAFMT(\*DLM) parameter and uses the default delimiter values as follows:

```
CPYTOIMPF FROMFILE(VIJAY/STAFF) TOFILE(VIJAY/PF72 STAFFDLM) MBROPT(*REPLACE)
```

Figure 6-14 shows a partial list of the resulting *STAFFDLM* member; this member shows the data ready for export in the most common CSV format that is used to port data between heterogenous databases.

```

Display Physical File Member
File . . . . . : PF72          Library . . . . . : VIJAY
Member . . . . . : STAFFDLM      Record . . . . . : 1
Control . . . . .           Column . . . . . : 1
Find . . . . .
*...+...1...+...2...+...3...+...4...+...5...+...6...+...7..
10  ,"Sanders " ,20  ,"Mgr " ,7    ,18357.50 ,300.00
20  ,"Pernal  " ,20  ,"Sales",8   ,18171.25 ,1412.45
30  ,"Marenghi " ,38  ,"Mgr " ,5    ,17506.75 ,500.00
40  ,"O'Brien " ,38  ,"Sales",6   ,18006.00 ,846.55
50  ,"Hanes   " ,15  ,"Mgr " ,10   ,20659.80 ,,
60  ,"Quigley " ,38  ,"Sales",,16808.30 ,650.25
70  ,"Rothman " ,15  ,"Sales",7    ,16502.83 ,1152.00
80  ,"James   " ,20  ,"Clerk",,13504.60 ,128.20
90  ,"Koonitz " ,42  ,"Sales",6   ,18001.75 ,1386.70
100 ,"Plotz   " ,42  ,"Mgr " ,7    ,18352.80 ,,
110 ,"Ngan    " ,15  ,"Clerk",5    ,12508.20 ,206.60
120 ,"Naughton " ,38  ,"Clerk",,12954.75 ,180.00
130 ,"Yamaguchi",42  ,"Clerk",6    ,10505.90 ,75.60
140 ,"Fraye   " ,51  ,"Mgr " ,6    ,21150.00 ,,
150 ,"Williams " ,51  ,"Sales",6   ,19456.50 ,637.65
More...

F3=Exit  F12=Cancel  F19=Left  F20=Right  F24=More keys

```

Figure 6-14 TOFILE with DTAFMT(\*DLM)

- Use the CPYTOIMPF command to copy the DDL source from the *STAFF* member in the *SQLSRC* source physical file in the *VIJAY* library and add the *STAFFDDL* member to the file *PF72*; the command specifies the *DTAFMT(\*FIXED)* parameter as shown here:

```

CPYTOIMPF FROMFILE(VIJAY/SQLSRC STAFF) TOFILE(VIJAY/PF72 STAFFDDL) MBROPT(*REPLACE)
DTAFMT(*FIXED)

```

Figure 6-15 shows a partial list of the resulting member *STAFFDLM*.

```

                                Display Physical File Member
File . . . . . : PF72                Library . . . . . : VIJAY
Member . . . . . : STAFFDDL          Record . . . . . : 1
Control . . . . . :                  Column . . . . . : 1
Find . . . . . :
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7..

CREATE TABLE VIJAY.STAFFI (
ID          SMALLINT NOT NULL ,
NAME       VARCHAR(9) CCSID 37 DEFAULT NULL ,
DEPT       SMALLINT DEFAULT NULL ,
JOB        CHAR(5) CCSID 37 DEFAULT NULL ,
"YEARS"    SMALLINT DEFAULT NULL ,
SALARY     DECIMAL(7, 2) DEFAULT NULL ,
COMM       DECIMAL(7, 2) DEFAULT NULL
);

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

                                                                Bottom

F3=Exit   F12=Cancel   F19=Left   F20=Right   F24=More keys

```

Figure 6-15 TOFILE for DDL source

### 6.3.3 Exporting the TOFILE

The members STAFFDDL and STAFFDLM in the import file *PF72* in the *VIJAY* library are exported to the external system. A TCP/IP connection exists between the iSeries server and the external database server. FTP is used for the data transfer between the two database servers. The FTP dialogue from the external system is shown here:

```

C:\>ftp as23 1
Connected to AS23.
220-QTCP at rchasm23.rchland.ibm.com.
220 Connection will close if idle more than 5 minutes.
User (AS23:(none)): vijay 2
331 Enter password.
Password: 3
230 VIJAY logged on.
ftp> get vijay/pf72.staffddl c:\vijay\staff.ddl 4
200 PORT subcommand request successful.
150 Retrieving member STAFFDDL in file PF72 in library VIJAY.
250 File transfer completed successfully.
ftp: 305 bytes received in 0.00Seconds 305000.00Kbytes/sec.
ftp> get vijay/pf72.staffdml c:\vijay\staff.csv 5
200 PORT subcommand request successful.
150 Retrieving member STAFFDLM in file PF72 in library VIJAY.
250 File transfer completed successfully.
ftp: 2136 bytes received in 0.00Seconds 2136000.00Kbytes/sec.
ftp> quit 6

```

**Notes:**

- 1 From a command line, type **FTP** to the iSeries server AS23.
- 2 Enter your user ID and press Enter.
- 3 Type your password and press Enter.
- 4 Type the **GET** sub-command to copy the *STAFFDDL* member in the file *PF72* in the *VIJAY* library to the *staff.ddl* file in the *vijay* directory. Note the use of the forward slash (/) and the period (.) in the target file name (library/file.member) format.
- 5 Type the **GET** sub-command to copy the *STAFFDLM* member in the *PF72* file in the *VIJAY* library to the *staff.csv* file in the *vijay* directory. Note the use of the forward slash (/) and the period (.) in the target file name (library/file.member) format.
- 6 Type **QUIT** and press Enter to exit the FTP session to iSeries server AS23.

### 6.3.4 Creating the import file (STMF)

This section shows the creation of the STMF import file in the integrated file system (IFS) on the iSeries server using the \*FIXED and \*DLM data formats. The DB2 UDB for iSeries file STAFF in library VIJAY are exported to another database server along with the DDL source for the file from the member STAFF in source physical file SQLSRC in library VIJAY.

1. Use the **make directory** command to create the *vijay* directory in the integrated file system:

```
md vijay
```

2. Use the **CPYTOIMPF** command to copy the *STAFF* file and add the *staffnl.txt* stream file to the *vijay* directory; the command specifies the *DTAFMT(\*FIXED)* and *NULLS(\*NO)* parameters as shown here:

```
CPYTOIMPF FROMFILE(VIJAY/STAFF) TOSTMF('/vijay/staffnl.txt') MBROPT(*REPLACE)  
RCDDL(*LF) DTAFMT(*FIXED)
```

Figure 6-16 shows a partial list of the resulting *staffnl.txt* stream file.

```

Browse : /vijay/staffnln.txt
Record :      1  of      36 by 14          Column :      1  63 by 79
Control :

....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
*****Beginning of data*****
10      Sanders  20      Mgr  7      18357.50 300.00
20      Pernal  20      Sales8    18171.25 1412.45
30      Marenghi 38      Mgr  5      17506.75 500.00
40      O'Brien  38      Sales6    18006.00 846.55
50      Hanes   15      Mgr  10     20659.80 0.0
60      Quigley 38      Sales0    16808.30 650.25
70      Rothman 15      Sales7    16502.83 1152.00
80      James   20      Clerk0    13504.60 128.20
90      Koonitz 42      Sales6    18001.75 1386.70
100     Plotz   42      Mgr  7      18352.80 0.0
110     Ngan    15      Clerk5    12508.20 206.60
120     Naughton 38      Clerk0    12954.75 180.00
130     Yamaguchi42      Clerk6    10505.90 75.60
140     Fraye   51      Mgr  6      21150.00 0.0

F3=Exit  F10=Display Hex  F12=Exit  F15=Services  F16=Repeat find
F19=Left  F20=Right
          (C) COPYRIGHT IBM CORP. 1980, 2000.

```

Figure 6-16 STMF DTAFMT(\*FIXED) NULLS(\*NO)

- Use the CPYTOIMPF command to copy the *STAFF* file and add the *staffnly.txt* stream file to the *vijay* directory; the command specifies the DTAFMT(\*FIXED) and NULLS(\*YES) parameters as shown here:

```

CPYTOIMPF FROMFILE(VIJAY/STAFF) TOSTMF('/vijay/staffnly.txt') MBROPT(*REPLACE)
RCDDL(*LF) DTAFMT(*FIXED) NULLS(*YES)

```

Figure 6-17 shows a partial list of the resulting *staffnly.txt* stream file.

```

Browse : /vijay/staffnly.txt
Record :      1  of      36 by 14          Column :      1  72 by 79
Control :

....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
*****Beginning of data*****
10      NSanders  N20      NMgr  N7      N18357.50 N300.00  N
20      NPernal  N20      NSalesN8    N18171.25 N1412.45 N
30      NMarenghi N38      NMgr  N5      N17506.75 N500.00  N
40      NO'Brien N38      NSalesN6    N18006.00 N846.55  N
50      NHanes   N15      NMgr  N10     N20659.80 N0.0    Y
60      NQuigley N38      NSalesN0    Y16808.30 N650.25  N
70      NRothman  N15      NSalesN7    N16502.83 N1152.00  N
80      NJames   N20      NClerkN0    Y13504.60 N128.20  N
90      NKoonitz  N42      NSalesN6    N18001.75 N1386.70  N
100     NPlotz   N42      NMgr  N7      N18352.80 N0.0    Y
110     NNgan    N15      NClerkN5    N12508.20 N206.60  N
120     NNaughton N38      NClerkN0    Y12954.75 N180.00  N
130     NYamaguchiN42    NClerkN6    N10505.90 N75.60   N
140     NFraye   N51      NMgr  N6      N21150.00 N0.0    Y

F3=Exit  F10=Display Hex  F12=Exit  F15=Services  F16=Repeat find
F19=Left  F20=Right
          (C) COPYRIGHT IBM CORP. 1980, 2000.

```

Figure 6-17 STMF DTAFMT(\*FIXED) NULLS(\*YES)

4. Use the CPYTOIMPF command to copy the STAFF file and add the staffdlm.csv stream file to the vijay directory; the command specifies the DTAFMT(\*DLM) parameter and uses the default delimiters as shown here:

```

CPYTOIMPF FROMFILE(VIJAY/STAFF) TOSTMF('/vijay/staffdlm.csv') MBROPT(*REPLACE)
RCDDL(*LF)

```

Figure 6-18 shows a partial list of the resulting staffdlm.csv stream file. This stream file shows the data ready for export in the most common CSV format that is used to port data between heterogenous databases.



```

Browse : /vijay/staffdml.csv
Record :      1  of      36 by 14          Column :      1  59 by 79
Control :

....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
*****Beginning of data*****
10  ,"Sanders  ",20  ,"Mgr  ",7  ,18357.50 ,300.00
20  ,"Pernal  ",20  ,"Sales",8  ,18171.25 ,1412.45
30  ,"Marenghi",38  ,"Mgr  ",5  ,17506.75 ,500.00
40  ,"O'Brien ",38  ,"Sales",6  ,18006.00 ,846.55
50  ,"Hanes   ",15  ,"Mgr  ",10 ,20659.80 ,,
60  ,"Quigley ",38  ,"Sales",,16808.30 ,650.25
70  ,"Rothman ",15  ,"Sales",7  ,16502.83 ,1152.00
80  ,"James   ",20  ,"Clerk",,13504.60 ,128.20
90  ,"Koonitz ",42  ,"Sales",6  ,18001.75 ,1386.70
100 ,"Plotz   ",42  ,"Mgr  ",7  ,18352.80 ,,
110 ,"Ngan    ",15  ,"Clerk",5  ,12508.20 ,206.60
120 ,"Naughton",38  ,"Clerk",,12954.75 ,180.00
130 ,"Yamaguchi",42  ,"Clerk",6  ,10505.90 ,75.60
140 ,"Fraye   ",51  ,"Mgr  ",6  ,21150.00 ,,

F3=Exit  F10=Display Hex  F12=Exit  F15=Services  F16=Repeat find
F19=Left  F20=Right
(C) COPYRIGHT IBM CORP. 1980, 2000.

```

Figure 6-18 STMF DTAFMT(\*DLM)

5. Use the CPYTOIMPF command to copy the DDL source from the *STAFF* member in the *SQLSRC* source physical file in the *VIJAY* library and add the *staff.ddl* stream file to the *vijay* directory; the command specifies the DTAFMT(\*FIXED) parameter as shown here:

```

CPYTOIMPF FROMFILE(VIJAY/SQLSRC STAFF) TOSTMF('/vijay/staff.ddl') MBROPT(*REPLACE)
RCDDL(*LF) DTAFMT(*FIXED)

```

Figure 6-19 shows a list of the resulting *staff.ddl* stream file.

```

Browse : /vijay/staff.ddl
Record :      1 of      11 by 14      Column :      1      59 by 79
Control :

....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
*****Beginning of data*****

CREATE TABLE VIJAY.STAFFI (
  ID      SMALLINT NOT NULL ,
  NAME    VARCHAR(9) CCSID 37 DEFAULT NULL ,
  DEPT    SMALLINT DEFAULT NULL ,
  JOB     CHAR(5) CCSID 37 DEFAULT NULL ,
  "YEARS" SMALLINT DEFAULT NULL ,
  SALARY  DECIMAL(7, 2) DEFAULT NULL ,
  COMM    DECIMAL(7, 2) DEFAULT NULL
);

*****End of Data*****

F3=Exit  F10=Display Hex  F12=Exit  F15=Services  F16=Repeat find
F19=Left  F20=Right
          (C) COPYRIGHT IBM CORP. 1980, 2000.

```

Figure 6-19 STMF for DDL source

### 6.3.5 Exporting the STMF

The stream files *staff.ddl* and *staffdlm.csv* in the *vijay* directory in the iSeries server Integrated File System are exported to the external system. A TCP/IP connection exists between the iSeries server and the external database server. FTP is used for the data transfer between the two database servers. The FTP dialogue from the external system is shown here:

```

C:\>ftp as23 1
Connected to AS23.
220-QTCP at rchasm23.rchland.ibm.com.
220 Connection will close if idle more than 5 minutes.
User (AS23:(none)): vijay 2
331 Enter password.
Password: 3
230 VIJAY logged on.
ftp> get /vijay/staff.ddl c:\vijay\staff.ddl 4
200 PORT subcommand request successful.
150-NAMEFMT set to 1.
150 Retrieving file /vijay/staff.ddl
250 File transfer completed successfully.
ftp: 294 bytes received in 0.00Seconds 294000.00Kbytes/sec.
ftp> get /vijay/staffdlm.csv c:\vijay\staff.csv 5

200 PORT subcommand request successful.
150 Retrieving file /vijay/staffdlm.csv
250 File transfer completed successfully.
ftp: 1987 bytes received in 0.03Seconds 66.23Kbytes/sec.
ftp> quit 6

```

**Notes:**

- 1 From a command line, type **FTP** to the iSeries server AS23.
- 2 Enter your user ID and press Enter.
- 3 Type your password and press Enter.
- 4 Type the **GET** sub-command to copy the *staff.dtl* stream file in the *vijay* directory in the integrated file system to the *staff.dtl* file in the *vijay* directory.
- 5 Type the **GET** sub-command to copy the *staffdlm.csv* stream file in the *vijay* directory in the integrated file system to the *staff.csv* file in the *vijay* directory.
- 6 Type **QUIT** and press Enter to exit the FTP session to the iSeries server AS23.

## 6.4 Moving data from DB2 UDB 7.2 to DB2 UDB for iSeries

This section illustrates two approaches for moving data, among many other valid options. The first approach is based exclusively on the Import and Export utilities in a very direct way. The second approach combines the Export utility with the CPYFRMIMPF CL command for better performance.

### 6.4.1 First approach: Using the Export and Import utilities

In the first approach we found a very direct way to move data that was also very friendly to users already familiarized with DB2 UDB 7.2:

1. Before starting, define the target iSeries database in DB2 UDB V7.2.
2. Use the DB2 UDB Export utility for exporting data from DB2 UDB V7.2 to an integrated exchange file (IXF).
3. Use the DB2 UDB V7.2 Command Center or other DB2 UDF V7.2 SQL interface to connect to DB2 Universal Database for iSeries.
4. Use the Import utility for importing the IXF file into a predefined target table.

The Export utility can be used to export DB2 UDB V7.2 information into an operating system file in one of the following formats:

- ▶ **Integrated exchange file (IXF):** A data format explicitly designed for exchanging relational data. IXF files are the preferred file format for exchanging information between DB2 UDB V7.2 databases.
- ▶ **Delimited ASCII file (DEL):** A very popular family of flat files for exchanging information, in which text fields are enclosed by quotations, fields are separated by commas, and decimals are delimited by a point. Those delimiter characters can be changed if needed. Delimited ASCII files are very popular also for exchanging information among personal productivity tools, such as spreadsheets, where they are known as CSV files.
- ▶ **Worksheet format file (WSF):** Highly used for exchanging information between spreadsheets and other tools, including relational databases.

In order to use the export utility, you can use the interactive Control Center interface or a DB2 UDB V7.2 SQL interface such as Command Center or DB2CMD.

#### Exporting relational data using the Control Center

From the Control Center, expand the object tree until you find the Tables or Views folder. Then right-click the table or view you want in the contents pane. Select **Export** from the pop-up menu, as shown in Figure 6-20.

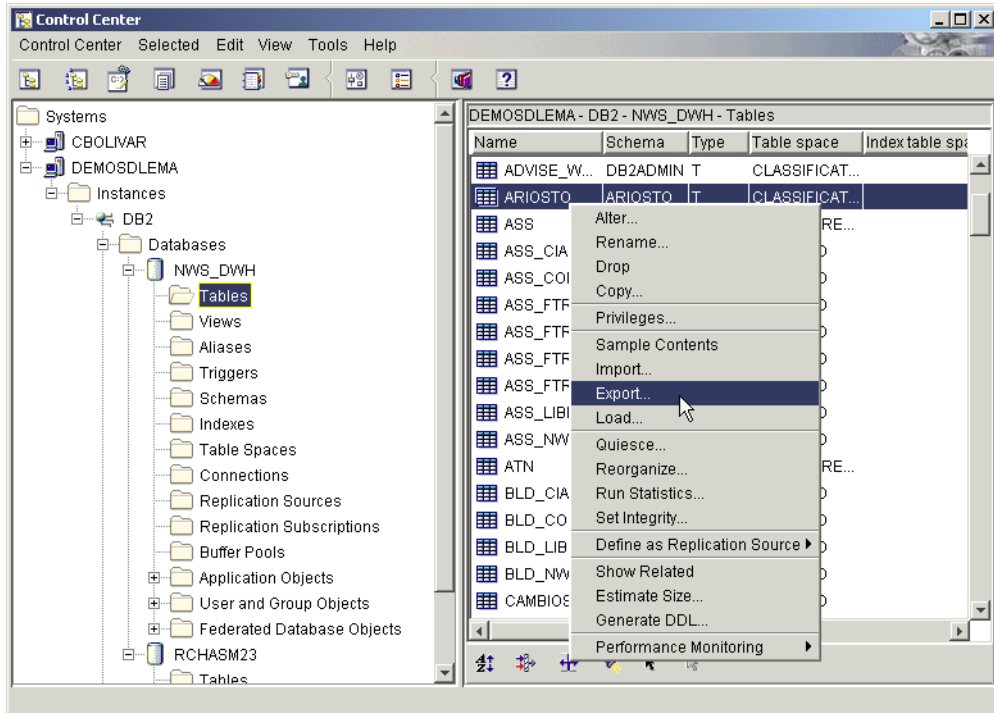


Figure 6-20 Starting Control Center's Export notebook

This takes you to the Export notebook. Follow the instructions for providing necessary information as output file, output file format, message file, select sentence, etc. as shown in Figure 6-21.

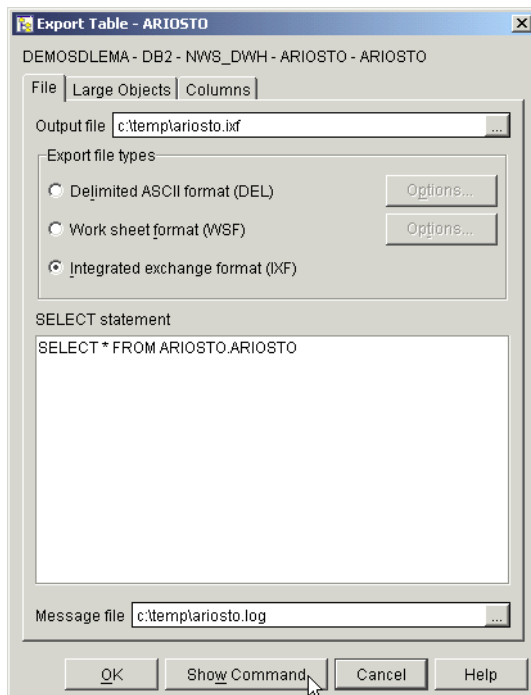


Figure 6-21 Export notebook

You can choose between running the Export command immediately by clicking the OK button or reviewing the export command by clicking the Show Command, as shown in Figure 6-22.

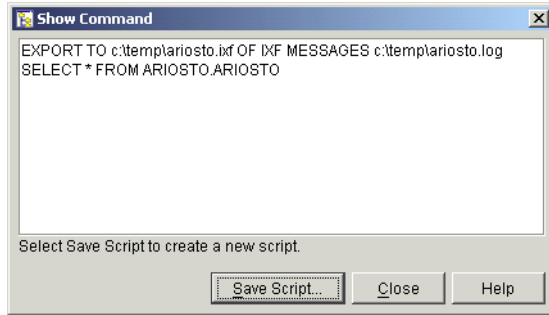


Figure 6-22 Export command as generated by the Control Center

For general information about the Control Center, you can find detailed information in online help facility within the Control Center.

### Exporting relational information using the Export command

You can also use the Export command from an SQL interface, such as DB2CMD, which is a very practical approach when you need to export data periodically, because it lets you create a batch file.

An example of the Export command issued through the CLP is shown here:

```
db2 export to staff.ixf of ixf select * from userid.staff
```

### Exporting relational information using Export API

You can also use the provided export application programming interface (API), `sqluexpr`. For general information about creating applications containing DB2 administrative APIs, see *DB2 UDB Application Development Guide V6, SC09-2845*.

### Importing an IXF file into DB2 UDB for iSeries

From a DB2 UDB V7.2 SQL interface, you connect to the target DB2 UDB for iSeries database using the following command:

```
CONNECT TO RCHASM23 USER DLEMA USING MYPASS
```

Then you use the following Import command to import data into the target table:

```
IMPORT FROM SOURCE_IXF_FILE.IXF OF IXF INSERT INTO DLEMA.DESTTABLE
```

Here *SOURCE\_IXF\_FILE.IXF* is the IXF file located in the workstation where the SQL interface is being used.

**Important:** A destination table with compatible columns must exist previously, and it must be journaled. If it is not, you will receive an SQL error -7008.

The Import utility in DB2 UDB for iSeries V5R1 is limited to importing IXF files into existing tables.

### All in a batch

As a convenient way to move multiple tables in a periodic way, you can create a batch file as shown in the following example:

```
-- CONNECTION TO DB2 UDB 7.2 SOURCE DATABASE
CONNECT TO NWS_DWH USER MYUSER USING MYPASWRD;
-- EXPORTING DATA INTO IXF FILES
```

```

EXPORT TO C:\TEMP\ASS.IXF OF IXF SELECT * FROM DB2ADMIN.ASS WHERE END_DT IS NULL;
EXPORT TO C:\TEMP\ACT.IXF OF IXF SELECT * FROM DB2ADMIN.ACT
    WHERE ACT_TS > '2001-01-01 00:00:00.000000';
EXPORT TO C:\TEMP\USR.IXF OF IXF SELECT USR_IP_ID, NM, DEPT_OU_ID, CC_OU_ID, BLD_ID
    FROM DB2ADMIN.USR;
-- AND MANY MORE...
--
-- CONNECTION TO DB2 UDB FOR ISERIES
CONNECT TO RCHASM23 USER AS400USR USING AS400PWD;
-- IMPORTING IXF DATA INTO ISERIES SERVER
IMPORT FROM C:\TEMP\ASS.IXF OF IXF INSERT INTO DLEMA.ASS;
IMPORT FROM C:\TEMP\ACT.IXF OF IXF INSERT INTO DLEMA.ACT;
IMPORT FROM C:\TEMP\USR.IXF OF IXF INSERT INTO DLEMA.USR;

```

You run this batch file with the following DB2 UDB V7.2 command:

```
db2cmd -w -c db2 -f c:\temp\export_import.sql -z c:\temp\export_import.log -t
```

Here *c:\temp\export\_import.sql* is the batch file and *c:\temp\export\_import.log* is a text file, where the informational, warning, and error conditions will be stored for review in case of failure.

## 6.4.2 Second approach: Using Export and CPYFRMIMPF

The second approach is more appropriate for moving large data sets because it performs better:

1. Use the DB2 UDB Export utility for exporting data from DB2 UDB V7.2 to a DEL file.
2. Move the DEL file to the iSeries server.
3. Use the CPYFRMIMPF CL command to load the DEL file into the target table.

### Using Export utility for exporting data to a DEL file

You can use the Export utility from the Control Center as shown in “Exporting relational data using the Control Center” on page 149, but export to a delimited ASCII file. You can also use an interactive or batch command interface for executing a sentence as shown in the following example:

```
EXPORT TO C:\TEMP\FILE.DEL OF DEL SELECT * FROM SAMPLEDB02.STAFF
```

The resulting delimited ASCII file can now be loaded into the iSeries server using FTP and then loaded into the target table using the CPYFRMIMPF command, as explained in 6.2, “DB2 UDB for iSeries Import utility” on page 126.

## 6.5 Moving data from DB2 UDB for iSeries into DB2 UDB 7.2

You can move data from DB2 UDB for iSeries into DB2 UDB V7.2 using the same two approaches shown in 6.4, “Moving data from DB2 UDB 7.2 to DB2 UDB for iSeries” on page 149.

### 6.5.1 Using the Import and Export utilities

Using any SQL interface to DB2 UDB V7.2 such as Command Center or DB2CMD, you can connect to the source iSeries server database and use the Export command to export any table to an IXF file, as shown here:

```

CONNECT TO RCHASM23 USER AS400USR USING AS400PWD;
EXPORT TO C:\TEMP\IXF_FILE.IXF OF IXF SELECT * FROM SCHEMA.TABLE

```

Here, *c:\temp\ixf\_file.ixf* is the target file on the workstation in which you execute the command.

After you export the iSeries server data into an IXF file, you can import into DB2 UDB V7.2 using the Import utility. In this case, you have some extra functionality, like importing into a new table as shown in the following example:

```
IMPORT FROM C:\TEMP\IXF_FILE OF IXF CREATE INTO DB2ADMIN.TARGETTABLE
```

Here, *c:\temp\ixf\_file* is the source IXF file, and *DB2ADMIN.TARGETTABLE* is the destination table.

You can find valuable options that enable you to import appending into an existing table, import replacing an existing table, or import updating an existing table. For a detailed discussion on the Import and Export utilities in DB2 UDB V7.2, refer to *Data Movement Utilities Guide and Reference*, which you can find on the Web at:

<http://www-4.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/document.d2w/report?fn=db2v7dmdm07.htm#HDREXP0VW>

## 6.5.2 Using the CPYTOIMPF command and the Import utility

Similarly, you can use the CPYTOIMPF command to create a delimited file (also known as a CSV file) and FTP it into a DB2 UDB V7.2 workstation as described in 6.3, “DB2 UDB for iSeries Export utility” on page 138. Then you can use the Import utility on the DB2 UDB V7.2 workstation as shown in the following example:

```
IMPORT FROM C:\TEMP\DEF_FILE OF DEF INSERT INTO DB2ADMIN.TARGETTABLE
IMPORT FROM C:\TEMP\DEF_FILE OF DEF CREATE INTO DB2ADMIN.TARGETTABLE2
IMPORT FROM C:\TEMP\DEF_FILE OF DEF INSERT_UPDATE INTO DB2ADMIN.TARGETTABLE2
```







## Part 3

# Database administration

Operations Navigator offers a Windows-like graphical interface to configure, monitor, and manage the OS/400 environment. This part gives you insight into the wide range of DB2 Universal Database for iSeries database administration functions available through the Operations Navigator graphical interface, which comes packaged with Client Access Express for Windows V5R1.

This part of the book covers the following topics:

- ▶ Database functions using Operations Navigator
- ▶ The use of Database Navigator
- ▶ Reverse engineering and Generate SQL
- ▶ Visual Explain





# Database administration

This chapter discusses using the Database component of Operations Navigator to administer, view, and manipulate your databases.

This chapter discusses:

- ▶ Basic database operations
- ▶ SQL scripts
- ▶ Query attributes
- ▶ SQL Performance Monitors

## 7.1 Database overview

The Database component of Operations Navigator provides a graphical interface for many DB2 Universal Database for iSeries database operations, including:

- ▶ Creating and managing tables, views, indexes, SQL, and stored procedures
- ▶ Creating and managing OS/400 journals (record changes to the database and other functions supporting journals)
- ▶ Entering new, or modifying already created, SQL statements
- ▶ Running and debugging previously created SQL statements (referred to as *scripts*)
- ▶ Saving SQL statements for later use
- ▶ Doing performance analysis of SQL statements
- ▶ Capturing current SQL statements for any job running on the system

The Database component of AS/400 Operations Navigator is not installed by default when choosing a *Typical* installation option of IBM AS/400 Client Access Express. If the Database component is not currently installed, you can run Selective Setup to install it as discussed in *Managing OS/400 with Operations Navigator V5R1 Volume 1: Basic Functions*, SG24-6226.

With proper authorization to the database objects, the user of the database graphical interface has easy access to OS/400 server administration tools, has a clear overview of the entire database system, can perform remote database management, and receives assistance for complex tasks.

For OS/400 V4R4, key enhancements to DB2 Universal Database for iSeries included an interface to the SQL-specific performance monitor and new Universal Database Object Relational Support functions, such as various types of binary large objects (LOBs), user defined data types (UDTs), user defined functions (UDFs), and DataLinks.

OS/400 V4R5 delivered a mix of enhancements across a wide variety of DB2 UDB functions and interfaces including:

- ▶ Distributed Relational Database Architecture (DRDA) password encryption to improve the security of Internet and intranet solutions.
- ▶ The iSeries Data Loader utilities (Copy From Import File (CPYFRMIMPF) and Copy To Import File (CPYTOIMPF) CL commands) were enhanced and made easier to use in V4R5.
- ▶ iSeries external stored procedure support was upgraded in V4R5 with the addition of Java as a supported language
- ▶ Further Java database improvements were made to the iSeries SQLJ support. For example, the implementation was re-engineered to deliver performance similar to static, embedded SQL and extended dynamic SQL.
- ▶ The iSeries SQL CLI was significantly enhanced to make it more compatible with the ODBC standard.
- ▶ On the performance front, the database engine was enhanced to reduce the number of cases where SQL open data paths (ODPs) are non-reusable.
- ▶ Operations Navigator was enhanced to improve the manageability of DB2 Universal Database for iSeries. The major additions include a Display Physical File Description (DSPFD) type of output for tables, views, and indexes and Visual Explain for graphical analysis of query implementations.

- ▶ Porting the database components of an application to the iSeries server was much improved in V4R5. Improvements to the SQL Stored Procedure language and the SQL Call Level Interface (CLI) top the list of portability enhancements. The SQL Stored Procedure language has been available since V4R2 and has been widely used to successfully port procedures written in proprietary languages such as Transact SQL and PL/SQL to DB2 Universal Database for iSeries.
- ▶ Full autocommit support that improves the integrity transactions performed by ODBC- and JDBC-based client applications and lifts the restrictions that prevented stored procedures and triggers from making committable database changes.

### 7.1.1 New in V5R1

The new features in V5R1 include:

- ▶ Database Navigator: A new component that gives a pictorial representation of a schema and DB objects and can generate SQL for those objects
- ▶ Support for SQL Triggers creation from table properties
- ▶ Generate SQL from existing DB objects (including DDS created)
- ▶ RUN SQL script enhancements
- ▶ The ability to print Visual Explain graphs
- ▶ Database (SQL) V5R1 functions
  - Database Text Extenders (including Text Search Engine and XML Extenders)
  - DRDA 2 Phase Commit Over TCP/IP
  - DRDA Result Sets
  - RIGHT OUTER Join
  - Expressions in INSERT
  - SQL Triggers
  - Up to 300 triggers per table
  - Column triggers
  - Read only triggers
  - Longer than 80 character columns in C and C++ pre-compilers
  - RUNSQLSTM support in OS/400 V5R1
  - Support for user-defined functions implemented in Java
  - Increased large object (LOB) size – now 2 GB up from 15 MB
  - Maximum total size for all large objects in a table row is now 3.5 GB, up from 1.5 MB
- ▶ The maximum number of rows allowed in a table increased from 2.1 billion to 4.2 billion in V4R5. The maximum table size remains half of a terabyte (TB). In addition, you can reference more tables – up to 256 – on a single SQL statement. A journal receiver maximum size increased from 2 GB to 1 TB, which reduces the frequency of changing journal receivers. Similarly, the maximum number of journal sequence numbers increased from 2 billion to 10 billion to reduce the frequency of sequence-number resets. These remain unchanged at V5R1.

Although OS/400 integrated DB2 Universal Database for iSeries support is one of the major strengths of iSeries servers, a complete description of this support is beyond the goal of this redbook. Good sources for details of DB2 Universal Database for iSeries capabilities are:

- ▶ iSeries Information Center: <http://www.iseries.ibm.com/infocenter>

Here you can select **Database and File Systems->Database management**. Under Database management, by selecting **DB2 Universal Database for iSeries books online**, you can find a list of publications that contain even more information. Most of these publications are listed here.

- ▶ *Database Programming*, SC41-5701  
This book describes database capabilities that are primarily outside of SQL terminology. This includes physical files (correspond to SQL tables), logical files (correspond to SQL views), fields (correspond to SQL columns), records (correspond to SQL rows), file management, and file security.
- ▶ *SQL Programming Guide*, SC41-5611
- ▶ *SQL Reference*, SC41-5612
- ▶ *DB2 UDB for AS/400 Database Performance and Query Optimization*:  
<http://submit.boulder.ibm.com/pubs/html/as400/bld/v5r1/ic2924/index.htm>
- ▶ *Distributed Data Management*, SC41-5307
- ▶ *Cross-Platform DB2 Stored Procedures: Building and Debugging*, SG24-5485
- ▶ *DB2/400: Mastering Data Warehousing Functions*, SG24-5184
- ▶ *DB2 UDB for AS/400 Object Relational Support*, SG24-5409
- ▶ DB2 Universal Database for iSeries home page:  
<http://www.iseries.ibm.com/db2/db2main.htm>  
Use this link to learn about the iSeries database, recent announcements, support information, and related products.  
This page features many useful links to database related issues and products (like Business Intelligence) and gives you access to a wealth of articles, white papers, coding examples, tips, and techniques.
- ▶ Self-study lab exercise with sample OS/400 database, installation instructions, and lab instructions that can be downloaded from PartnerWorld for Developers iSeries Web site at: <http://www.iseries.ibm.com/developer>  
Select **Education->Internet Based Offerings->DB2 UDB->Piloting DB2 UDB for iSeries with Operations Navigator in V5R1**.

Under OS/400, you can use SQL interfaces to access a *database file* or an *SQL table* since these terms refer to the same object, classified within OS/400 as a \*FILE object type. You can use SQL interfaces to access the file regardless of whether the object was created with the OS/400 Create Physical File (CRTPF) command or the CREATE TABLE SQL statement. OS/400 also supports access to the physical file or table through a logical file (Create Logical File (CRTLF) command) or an SQL view (SQL CREATE VIEW).

Table 7-1 shows the corresponding OS/400 term and SQL term for physical files or tables, records or rows, fields or columns, logical files or views, aliases, and indexes.

Table 7-1 OS/400 term and SQL term cross reference

OS/400 Create statement or term	SQL Create statement	OS/400 object type	OS/400 object attribute	SQL term
CRTPF	CREATE TABLE	*FILE	Physical File (PF)	Table
CRTLF	CREATE VIEW	*FILE	Logical File (LF)	View
CRTDDMF	CREATE ALIAS	*FILE	DDM File (DDMF)	Alias
CRTLF	CREATE INDEX	*FILE	Logical File (LF)	Index
Field				Column
Record				Row

**Note:** A DDM File represents a Distributed Data Management File. This is the original OS/400 object on the local iSeries server used to provide a link to a file on a remote system. In the context of Table 7-1, an alias created by SQL has no remote system specification. To determine if the DDMF/alias has any remote system specification, you can use the Work with DDM File (WRKDDMF) command.

**Note:** OS/400 supports an object type of table (\*TBL). This object type is for data translation.

Throughout the remainder of this chapter, the SQL terms table, row, and column are used more frequently than their corresponding OS/400 terms file, record, and field. In some cases, both corresponding terms, such as field or column, are used.

## 7.2 DB2 Universal Database for iSeries through Operations Navigator overview

In the Operations Navigator window, click the + (plus) sign next to the **Database** function for the system to which you are attached to see the three major function areas as shown in the left pane and right pane in Figure 7-1.

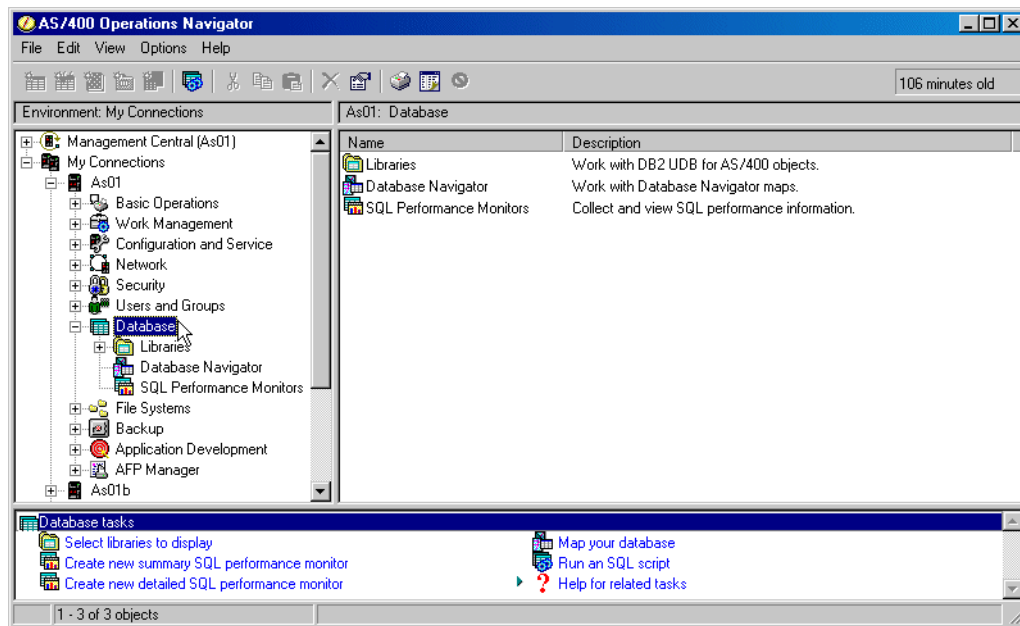


Figure 7-1 Database function list view

There are several other ways to make the same three database function areas also appear in the right pane as shown. This chapter discusses some of these ways. However, Operations Navigator database capabilities are actually grouped under *four* functional branches:

- ▶ Database
- ▶ Libraries
- ▶ Database Navigator
- ▶ SQL Performance Monitors

The following sections summarize the capabilities under each of these four major database function groupings. Examples and tips on usage are given for selected sub-functions under each major function group to highlight Operations Navigator interfaces into the wide range of DB2 Universal Database for iSeries capabilities.

These sections do not explain every action on every pull-down menu, but instead emphasize the actions that are most significant. Such actions as Explore, Open, Shortcuts, and Print options are very similar to these same actions described for Operations Navigator interface in *Managing OS/400 with Operations Navigator V5R1 Volume 1: Basic Functions*, SG24-6226. For some other database-specific actions or options, you must refer to Operations Navigator online help information.

For the database functions described in the following sections, you need the appropriate authority to perform the functions. You can use the SQL GRANT and REVOKE statements to define authority to a table, view, procedure, user-defined functions, and user-defined types. For tables and views, these statements may also specify processing authority, such as SELECT (read), INSERT (write), DELETE, and UPDATE. SQL GRANT and REVOKE can also specify column-level authorities.

The Operations Navigator Database interface supports table, view, index, procedure, column, etc. database-related object levels of authority through the Permissions action by right-clicking the database object name within a library. You can specify permissions for all OS/400 objects, including database-related objects through Operations Navigator File Systems interface.

An alternative to column-level authority is to use an SQL CREATE VIEW to a table or a Create Logical File (CRTLF) command based on a file and specify only certain columns or fields. Then you specify authorities or permissions to the logical file or view. SQL CREATE VIEW or CRTLF can also specify compare values for columns or fields that limit the rows or records that can be seen by those authorized to the view or logical file.

For additional details on the Object Relational Support items (functions and types), refer to *DB2 UDB for AS/400 Object Relational Support*, SG24-5409. For authority implications of using \*SYS or \*SQL naming convention when creating new DB objects with Operations Navigator, refer to document number 9510127 in the *Support Line Knowledge Base* at: <http://as400service.ibm.com/supporthome.nsf/document/10000051>



**Important iSeries software requirements:** Base OS/400 provides SQL “run time support”, not “program development for SQL support”. Run time support includes the following uses of SQL with no SQL software installation required:

- ▶ All Open Database Connectivity (ODBC) support, which includes Operations Navigator functions and Run SQL Scripts jobs and client workstation jobs using Client Access ODBC support, such as a Visual Basic program
- ▶ All Java Database Connectivity (JDBC) support, which includes client workstation Java applets and local iSeries Java servlets accessing JDBC
- ▶ DB2 Universal Database for iSeries support from an already compiled (created) local iSeries program using embedded SQL in the RPG, COBOL, or C program
- ▶ DB2 Universal Database for iSeries support from an already compiled (created) local iSeries program using the SQL Call Level Interface (CLI) in RPG, COBOL, C, or Java
- ▶ Use of the RUNSQLSTM command

To use DB2 Query Manager support or to compile (create) local iSeries programs using embedded SQL, such as iSeries RPG, COBOL, and C programs, you must have licensed program DB2 Query Manager and SQL Development Kit, 5722-ST1 (5769-ST1 for releases prior to V5R1M0). This is for program development support.

## 7.2.1 Database functions overview

In the Operations Navigator window, right-click **Database** to access the pop-up menu shown in Figure 7-2.

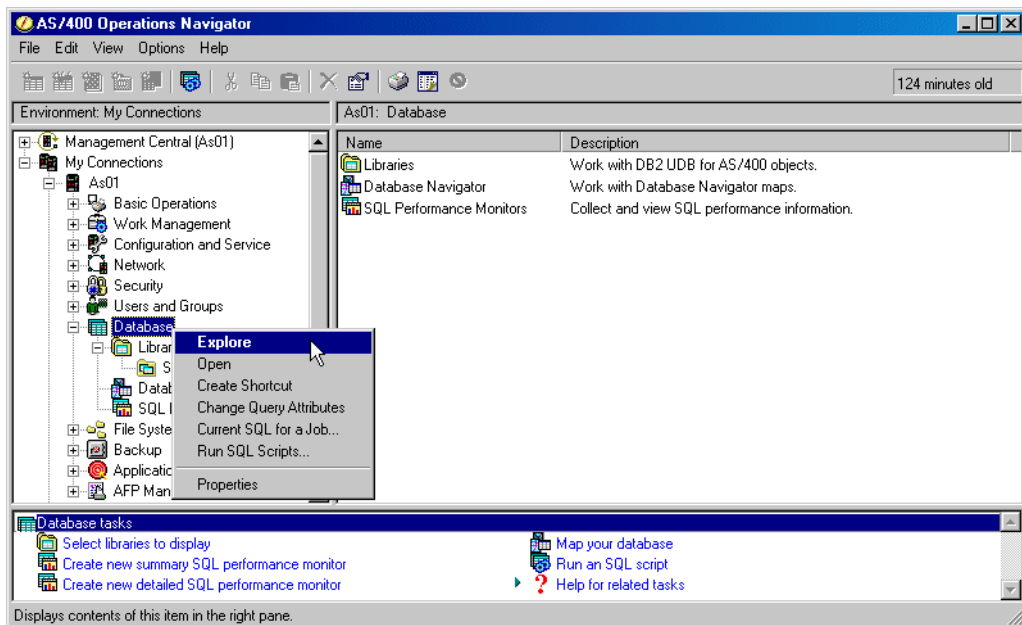


Figure 7-2 Database pop-up menu functions

The possible actions are:

- ▶ **Explore:** The right pane displays the three other major database function areas:
  - Libraries
  - Database Navigator
  - SQL Performance Monitors

- ▶ **Open:** This is the same as choosing *Explore*, except that the contents of the selected file system are displayed in a separate window.
- ▶ **Change Query Attributes:** This enables you to specify attributes for database queries and database file keyed access path (index) builds, rebuilds, and maintenance that are *run in a job*. Query attributes may be specified through the OS/400 Change Query Attributes (CHGQRYA) command.

In Operations Navigator, Change Query Attributes provides a graphical interface to apply a superset (more than CHGQRYA provides) of query attributes as stored in a file. These attributes can be applied to one or more active jobs that can be selected from a list.

OS/400 supplies a read-only version of the query attributes file—QAQQINI in library QSYS. Run SQL Scripts within Operations Navigator defaults to using the QAQQINI file in library QUSRSYS. You must copy the base QAQQINI file in QSYS into library QUSRSYS if you want Operations Navigator to use its values system wide. Or use the following CL command in the Run SQL Scripts window to default your job to another library where you previously copied the QAQQINI file:

```
CL: CHGQRYA QRYOPTLIB (yourlib);
```

If there is no QAQQINI file in QUSRSYS, internal defaults are used.

You can use the *Change Query Attribute* graphical interface to easily make a copy of the default QAQQINI file in a library of your choice and to change the default values to what is most suitable for your job. We document this interface in 7.4, “Change Query Attributes” on page 217. Any changes to the attribute values are typically determined by an experienced query programmer. You can find the best explanation of how to use these attributes in *DB2 UDB for iSeries Database Performance and Query Optimization*, which you can find on the Web at:

<http://submit.boulder.ibm.com/pubs/html/as400/bld/v5r1/ic2924/index.htm>

In addition to CHGQRYA, you can specify a subset of the query attributes available under Operations Navigator through the OS/400 system values QQRVTIMLMT (time limit) and QQRVDEGREE (degree).

**Restriction:** You must have job control (\*JOBCTL) special authority to use this function.

- ▶ **Current SQL for a Job:** With this feature, you can select any active job on the iSeries server and display it through the automatically linked *Run SQL Scripts* option, the SQL statement, if any, currently being executed in the job. In addition to displaying the SQL statement, you can edit or rerun it; you can also display the job log for the selected job or end the job.

This can also be used for database usage and performance analysis, linking into the Visual Explain tool documented in Chapter 10, “Visual Explain” on page 301.

- ▶ **Run SQL Scripts:** This enables you to enter, edit, run, save, and debug SQL statements across tables within all libraries (includes SQL collections).

You can run all supported SQL statements from this action. OS/400 provides a set of base SQL statements for all supported functions that you can select and insert into your SQL statements. You can enter a completely new SQL statement or modify an already available statement for your own unique queries. You can also run CL commands. You can save your own newly created or modified base statements for later use.

You must have appropriate file or table and field or column authorities (permissions) to perform the functions at run time.

Section 7.3, “Run SQL Scripts” on page 197, shows several examples of building and running SQL script.

- ▶ **Properties:** This enables you to specify to refresh the current display every time a list is displayed or after a time interval is specified in minutes.

There are several actions or functions available from the menu bar options for the Database, Libraries, Database Navigator, and SQL Performance Monitors function groupings. This chapter discusses a subset of all of these actions or functions. You must review the online help text for a description of the entire set of actions or functions.

## 7.2.2 Database library functions overview

You can create, delete, and assign permissions (authority) to an OS/400 library under this group of functions. You can also display the list objects within a library and create, change, delete, or assign authorities (permissions) to an SQL table, view, alias, index or OS/400 journal, or OS/400 journal receiver listed within the library.

Figure 7-3 shows an example display after expanding the Libraries function and then right-clicking Libraries to see the context menu.

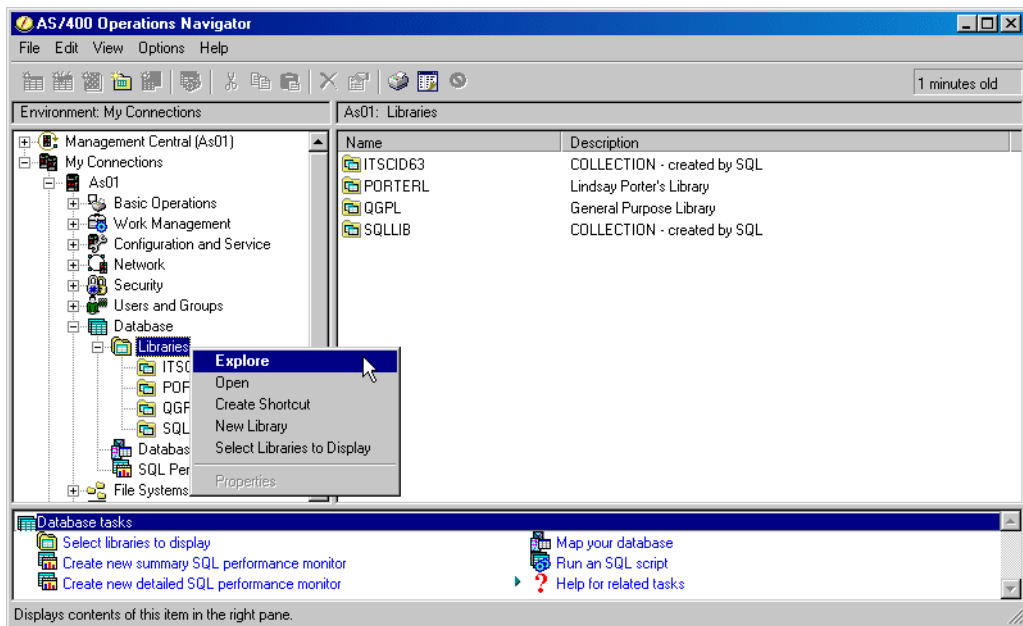


Figure 7-3 Database library actions

In this example, you see the library names ITSCID63, PORTERL, QGFL, and SQLLIB. These libraries were currently specified in the Initial Library List (INLLIBL) parameter of the OS/400 job description object used by the Operations Navigator session to the iSeries server. The job description is associated with the OS/400 user profile you used to sign on under Operations Navigator when connecting to your iSeries server. By default, only the libraries in the user portion of your iSeries library list are included under the Database component, plus any other library you asked to add here in previous Operations Navigator working sessions.

You can add more libraries. Simply click **Select Libraries to Display** in the pop-up window by either entering a library name or selecting from a list of library names on the system. Then, click the **Add** button as shown in Figure 7-4.

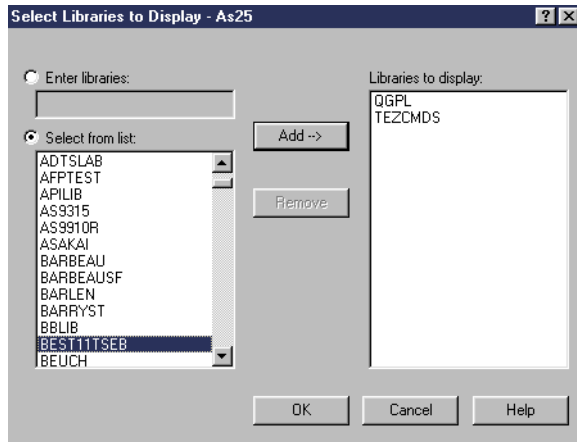


Figure 7-4 Database: Adding a library to your library list

This change is retained across subsequent new Operations Navigator working sessions. This is done by maintaining a table on the host iSeries, QAUGDBLL in QUSRSYS, which lists all users for the Database portion of Operations Navigator and all the libraries that were chosen to work with using this interface.

If you want to remove a library from this list, repeat the previous steps, but select **Remove**.

**Note:** Any library added here may be used when you perform actions or functions under this Libraries function of Database. Any library added here is *not* automatically used by the actions or functions under other database sub-components such as Run SQL Scripts. Nor is it added to the user's library list on the iSeries server. In other words, while the original list of libraries shown in Operations Navigator is built on the user's library list, a change in the library list for this interface is not going to affect the user's library list on the iSeries server.

### 7.2.3 Creating an OS/400 library or collection

There are several Operations Navigator higher level branches from which you can create an OS/400 library. This section discusses creating a library by selecting **Database->Libraries** and going to the **New Library** function as shown in Figure 7-5.

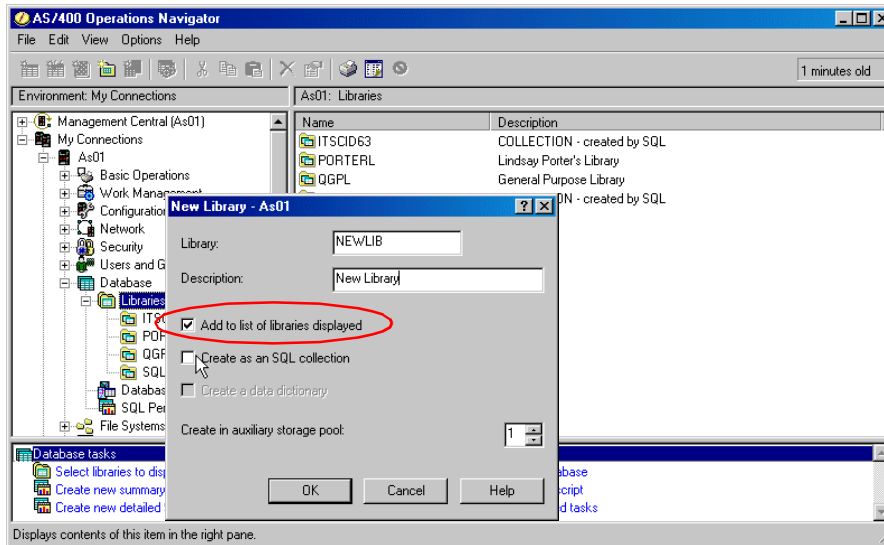


Figure 7-5 Database: Creating a new library

A library can contain any supported iSeries object type. However, under the Operations Navigator's Database interface, you only work with objects related to database support. Under the New Library function, you can create a new library, or you can create an SQL collection. In OS/400, an SQL collection automatically builds a library, and within that library, it creates:

- ▶ A journal
- ▶ A journal receiver
- ▶ A catalog
- ▶ Optionally, a data dictionary

A data dictionary is used for migrated System/36 application environments.

When you select the **Add to list of libraries displayed** check box (circled in Figure 7-5), the newly created library is added to the user's list of libraries in Operations Navigator working session. This has no affect on the iSeries library list.

The library can be placed into the system auxiliary storage pool, ASP1 (default) or a user-defined ASP2 (up to 32). An ASP is a defined set of disk devices that contain only objects created into a library within that ASP. As shipped from IBM, ASP1 contains all disk devices. A user-defined ASP is typically used for a specific performance requirement to reduce disk arm movement or for a specific backup and recovery procedure.

For more information about ASP support, journaling support, and overall backup and recovery, please refer to:

- ▶ iSeries Information Center at: <http://www.iseries.ibm.com/infocenter>  
When you reach this site, select **Backup, Recovery, and Availability**.
- ▶ *Backup and Recovery*, SC41-5304

All database-related objects, such as tables, views, journals, and other system objects, like programs, message queues, output queues, and so on, can be created, moved, or restored into any iSeries library. All iSeries tables or files can be created or moved into an SQL collection if the SQL collection does not contain a data dictionary.

An SQL collection can also contain catalog views that have descriptions and information for all tables, views, indexes, files, packages, and constraints created in the library. All tables created in the SQL collection automatically have journaling performed on them. When referring to an SQL collection in iSeries documentation and screen panels, the collection name and the library name refer to the same object.

## 7.2.4 Library-based functions

Refer to the display shown in Figure 7-6 to see an overview of the functions that are available when you select a specific library. These database functions include:

- ▶ Assigning authorities (Permissions) to the library and objects within the library
- ▶ Totaling the number of files and folders (directories) in the library and the storage size of all objects within the library (Properties)
- ▶ Creating new tables (Table) in the library
- ▶ Creating new views (View) and aliases (Alias) in the library

A *view* is an object that permits access to a subset of all rows in a table and columns within a row. An alias is an object that allows SQL applications to reference a table or view by another name.

In addition, aliases provide an easy way for SQL applications to access data in multiple-member DB2 UDB for iSeries files.

In SQL standards, a table represents only one set of data (rows). OS/400 file support includes multiple members, which are sets of records or rows that contain the same field or column definitions, but different sets of data. For example, the MONTHS file can contain a set of rows for January data (member name JAN) and another set of rows for February data (member name FEB). At run time, a command parameter for Member Name (MBR) could specify JAN one time and FEB another time. Opening an SQL alias provides an equivalent function.

- ▶ Creating new journals to be used with the tables, views, or aliases
- ▶ Creating new SQL procedures
- ▶ Creating new user defined functions (Function)
- ▶ Creating new user defined types (Type)

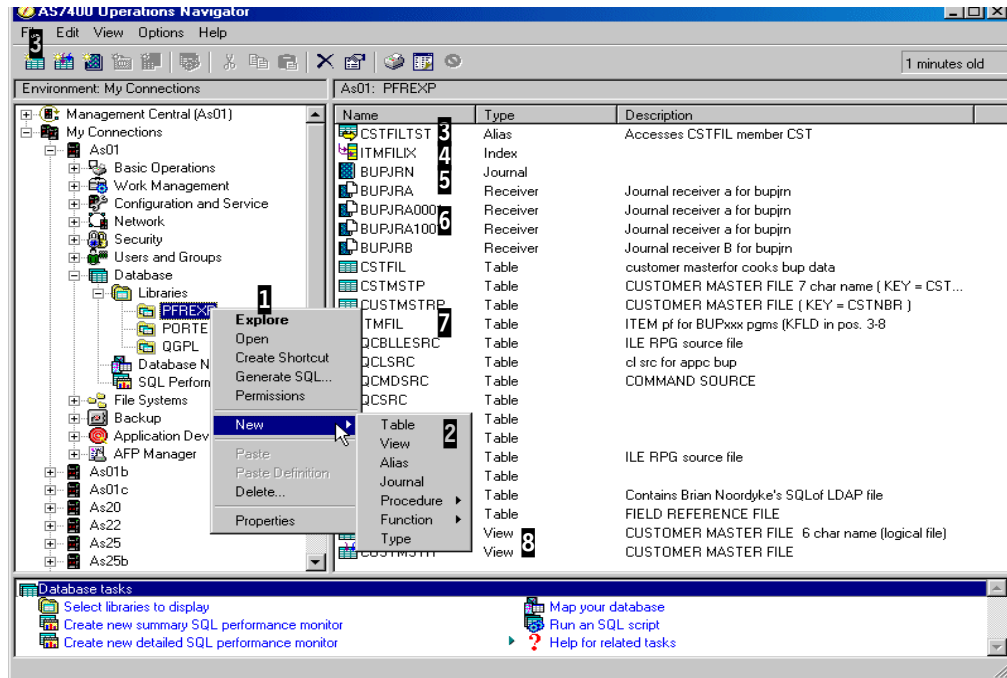


Figure 7-6 Database Library functions

To bring up the Library functions, right-click a specific library (PFREXP in our example). In this example, we double-clicked **PFREXP** or selected the **Open** option in the pull-down menu (1) to see the database-related objects in this library in the right pane.

By selecting **New** in the Library pull-down menu, the next level of objects to create (Table, View, and so forth) are shown in the menu (2).

Before we explain more about creating these objects, we discuss the existing objects shown for library PFREXP.

We created an alias **CSTFILTST** (3), which accesses the **CSTFIL** file, with the member name **CST**. **CSTFIL** is shown as a table (7), but was originally created with the OS/400 Create Physical File (CRTPF) command.

SQL index object **ITMFILIX** is at 4. This object was created, based on the **ITMFIL** table. Section 7.2.5, “Object-based functions” on page 181, explains how to create an index. OS/400 files created through the Create Physical File (CRTPF) and Create Logical File (CRLF) OS/400 commands have access paths (indexes) if key fields are specified, but they are not visible as a separate object of type index.

The **BUPJRN** journal is at 5. Each journal can have one or a pair (dual) of attached journal receivers (where actions on the table or data within the table are actually recorded). **BUPJRA** and **BUPJRB**, as the original dual journal receivers, are shown at 6. In our example, **BUPJRA** and **BUPJRB** already reached their maximum space for journal entry information. Through journal configuration parameters, a second set of dual journal receivers, **BUPJRA1000** and **BUPJRB1000**, have been created by OS/400, with the system generated 1000 suffix. They are now the *attached* (receiving entries) journal receivers.

A series of tables including **CSTFIL**, **CSTMSTP**, **CSTMSTRP**, and **ITMFIL** are shown at 7.

The two views **CSTMST** and **CSTMSTR** are shown at 8.

## Physical file and SQL TABLE differences

The OS/400 Create Physical File (CRTPF) command and the SQL CREATE TABLE statement (implicitly used by the Operations Navigator New->Table dialogue) create an OS/400 object type of \*FILE. There are CRTPF command OS/400 parameters that have no corresponding CREATE TABLE parameter. These parameters are part of every \*FILE object within OS/400 and affect the operating environment when accessing the file or table. Therefore, when you use an SQL-based interface to create a table, OS/400 uses default values for these CRTPF-only parameters. These CRTPF-only parameters include:

- ▶ **Maximum members (MAXMBR parameter):** OS/400 physical files can have multiple members (same record layout and field attributes, different sets of records or rows). All SQL tables default to a value of 1. This is also the default for CRTPF, but the user can specify a number or \*NOMAX (no limit on the number of members).
- ▶ **Member size (SIZE parameter):** OS/400 uses the number of records or rows value to implicitly allocate the initial amount of storage for the file or table. Other values in this parameter optionally specify how to allocate additional storage when the initial storage is exceeded.

CRTPF defaults to 10000 records with up to an additional allocation of up 3000 records in 1000 record increments. A system operator message communicates each additional allocation.

CREATE TABLE defaults to \*NOMAX.

- ▶ **Reuse of deleted record or row storage (REUSEDLT and DLTPCT parameters):** When a row or record is deleted, the storage previously occupied by the record or row remains as part of the total file or table storage allocation.

DLTPCT is the percent of deleted records or rows compared to all active records or rows in the file or table. At file or table close time, if the number of deleted records or rows exceeds this percentage, a message is issued to the OS/400 History Log (viewed with the Display Log (DSPLOG) command).

REUSEDLT specifies to OS/400 whether to insert a new record or row into a new physical storage space (REUSEDLT(\*NO)) or into the storage of a previously deleted record or row (REUSEDLT(\*YES)).

CRTPF defaults to DLTPCT(\*NONE) and REUSEDLT(\*NO). CREATE TABLE defaults to DLTPCT(\*NONE) and REUSEDLT(\*YES).

**Note:** Regardless of the DLTPCT and REUSEDLT parameter values for a file or table, you may have an application environment that you know or suspect may have files or tables with a large number of deleted records (for example, disk storage is increasing with no known increase in the number of new records). In this case, you should consider running the OS/400 Reorganize Physical File Member (RGZPFM) command or its equivalent Operations Navigator Database Reorganize function (see “Managing tables and views” on page 182) on a specific file or table. You can use the DLTPCT parameter message to assist you. Alternatively, you can periodically use the Display File Description (DSPFD) command with TYPE parameter specifying \*MBRLIST to see both the number of records or rows in the file or table and the number of deleted records in each member of the file or table.

You can specify, change, and view the values for these and additional OS/400 parameters for a file or table by using the following OS/400 commands:

- ▶ Create Physical File (CRTPF) command
- ▶ Change Physical File (CHGPF) command
- ▶ Display File Description (DSPFD) command



For more information on these and other file attributes, refer to *Database Programming*, SC41-5701, and *CL Reference*, SC41-5722.

You can view the above mentioned settings and other file or table parameters, such as database constraints and triggers, in Operations Navigator by right-clicking the table and selecting the menu options Table Description and Properties.

## Create Table example

To create a new table (or file) on the iSeries server with the traditional interface, you can use DDS or the CREATE TABLE SQL statement. In both cases, you need the appropriate skill, whether it is programming with DDS or SQL knowledge.

Follow these steps in Operations Navigator to create a new table:

1. Click **Database->Libraries**. Then, right-click the library **PORTERL** in which you want to create the new object. You are presented with a list of choices, as shown in Figure 7-7.

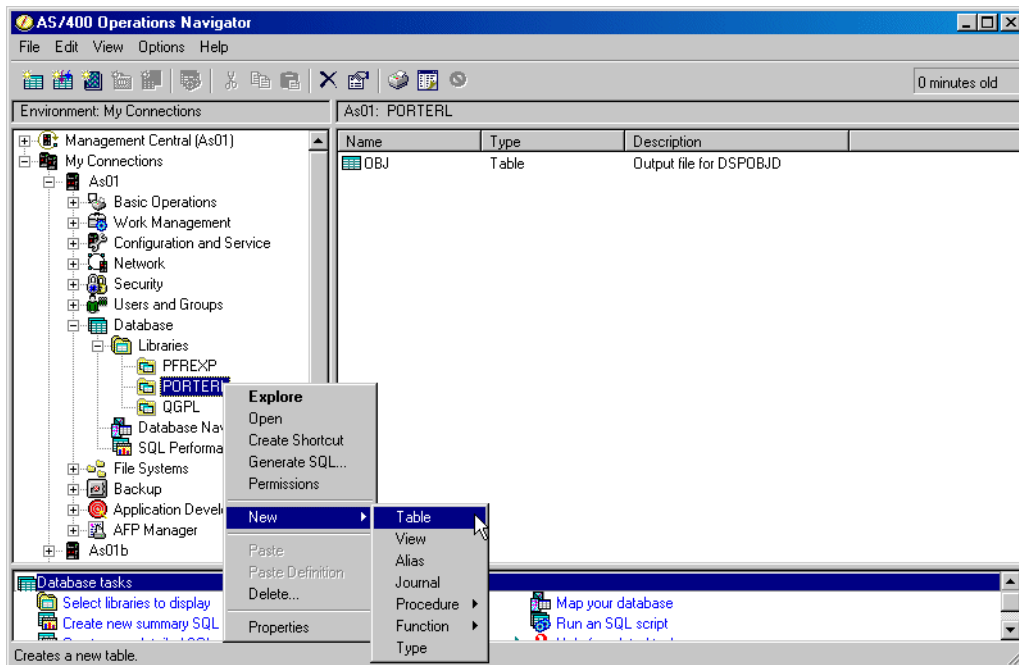


Figure 7-7 Create Table example (Part 1 of 3)

2. Select **New->Table** to access the panel where you specify the table name and description for the new table (see Figure 7-8).

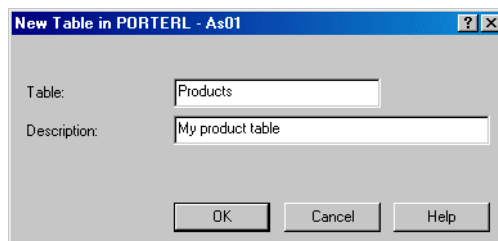


Figure 7-8 Create Table example (Part 2 of 3)

3. Click **OK** and you see the panel where you can specify the columns for the new table (see Figure 7-9). Click the **Insert** button (I) to insert a new column, and specify the column

name, type, length, and an optional description. You can also specify a short name (up to 10 characters), column heading (up to three lines of 20 characters each), must contain a value (not null), default value, CCSID and a length to allocate (for VARCHAR and VARGRAPHIC datatypes).

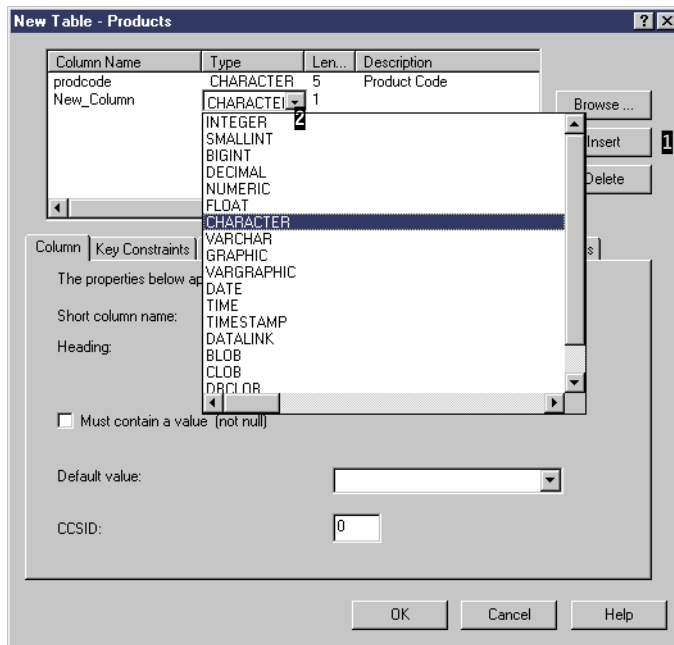


Figure 7-9 Create Table example (Part 3 of 3)

- Use the pull-down list in the **Type** column (2) to choose the data type for the column. The content of this list depends on the version and release of OS/400 installed on your iSeries server. Since V4R4 DB2 UDB for iSeries added support for BLOB, CLOB, DBCLOB, and datalink data types, these values only appear in the list if your iSeries server is running V4R4 or a later release.
- When finished with inserting columns, click **OK** to create the table or select any other item you may need to work on (constraints, indexes, triggers, etc.).

### Create SQL View example

A view is typically used to represent a subset of the columns in a table and, if specified, a subset of the rows in the table.

For example, you have a customer table file that has several columns describing the customer, including customer number (key field), customer description, customer address, and customer telephone number. You want to show someone the customer number, customer name, and customer telephone number, but not their address. You also know that customers with a customer number greater than 500 do not want their telephone numbers known.

The following steps show you how to create a view (CUST\_DIMVU) over the CUST\_DIM table:

- Right-click the library (**PORTERL**), and select **New->View** to access the new view panel shown in Figure 7-10.

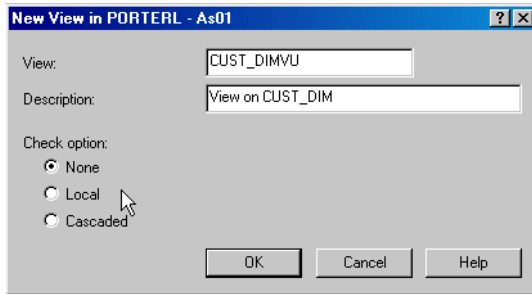


Figure 7-10 Create View example (Part 1 of 6)

2. Enter the name, **CUST\_DIMVU**, for the new view and the description.

The *Check option* specifies whether some type of data validity checking will be performed on an update or insert operation. You must view the help information for additional details. We selected **None** (default) in our example.

3. Click **OK** to see the panel with blank input areas, as shown in Figure 7-11. Click the **Select Tables** button (1) to bring up the current library list for your current Operations Navigator working session.

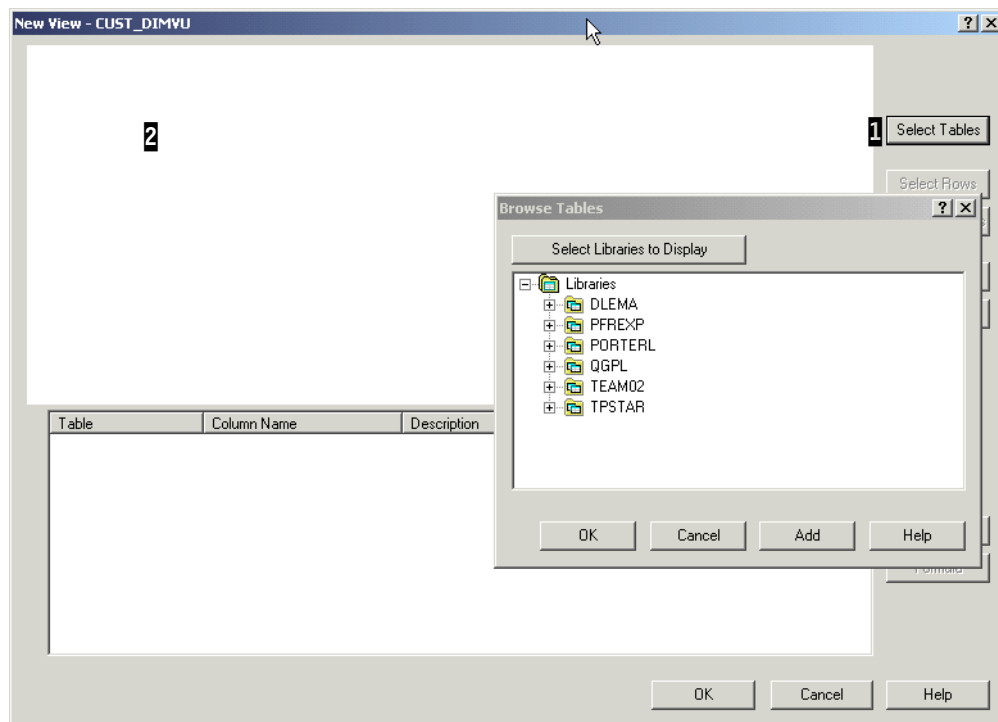


Figure 7-11 Create View example (Part 2 of 6)

4. Previously we selected library PORTERL to create the view. However, the view can be created to use tables in various libraries. To keep this example simple, we select the tables from library PORTERL.
5. To see the tables within a library, either click the + (plus) sign next to the library name or position the mouse on the library and double-click. Select the table, and click the **OK** button, which places the column names in the upper pane in the area (2) in Figure 7-11).
6. As shown in Figure 7-11, select your table from the library. We selected the **CUST\_DIM** table from the PORTERL library, and clicked **OK**. We can select another table from the

library. We selected the **PART\_DIM** table from the PORTERL library and then clicked the **Add** button.

This places the columns of both the CUST\_DIM (1) and PART\_DIM (2) tables into the upper pane (see Figure 7-12). We chose two tables to show an example of how Operations Navigator assists you in building SQL statements that could become quite complex.

As shown in Figure 7-12, we selected the **CUSTKEY** and **CUSTOMER** columns from CUST\_DIM and dragged and dropped them into the lower pane. You can see the arrow to the left of the CUSTOMER column (3), which indicates that the next column selection will be inserted after this statement.

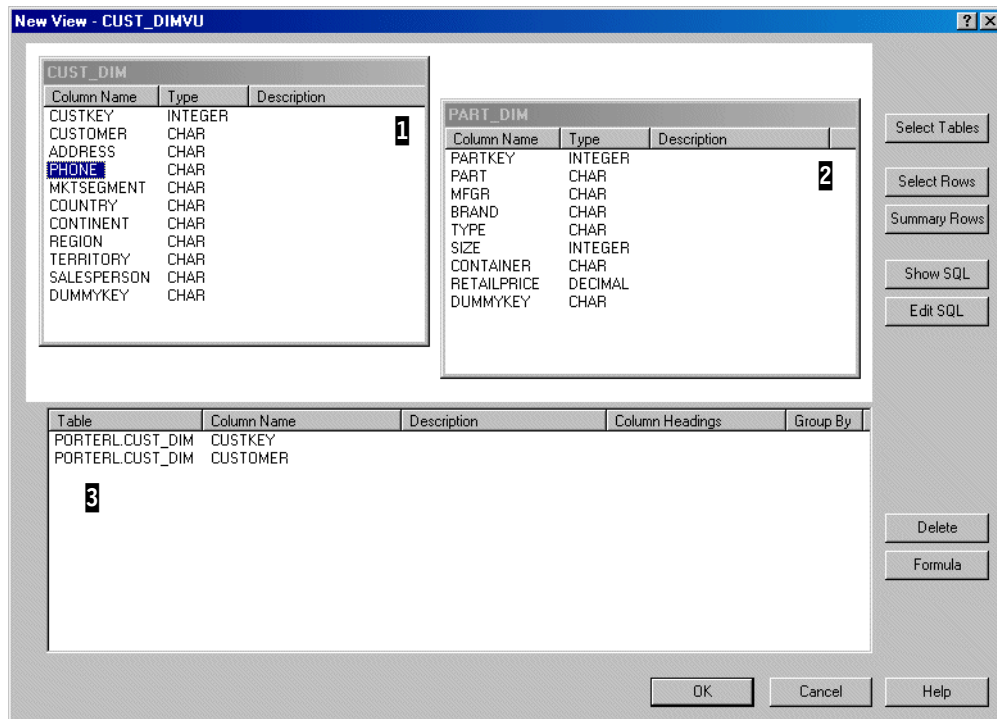


Figure 7-12 Create View example (Part 3 of 6)

You can reposition this arrow for the next insert of a new column by clicking any existing column in the lower pane. In this example, we selected the PHONE column, but have not yet dragged it to the lower pane.

In this example, we create a view using only columns from the CUST\_DIM table. If you select multiple tables to appear in the upper pane, Operations Navigator expects a JOIN clause in the VIEW statement and issues a message indicating this later if you continue showing more than one table in the upper pane. Since we are only going to use the CUST\_DIM table, we select the **PART\_DIM** table in the upper pane and press the Delete key to delete this table from the upper pane. The PART\_DIM column names no longer appear in the following displays.

- As shown in Figure 7-13, we completed a column selection for the CUST\_DIMVU view and clicked the **Select Rows** button. The **Select Rows** button enables a WHERE clause. The Select Rows window shows the table columns, operators, and functions available in the upper pane. Once a column operator or function is selected (by double-clicking), it is inserted into the Clause pane. You may also manually enter your own text into the Clause area as we did by entering the value 500.

**Note:** If you click the Summary Rows button, the HAVING clause is enabled.

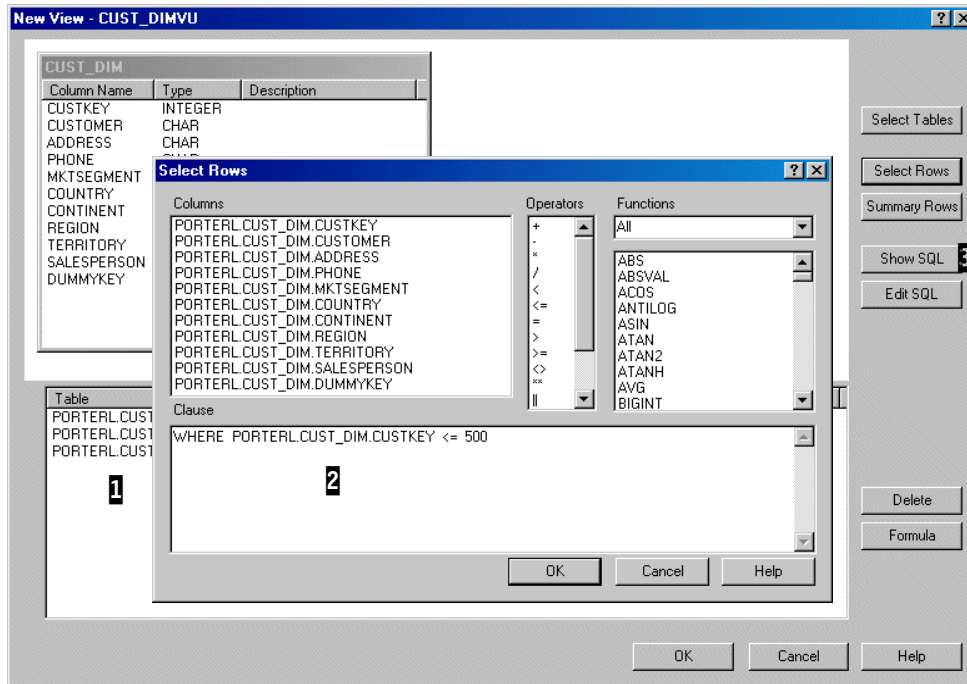


Figure 7-13 Create View example (Part 4 of 6)

As soon as you have at least one SQL column in the Table pane (1) or text in the Clause pane (2), you can use the Show SQL (3) button to view the current SQL statement.

We clicked the **Show SQL** button to generate the Show Generated SQL window shown in Figure 7-14.

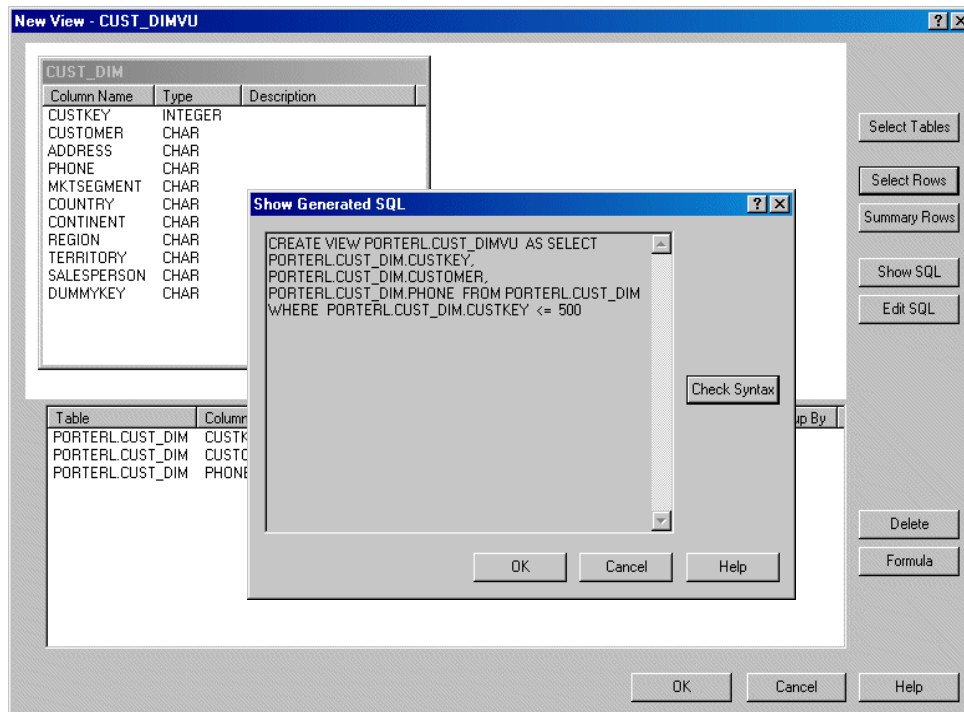


Figure 7-14 Create View example (Part 5 of 6)

8. In this window, click the **Check Syntax** button to view the generated SQL and have syntax checking performed. You cannot edit any text on this window.
9. If you are satisfied with the current SQL statement, you can click the **OK** button twice on successive windows, and the View is created, assuming no errors are detected. Depending on your Operations Navigator refresh setting, a new view appears in an updated display showing the contents of the library, such as the example shown in Figure 7-7 on page 171.
10. To edit the generated SQL, click the **Edit SQL** button (1), which opens the Edit Generated SQL window shown in Figure 7-15.

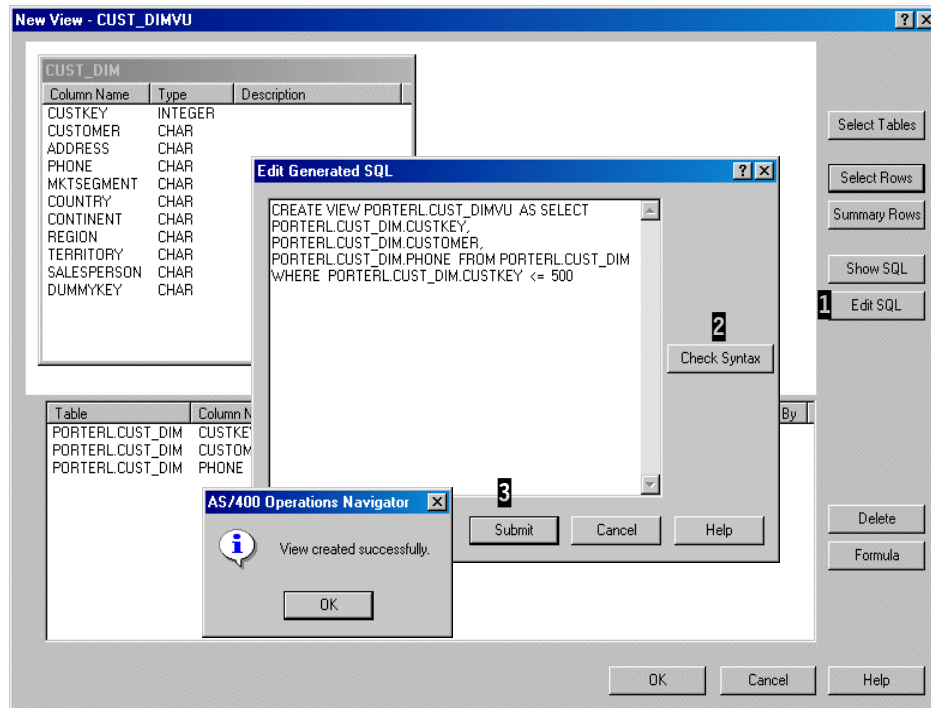


Figure 7-15 Create View example (Part 6 of 6)

11. In Figure 7-15, the SQL statement area now has a white background. Here, you can enter any characters and also have your syntax checked by using the **Check Syntax** button (2).
12. After we validated the SQL syntax, we clicked the **Submit** button (3) in Figure 7-15). Then, the view was created successfully as indicated by the Information window shown.

**Edit SQL tip:** If you make changes through this Edit SQL process, the changes are not saved. You may make changes and successfully create the view as we have done by using the Submit button. However, the changes are not saved in this dialogue because you must exit the Edit SQL function by clicking the **Cancel** button or using the Windows cancel (X button).

SQL changes are not saved because they could be extensive. You can even change the name of the view and the library that is already specified.

## Create journal example

A journal is an object used to record actions on database tables or files and other objects or software that support journaling, such as system auditing. For DB2 UDB for iSeries, journals are typically used to recover from application errors or unscheduled iSeries server outages. Commitment control, as discussed in 7.3.1, “ODBC and JDBC connection” on page 202, requires journaling to implement its COMMIT and ROLLBACK functions.

OS/400 uses the journal object as a *front-end interface* to an attached object, which is a journal receiver that actually contains the journaled data. Each set of related journal data is recorded as a *journal entry*.

Examples of non-DB2 UDB for iSeries software functions that optionally use journals and journal receivers include:

- ▶ OS/400 security: Action auditing
- ▶ OS/400 job accounting
- ▶ TCP/IP-based functions, including IP filters, IP network address translation (NAT), and virtual private network (VPN)
- ▶ OS/400 software license management tracking

Applications can also use OS/400 commands and System Application Program Interfaces (APIs) to write to and read journal entries.

OS/400 supports defining and using *remote journals* as well. A journal associated with a local journal can be defined to reside on a remote iSeries server. The remote journal can be defined so that OS/400 automatically sends journal entries made on the local iSeries server to the corresponding remote iSeries server journal. The primary intent of remote journal support is to quickly and easily replicate data onto a backup iSeries server in a high availability environment where the backup iSeries server can switch over to become the production iSeries server, if an unscheduled outage occurs on the primary iSeries server.

To create and set up remote journaling through Operations Navigator, you must first create the local journal and journal receiver. Then use the Properties support for the journal to access actions that set up a remote journal. “Managing journals and journal receivers” on page 192 shows an example of journal and journal receiver properties.

The following example shows how to create a local journal (CUST\_DIMJ) in the PORTERL library and create its associated journal receiver in the JRNLIB library:

1. Right-click the **POTERL** library. Select **New->Journal** to access the New Journal panel (1 in Figure 7-16).

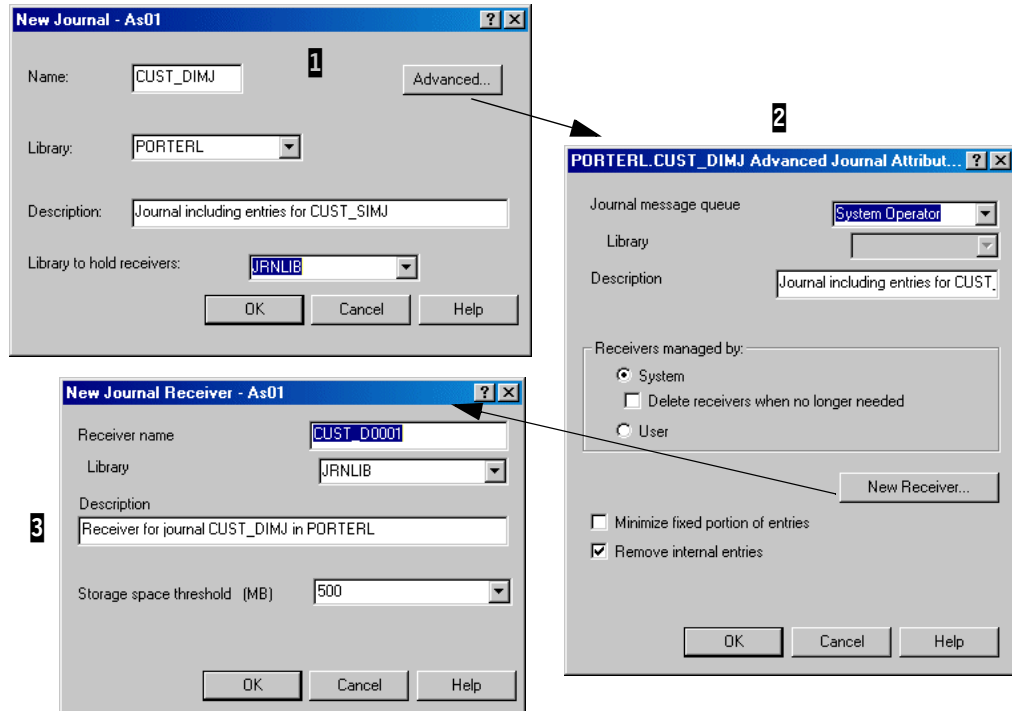


Figure 7-16 Creating the Journal and Journal Receiver

2. In the New Journal panel, enter the journal name, library, and description. Name the library to hold the journal receiver. You can select a library from a list of the current Operations Navigator session's library list, except for the library named to contain the journal. In our example, the PORTERL library would not appear in the list. Although you can place a journal receiver in any library you want, including PORTERL in our example, the OS/400 recommendation is to place the journal receiver in a library separate from the library that contains the journal itself.

Another recommendation for OS/400 journaling support is to place the library used for the journal receivers in its own user-defined ASP.

In our example, we specify library **JRNLIB** to emphasize a different library for the receiver. JRNLIB must already exist.

3. Click the **OK** button in the New Journal panel (1). The journal is created, along with an attached journal receiver with a default name and default attributes.

If you click the **Advanced** button, you see the Advanced Journal Attributes panel (2 in Figure 7-16). You see the default attributes that were used in our example to create the CUST\_DIMJ journal. If you click the **New Receiver** button, you see the New Journal Receiver pane (3 in Figure 7-16), which shows the default new journal receiver attributes.

Once the journal is created, right-click it and select **Properties** from the drop-down menu. Then the Properties window appears as shown on the right-hand side in Figure 7-17. On this window, you can start journaling for a table or a group of tables. To do this, click the **Tables** button.



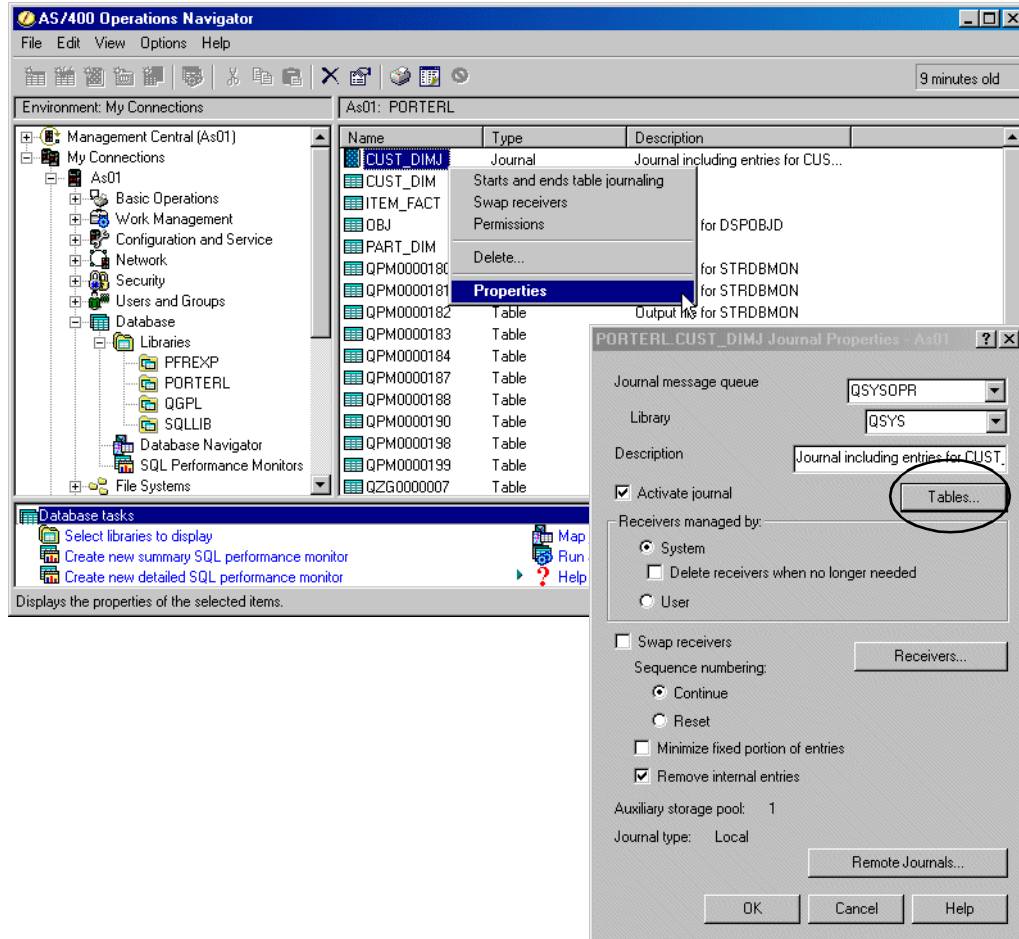


Figure 7-17 Selecting the tables to journal (Part 1 of 2)

When you click the Tables button, the Start/End Journaling panel (Figure 7-18) appears. On this display, select the **CUST\_DIM** table and click the **Add** button to the left of the Tables to journal pane to add it to the list of tables to be journaled in the CUST\_DIMJ journal. Click **OK** to start journaling the CUST\_DIM table.

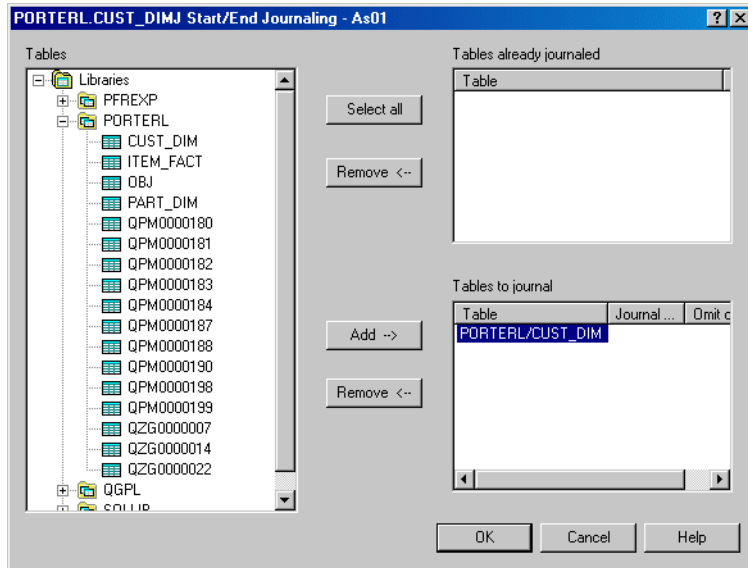


Figure 7-18 Selecting the tables to journal (Part 2 of 2)

The following summary describes the key journal and journal receiver attributes. For a full discussion on these journaling attributes, refer to *Backup and Recovery*, SC41-5304.

### Advanced journal attributes

Listed here are the advanced journal attributes. Refer to the Advanced Journal Attributes window (2) in Figure 7-16).

- ▶ **Journal message queue:** OS/400 issues specific messages for specific changes to the journaling environment. The typical reason for a journaling message is when a journal receiver is reaching its threshold of maximum entries, a message is issued indicating that the current receiver should be detached and a new, fresh journal receiver should be attached.

The default message queue is “System Operator”, which is actually message queue QSYSOPR.

In some environments, you may choose to manage your own journaling support, or you may have an application that manages the journaling through software. In those cases, you may want to use a message queue other than QSYSOPR.

- ▶ **Receiver managed by – System:** By clicking System, you tell OS/400 to automatically detach the current journal receiver and attach a new one when the journal receiver storage space threshold has been reached or when the attached journal receiver’s sequence number has reached a value of 1 TB. Each time the system attaches a new journal receiver to the journal, the journal receiver sequence number is incremented by one. In addition, the system resets the receiver sequence number during IPL, provided the receiver is not required for commitment control recovery. See Commit mode under 7.3.1, “ODBC and JDBC connection” on page 202, for information on commitment control.

Under system managed receivers, you can also specify that OS/400 delete receivers when they are no longer needed. If you do not choose this option, the detached receivers remain on the system until you delete them.

- ▶ **Receiver managed by – User:** By clicking User, you assume the responsibility for changing journal receivers and determining when to delete receivers you no longer need.
- ▶ **Minimize fixed portion of entries:** By clicking this option, you remove job, program, and user profile information from each journal receiver entry. In a busy journaling environment,

this can significantly reduce storage space required, but restricts selectivity by other OS/400 journal entry support.

- ▶ **Remove internal entries:** Depending on what is being journaled, OS/400 sometimes puts its own entries into a journal receiver. By selecting this option, OS/400 deletes these entries from the journal receiver when the system determines they are no longer needed.

One good example of these internal entries is those made to support System Managed Access Path (table index) Protection (SMAPP) support. SMAPP journals changes to access paths (that is, key columns or fields) independent of whether you use journaling of database tables or files. SMAPP is intended to minimize access path recovery following an abnormal system termination. Journaling access path changes helps SMAPP do this.

To enable SMAPP, you use the OS/400 Edit Recovery for Access Path (EDTRCYAP) command as explained in *Backup and Recovery*, SC41-5304.

### ***New journal receiver attributes***

Listed here are the new journal receiver attributes. Refer to the New Journal Receiver window (3) in Figure 7-16 on page 178).

- ▶ **Journal name and description:** Enter here the journal receiver name and journal receiver descriptive text. As shown, the journal name and description are the default values generated by Operations Navigator. These values are used if you never select the New Receiver button in the Advanced Journal Attributes pane.
- ▶ **Library:** Enter the journal receiver library. The default value shown (JRNLIB) was specified on the initial New Journal panel (1) in Figure 7-16 on page 178).
- ▶ **Storage space threshold:** Enter the maximum storage in megabytes that the journal receiver can take. You see the default value of 500 MB. The value 500 MB is specified as 500000 KB on the corresponding OS/400 Create Journal Receiver (CRTJRNRCV) command Threshold parameter.

The number of journal receiver entries this space can contain depends on the amount of data contained in each entry. When this threshold is reached, a message is sent to the message queue specified in the window pane (2) in Figure 7-16 on page 178). See the online help information for additional details.

In addition to the powerful Operations Navigator interface for creating and managing journals and journal receivers discussed in this section and in 7.2.5, “Object-based functions” on page 181, there are several OS/400 journal creation and management commands. To view these commands and access the related online 5250 display-based help information, enter the following command on a 5250 command line:

```
GO CMDJRN
```

## **7.2.5 Object-based functions**

When you right-click a specific database-related object, a pull-down menu appears with functions that are unique for that object type. At this specific object-level interface, you have some additional create functions and a wide range of management functions. Object-based functions for a database include:

- ▶ Managing a table and view
- ▶ Adding and managing constraints and triggers for a table
- ▶ Assigning and changing authorities and permissions to these objects
- ▶ Creating and managing an index for a table
- ▶ Managing a journal
- ▶ Adding and managing an associated journal receiver or a remote journal

## Managing tables and views

Right-clicking a table brings up a menu similar to the example shown in Figure 7-19.

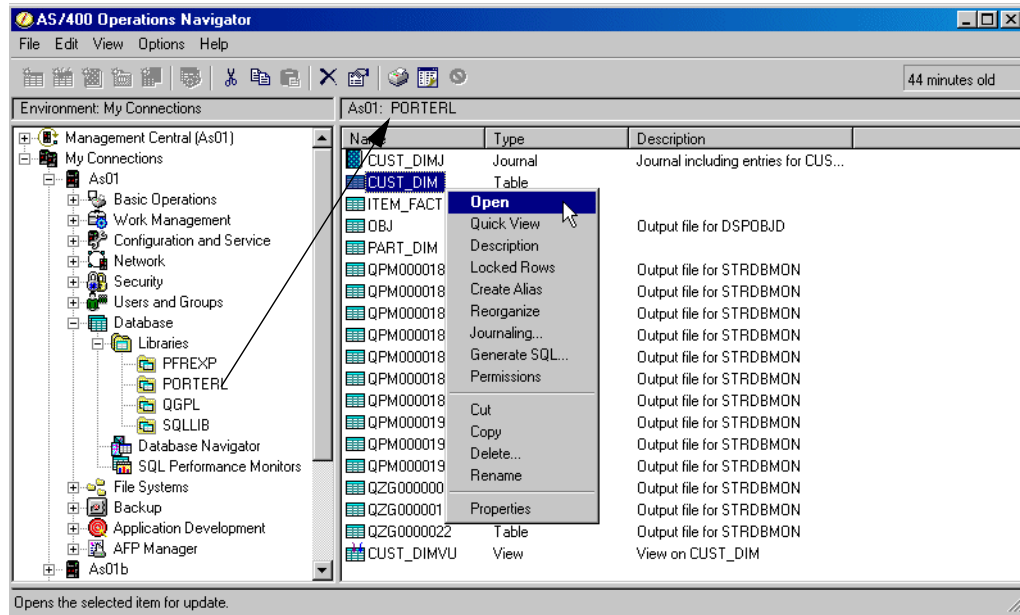


Figure 7-19 Managing table actions

For the CUST\_DIM table, these are the actions:

- ▶ **Open:** This displays, in the right pane, the first “n” rows of the table. The number of rows displayed and the number of columns displayed for each row depend on the window size, which can be adjusted to be shorter or longer (less or more rows) or narrower or wider (less columns or more columns). With the right permissions, you can update columns, delete rows, and insert new rows.

OS/400 issues an error message if you try to make invalid changes to a table. See “Open table example” on page 183.

- ▶ **Quick View:** This displays the table data as Open does, but is a read-only view. No changes can be made to the data.
- ▶ **Table Description:** Using this item, you can gather similar information as you would using the DSPFD command on the iSeries server. However, in Operations Navigator, you are also allowed to change some attributes such as Reuse of Deleted Records and Share Open Data Path. See “Table Description example” on page 184 for details.
- ▶ **Locked Rows:** When the table is in use, this displays whether records are locked, their relative number, which job (fully qualified job name) is actually locking them, and whether the lock type is Read or Update. From this panel, it is also possible to access the locking job’s job log, check what SQL statement is used, and copy it to a Run SQL Scripts instance to work with it. The interface also allows you to end the locking job. “Working on locked rows” on page 196 explains how to use it.
- ▶ **Create Alias:** An alias is an object that allows SQL applications to reference a table or view by another name. In addition, aliases provide an easy way for SQL applications to access data in multiple-member iSeries physical files.
- ▶ **Reorganize:** This enables you to reorganize the rows within the table according to a specified table key or a named index, or by compressing storage currently occupied by deleted rows.

If your application frequently inserts new rows and then deletes them, such as in a work file, you should consider using the compression of deleted rows function.

**Note:** Although both the OS/400 Create Physical File (CRTPF) command and SQL CREATE TABLE create an OS/400 object of type \*FILE, there are CRTPF command parameters that have no corresponding SQL CREATE TABLE parameter. Therefore, creating a table either via CREATE TABLE or by using the Operations Navigator New->Table interface requires OS/400 to use default values for these physical file parameters. One such parameter is the Reuse deleted record storage (REUSEDLT) parameter. See “Physical file and SQL TABLE differences” on page 170 for notes on creating a new table.

New starting in  
V5R1

- ▶ **Journaling:** This option displays information about any journal that is currently or last associated with a table. If the status shows “Never journaled”, you can start journaling the table by specifying the name of an existing journal in an existing library, selecting the **Journal images before change** option (to journal both before and after images), and clicking **Start**.
- ▶ **Generate SQL:** Create SQL source statement for the table. See Chapter 10, “Visual Explain” on page 301.
- ▶ **Permissions:** This enables you to view and change user profile and public authority or permissions to the table and its columns. See the security chapter in *Managing OS/400 with Operations Navigator V5R1 Volume 1: Basic Functions*, SG24-6226, for a general discussion on Operations Navigator Permissions support.
- ▶ **Cut:** This enables you to select a database object and drag and drop it to a different library. When the drop is completed, the database object is deleted (cut) from the original library.
- ▶ **Copy:** This enables you to select a database object and drag and drop it to a different library or in the same library. When the drop is completed, the database object exists in both the source and the target libraries.
- ▶ **Delete:** This enables you to select a database object and permanently delete it after you confirm the delete of the object.
- ▶ **Rename:** This enables you to select a database object and rename it.
- ▶ **Properties:** This enables you to select a database object and display its properties. Different property values are displayed depending on the object type. Also, depending on the object type, you may be able to add, change, or remove property values. For example, when you click Properties for an SQL-created view, you can see a read-only view of the SQL used to create the view. If you click Properties for a view created by Create Logical File (CRTLF) command, you see only a message panel that states there is no SQL statement available.

### Open table example

Figure 7-20 shows some example windows when performing an insert, delete, or update to a table through Operations Navigator. This is the equivalent of using the Update Data (UPDDTA) command to make changes to the records in a physical file.

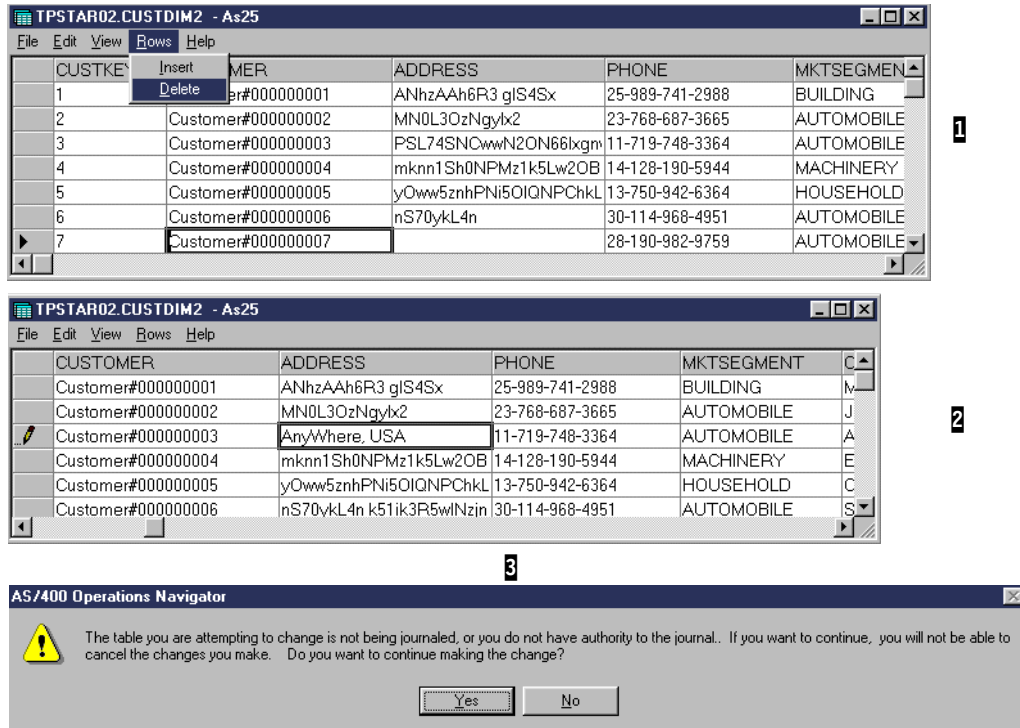


Figure 7-20 Open table example

In the Insert or Delete window (1), you can insert a completely new row or delete an existing row, such as row 7 (customer number 7) in this example. For insert, you must enter the data according to each column's valid data format or you receive a warning message. If you attempt to delete a row or update a column in a row (update window (2)), you see a warning message window similar to the one that is shown (3). This message cautions about recovering the original data if the table is not being journaled.

### Table Description example

Right-click a table and select **Table Description** to see context information similar to what you see with the OS/400 DSPFD command.

This interface provides information formatted on different notebook pages and is structured as follows:

- ▶ **General tab** (Figure 7-21): Displays such information as the member name, its description, size, number of current and deleted rows (therefore, making it easy to judge whether reorganize can be useful), and maximum percentage of deleted rows allowed. Plus it gives you the ability to change the Reuse Deleted Rows (REUSEDLT) parameter and the table description.

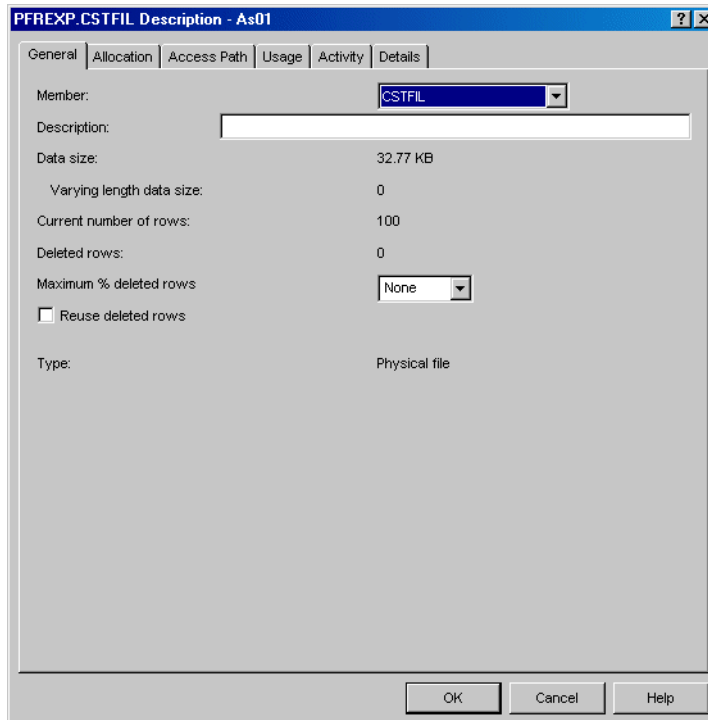


Figure 7-21 Table Description panel: General

- ▶ **Allocation tab:** Allows you to verify if there is a maximum number of rows set on the table, what was the initial number of rows, its subsequent increase, if there is a value set for forcing the writing of updates to auxiliary storage, and to change these settings.
- ▶ **Access Path:** Shows the current size of the access path, the maximum size, the maximum key length, whether the access path is valid or shared, whether it is journaled, what the maintenance and recovery of the access path is set to, and other options for further details.

**Note:** The Access Path tab is only visible for non-SQL described files. For SQL files, use the **properties** option to see indexes and views or display their individual descriptions.

- ▶ **Usage tab:** Contains information on creation, modifications, backups and if this files data can share open data paths across jobs (SHARE \*YES/NO).

The *Share open data path* check box offers an easy way for you to change this setting for the table.

- ▶ **Activity tab:** Allows you to record the level of activity, documenting such information as the number of insert, update and delete operations, the logical and physical reads, clear operations, index builds/rebuilds, a full open and close, reorganize operations, and the number of rows being rejected in open operations (by key, non-key, group by/having selection methods).

This tab also contains important information regarding the number of *valid* and *invalid indexes* built over the table.

- ▶ **Table Details** tab (Figure 7-22): The last tab provides information on:
  - Creation
  - Number of allowed members

- Maximum time a program is to wait for the file and its rows to be available
- Maximum row length
- Sort sequence
- Language identifier
- Format level check and identifier
- Allowed activity level
- Unique identifier for this table in the system
- Disk pool it is using and if it is a distributed file

Some of the above mentioned values can be changed using this interface.

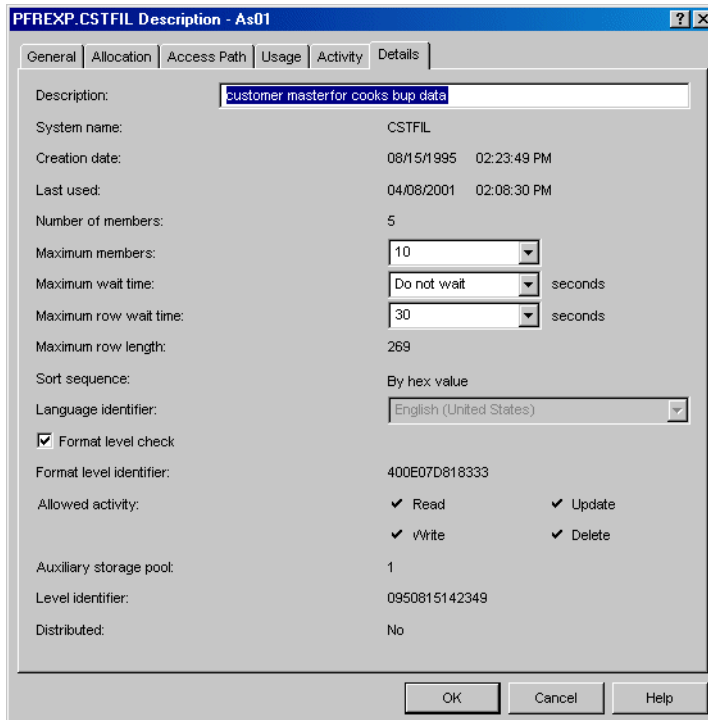


Figure 7-22 Table Description: Table details

**Changing properties:** You must ensure that proper authorization or permission has been given to the Operations Navigator user to access the Table Description and Properties function for the object. You must also ensure the authorized user understands the importance of any table modifications they make. For example, the properly authorized user can delete fields or columns, and therefore, lose the associated data.

Programs created (compiled) against the table that has a field or column added or removed may encounter an error during the next file or table open function. Through the Create Physical File (CRTPF) or Change Physical File (CHGPF) command, you can specify the Level Check (LVLCHK) parameter. A table with LVLCHK(\*YES) specified detects the added or removed column during file open. Re-creating the program usually resolves the problem if the program does not need to use the column.

A program that may have already been performing column validity checking performs unnecessary duplicate processing if a check constraint is added to the table.



## Table Properties example

Right-click a table, and select **Properties** to display all the table properties. We use the initial properties panel (Column information) as shown in Figure 7-23 to discuss table properties:

- ▶ Column properties
- ▶ Key constraints
- ▶ Indexes
- ▶ Referential constraints
- ▶ Triggers
- ▶ Check constraints

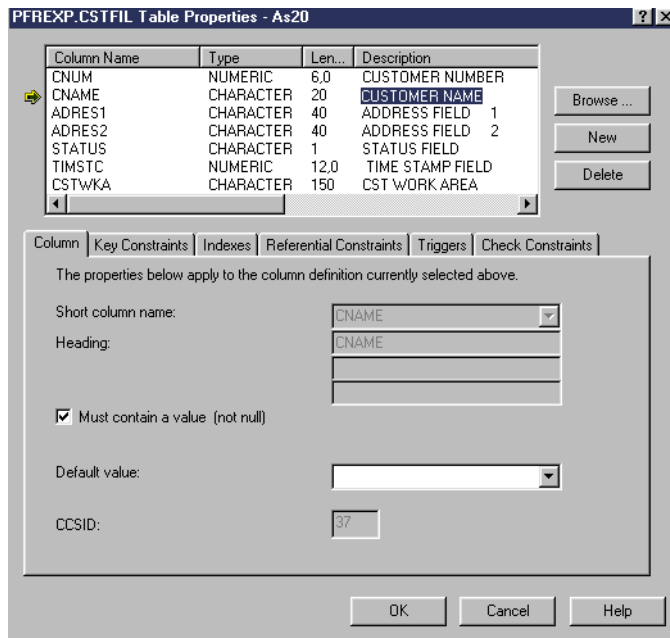


Figure 7-23 Table Properties example

### Column properties

As shown in Figure 7-23, we moved the cursor to the CNAME field or column, as indicated by the arrow to the left of the column and the highlighted column description. In the upper column list, you see the column data type and length. In the lower pane, you see some information about the column, including the Coded Character Set Identifier (CCSID).

The CCSID numeric value specifies how character data is stored on your system. For user-created tables, the character data is defaulted to be stored in the format according to your primary language ID. For example, on the systems used for this redbook, the OS/400 Language ID system value QLANGID is set to ENU – English for United States (uppercase and lowercase). The default CCSID value for ENU is 37, as shown in our Properties example. For more details on CCSID support, refer to *AS/400 National Language Support*, SC41-5101.

The Browse button leads to a dialogue in which you can view other tables that you may want to use as a base definition to add (copy in) a new column to table CSTFIL.

The New button enables the Column window shown at the top of Figure 7-23 to accept a new column definition. In the Column window, you enter the appropriate definition information. Select a column in the Column window, and click the **Delete** button to remove the column from the table.

You can make other changes or additions to the table and, when finished, click the **OK** button to make the changes permanent. The changes or additions are run as if you entered the ALTER TABLE SQL statement. If the table was created with the CRTPF command, the original file is deleted and the new file is recreated. The field or column deleted also deletes the associated data.

### **Key Constraints**

Constraints place some controls on the action to an object or portion of an object. This Key Constraints tab enables you to add, modify, view, or delete the primary key and unique keys for a table. You may modify a constraint if it was defined during your current table editing session. If you added the constraint and then clicked OK on either the New Table dialog or Table Properties dialog, you may only view the constraint.

A unique constraint is the rule that the values of the key are valid only if they are unique. Unique constraints can be created using the CREATE TABLE and ALTER TABLE statements. Unique constraints are enforced during the execution of INSERT and UPDATE statements.

A PRIMARY KEY constraint is a form of the UNIQUE constraint. The difference is that a PRIMARY KEY cannot contain any nullable columns.

### **Indexes**

Indexes are your specific definition of key fields or columns and the order of those fields or columns within the complete key. During performance analysis, the OS/400 query optimizer may issue a job log message that recommends a new index be created to improve performance. You can use SQL CREATE INDEX or this tab dialogue to create a new index.

The Indexes tab enables you to add modify, view, or delete an index for the table with which you are currently working. You may modify an index only if it was defined during your current table editing session. If you added the index and then clicked OK on either the **New Table** button or Table Properties button, you can only view the index.

### **Referential Constraints**

A referential constraint is where one or more columns of a table refer to values of columns in the table you are currently working on or another table that is referred to as the *parent table* for the current table.

The Referential Constraints tab enables you to add, modify, view, or delete referential constraints for the table on which you are currently working. You may modify a constraint only if it was defined during your current table editing session. If you added the constraint and then clicked OK on either the New Table button or Table Properties button, you may only view the constraint.

### **Triggers**

DB2 Universal Database for iSeries has supported native high-level language (HLL) system (external) triggers since V3R1. A *trigger* is program to initiate an action (trigger) when an event occurs on a database file/table (insert, update, or delete). Triggers can be initiated either before or after the event. Update triggers can differentiate whether a record/row was actually changed.

You need to be cautious when using triggers. They offer powerful functions without knowledge of the current program, but they are called synchronously. If they do too much work before returning control to the original program, you may observe performance degradation.

SQL Triggers are new for V5R1. For SQL Triggers, SQL code is used to create the trigger using SQL syntax. With native triggers, a program name is specified to execute (which could, of course, contain SQL).

New for V5R1 is the support for up to 300 triggers per table. You are now provided with the option to add or replace triggers when you associate a trigger with a database table. Also new in V5R1 is the support for READ event system (external) triggers only – a trigger program that executes when a record is read from a database table.

Since DB2 Universal Database for iSeries now supports both system (external) and SQL Triggers, the Operations Navigator Database component now interfaces to both kinds of trigger.

Trigger definition and properties are part of the table Properties dialogue. See Figure 7-24. The Properties page for triggers has been changed to display a list of triggers for the table, which could include a mixture of both Native and SQL Triggers. Then the user can select a trigger and go to a more detailed Properties dialogue for that specific trigger. The properties dialogues are different for native and SQL Triggers.

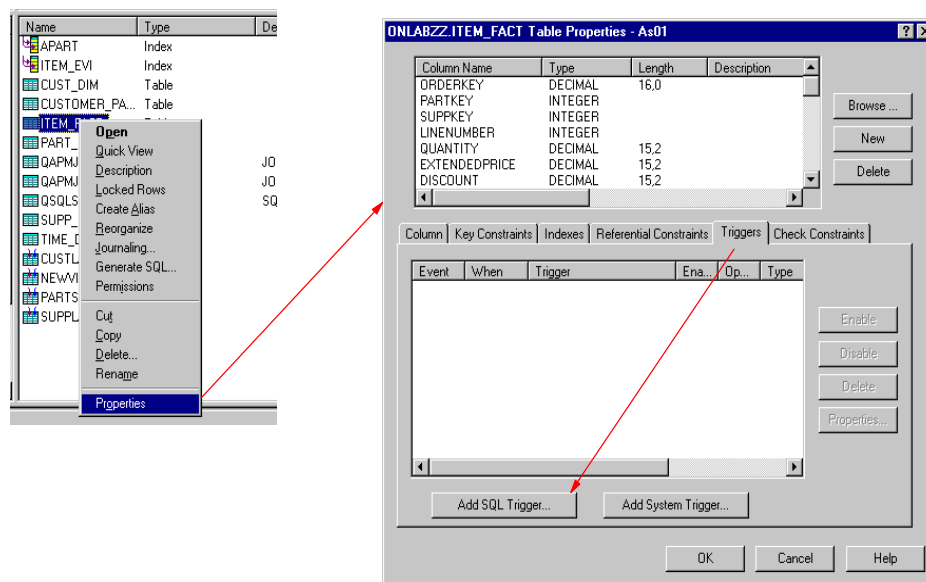


Figure 7-24 Defining an SQL Trigger

From the General tab screen (Figure 7-25), you can add an SQL Trigger to your table, specify the event to fire the trigger, and indicate whether the trigger applies to the whole table or just the selected columns.

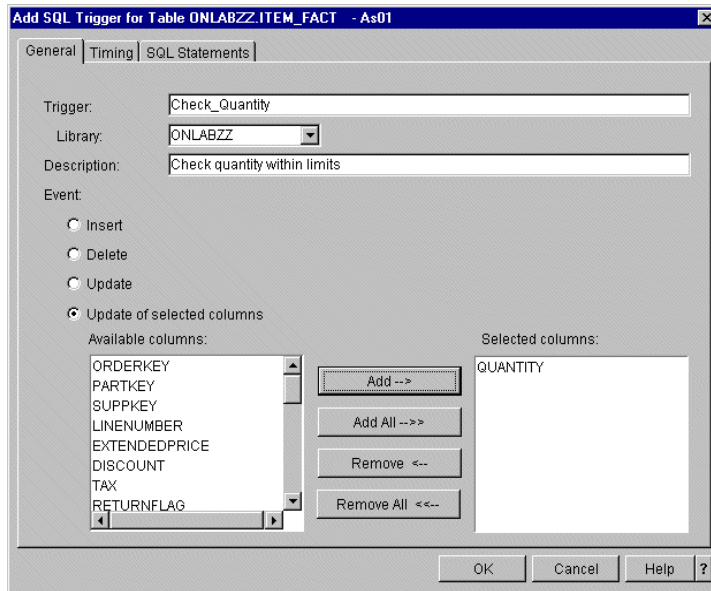


Figure 7-25 Defining an SQL Trigger: General tab

The Timing page specifies the timing and frequency of the trigger and the resulting correlation names (Figure 7-26).

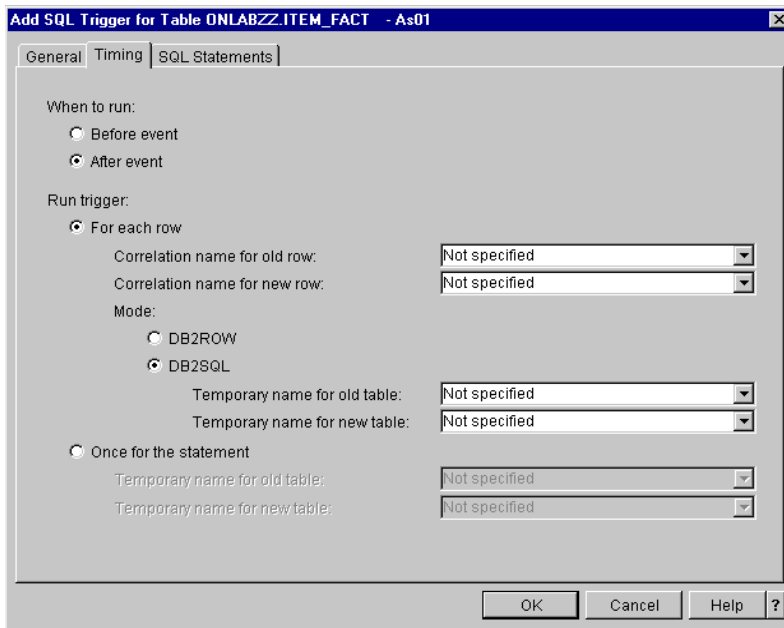


Figure 7-26 Defining an SQL Trigger: Timing tab

The SQL Statements page (Figure 7-27) contains the code for the SQL program that you are defining as a trigger. You can use the SQL statement examples and fill in the necessary information to make coding SQL easier. If you are adding a trigger to an existing table, you can check for syntax errors by clicking Check Syntax once you have the statement defined. A message is displayed for the first error detected, if any. To check for additional errors, click **Check Syntax** after the first error is fixed. This button is disabled when you add a trigger to a new table.

After an SQL trigger has been created, the SQL statements cannot be changed. You will have to delete and recreate the trigger to change the SQL.

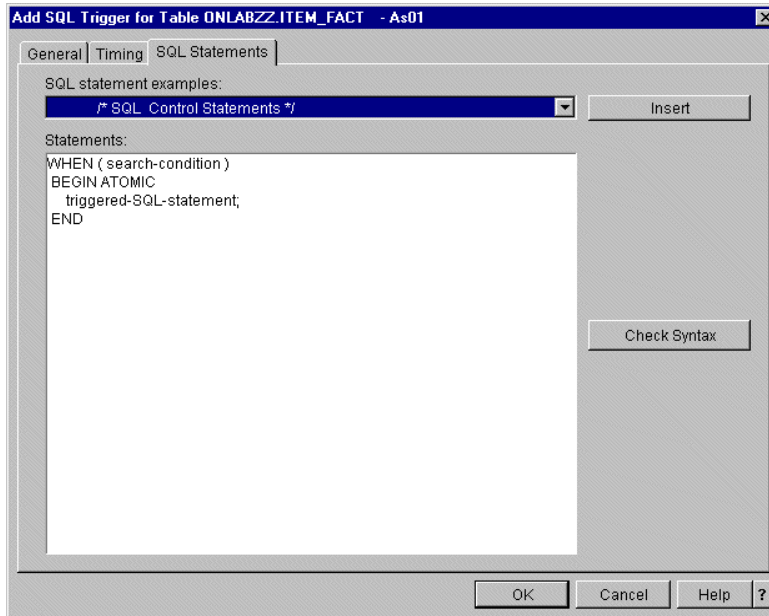


Figure 7-27 Defining an SQL Trigger: SQL Statements tab

On the Add System Trigger dialog, you can add a system (external) trigger to your table. System (external) triggers use a program object that already exists on the system. The program must exist before the trigger can be added. On pre-V5R1 systems, only the program name and library fields are active. See Figure 7-28.

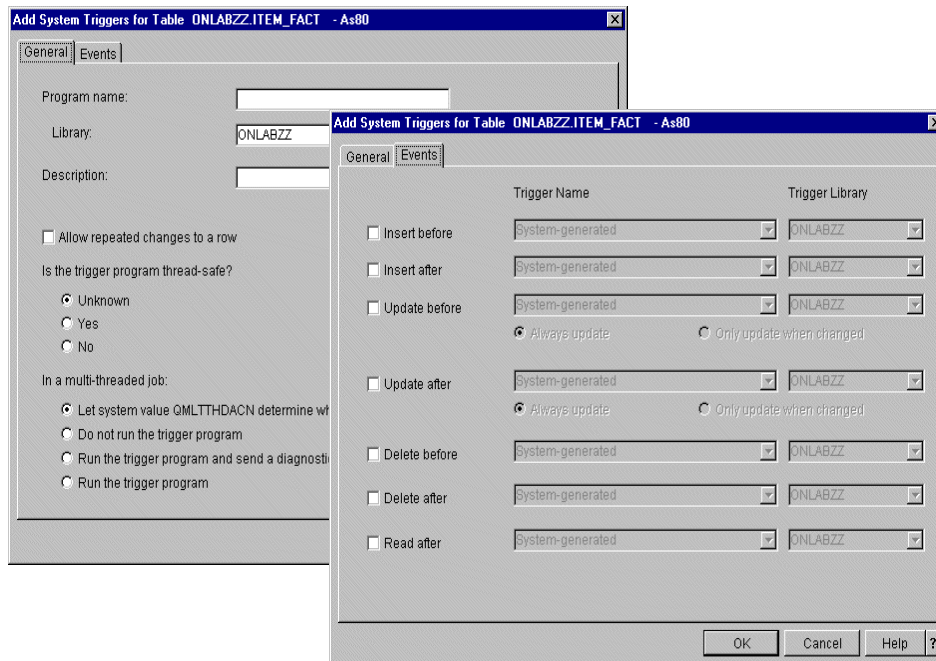


Figure 7-28 Defining a system trigger

### **Check Constraint**

A check constraint is specified at the field or column level. A check constraint examines the validity of the data in one or more of the columns in the same table.

The Check Constraints tab enables you to add, modify, view, or delete check constraints for the table on which you are currently working. You may modify a constraint only if it was defined during your current table editing session. If you added the constraint and then clicked OK on either the New Table dialog or Table Properties dialog, you may only view the constraint.

### **Database table constraints tips**

Constraints offer powerful system-provided DB2 UDB functions that need to be understood before you use them. In addition to the Operations Navigator graphical interface to constraints, OS/400 provides several commands to support constraints, such as the Add Physical File Constraint (ADDPFCST) and Remove Physical File Constraint (RMVPCST) commands.

You can access the full range of OS/400 constraints support by using the OS/400 Work with Physical File Constraints (WRKPCST) command. For additional constraints information, refer to:

- ▶ Operations Navigator online help information
- ▶ iSeries Information Center (<http://www.iseries.ibm.com/infocenter>). You can use the search word *constraints*
- ▶ Chapter 15, “Controlling the integrity of your database with constraints”, in *Database Programming*, SC41-5701
- ▶ Online help for the OS/400 commands on constraints, accessed through the Work with Physical File Constraints (WRKPCST) command

### **Managing journals and journal receivers**

As discussed in “Create journal example” on page 177, a journal and its attached journal receiver record the changes and actions made to a table. Once you create a journal and its initial journal receiver, you can perform additional journal management by right-clicking either the journal or a journal receiver within a library.

Figure 7-29 shows the actions that are possible on an existing journal.

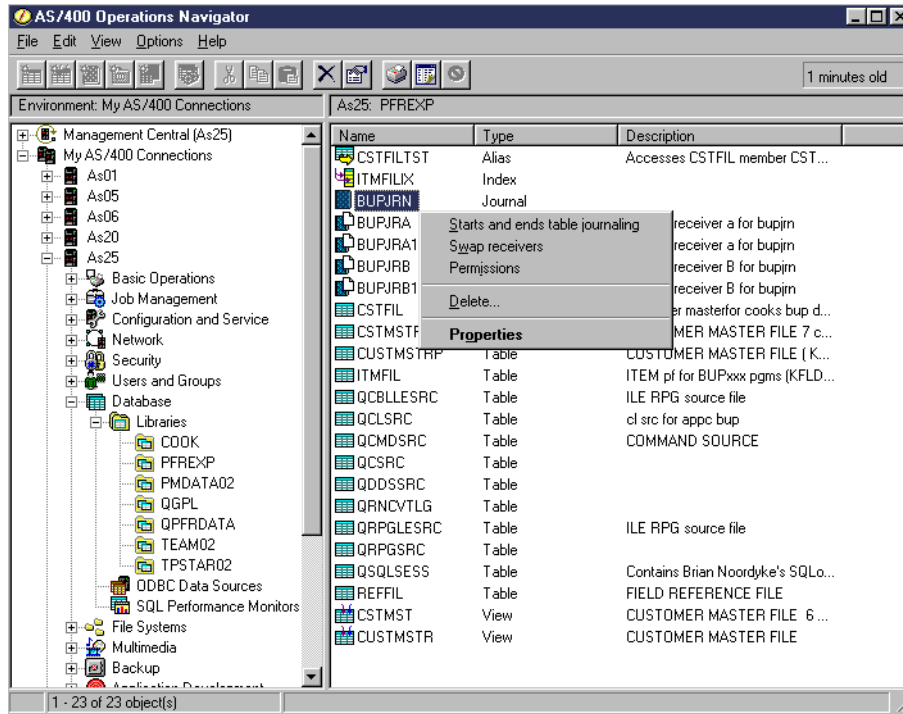


Figure 7-29 Managing a journal

The actions are explained in the following list:

- ▶ **Starts and ends journaling:** This action starts or ends journaling for one or more specific files or tables. Clicking this action brings up the Start/End Journaling panel shown in Figure 7-30 on page 194.
 

The start and end functions correspond to the OS/400 Start Journaling Physical File (STRJRNPf) and End Journal Physical File Change (ENDJRNPf) commands.

Journaling can be started and ended from the item you obtain by right-clicking a table name.
- ▶ **Swap receivers:** Clicking this action immediately detaches the currently attached journal receiver and creates a new journal receiver by adding 1 to a sequential number suffix to the journal receiver name. You can also manually swap receivers by using either an option from the Properties action or by using the OS/400 Change Journal (CHGJRN) command.
- ▶ **Permissions:** This action lets you view and change the authorities to the journal
- ▶ **Delete:** Clicking this action brings up a confirmation window for completing the journal deletion request or canceling it. The journal can only be deleted if *all* the objects being journaled to the journal have had journaling ended for them.
- ▶ **Properties:** This action brings up a panel that shows the original create journal attributes including journal receiver attributes and remote journal attributes, if any. You can also create a new journal receiver or remote journal by using the buttons that lead to additional panels. Figure 7-31 on page 195 shows an example of Journal properties information.

We right-clicked the **BUPJRN** journal and selected **Starts and ends table journaling**. Figure 7-30 shows the Start/End Journaling display for BUPJRN after we performed some journal-related operations earlier.

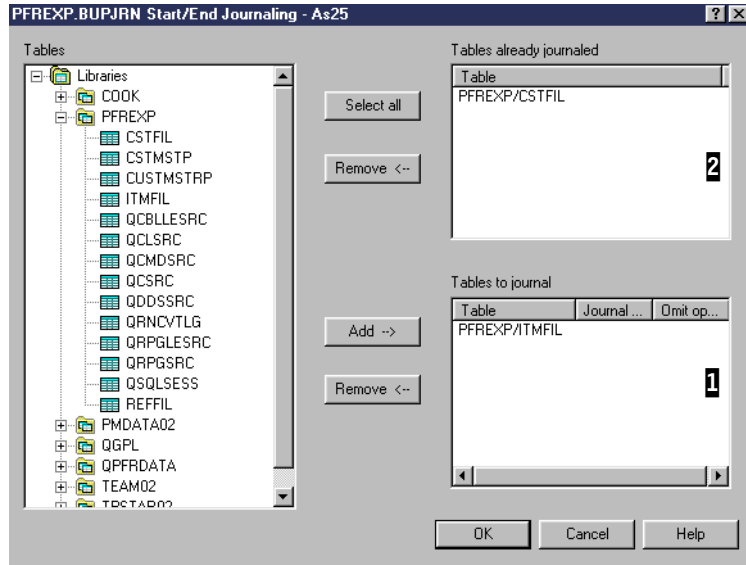


Figure 7-30 Start/End journaling display

To start journaling for a file or table, you can select the table and either click the **Add** button or drag and drop the file name into the list box (1 in Figure 7-30). When all tables you want journaled have been added to the list box, click the **OK** button. This starts journaling for these files or tables.

Alternatively, you could have used the OS/400 Start Journaling Physical File (STRJRNPF) command.

Notice the “Journal...” and the “Omit op...” column headings in the list box (1). The “Journal...” heading corresponds to the STRJRNPF command IMAGES (Record images) parameter. The “Omit op...” column heading corresponds to the STRJRNPF command OMTJRNE (Omit journal entries) parameter.

If you click *under* the “Journal” or “Omit op” heading to the right of a file or table name, an “X” character appears. If you click again, the “X” disappears. An “X” under Journal means that both before and after record images are written to the journal receiver. If no “X” appears, only an *after image* is recorded in the journal receiver. An “X” under Omit op... means that file or table open and close actions are not recorded in the receiver. If no “X” appears, all actions on the journaled file or table are recorded in the receiver.

In the list box (2 in Figure 7-30), you see the PFREXP/CSTFIL (system naming convention) table is already being journaled at the time the Properties action was selected.

You can stop journaling for a file or table by selecting the file or table (listed in 2) and clicking the Remove button and then clicking the OK button. This function corresponds to the OS/400 End Journaling Physical Files (ENDJRNPF) command.

### **Journal Properties example**

When we right-clicked the **BUPJRN** journal and selected **Properties**, the Journal Properties panel appeared as shown in Figure 7-31 on page 195. This shows the original parameters used to create the journal and enables you to make some changes and additions.

The *Tables* button shows you the Start or End journaling panel we already described.



The *Receivers* button shows you the currently attached receiver and previously detached journal receivers still on the system. You can also add a new journal receiver.

The *Remote Journals* button shows you the current status of a remote journal, if any. You can also add a new remote journal.

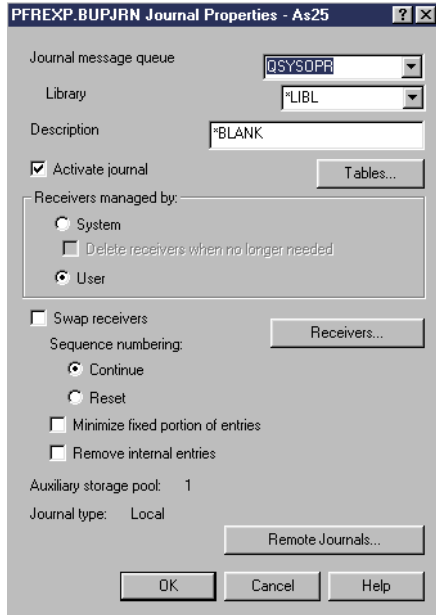


Figure 7-31 *Journal Properties example*

You can select the **Swap receivers** box and optionally specify either **Continue** or **Reset** to specify the sequence numbering to be used with the new receiver. Then click the **OK** button to have an immediate detach of the current journal receiver and creation of a new receiver that is immediately attached to the journal. Review online help information (click the Windows **?** button and place it on the **Swap receivers** text; this is the equivalent of context sensitive help on 5250 command screen when you move the cursor to a particular keyword parameter and press F1 for help) to determine if Swap receivers applies to your journaling environment.

In this example, we clicked the **Receivers** button to show you the panel in Figure 7-32. In our example, we have three online, but detached, receivers. The currently attached receiver is BUPJRA0002.

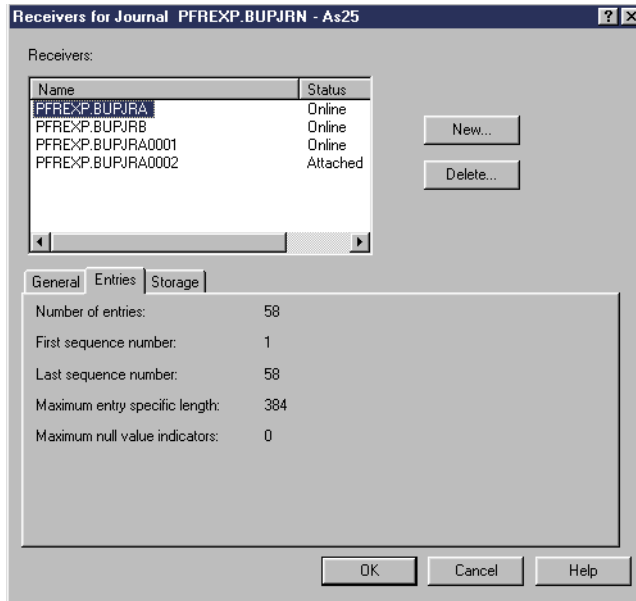


Figure 7-32 Journal receivers list and properties example

By selecting the BUPJRA journal receiver, the lower portion of the panel automatically displays the General properties of this receiver. We already selected the **Entries** tab information.

Select an online detached journal receiver. Click the **Delete** button to remove the journal receiver and its entries from the system when you no longer need this journaled information.

When you click the **New** button, you see an Add Journal Receiver panel. Clicking the **OK** button makes any new or delete function permanent.

You may also find the journal receiver General, Entries, and Storage information in a separate Properties panel for a specific receiver by performing *either* of the following actions from the *library* panel:

- ▶ Double-clicking the **journal receiver object**
- ▶ Right-clicking the **journal receiver object** and selecting **Properties**

### Working on locked rows

To gain access to this function, right-click a table. The Locked Rows dialog (Figure 7-33) displays the row number, job, user, job number, current user, status, and lock type for rows that have a row lock placed on them. A row lock is placed on a row when you read a table that is opened for update. While the row lock is in effect, no other job can read the same row for update, which keeps another job from unintentionally deleting the first job's update.

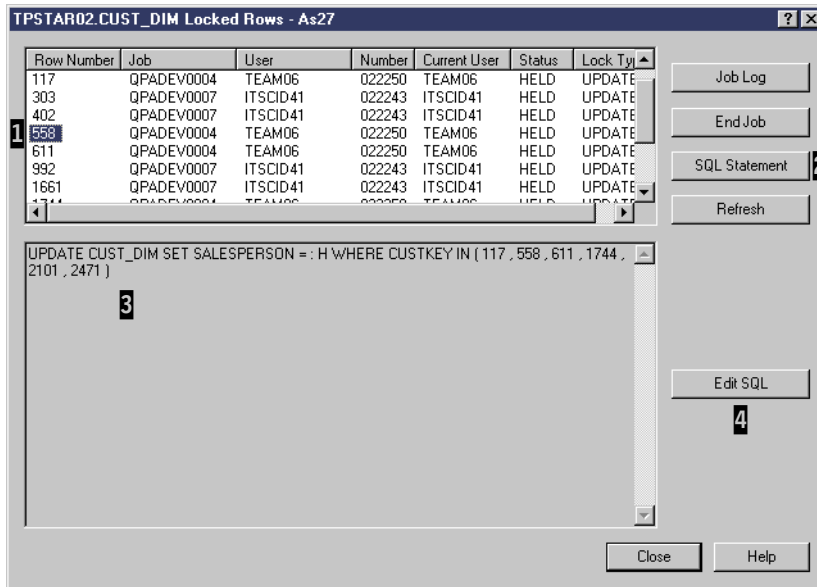


Figure 7-33 Locked Rows example

The Locked Rows panel allows you to perform various tasks:

- ▶ Check which jobs are locking which rows
- ▶ View the job log for a job
- ▶ View an SQL statement that is running or has run in the job
- ▶ Use the above mentioned SQL Statement with the Run SQL Scripts center
- ▶ End a job that is listed (provided you have the right authority)

Since most of these tasks are rather intuitive, we only document how to link to the SQL Script center to investigate on what is happening in the database.

After you start the Locked Rows function, select the job (1) in Figure 7-33) you want to examine. Click the **SQL Statement** button on the right-hand side of the picture (2) to bring it into the bottom part of the panel (3).

At this point, when you click the **Edit SQL** button (4), the Run SQL Scripts center starts, and the SQL statement is brought into it for you to use. Refer to “Running a single SQL statement” on page 211 for a discussion on how to use this tool. You should also refer to “Linking to the Visual Explain component” on page 212 to see how to use it to conduct database performance analysis.

## 7.3 Run SQL Scripts

The Run SQL Scripts center is a powerful interface to your iSeries database. With it, you can use any SQL statements to issue any kind of operations you are authorized to on the iSeries database objects. Licensed product program 5722-ST1, DB2 Query Manager and SQL Development Kit for iSeries, is not a prerequisite for using Run SQL Scripts. This component of Operations Navigator uses JDBC to access the server.

To use SQL from Operations Navigator, right-click the **Database** component under the iSeries server that contains the data. Figure 7-34 shows the Database context menu with the Run SQL Scripts action highlighted.

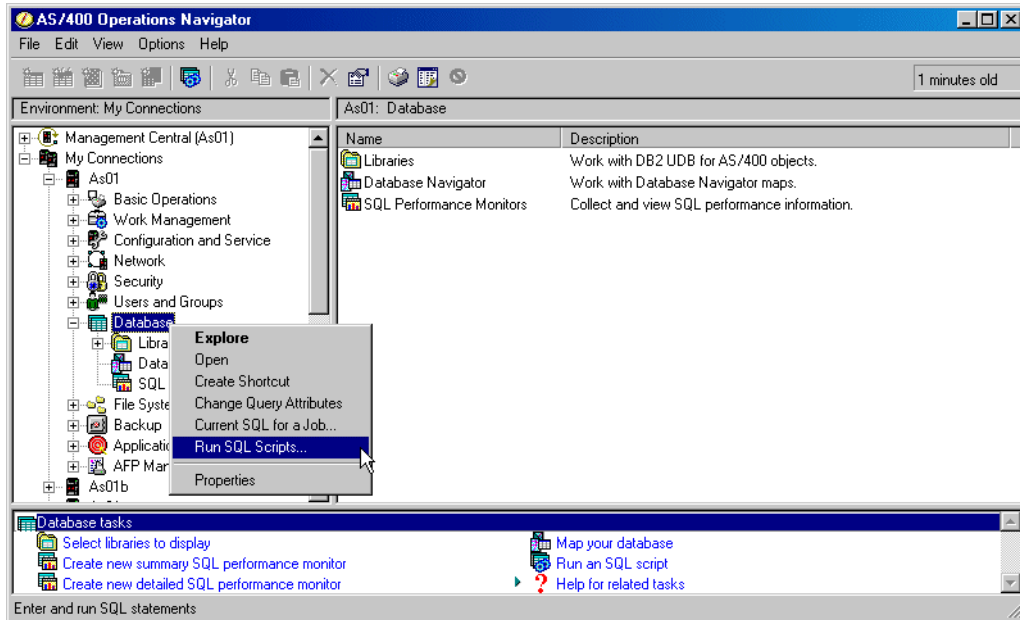


Figure 7-34 Run SQL Script

**Important:** This component has been entirely re-written in V5R1 using Java. It has an enhanced layout and supports these new features as well:

- ▶ Result data is displayed in the same or a separate window via the **Options** menu
- ▶ Run SQL Statement icons
- ▶ Run SQL statement by double-clicking instead of single-clicking via the **Options** menu

Right-click **Database** to bring up the pull-down menu. *Do not click* Libraries, because Operations Navigator enables you to potentially access the entire system, rather than limiting you to just the data within a library.

Figure 7-35 shows an example of the initial Run SQL Scripts panel.

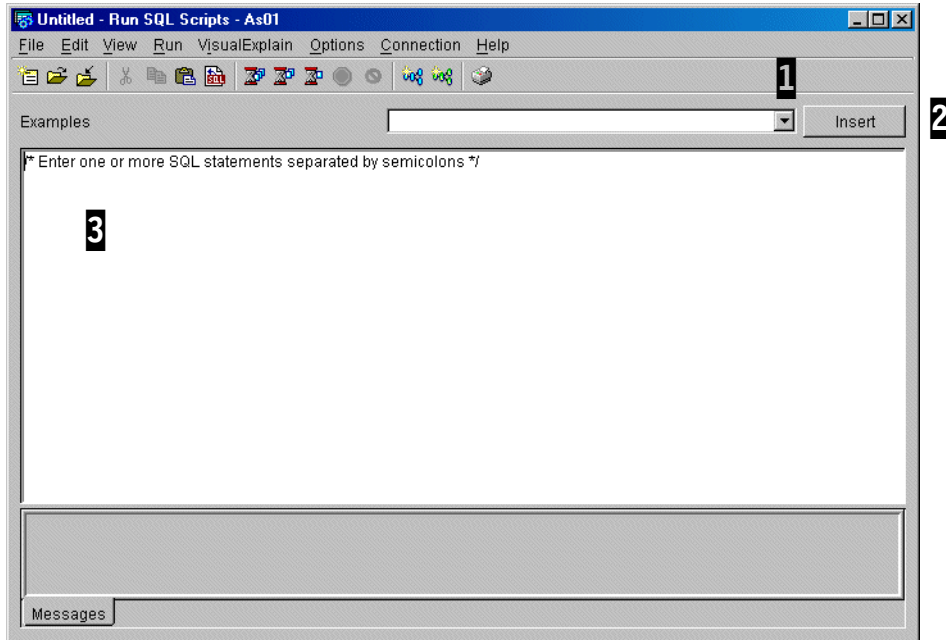


Figure 7-35 Run SQL Scripts: Initial input panel

The Run SQL Scripts window lets you create, edit, run, and troubleshoot scripts of SQL statements. You can also save the SQL scripts with which you work into a PC file on your PC workstation. There are several run options for the SQL statements that are entered into the SQL statement input area (3). We discuss them later in this section.

As shown at 1 in Figure 7-35, you can select to review a list of already provided SQL statements. OS/400 provides a large set of base syntax for almost every possible SQL statement that can be used. You can display the list of existing SQL statements by clicking the down arrow in this area of the panel. You can then select an SQL statement from the list shown and have it *inserted* into the statement input area (3) by clicking the Insert button (2).

You can modify the selected SQL statement or enter your own SQL statement. You can run one or more of your entered your SQL statements in different ways and stop between statements.

Before we discuss the run actions, refer to Figure 7-36 to see the different panels within the Run SQL Scripts function.

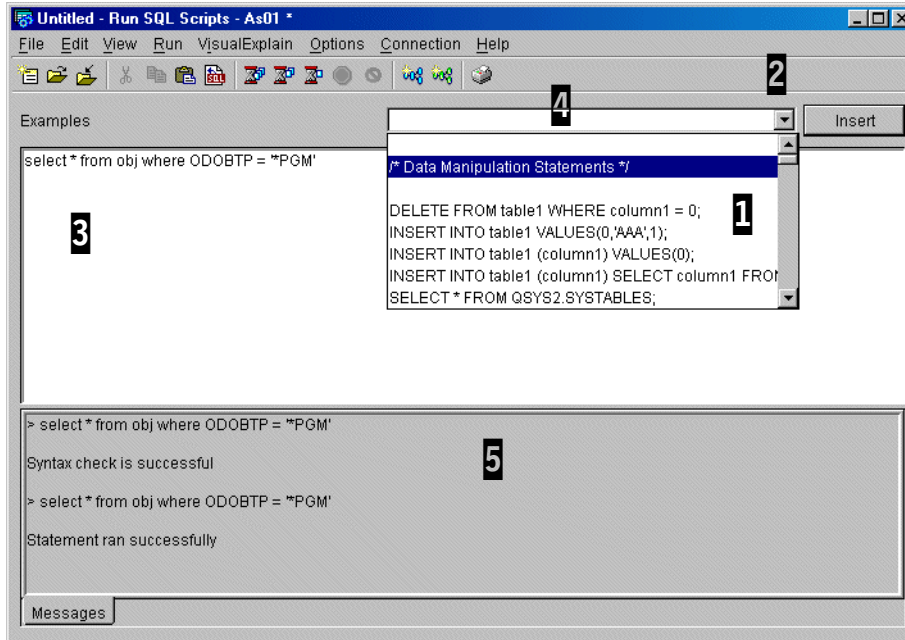


Figure 7-36 Run SQL Scripts window pane example

The beginning of the list of provided SQL statements is shown at 1. This list was produced by clicking the down arrow (2). In this example, we do not select an SQL statement to be placed into the statement input area (3). However, if we selected one or more SQL statements in the window at 1, the statement or statements would appear in the “SQL statement example” area (4), and you could click the Insert button to place the statements into SQL input area (3).

In the SQL statement input area (3), we already entered two simple SQL statements that are partially hidden. We separately ran the following SQL statements:

```
select * from item_fact;
select * from cust_dim;
```

Then we viewed the results on a panel (not shown), prior to selecting the list of SQL statements (1).

The Run History panel (5) shows you the success and any messages of the SQL statements run. When you select the Edit option from the menu bar, you have the option to *clear run history* information.

Figure 7-37 includes the previous SQL SELECT statements. But, we added SQL statements to illustrate more of the power of DB2 Universal Database for iSeries accessible through Operations Navigator.

Figure 7-37 also illustrates some of the *run options* for the SQL statements we showed under Run SQL Scripts support.

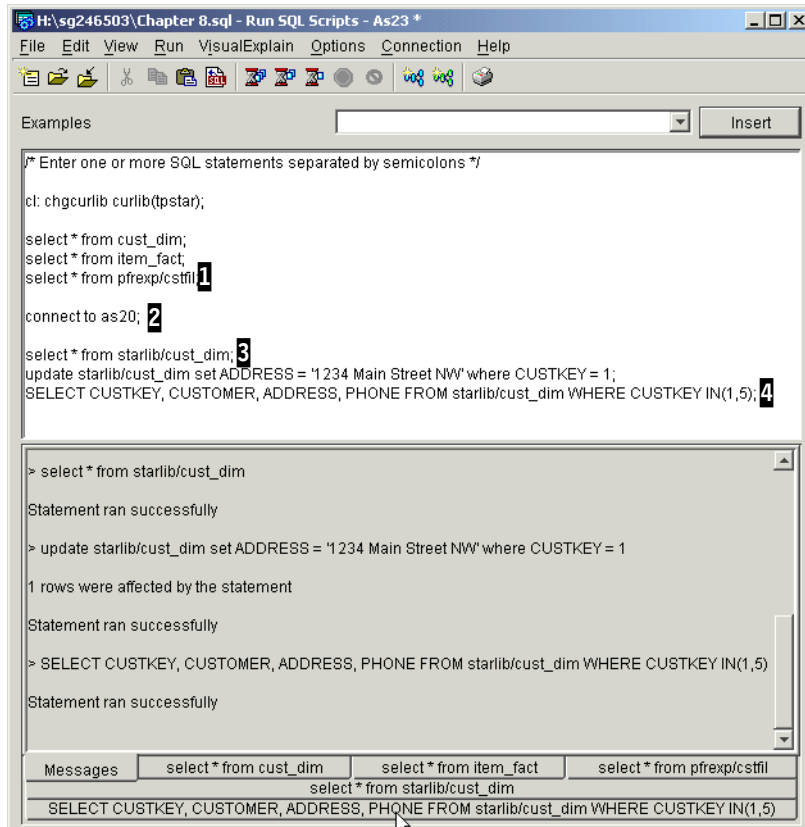


Figure 7-37 Run SQL Scripts: Additional sources

Figure 7-37 at 1 shows that we did a select from a table that is in a different OS/400 library (*pfrexp*) than the libraries included in our job description's initial library list. We did this by qualifying the table with **pfrexp/**.

The slash separator character (/) is valid because we changed from the default SQL naming convention to the system naming convention.


By default, we are running SQL statements on the system to which we are connected. The CONNECT SQL statement used to connect to a remote system as20 ("As20" in our Operations Navigator screen example figures), using OS/400 Distributed Relational Database Architecture (DRDA) over TCP/IP is shown at 2. Assuming this CONNECT statement is successful, all SQL statements thereafter are directed to remote system as20 until an SQL "release all" statement is issued, when the connection returns to access only the local As25 system.

OS/400 supports connections to multiple remote systems during the same session. For example, following the statement shown at 4, you can issue a "connect to as05" statement. Assuming this is successful, all the following SQL statements are directed to system As05. You can then issue a "set connection to as20" statement that *resets* the current dialogue back to system As20. You need to keep track of which system (remote database) you are connected to and on which system you are performing operations.

The next statement (3) selects the *cust\_dim* table in the library *tpstar01* on the remote system *as20*.

**Note:** DRDA is the IBM-defined architecture for accessing remote databases. It is implemented on all IBM operating systems, and some non-IBM operating system databases support it. At a base set of functions level, it is similar to the ODBC and Java Database Connectivity (JDBC) set of capabilities. On IBM systems, Distributed Data Management (DDM) is a higher level interface to DRDA capabilities.

While we cannot go into the details of DDM/DRDA in this book, we discuss basic setup requirements for the DDM/DRDA example shown here to work over TCP/IP. Refer to 7.4, “Change Query Attributes” on page 217, for more information.

A select statement that uses only some of the fields or columns in the *cust\_dim* table and displays only the records or rows where the key field or the CUSTKEY column has a value of 1 or a value of 5 is shown at  in Figure 7-37.

**Note:** With our examples, each table index (set of key fields or columns) structure is relatively simple, and the number of rows is small relative to a million or more rows that would be present in a data warehouse environment. We also do not have complex join statements (columns joined together from two or more tables).

In a more complex data structure and performance critical environments, you would want to use a combination of the following options:

- ▶ The Run SQL Scripts option to include query optimizer debug messages in the job log (see 7.3.4, “Run SQL Scripts Run options” on page 210)
- ▶ SQL Performance Monitor support (see 7.6, “SQL Performance Monitors” on page 220)
- ▶ Visual Explain (see Chapter 10, “Visual Explain” on page 301)

By reviewing the job log, using Visual Explain or going through monitored data, you can determine if the most efficient method is used by OS/400 query support to perform the SQL function.

### 7.3.1 ODBC and JDBC connection

Open Database Connectivity (ODBC) is a standard interface for database connectivity defined by the Microsoft Corporation. ODBC establishes the standard interface to any database as SQL. In general, the ODBC architecture accounts for an application using the ODBC interface, an ODBC Driver Manager, one or more ODBC Drivers, and an ODBC Data Source (place where the data is stored). Java Database Connectivity (JDBC) is an equivalent standard interface for database connectivity from Java applications.

Client Access Express provides the iSeries ODBC and JDBC drivers that runs on the PC workstation and the ODBC and JDBC Data Source support that runs on the iSeries server. Production mode job name starts with QZDASOINIT (or QZDASSINIT if SSL is being used). In version 4, with ODBC Data Sources, you can set up a Client Access Express ODBC data source by providing a data source name (a name meaningful to you) and an iSeries server name. Starting in version 5, the setup and administration of Client Access-provided ODBC driver is done by using the standard ODBC data source administrator, provided with the Windows operating system.

An ODBC data source consists of the data that the user wants to access and its associated operating system, Database Management System (DBMS), and network platform (if any) used to access the DBMS.



Setup information is associated with a data source and may include, for example, data formatting and performance options. Data formatting options include qualified name separators, date and time formats, and data translation. Performance options include when to use record blocking, data compression, or an SQL Package. An SQL package stores previously parsed SQL statements to improve performance when used later.

You can also specify if Secure Socket Layer (SSL) is to be used with the ODBC connection.

Some client applications (including Operations Navigator) may provide their own unique data source definition.

A good source for more information on ODBC support is *Client Access Express for Windows*, SC41-5509.

You can create your own data source to limit the libraries that can be used and, as previously described, your own set of name separators, date and time formats, performance options, and so on.

OS/400 provides two data sources that you should understand even if you are not creating your own data source:

- ▶ *A data source used by Operations Navigator itself to perform its functions:* This data source is identified by the system name to which you are first connected. For example, if the first system you connect to is called As25, the data source used by Operations Navigator is named QSDN\_As25.

**Important:** Unless you are an ODBC expert, *do not change* any of the default settings for this data source. If you change them, Operations Navigator may fail to operate correctly.

- ▶ *A data source is used if you use Database-> Run SQL Scripts:* The first time you select the action to Run SQL Scripts to a specific iSeries server, OS/400 creates a JDBC data source for the system (ODBC in V4R5 or previous releases), which can be changed by selecting **Connections -> JDBC Setup** (Figure 7-38). One JDBC data source is created for each system on which SQL scripts are run. You do not have to create your own JDBC Data Source and understand the data source parameters to run SQL statements against libraries and files or tables to which you are authorized.

In 7.3, “Run SQL Scripts” on page 197, we use the default IBM-created data source in our JDBC Data Source Translation parameters.

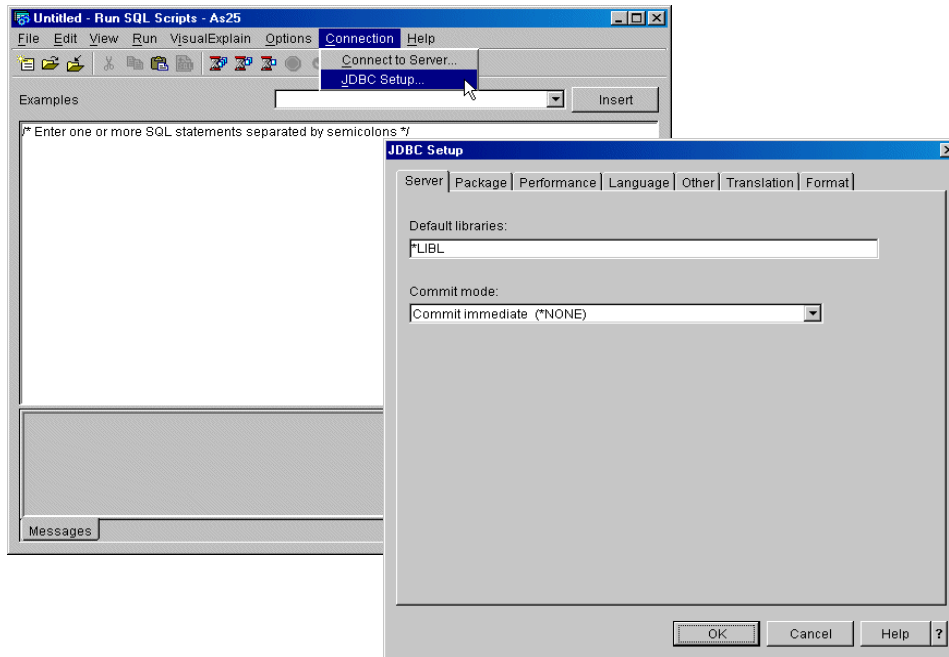


Figure 7-38 JDBC data source panel

## Server tab

*Default libraries* enables you to change the set of libraries available to the user of this JDBC data sources. The default (\*USRLIBL) means to use the initial library list (INLLIBL) parameter specified on the job description for the OS/400 user profile using this JDBC data source.

*Commit mode* controls the level of DB2 Universal Database for iSeries commitment control, including when database changes are considered permanent and whether other users of the same database rows can see column updates that are not yet permanent.

A complete description of commitment control is beyond the scope of this redbook. However, you should understand that in the industry, users of SQL typically expect commitment control to be active. That is, an application design determines what a completed transaction (also called a unit of work) is. Any database row changes (column updates, rows deleted, rows inserted) are not considered permanent until a successful transaction has been completed (transaction boundary). At that time, the application performs a *commit* and all changes are now made permanent. If the application determines that an in-progress transaction should be terminated, it performs a *rollback*. All changes are as if they had never occurred. If the application abnormally terminates before issuing a commit or rollback, the underlying SQL support performs the rollback.

To support commitment control on OS/400, you must also have the tables journaled and the job using these tables must issue a system operation that starts commitment control for the job. This system operation can be invoked by using the OS/400 Start Commitment Control (STRCMTCTL) command or be implicitly invoked by this parameter for values other than \*NONE.

A *commit group* refers to the rows that are in the process of being updated, deleted, or inserted. As the help text shows, objects referred to on the COMMENT ON, CREATE, and so forth are also part of this commit group. The commit or rollback applies to all of these rows and objects.

We include the help text here because the OS/400 default is \*NONE, which is not generally supported in the industry. This provides a very flexible operating environment, such as letting other applications or users access the latest database changes. However, \*NONE exposes the table rows, even while being processed by the properly authorized Operations Navigator user, to be modified without a required database Commit or Rollback operation sequence to make any database changes permanent.

For example, using \*NONE means any valid SQL statement that changes column data has made a permanent change to the data. If the properly authorized Operations Navigator user mistakenly updates a column using a wrong value for a key, there is no rollback function available to undo the change to the wrong row. You need either a backup copy of the data or an OS/400 journal to recover the original data.

The other commit values specify row locking rules (other applications prevented from updating the same row) and visibility of in-progress changes among applications accessing the same rows.

### **Package tab**

This tab specifies whether extended dynamic support is enabled. Extended dynamic support provides a mechanism for caching certain dynamic SQL statements on the server.

The first time a particular SQL statement is run, it is stored in an SQL package on the server. On subsequent runs of the same SQL statement, the server can skip a significant part of the processing by using information stored in the SQL package. By default, it is not enabled.

### **Performance tab**

This tab allows you to set performance options.

### **Language tab**

This tab allows you to specify language options.

### **Other tab**

The Other tab allows you to set the *access type* and *remarks source* options for your connection.

### **Translation tab**

In most cases, you never need to view or change the JDBC (or ODBC) data source translation parameters. This is because your application tables or files are typically stored as using the Coded Character Set Identifier (CCSID) numeric value that stores the data according to your national language encoding. In these cases, any OS/400 data accessed by the client workstation is translated into the appropriate ASCII format as required for viewing or processing on the client.

However, certain OS/400 system files or tables are defined to use the special CCSID 65535. By default, JDBC data source processing does not translate data from a file or table with CCSID 65535.

For example, if you want to use Run SQL Scripts against the performance collection files (prefix QAPM...) or a table generated from a virtual private network (VPN) journal (copied to a database file or table), you need to have the character columns translated in most cases. Select the JDBC data source **Translate** tab and select the **Translate CCSID 65535** check box.

For more information on CCSID support, refer to *AS/400 National Language Support*, SC41-5101.

### **Format tab**

There is an important operational difference between using the *SQL naming convention* and the *System naming convention* when running SQL statements under Operations Navigator Run SQL Scripts. If you are using the system naming convention and use a non-qualified name, such as a table name with no library qualifier, the system searches for the table within all libraries currently in the session's (job's) current library list. If you are using the SQL naming convention, the ANSI standard specification causes the system to look *only in the current library* within the session's current library list.

For example, assume the user portion of the session's library list is in the order of TEAM02, followed by library TPSTAR02. Also, assume the unqualified table name is CUST\_DIM and is stored in library TPSTAR02. Using the SQL naming convention, the system looks for CUST\_DIM only in library TEAM02 and does not find it, which results in an error condition. Using the system naming convention, the system first searches library TEAM02 and then library TPSTAR02. The CUST\_DIM table will be found and the SQL statement will run successfully.

Format parameters are important if you have a special operating environment, such as your system requiring country specific or multiple country support.

You must review the online help text to get the details for all of these parameters. The settings are determined by your requirements.

If you want to modify either data source, refer to the online help or consult *Client Access Express for Windows*, SC41-5509.

## **7.3.2 Running a CL command under SQL script**

In addition to running SQL statements under Run SQL Script, Operations Navigator allows the properly authorized user to run any OS/400 Control Language (CL) statement that can be validly run in a batch (no 5250 workstation required) environment. You must precede the OS/400 command syntax with the prefix **CL:** (uppercase or lowercase) as shown in Figure 7-39.

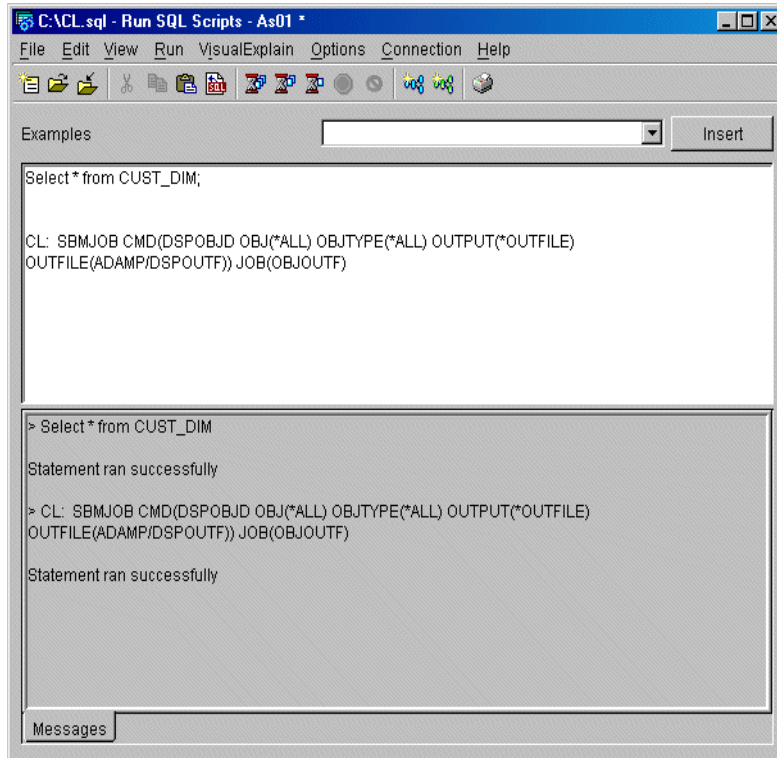


Figure 7-39 Run SQL Scripts: Running a CL command

The selected CL command is an OS/400 command that submits the job to job queue QBATCH, which is one of the IBM-supplied job queues associated with the IBM-provided subsystem QBATCH. The submit job command parameter (CMD) value can be any OS/400 command or user-defined command. In our example, we used the DSPOBJD command with its own set of parameters.

You may also use much simpler OS/400 commands, such as:

- ▶ Adding a new library to the current library list of the Operations Navigator session using the CL command:
 

```
CL: ADDLIB LIB(PFREXP);
```
- ▶ Sending a message to the system operator using the CL command:
 

```
CL: SNDMSG MSG('This message is from an Operations Navigator session from user TEAM02.')
TOUSR(*SYSOPR);
```

### Tips for running CL in Run SQL Scripts

Running SQL Scripts is a powerful way to test new SQL statements, especially in the sequence you may want to run them in a program. In an actual application environment, you may also want to integrate running system functions through CL commands with your SQL statements. Here are some tips:

- ▶ Starting with Client Access Express Service Pack 5 (SP5) for V4R4, the following restriction has been removed:

*For the CL command to be recognized successfully, you must remove (delete) any comment statement, such as:*

```
"/* Enter one or more SQL statements separated by semicolons */."
```

- ▶ The IBM-supplied SQL statement examples include some CL command examples at the end of the SQL statements.
- ▶ The key to making the OS/400 command work from an Operations Navigator Run SQL Scripts session is to ensure the objects referenced in the command can be found in the Operations Navigator session's (job's) library list or the system library list (system value QSYSLIBL).

Adding a library name under the Database->Libraries branch does not carry over to the Run SQL Scripts function. OS/400 commands can always be found through the system value QSYSLIBL. However, objects, such as user-defined commands, may require the appropriate library to be in the Operations Navigator Run SQL Scripts session's library list. Use **Connection -> JDBC Setup** to amend the user part of the library list.

### 7.3.3 Run SQL Scripts example using a VPN journal

This section shows an example of using Run SQL Script to identify the IP packets, if any, that were *denied* routing based on OS/400 VPN filtering rules. The standard OS/400 VPN support records *permit*, *deny*, and *filter rule* change occurrences in a system journal named QIPFILTER, stored in library QSYS. The OS/400 Display Journal (DSPJRN) command provides a journal entry time stamp and other compare values to selectively display, print, or copy journal entries to a database file or table.

If you choose the database option, you can process the copied journal entries several different ways through SQL. Run SQL Scripts is a good way to experiment with viewing different journal entry field or column data. Once you see a view of the data you want to use repetitively, you can save the SQL statements for later reuse or copy the SQL statements into a program that does further processing or graphical display.

This section uses the journal data discussed in the "AS/400 VPN problem determination," chapter in *AS/400 Internet Security: Implementing AS/400 Virtual Private Networks*, SG24-5404.

We performed the following steps to query the VPN logging data originally placed into the QIPFILTER journal. The query results show the journal entries for packets that have been denied routing, since a large number of *deny entries* may require further investigation by your security personnel.

1. Create a copy of the IBM-supplied file QSYS/QATOFIPF into a library of your choice, using the OS/400 Create Duplicate Object (CRTDUPOBJ) command, for example:

```
CRTDUPOBJ OBJ(QATOFIPF) FROMLIB(QSYS) OBJTYPE(*FILE) + TOLIB(mylib) NEWOBJ(myfile)
```

The system file or table QATOFIPF provides the column definitions used by the IBM-supplied queries. In our example, we duplicate this table as ON\_IPFTRT.

- Use the DSPJRN command to copy the journal entries from the QUSRSYS/QIPFILTER journal to the output database file created in the preceding step:

```
DSPJRN JRN(QIPFILTER) JRNCODE(M) ENTTYP(TF) OUTPUT(*OUTFILE) +
OUTFILFMT(*TYPE4)
OUTFILE(mylib/myfile) ENTDTALEN(*CALC)
```

The DSPJRN command has both starting and ending time-stamp values and starting and ending journal entry sequence numbers so you do not need to copy the entire set of journal entries to the file or table.

- You need to review the field or column names and descriptions for file or table ON\_IPFTRT to determine which columns to select and use for row selection. You may use the OS/400 Display File Field Description (DSPFFD) command or use Operations Navigator to display the table Properties by right-clicking the table name.

*AS/400 Internet Security: Implementing AS/400 Virtual Private Networks*, SG24-5404, provides good background information to help select the appropriate fields or columns.

- Using **Run SQL Scripts**, build the SQL statement and view the results.

Figure 7-40 shows our example SQL statement and sample output.

The screenshot shows the 'Run SQL Scripts' window with the following SQL query:

```
SELECT ADAN.ON_IPFTRT.TFRNUM, ADAN.ON_IPFTRT.TFFACT, ADAN.ON_IPFTRT.TFPDIR,
ADAN.ON_IPFTRT.TFSRCA, ADAN.ON_IPFTRT.TFSRCP, ADAN.ON_IPFTRT.TFDSTA,
ADAN.ON_IPFTRT.TFTIME
FROM ADAN.ON_IPFTRT
WHERE ADAN.ON_IPFTRT.TFFACT = 'DENY'
AND ADAN.ON_IPFTRT.TFSEQN >= 6300
```

The output table is as follows:

	TFRNUM	TFFACT	TFPDIR	TFSRCA	TFSRCP	TFDSTA	TFDSTP	TFTIME
1	8	DENY	O	10.196.8.5	137	10.196.8.255	137	1999-07-24 13:32:52.878560
2	8	DENY	O	204.146.18.5	137	204.146.18.255	137	1999-07-24 13:32:52.972128
3	8	DENY	O	10.196.8.5	137	10.196.8.255	137	1999-07-24 13:32:53.082448
4	8	DENY	O	204.146.18.5	137	204.146.18.255	137	1999-07-24 13:32:53.169424
5	8	DENY	O	10.196.8.5	137	10.196.8.255	137	1999-07-24 13:32:53.582032
6	8	DENY	O	204.146.18.5	137	204.146.18.255	137	1999-07-24 13:32:53.653728
7	8	DENY	O	10.196.8.5	137	10.196.8.255	137	1999-07-24 13:32:54.085984
8	8	DENY	O	204.146.18.5	137	204.146.18.255	137	1999-07-24 13:32:54.175344
9	8	DENY	O	10.196.8.5	137	10.196.8.255	137	1999-07-24 13:32:54.409520
10	8	DENY	O	204.146.18.5	137	204.146.18.255	137	1999-07-24 13:32:54.459792
11	8	DENY	O	10.196.8.5	137	10.196.8.255	137	1999-07-24 13:32:54.935600
12	8	DENY	O	204.146.18.5	137	204.146.18.255	137	1999-07-24 13:32:54.989056

Figure 7-40 Run SQL Scripts: Viewing 'denied' VPN packets

The TFACT (filter action) column (1 in Figure 7-40), records values such as PERMIT, DENY, or additional values for adding and changing filter rules and starting and stopping filtering. You also see our SQL compare value for 'DENY'. You can see that we did not want to look at all (13,000) journal entries, so we started around the middle of the entries with journal entry sequence number 6300 (2 in Figure 7-40).

The TFPDIR (packet direction) column specifies “O” for output packet and “I” for input packet.

Using the source IP address and port number (3) and the destination IP address and port number (4), a TCP/IP expert can determine the actual workstation and TCP/IP function. A TCP/IP expert may also choose different columns to include in the SQL SELECT statement.

### 7.3.4 Run SQL Scripts Run options

This section explains the Run options available for these SQL statements. We use Figure 7-41 as a basis for explaining the run options.

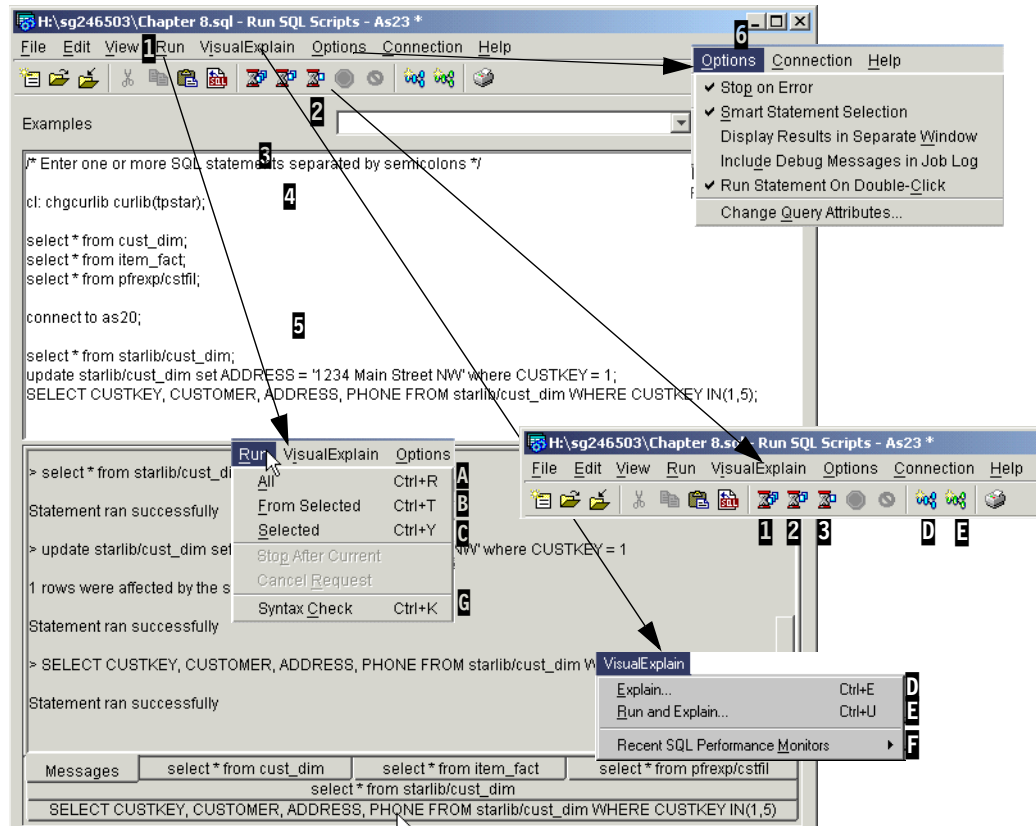


Figure 7-41 Run SQL Script: Run options

There are two “selection lists” types from which you can choose to run one or more SQL statements at a single time. You can select the Run option (1) from the Run SQL Scripts menu bar or select one of the *green arrow* or *hour glass* Run action icons (2) from the toolbar. These have corresponding functions. You can also select the Run option with a key sequence as shown under the Run pull-down menu.

You can pre-specify (defaults are provided) some controls over the Run function through the Options action in the menu bar (6). We discuss these controls in “Controlling SQL run options” on page 213 after we explain the three levels of run options:

- ▶ Running a single SQL statement
- ▶ Running a set of SQL statements
- ▶ Running all SQL statements currently specified



## Running a single SQL statement

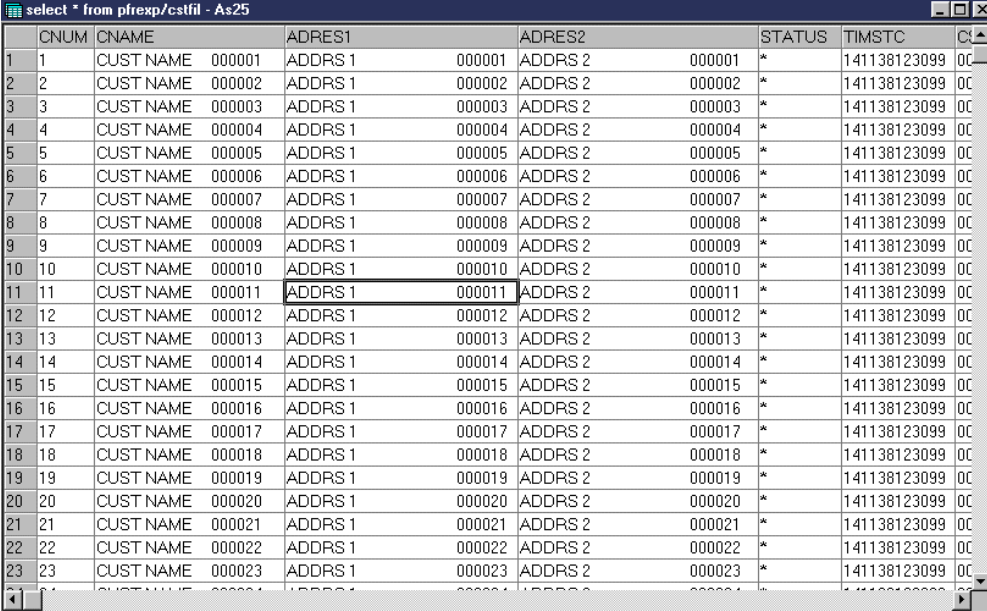
Place the active screen cursor within the SQL statement text you want to run, for example:

```
select * from pfrexp/cstfil;
```

This is referenced as **4** in Figure 7-41. You can run only this statement by using *one* of the following actions:

- ▶ Click the **Selected** action (**6**).
- ▶ Click the “select one line” or “select one line hour glass” icon associated with **6** in our example in Figure 7-41.
- ▶ Press Ctrl+Y from the workstation keyboard.

Only the single statement will run. If it is a SELECT statement, the results are presented as a window on your Operations Navigator workstation. The column names are presented as column headings. If you want to select only a subset of columns later, you can use these headings and displayed column data to help you select the appropriate columns. Figure 7-42 shows some of the column headings and associated data for the *pfrexp/cstfil* table.



CNUM	CNAME	ADDR1	ADDR2	STATUS	TIMSTC
1	CUST NAME 000001	ADDRS 1 000001	ADDRS 2 000001	*	141138123099
2	CUST NAME 000002	ADDRS 1 000002	ADDRS 2 000002	*	141138123099
3	CUST NAME 000003	ADDRS 1 000003	ADDRS 2 000003	*	141138123099
4	CUST NAME 000004	ADDRS 1 000004	ADDRS 2 000004	*	141138123099
5	CUST NAME 000005	ADDRS 1 000005	ADDRS 2 000005	*	141138123099
6	CUST NAME 000006	ADDRS 1 000006	ADDRS 2 000006	*	141138123099
7	CUST NAME 000007	ADDRS 1 000007	ADDRS 2 000007	*	141138123099
8	CUST NAME 000008	ADDRS 1 000008	ADDRS 2 000008	*	141138123099
9	CUST NAME 000009	ADDRS 1 000009	ADDRS 2 000009	*	141138123099
10	CUST NAME 000010	ADDRS 1 000010	ADDRS 2 000010	*	141138123099
11	CUST NAME 000011	ADDRS 1 000011	ADDRS 2 000011	*	141138123099
12	CUST NAME 000012	ADDRS 1 000012	ADDRS 2 000012	*	141138123099
13	CUST NAME 000013	ADDRS 1 000013	ADDRS 2 000013	*	141138123099
14	CUST NAME 000014	ADDRS 1 000014	ADDRS 2 000014	*	141138123099
15	CUST NAME 000015	ADDRS 1 000015	ADDRS 2 000015	*	141138123099
16	CUST NAME 000016	ADDRS 1 000016	ADDRS 2 000016	*	141138123099
17	CUST NAME 000017	ADDRS 1 000017	ADDRS 2 000017	*	141138123099
18	CUST NAME 000018	ADDRS 1 000018	ADDRS 2 000018	*	141138123099
19	CUST NAME 000019	ADDRS 1 000019	ADDRS 2 000019	*	141138123099
20	CUST NAME 000020	ADDRS 1 000020	ADDRS 2 000020	*	141138123099
21	CUST NAME 000021	ADDRS 1 000021	ADDRS 2 000021	*	141138123099
22	CUST NAME 000022	ADDRS 1 000022	ADDRS 2 000022	*	141138123099
23	CUST NAME 000023	ADDRS 1 000023	ADDRS 2 000023	*	141138123099

Figure 7-42 Run SQL Script: Sample SQL SELECT output

## Running a set of SQL statements

You can run a set of SQL statements that are currently active in your Operations Navigator session to the iSeries server. Using our example in Figure 7-41, you would run:

```
select * from pfrexp/cstfil; 4 through SELECT CUSTKEY ... IN(1,5); 5
```

You do this by placing the active screen cursor within the SQL statement text (**4**) and performing *one* of the following actions:

- ▶ Click the **From Selected** action (**5**).
- ▶ Click the **From Selected** icon (the middle down arrow or the middle hour glass) associated with **2** in our example in Figure 7-41.
- ▶ Press Ctrl+T from the workstation keyboard.

This runs each statement sequentially, beginning with:

```
select * from pfrexp/cstfil;
```

We have three SELECT statements in our example. For each SELECT statement, a window of data is presented; all three windows are produced. However, if the SELECTs are fast enough, you may notice only the last SELECT output.

The three windows are active on the screen and can be viewed by selecting the appropriate task from the windows task bar, typically at the bottom of a window.

If an error occurs and a Stop on error option is selected (as specified under the Options pull-down menu (6) in Figure 7-41), the program stops and the statement where the error occurred remains selected. The statement is ready to be run after it is corrected.

### Running all SQL statements currently active

You can run sequentially all the SQL statements that are currently active in your session to the iSeries server. Using our example, this would start with

```
select * from cust_dim; 3 through SELECT CUSTKEY ... IN(1,5); 5
```

You run all the SQL statements by doing *one* of the following tasks:

- ▶ Click the **All** action (A).
- ▶ Click the **All** icon (leftmost down arrow or leftmost hour glass) associated with 1 in our example in Figure 7-41.
- ▶ Press Ctrl+R from the workstation keyboard.

If an error occurs and a Stop on error option is selected (as specified under the Options pull-down menu (6) in Figure 7-41), the program stops, and the statement where the error occurred remains selected.

### SQL statement syntax check

Using this option (C) in Figure 7-41), it is possible to validate a selected SQL statements or statements. This function performs a formal syntax check of the statement, while validating that the objects referenced (libraries, tables, columns) actually exist in the linked database. Resulting messages appear in the result panel. This option can also be invoked by pressing Ctrl+K after selecting an SQL statement.

### Linking to the Visual Explain component

In V4R5, two more icons (D and E in Figure 7-41) were added to the Run SQL Script tool bar. These icons provide access to the Visual Explain function, as do the two new menu items (D and E) under Visual Explain. For more information, refer to Chapter 10, “Visual Explain” on page 301.

The *Explain* option (D), or using Ctrl+E, allows you to review the Optimizer access plan that will be used when executing an SQL statement; the statement is not actually run but optimized with the query attribute Time Limit set to 0. For details on query attributes, see 7.4, “Change Query Attributes” on page 217. It produces a visual explanation of the statement but does not access the actual data from the database, therefore avoiding the unnecessary I/O load.

The *Run and Explain* option (E), or using Ctrl+U, runs the SQL statement and gathers execution time statistics from the statement. It uses the actual access plan from the statement and the statistics and presents these in a graphical format. With this option, the statements are executed before the analysis graph is reported.

## Linking to the SQL Performance Monitor component

Using the Recent SQL Performance Monitors option (f) under Visual Explain in Figure 7-41 on page 210, you can obtain a list of the most recent SQL Performance Monitor collections and can then link into the tool to analyze collected data. See 7.6, “SQL Performance Monitors” on page 220, for a discussion on the characteristics and usage of this tool.

## Controlling SQL run options

By selecting Options from the Run SQL Scripts menu bar (g) in Figure 7-41 on page 210), you can control what to do if an SQL error occurs and what levels of additional information should be included in your session to the iSeries server:

- ▶ **Stop on Error:** This turns stopping on or off when there is more than one SQL statement to run and an error occurs. If it is turned on (default), the SQL statements are stopped at the SQL statement in error, which remains selected. If it is turned off, all SQL statements continue to run until the end of the script has completed.
- ▶ **Smart Statement Selection:** This turns on or off treating the selected SQL statement as a complete statement or attempting to run only the selected text. If it is turned *on* (default), the complete statement, up to the ending semi-colon (;) character, is attempted. If it is turned *off*, only the selected text is attempted. If you attempt to run only a portion of the original statement, the statement may complete successfully. However, you are subject to at least two error conditions:
  - Omitting some text may make the SQL statement fail, because the statement is incomplete.
  - Omitting some text may still result in successful completion. However, if the JDBC data source used for your session is set to \*NONE for commitment control, omitting a phrase an UPDATE statement, such as WHERE CUSTKEY = 1, may update all the rows in the table, which is not what was intended.See 7.3.1, “ODBC and JDBC connection” on page 202, for additional information about commitment control. The most complete OS/400 documentation on commitment control is in *Backup and Recovery*, SC41-5304.
- ▶ **Include Error Message Help in Run History:** This turns on or off (default) the inclusion of additional error message information in the Run History pane when an error occurs. Figure 7-43 shows an example where we specified an invalid column name (WRONGCOL) for the table.

**Note:** This option is no longer available in Operations Navigator Run SQL Scripts in V5R1. Detailed error messages are always displayed in the messages tab on bottom frame.

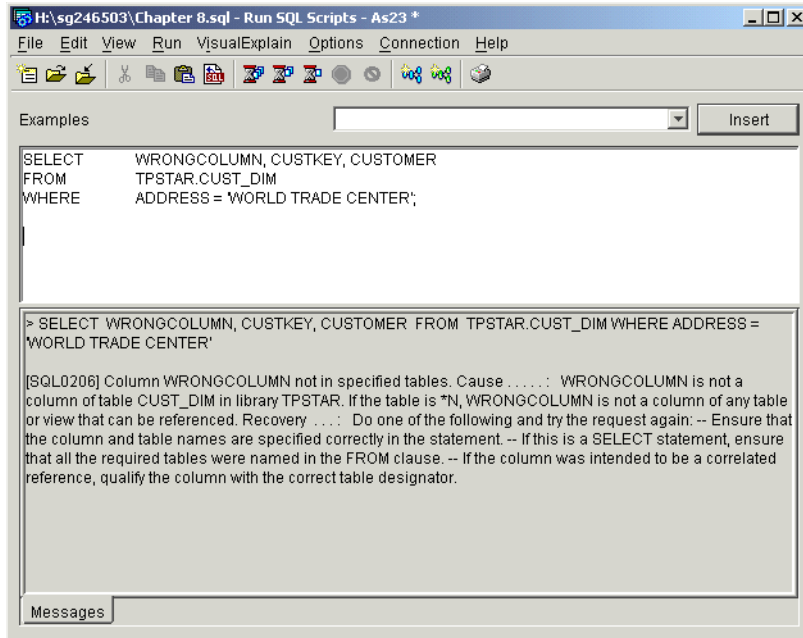


Figure 7-43 Run SQL Scripts: Include Error Message Help in Run History

- ▶ **Include Debug Messages in Job Log:** This option tells the OS/400 query optimizer support to record its decisions on how to process the SQL request, including any recommendation for creating an index that may improve performance. The option is typically used only when debugging new and complex SQL statements or while analyzing a suspected performance problem.

Analyzing the job log messages may be sufficient to determine if a performance problem exists and what action should be taken to resolve the problem. You may also consider using the Operations Navigator interface to the SQL Performance Monitor, which is described in 7.6, “SQL Performance Monitors” on page 220, and Visual Explain, described in Chapter 10, “Visual Explain” on page 301.

Figure 7-44 shows an example of an SQL JOIN statement and the associated job log messages that should be reviewed.

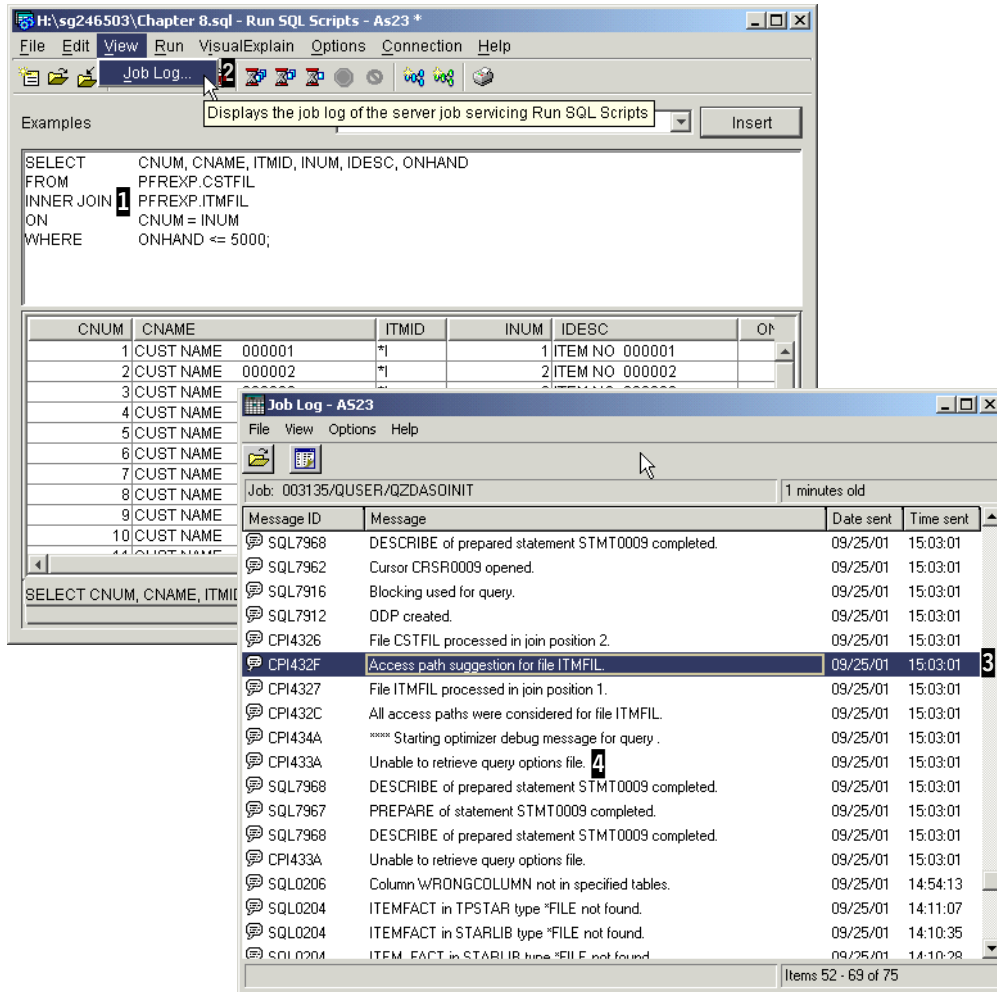


Figure 7-44 Run SQL Scripts: Include Debug Messages in Job Log

We use the selected SQL SELECT with JOIN statement (1) to show the associated job log debug messages issued by the query optimizer. To see the current Operations Navigator session's job log, complete these tasks:

- Click **View** in the Run SQL Scripts panel.
- Click **Job Log** (2).

In our example job log, we discuss two messages: the optimizer's suggestion for an access path (index) to file ITMFIL with message ID CPI432F (3) and error message CPI433A (4).

By double-clicking message CPI432F, the message details or "second-level text" is displayed. The message text describes why the create index function is recommended and the recommended column names to include in the new index.

Message CPI433A may appear multiple times in the job log of a job that has run several SQL statements. Each time an SQL statement is run, the system looks for a file or table by the name of QAQQINI in the QUSRSYS library. This table can be set up by you to specify query attributes that the OS/400 query optimizer will use while processing each SQL statement.

If you are not attempting to modify the default OS/400 query processing algorithm through this table, the table will not be in the QUSRSYS library, and this message is considered for information only.

- ▶ **Run Statement On Double-Click:** This option has been added in V5R1. When it is turned on, it allows the running of a SQL statement by double-clicking the SQL statement.
- ▶ **Change Query Attributes:** This allows you to easily modify the query options file QAQQINI for your job, provided you remember the job number previously checked in the job log, or for any other job in the system.

This is done using the same interface as documented in 7.4, “Change Query Attributes” on page 217.

### 7.3.5 DDM/DRDA Run SQL Script configuration summary

Using Figure 7-37 on page 201, at [2](#), we showed and discussed an SQL CONNECT statement (“connect to as20;”) to access data on a remote system. For ease of reference, this statement also appears in Figure 7-44. This section provides overview information on configuration parameters required to successfully access remote data.

For DRDA to work between a source system (function requester) and target system (request server) where the actual data is and the SQL function is performed, you need a certain DRDA configuration to be set up correctly. The following steps summarize the configuration required (using As25 as the source or requester system and As20 as the target or server system).

On the As25 (source system), complete the following steps:

1. Start TCP/IP.
2. Enter the OS/400 Add Relational Database Directory Entry (ADDRDBDIRE) command:

```
ADDRDBDIRE RDB(AS20) RMTLOCNAME(AS20 *IP) TEXT('Remote DB system via TCP/IP')
```

This relational database entry identifies a database name (RDB parameter), the remote system name, and that the connection is over TCP/IP. TCP/IP must be active on both the source and target systems. A Domain Name Services (DNS) server must be active in the network to resolve to the actual IP address.

Note that DRDA runs over SNA connections as well as TCP/IP.

3. Enter the Add Server Authentication Entry (ADDSVRAUTE) command:

```
ADDSVRAUTE USRPRF(TEAM02) SERVER(AS20) PASSWORD(T02EAM)
```

The SQL CONNECT TO *target system (remote server)-database* statement can explicitly specify USER (user ID) and USING (password) information. If it does not, the user ID and password information specified in the ADDSVRAUTE command are passed to the remote server. Depending on the target system’s (remote server) security requirements for clients to connect to it, a user ID and, optionally, a user password are required that must be successfully validated on the remote server.

We strongly recommend that you enter the user profile, server name, and password values in uppercase.

4. To specify a password value for the ADDSVRAUTE command’s PASSWORD parameter, the source system Retain server security data (QRETSVRSEC) system value must be set to 1.

**Note:** To use ADDSVRAUTE support, your user profile must specify \*SECADM special authority. You must also have \*OBJMGT and \*USE authorities to the user profile specified on this command.

On the As20 target (remote server) system, follow these steps:

1. Start TCP/IP.
2. Start the TCP/IP DDM server.

The DDM server jobs run in subsystem QSYSWRK. The jobs are named QRWTLSTN (daemon) and QRWTSRVR (server, one per connection).

The network attributes' DDM/DRDA Request (DDMACC) parameter for processing received DDM/DRDA requests is set to \*OBJAUT. This means normal OS/400 processing user profile authority to the requested file or table is performed.

This target or server system can be configured to not require a password from the source system. You do this by using the OS/400 Configure TCP (CFGTCP) command interface. Then select **Configure TCP/IP applications->Change DDM TCP/IP Attributes**.

3. A target system user ID and password must correspond to the user ID and password, if used, received from the requesting source system.

## 7.4 Change Query Attributes

Using the Change Query Attributes item that becomes available when you right-click Database gives you an easy way to change your query options for accessing the database. However, you must be aware that some of the options that are available here can be manipulated using the Change Query Attribute (CHGQRYA) CL command on the iSeries server. There is not an exact one-to-one correspondence. For a detailed discussion on the implications of changing query attributes, refer to the manual *DB2 UDB for iSeries Database Performance and Query Optimization* in the iSeries Information Center.

You can access the Change Query Attributes panel in two ways:

- ▶ From Operations Navigator, right-click **Database** and select **Change Query Attributes**.
- ▶ From the Run SQL Scripts center, select **Options** and **Change Query Attributes**.

You then see the Change Query Attributes dialog as shown in Figure 7-45.

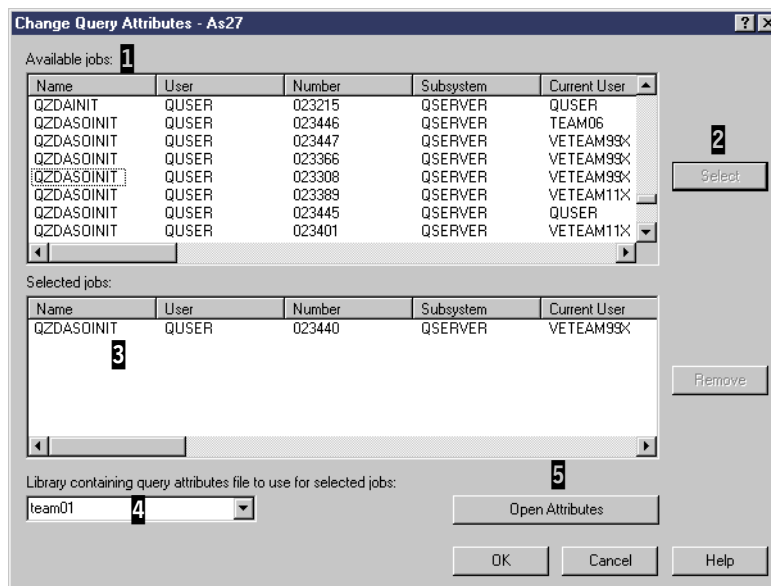


Figure 7-45 Change Query Attributes panel

Now proceed with the following steps:

1. In the upper part of the window (1), you see a list of all jobs currently active in the system. Scroll through the list to locate the job you are interested in, click on its name, and use the

Select button (2) to move the selection in the bottom part of the panel (3). You can select more than one job and set common query attributes for all of them at the same time.

- At this point, you can specify a library (4) in which you want the original QAQQINI file to be copied. Click the **Open Attribute** button (5), which allows you to edit the copy of QAQQINI you just made in the library (4), as shown in Figure 7-46.

QQPARAM	QQVAL	QQTEXT
APPLY_REMOTE	*DEFAULT	
QUERY_TIME_LIMIT	*DEFAULT	
PARALLEL_DEGREE	*DEFAULT	
ASYNC_JOB_USAGE	*DEFAULT	
MESSAGES_DEBUG	*YES	
PARAMETER_MARKER_CONVEF	*DEFAULT	
UDF_TIME_OUT	*DEFAULT	
OPTIMIZE_STATISTIC_LIMITATIC	*DEFAULT	
FORCE_JOIN_ORDER	*DEFAULT	

Figure 7-46 Editing QAQQINI

- Click the cell you want to change and type the new value. As shown in Figure 7-46, we change the setting for MESSAGE\_DEBUG from the original value \*DEFAULT to \*YES, therefore, stating that we want debug messages to be recorded for the selected job. When you press Enter to activate your changes, you receive a warning message (Figure 7-47).
- Click **Yes**.

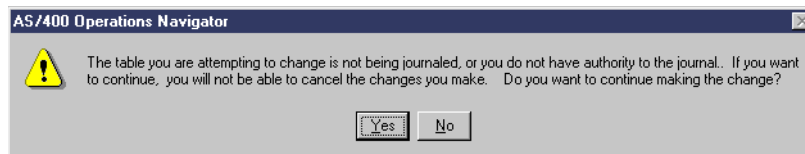


Figure 7-47 Warning message on modification of the QAQQINI file

- You are brought back to the Change Query Attributes panel. Click **OK** to make the change effective.

The options that are currently managed in the QAQQINI file and their values are documented in *DB2 UDB for iSeries Database Performance and Query Optimization*.

## 7.5 Current SQL for a job

You can use this function to select any job running on the system and display the current SQL statement being run, if any. Besides displaying the last SQL statement being run, you can edit and rerun it through the automatically linked Run SQL Scripts option and display the actual job log for the selected job or, even end the job. This can also be used for database usage and performance analysis, with the Visual Explain tool documented in Chapter 10, “Visual Explain” on page 301.

To start it, right-click the **Database** item in Operations Navigator and select **Current SQL for a Job**. You are presented with the dialog shown in Figure 7-48.



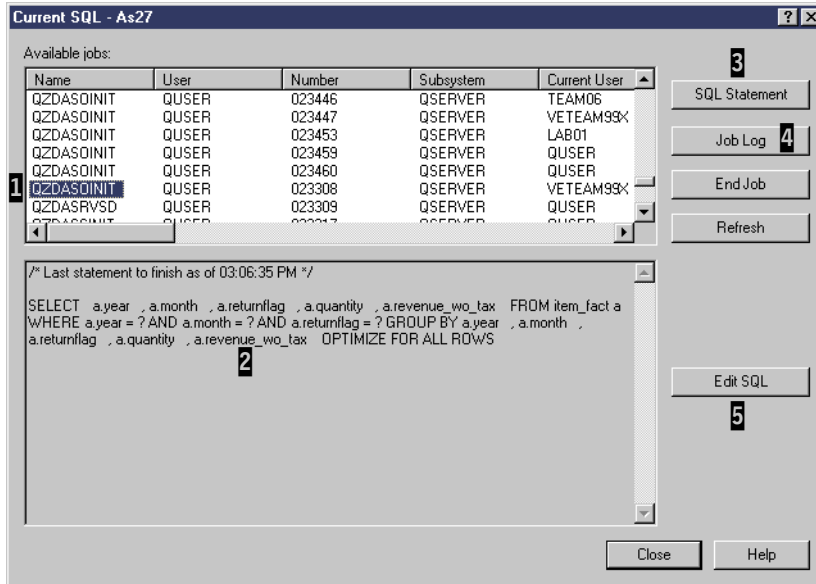


Figure 7-48 Current SQL for a Job

The Current SQL 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 Scripts center, or even end the job, provided you have sufficient authority.

In our example, we selected an ODBC job (1) and displayed the last SQL statement it ran in the bottom part of the panel (2) using the SQL Statement button (3). To go to its job log, we would use the Job Log button (4).

After the SQL statement is brought in the bottom part of the panel, it is possible to use the Edit SQL button (5) to work on this same statement with the Run SQL Scripts center that was previously documented in this redbook. See Figure 7-49.

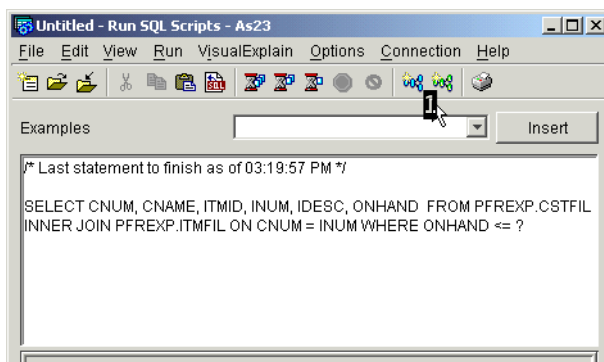


Figure 7-49 Working with current SQL for a job

From here, it is also possible to link into Visual Explain, using the appropriate menu item or the icons (1) to help you with database performance analyses. For a discussion on this tool, refer to Chapter 10, “Visual Explain” on page 301.

As you may have already noticed, all Operations Navigator database tools are tightly integrated into each other to make it easier for the user to fully exploit their capabilities.

## 7.6 SQL Performance Monitors

You can analyze the performance of iSeries SQL statements by putting the appropriate OS/400 job into debug mode, running the SQL statements, and viewing the query optimizer messages in the job log. You can see an example of using job log messages in “Controlling SQL run options” on page 213.

This section describes a more powerful SQL performance analysis tool that initially appeared in V4R4 Operations Navigator and was further enhanced in V4R5. This support provides a graphical interface to IBM-provided SQL queries against data collected by the *Memory Resident Database Monitor* that was introduced in V4R3. In addition to output equivalent to the debug mode optimizer messages, this monitor can monitor multiple jobs and show the actual SQL statement. This interface is referred to as the *SQL Performance Monitors*.

The SQL Performance Monitor, which was originally available in V4R4, only allowed gathering *summary* performance information from the Memory Resident Database Monitor. In V4R5, it is possible to enhance the usability of this interface by collecting *detailed* performance information. For a detailed discussion on the Memory Resident Database Monitor, refer to *DB2 UDB for iSeries Database Performance and Query Optimization*.

Before you start an SQL Performance Monitor, you need to determine which job or jobs you want to monitor. There are several techniques you can use to determine the job. We list some of them here:

- ▶ If you are using SQL statements running Operations Navigator **Database-> Run SQL Scripts**, you can click the **View** option from the menu bar. On the drop-down menu that appears, click **Job Log** to see your current job’s job log. Included in the gray header portion of the job log messages is the name of the job, for example, *139224/QUSER/QZDASOINIT*. You can scan down to the earliest job log messages to confirm this job is actually running under the user profile you think it should be.
- ▶ If you are not running the job that needs to be monitored, you can get the job name from the user of the job, if possible.
- ▶ If you know the user profile running the SQL jobs, but do not know which job is the one you want to monitor, you can use the OS/400 Work with Object Locks (WRKOBJLCK) command to find the jobs running with that user profile. You may receive more jobs than you anticipated. Then, you may need to look in the job logs of each job for some SQL-like messages to determine which job or jobs to monitor, for example:

```
WRKOBJLCK OBJ(QSYS/TEAM02) OBJTYPE(*USRPRF) MBR(*NONE)
```

This command resulted in five jobs running with user profile TEAM02: one job name starting with QPADEV000L (5250 emulation), two jobs running Client Access Express database serving with job name starting with QZDASOINIT (not using SSL), and two jobs with the job name starting with QZRCSRVS (central server functions). We looked in the job logs for the two QZDASOINIT jobs and in one of them found the message:

```
148 rows fetched from cursor CRSR0002.
```

This QZDASOINIT job was set by Operations Navigator Run SQL Scripts to Include debug messages in a job log.

- ▶ You can use the Operations Navigator server jobs interface to find the job by selecting from Operations Navigator **Network->Servers-> Client Access Database** and select **Server Jobs** to view the Client Access Express servers (circled in Figure 7-50).

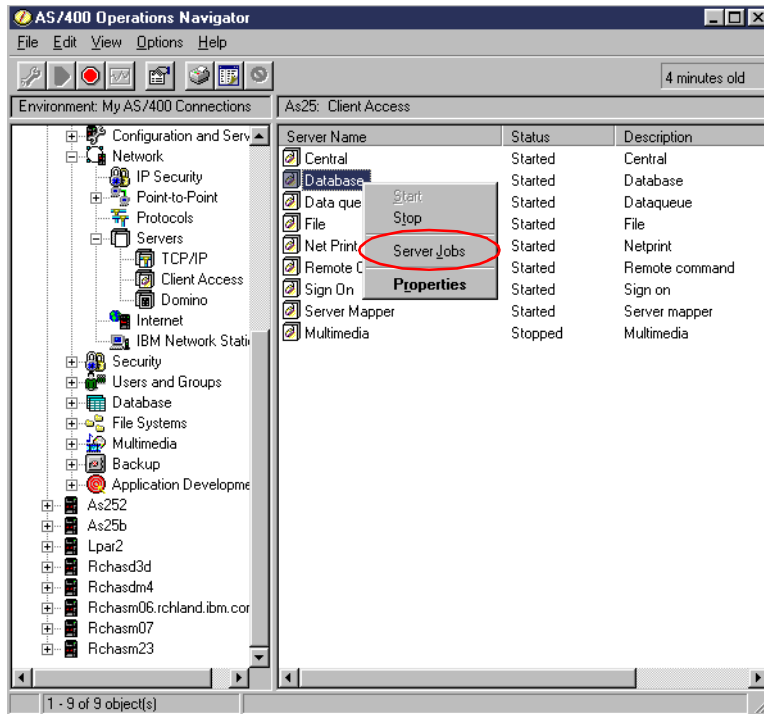


Figure 7-50 Finding the database server job (Part 1 of 2)

When you click Server Jobs, a window appears similar to the one shown in Figure 7-51. This display shows the database server jobs, QZDASOINIT (not using SSL), that are currently started and shows a current user ID for jobs currently doing active database functions.

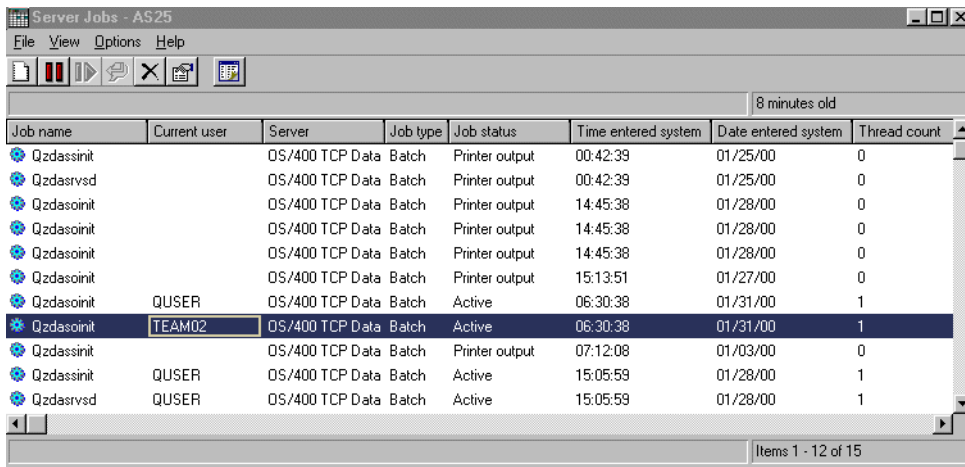


Figure 7-51 Finding the database server job (Part 2 of 2)

This display illustrates an advantage of using the Operations Navigator “servers” support to find a job, compared to using OS/400 5250-display based commands such as the Work with Subsystem Jobs (WRKSBSJOB), Work with Active Jobs (WRKACTJOB), or Work with Object Locks (WRKOBJLCK) commands.

The Operations Navigator interface lists the jobs based on their function. With the OS/400 commands, you need to understand what OS/400 subsystem the server jobs run in and the job name that identifies the server function. In our example, you need to know that the QZDASOINIT jobs do the database serving (in this case ODBC-based) work. You also need to look into the job logs of each active job to find the actual user ID (profile) using the job.

The OS/400 commands we discussed show equivalent jobs with the user ID as QUSER. QUSER is the user profile assigned by the system for pre-started Client Access database server jobs. The user profile name actually using the job is indicated in a job log message. The Operations Navigator interface examines the job log messages and shows the active user profile (TEAM02, in our example) if the pre-started job is currently in session with a signed on client.

## 7.6.1 Starting the SQL Performance Monitor

To run an SQL Performance Monitor, you need to:

1. Define a new monitor.
2. Determine whether it's going to be a *Detailed* collection or a *Summary* collection.
3. Specify the jobs to be monitored and the data to be collected for a Summary collection.

The Detailed collection is discussed later in “Detailed SQL Performance Monitor example” on page 226.

### Summary SQL Performance Monitor example

To start the SQL monitoring process, follow these steps:

1. Right-click **SQL Performance Monitors**, and select **New** as shown in Figure 7-52.

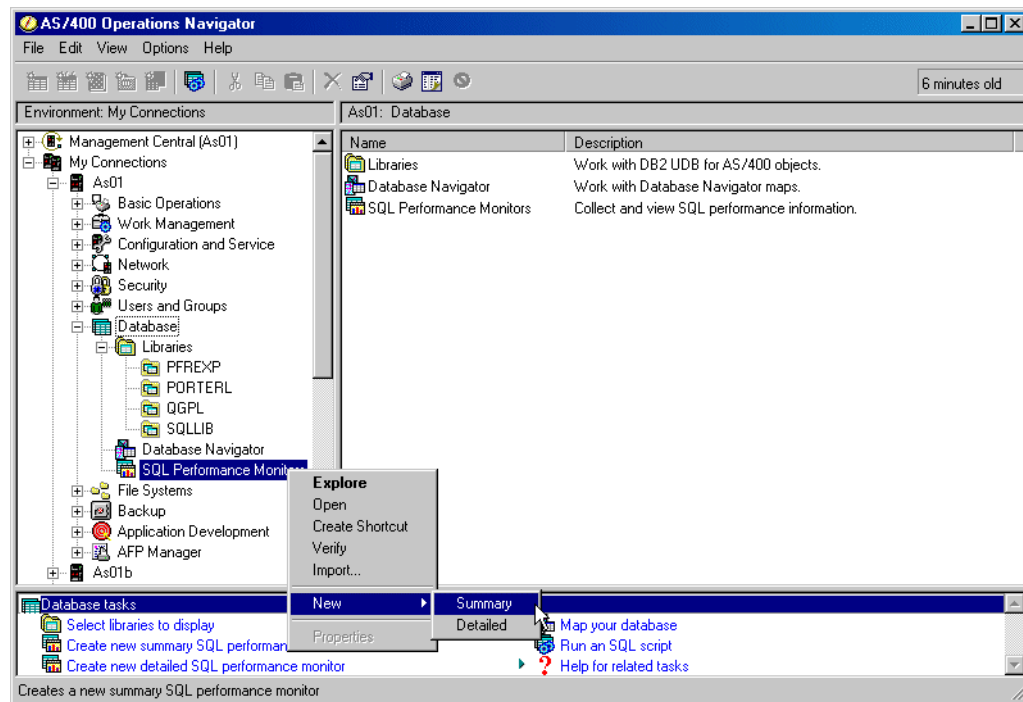


Figure 7-52 SQL Performance Monitor (Part 1 of 5)

2. Select **Summary**. This brings up the New SQL Performance Monitor dialogue panel with three tabs: General, Monitored Jobs, and Data to Collect.

The General tab is shown in Figure 7-53.

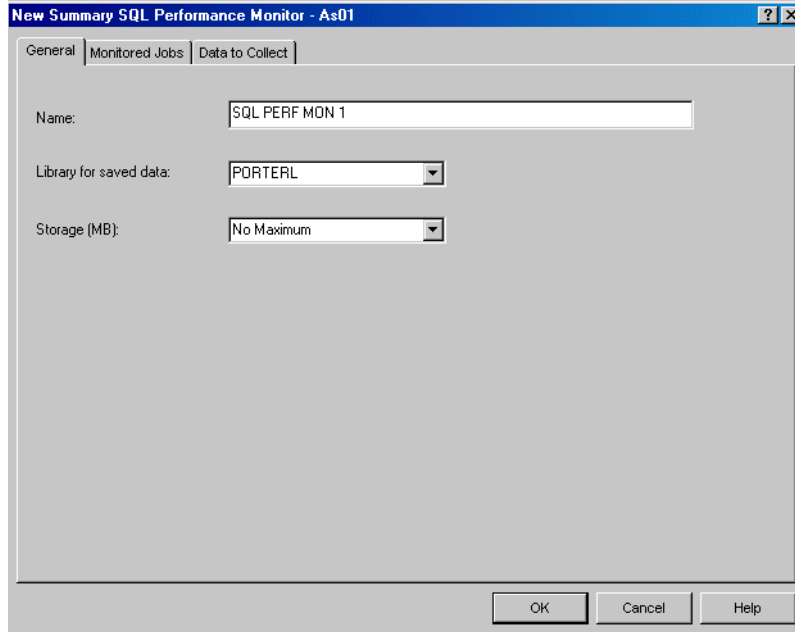


Figure 7-53 Starting a Summary SQL Performance Monitor (Part 2 of 5)

We entered the monitor name, the library name that is used to contain the collected data, and the amount of main storage allocated to the monitoring process.

Do *not* click the OK button yet. Monitoring all jobs will start if you have not selected specific jobs under the Monitored Jobs tab. Monitoring all jobs is not recommended on a system with hundreds of active jobs because the monitoring process can degrade performance.

3. To specify which OS/400 jobs to manage, click the **Monitor Jobs** tab, which brings up the panel shown in Figure 7-54.

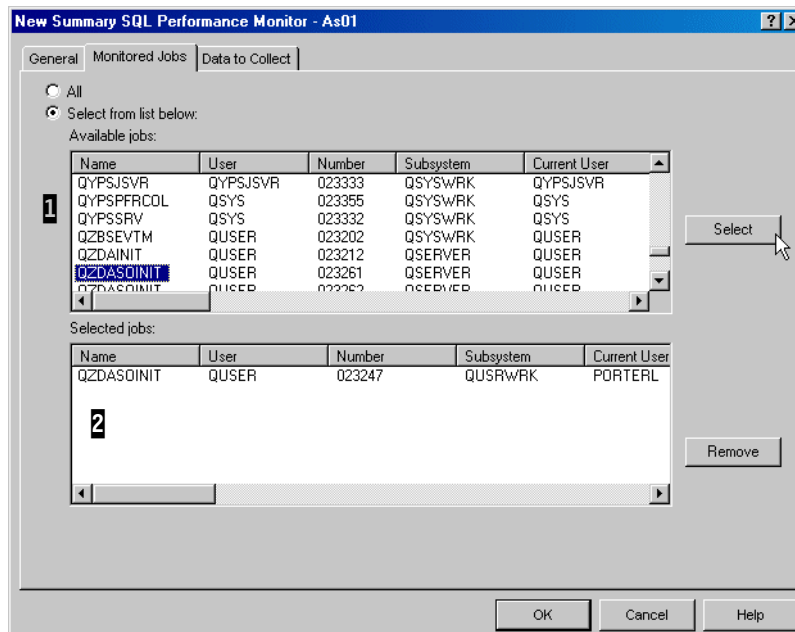


Figure 7-54 Starting a Summary SQL Performance Monitor (Part 3 of 5)

4. You can select to monitor all jobs or to select jobs from the Available jobs list pane (1 in Figure 7-54). After you select a job and click the **Select** button, the job information is entered into the Selected jobs list pane (2 in Figure 7-54). You remove selected jobs by selecting a job in the Selected jobs pane and clicking the **Remove** button.

In this example, we scrolled down the active job names to display the ones shown in 1. We select to monitor only job QZDASOINIT/QUSER/023247 with PORTERL as the current user. We recommend that you monitor as few jobs as possible, because monitoring a large number of active jobs could impact normal productivity.

5. When you are finished selecting jobs, click the **Data to Collect** tab. This brings up the panel shown in Figure 7-55.

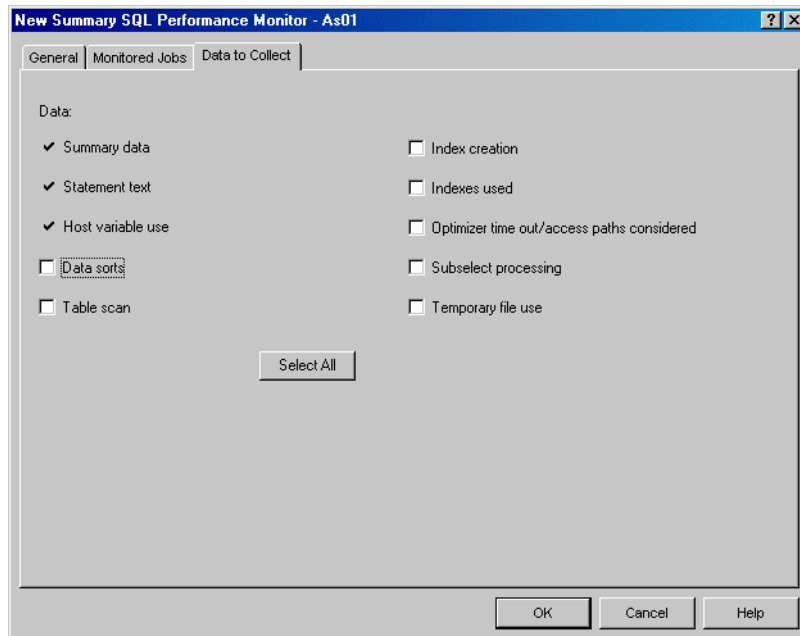


Figure 7-55 Starting a Summary SQL Performance Monitor (Part 4 of 5)

6. This panel shows three sets of SQL monitor data collected during every monitor collection period. You can specifically include other sets of data or simply click the **Select All** button. You should select *all*, unless you understand the application implementation in detail so that you need to collect only specific information.

When you are satisfied with your monitor collection specification, click the **OK** button to return to the original SQL Performance Monitor window, which shows the monitor status on the right pane in Figure 7-56.

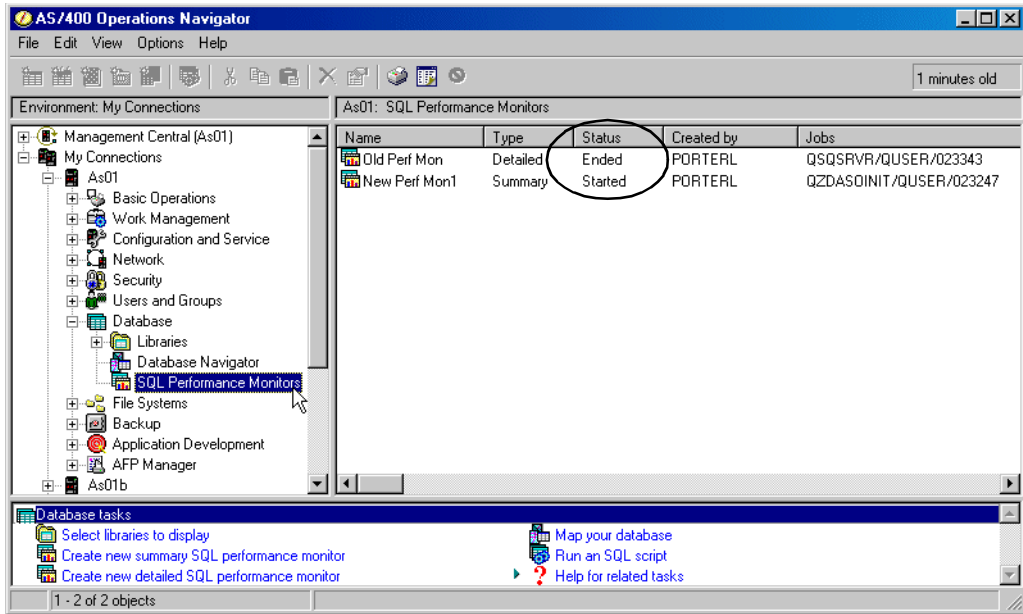


Figure 7-56 Starting a Summary SQL Performance Monitor (Part 5 of 5)

In our example, we used Run SQL Scripts to run the SQL statement. This statement has a relatively complex WHERE clause as shown in Figure 7-57. Run SQL Scripts is discussed in more detail in 7.3, “Run SQL Scripts” on page 197.

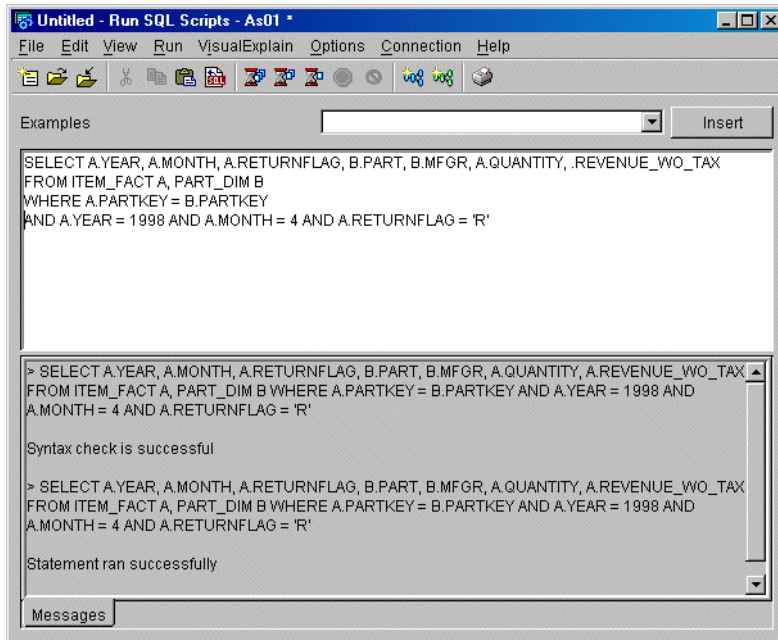


Figure 7-57 SQL Performance Monitor: SQL Statement which was monitored

Operations Navigator Run SQL Scripts support uses JDBC support. In our example figure, the SQL statement was already run based on the evidence of its appearance within the Run History pane. The message Opening results viewer... indicates the results of the SQL select statement has already been displayed to the Operations Navigator user.

The SQL Performance Monitor can monitor all SQL work performed on OS/400. In addition to Operations Navigator Run SQL Scripts jobs, other users of OS/400 SQL support would include a client workstation Visual Basic program accessing the OS/400 via ODBC, a client workstation Java applet accessing the OS/400 via Java Database Connectivity (JDBC), a local iSeries program using embedded SQL in the RPG, COBOL, or C program, a local iSeries program using the SQL CLI (Call Level Interface) in RPG, COBOL, C, or Java.

OS/400 also has a 5250 workstation-based SQL interface running under the Start SQL (STRSQL) command.

### Detailed SQL Performance Monitor example

To start the SQL monitoring process, right-click **SQL Performance Monitors**, and select **New**. Select **Detailed** as shown in Figure 7-58.

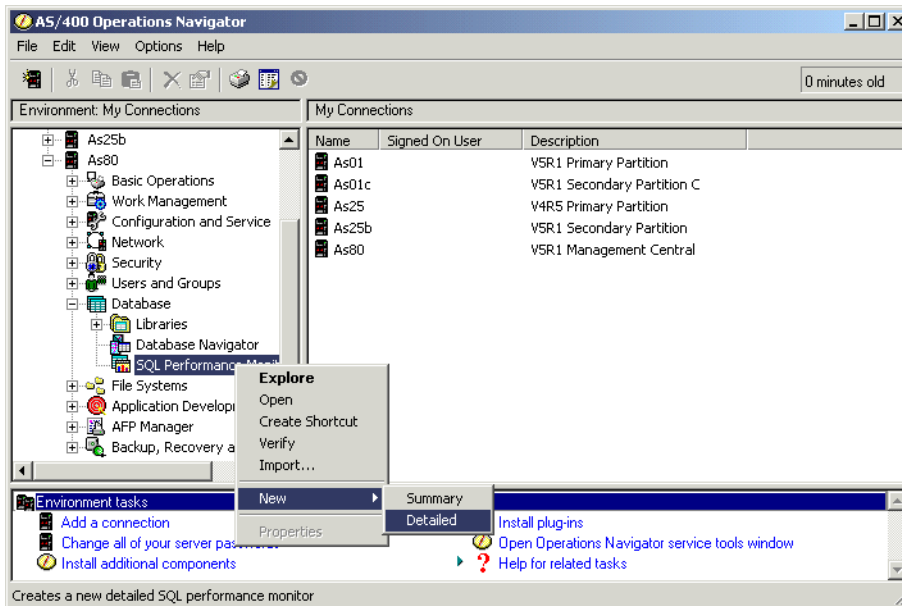


Figure 7-58 Starting a Detailed SQL Performance Monitor

Name the monitor. Select a library for the collected data and proceed to select the jobs you want to monitor as previously documented in “Summary SQL Performance Monitor example” on page 222.

## 7.6.2 Reviewing the SQL Performance Monitor results

The SQL Performance Monitor statistics are kept in main storage for fast recording, but need to be written to database files to use the Operations Navigator interface to review the results. You can have the statistics written to database files by either *pausing* or *ending* the monitor.

Right-click the active SQL Performance Monitor. A pop-up window appears that lists the Pause, End, and other monitor actions as shown in Figure 7-59.



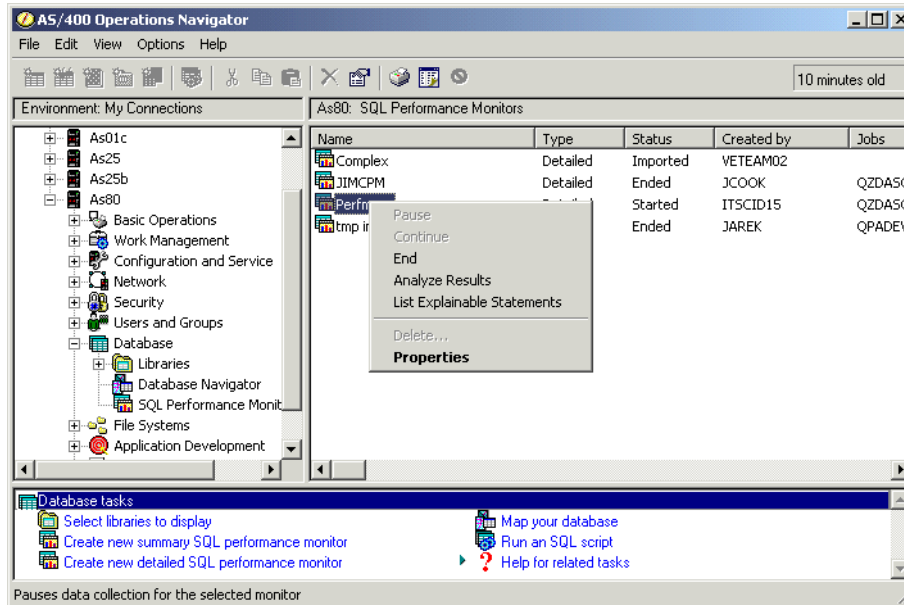


Figure 7-59 Managing the SQL Performance Monitor

The possible managing functions are:

- ▶ **Pause:** This 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 restarts the collection of statistics for a monitor that is currently paused.
- ▶ **End:** This stops and ends the monitor and writes the current collection of statistics to the database files or tables.
- ▶ **Analyze Results:** This brings up a window with three tabs for selecting ways to look at (query) the collected statistics in the database files or tables:
  - Summary Results
  - Detailed Results
  - Composite View
- ▶ **List Explainable Statements:** This opens a dialog listing the SQL statements for which the detailed SQL Performance Monitor has collected data and for which a Visual Explain diagram can be produced. See “Listing Explainable Statements” on page 231 for an example.
- ▶ **Properties:** This brings up a window with three tabs that represent the original monitor definition:
  - General
  - Monitored Jobs
  - Saved Data

An example of the Saved Data tab with the details for our monitor is shown in Figure 7-60.

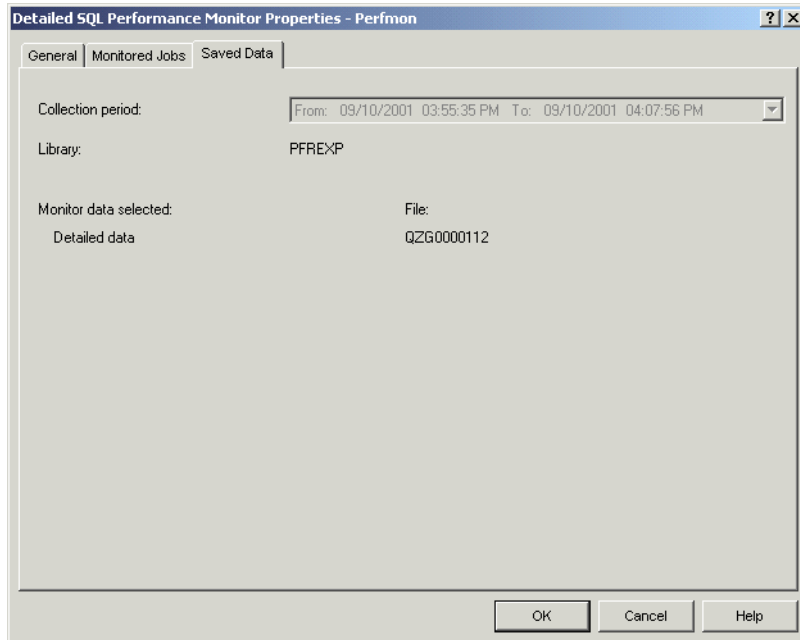


Figure 7-60 SQL Performance Monitor: Properties

The SQL Performance Monitor file name numeric suffix is updated when each monitor is started.

### Analyzing a summary of SQL Performance Monitor results

OS/400 provides many pre-defined queries to view the recorded statistics. You can select these queries by checking the various query types on the Analyze Results panels. To begin viewing the results, right-click the paused or ended monitor. Select **Analyze Results** from the pop-up window. Here we analyze results for a Summary Monitor.

Figure 7-61 shows the first results panel that groups queries according to three tabs:

- ▶ Summary Results
- ▶ Detailed Results
- ▶ Composite View

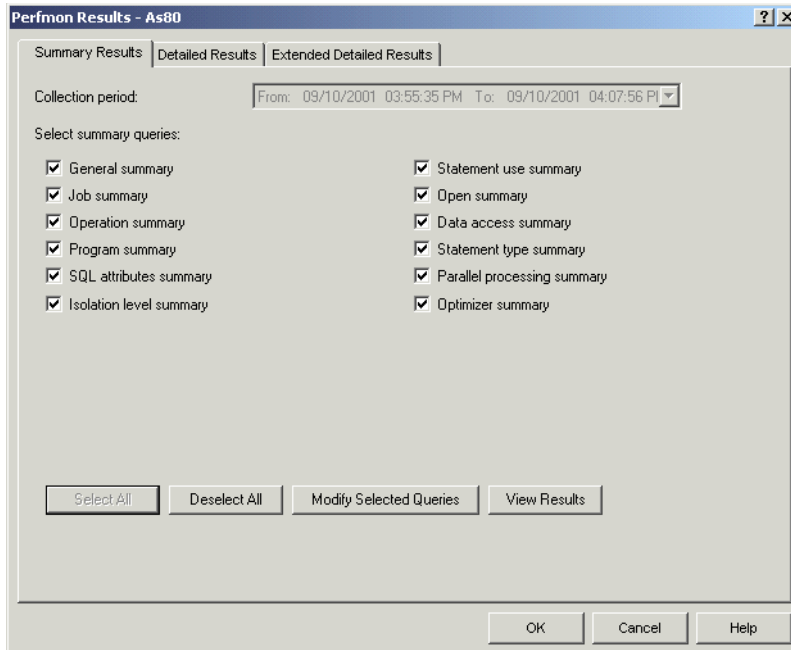


Figure 7-61 SQL Performance Monitor: Analyze Results - Summary results

You can select individual queries or use the **Select All** button. After you select the queries you want to run, click the **View Results** button. You can even choose to modify the pre-defined queries and run the new queries by clicking the **Modify Selected Queries** button.

An in-depth discussion of using the SQL Performance Monitor results to improve performance is beyond the scope of this redbook. However, we do show sample query results output.

To obtain the query results shown in Figure 7-63, you must first select the **Detailed Results** tab on the Performance Monitor Results window shown in Figure 7-61. This brings up the Detailed Results panel shown in Figure 7-62.

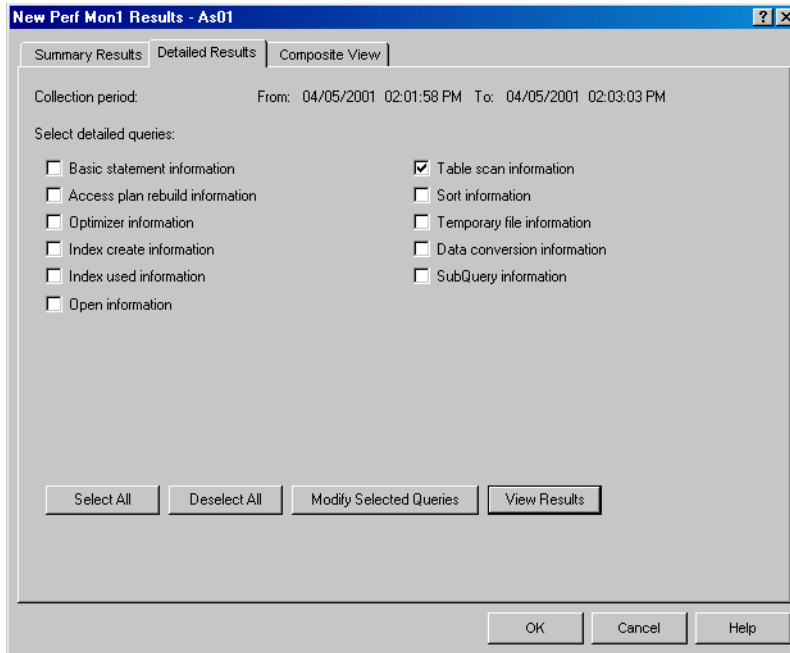


Figure 7-62 SQL Performance Monitor: Detailed Results

You can select individual detail query reports, select *all* queries, and even modify the provided queries. When you are finished selecting the queries you want, click the **View Results** button.

The OS/400 query optimizer support includes an Index Advisor function. This support includes, when appropriate, a recommendation that a new index should yield improved performance. Columns that should be used in the index are listed. To view this detailed information, you must first select to view Arrival Sequence Information (1 in Figure 7-63). Click the **View Results** button to access a panel similar to the one shown in Figure 7-64.

	Time	Estimated Processing Time	Reason Code	Total Rows	Estimated Rows Selected	Advised Index	Advised Index Keys
1	2001-04-05 14:02:20.146662	.014	T1	600572	601	Yes	YEAR, MONTH, RETURNFLAG
2	2001-04-05 14:02:01.435231	.001	T1	2	2	No	-
3	2001-04-05 14:02:01.435231	.001	T1	2	2	No	-
4	2001-04-05 14:02:01.435231	.001	T1	2	2	No	-
5	2001-04-05 14:02:01.435231	.001	T1	2	2	No	-

Figure 7-63 SQL Performance Monitor: Table Scan Information

To view the information in Figure 7-63, we had to scroll to the right to find the columns Advised Index and Advised Index Keys (1). You can see that we compressed several columns in the results to make the index path information fit within the window (2).

A lab exercise can be downloaded to your iSeries server on a PC workstation as listed in 7.1.1, “New in V5R1” on page 159. The “Self study lab” can be used to familiarize yourself with the power of the SQL Performance Monitor, as well as most of the Operations Navigator Database support. It also includes tips on tuning SQL performance.

### Analyzing a detailed SQL Performance Monitor

Most of the discussion in “Analyzing a summary of SQL Performance Monitor results” on page 228 also applies to a detailed monitor.

Figure 7-64 shows the first results panel that groups queries according to three tabs:

- ▶ Summary Results
- ▶ Detailed Results
- ▶ Extended Detailed Results

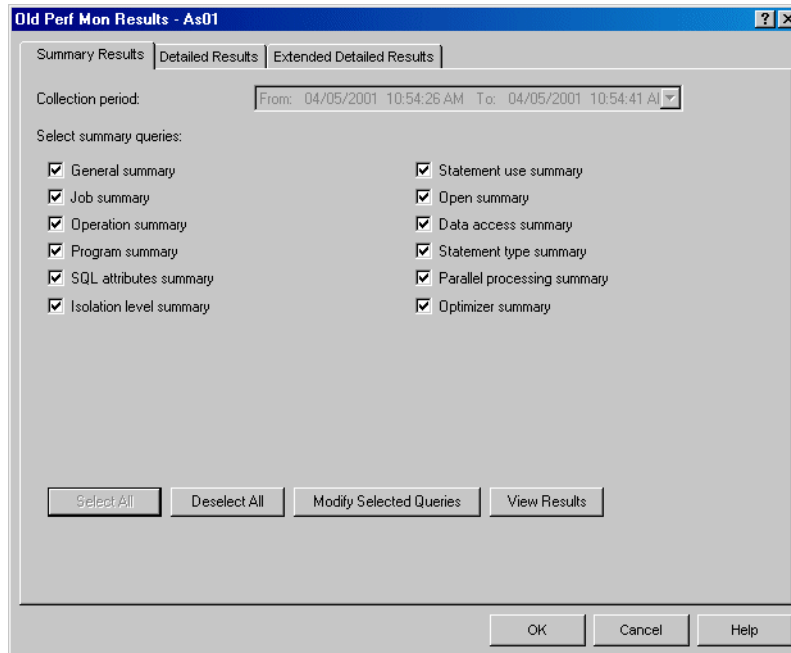


Figure 7-64 Detailed SQL Performance Monitor: Analyze Results - Detailed Monitor

For each of these options, you can run any of the pre-prepared queries or modify them for your own analysis.

Although the items listed under the Detailed and Extended Detailed Results tabs have the same names and descriptions, the underlying queries are different. The Extended ones allow you a more complete understanding of the Optimizer choices. You can easily verify this by selecting the same item in both lists and clicking the **Modify Selected Query** button to have the SQL statement opened with the Run SQL Script center.

### Listing Explainable Statements

The Explainable Statements for SQL Performance Monitor dialog lists the SQL statements for which an SQL Performance Monitor has collected detailed data and for which a Visual Explain graph can be produced.

To access this function, click **Operations Navigator->Database->SQL Performance Monitors**. Then you see a list of the SQL Performance monitors that are currently on the system. Right-click a *detailed* SQL Performance Monitors collection and select **List Explainable Statements**, as shown in Figure 7-65.

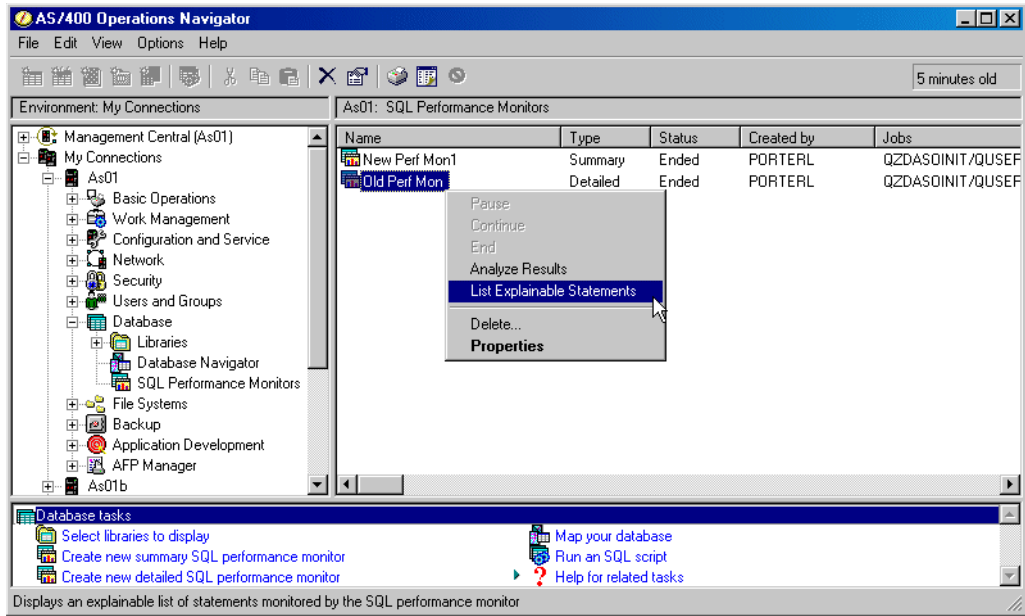


Figure 7-65 List Explainable Statements

When you select this item, you see the panel in Figure 7-66. The upper half of the panel displays the SQL statements monitored during the data collection session. Click to select the statement (1) you are interested in analyzing. The selected statement appears in the lower half of the dialog.

Once the statement is in focus, it is possible to have it analyzed and explained. Click the **Run Visual Explain** button (2). Refer to Chapter 10, “Visual Explain” on page 301, for a detailed discussion on this tool.

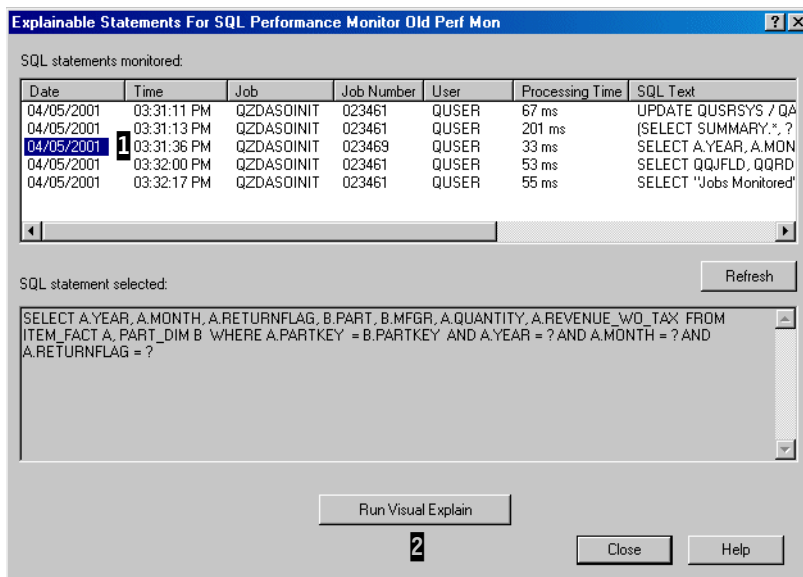


Figure 7-66 Using List Explainable Statements

As you may have already noticed, the set of database analysis options and tools provided by Operations Navigator are well interconnected and meant to be used together in an iterative fashion.

### 7.6.3 Importing data collected with Database Monitor

It is possible to import Database performance data collected with the more traditional green-screen interface tool, known as Database Monitor. You can start Database Monitor by using either the STRDBMON or STRPFRMON STRDBMON(\*YES) CL commands on the iSeries server. While it is beyond the purpose of this redbook to discuss the traditional collection methods, you will certainly be pleased to discover that this data can be analyzed with the simpler and more intuitive instruments made available with Operations Navigator, rather than with the traditional approach.

To import data collected with Database Monitor, select **Database->SQL Performance Monitor->Import** as shown in Figure 7-67.

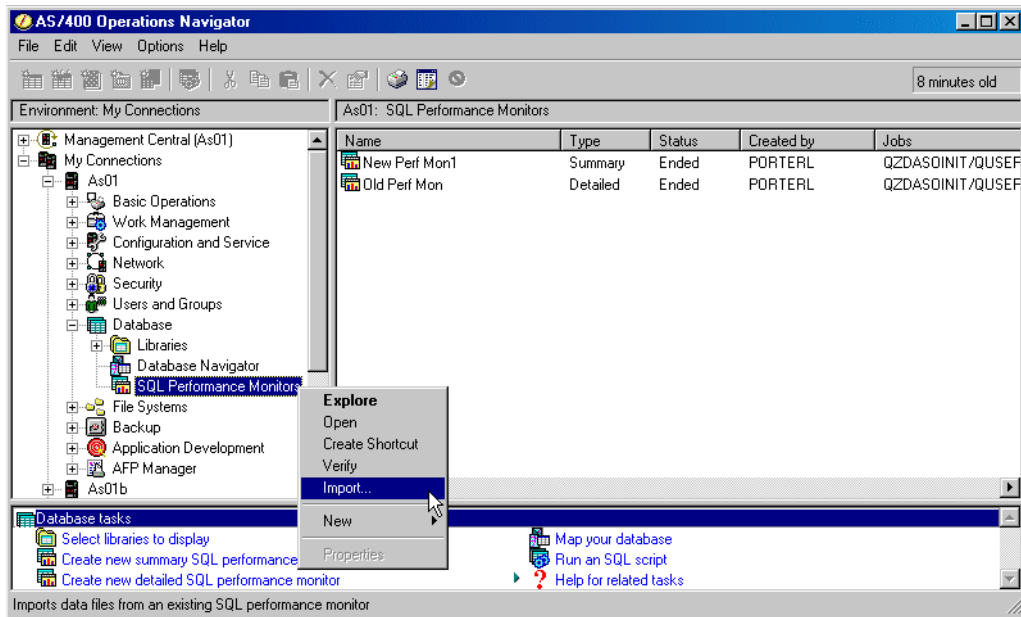


Figure 7-67 Importing data in SQL Performance Monitor (Part 1 of 2)

A panel appears like the example in Figure 7-68. On this display, you give a name to new monitor and specify the library and file containing the collected data.

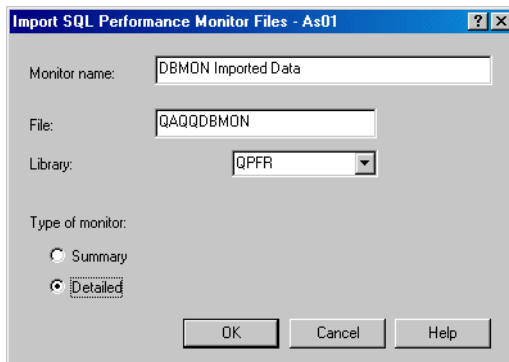


Figure 7-68 Importing data in SQL Performance Monitor (Part 2 of 2)

Due to some new fields being added to the Database Monitor file in V5R1, SQL Performance Monitor only fully supports importing and analyzing database performance data collected in V5R1. Data collected from earlier releases will not have all of the information needed by Visual Explain. The system imports data from earlier releases and converts the data to a V5R1 format. However, it can only use default values for information that was not recorded at the earlier release, and full results cannot be guaranteed when using Visual Explain.

Please refer to 7.6.2, “Reviewing the SQL Performance Monitor results” on page 226, for a discussion on how to analyze the collected data.

Exit the Visual Explain window and the Explainable Statements window after you complete your analysis. Depending on your future needs for further investigation, you may either retain the performance data or delete it from the system at this time. To delete an SQL Performance Monitor collection, right-click the data collection you are interested in, and select **Delete**.

## Importing performance data for Query/400

Query/400 is not included in the list of queries that can be monitored by the SQL Performance Monitor, even though debug messages can be used with Query/400 queries.

Because Query/400 queries are often blamed for poor performance, and sometimes even banned from execution during daylight hours, it was thought appropriate to provide guidance to bring Query/400 queries into the scope of Visual Explain.

There is no direct Query/400 to SQL command. However, the STRQMQRy CL command will run a query definition (object type \*QRYDFN) as an SQL statement, as long as the parameter ALWQRYDFN is set to either \*YES or \*ONLY.

To use this SQL statement with Visual Explain, either start an SQL Performance Monitor for this job before you issue the STRQMQRy command or use the native STRDBMON CL command to collect detailed data for the job. See 7.6.1, “Starting the SQL Performance Monitor” on page 222, for further information.

Alternatively, you can access the SQL statement by using the Current SQL for a Job option (obtained by right-clicking the database icon in Operations Navigator).

Here we document the actions you need to follow to import the database performance data collected for Query/400 using the STRQMQRy command as a workaround. For documentation on the differences between STRQRY and STRQMQRy, see: <http://publib.boulder.ibm.com/pubs/html/as400/v5r1/ic2924/info/index.htm>

Use **STRQMQRy** as a search word.

In our example, we collect database performance data on a pre-existing Query/400 definition, named TESTQRY in library ITSCID41 using Database Monitor on the iSeries server. We are going to create an SQL Performance Monitor named QRYIMPORT. We perform the following steps:

1. Start the traditional green-screen tool to collect database performance data, Database Monitor, into a file named QAQQDBMN in library ITSCD41, using the STRDBMON CL command:

```
STRDBMON OUTFILE(ITSCID41/QAQQDBMN) TYPE(*DETAIL) COMMENT('Collecting Data for Query/400  
- to be Imported in SQL Monitor')
```

See Figure 7-69 for a sample of this command. All traditional 5250 commands and activities documented here have been performed in the *same working session*. Had it been otherwise, the STRDBMON command should point to the correct job (STRDBMON JOB(nnnnnn/USER/JOBNAME)).



```

                                Start Database Monitor (STRDBMON)

Type choices, press Enter.

File to receive output . . . . . > QAQQDBMN      Name
  Library . . . . . > ITSCID41      Name, *LIBL, *CURLIB
Output member options:
  Member to receive output . . . *FIRST      Name, *FIRST
  Replace or add records . . . . *REPLACE  *REPLACE, *ADD
Job name . . . . . *                Name, *, *ALL
User . . . . .                        Name
Number . . . . .                        000000-999999
Type of records . . . . . > *DETAIL      *SUMMARY, *DETAIL
Force record write . . . . . *CALC      0-32767, *CALC
Comment . . . . . > Collecting Data for Query?400 -
                                to be Imported in SQL Monitor

                                                                    Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys

```

Figure 7-69 Start Database Monitor

- Now that we are recording all the database activities for our current job, we can run the previously prepared Query/400 definition using Query Management Query, as shown in Figure 7-70.

```

                                Start Query Management Query (STRQMQR)

Type choices, press Enter.

Query management query . . . . . > TESTQRY      Name
  Library . . . . . > ITSCID41      Name, *LIBL, *CURLIB
Output . . . . . *                *, *PRINT, *OUTFILE
Query management report form . . *SYSDFD      Name, *SYSDFD, *QMQR
  Library . . . . .                        Name, *LIBL, *CURLIB

                                Additional Parameters

Relational database . . . . . *NONE
Connection Method . . . . . *DUW          *DUW, *RUW
User . . . . . *CURRENT                Name, *CURRENT
Password . . . . .                        Character value, *NONE
Naming convention . . . . . *SYS          *SYS, *SAA
Allow information from QRYDFN . > *YES      *NO, *YES, *ONLY

                                                                    More...
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys

```

Figure 7-70 Start Query Management Query

Use the STRQMQRCL command with the ALLQRYDFN parameter set to \*YES to enable it to exploit the pre-existing query definition:

```
STRQMQRCL QMQRCL(ITSCID41/TESTQRY) ALWQRYDFN(*YES)
```

3. After the query has finished executing, you notice message QWM2704 posted into your job log, as shown in Figure 7-71.

This is an informational message documenting that no query management object was found. However, a query definition having the same name existed and it was used, as allowed by the ALLQRYDFN(\*YES) parameter.

```
Additional Message Information

Message ID . . . . . : QWM2704      Severity . . . . . : 00
Message type . . . . . : Diagnostic
Date sent . . . . . : 10/18/00      Time sent . . . . . : 16:14:21

Message . . . . . : STRQMQRCL command completed using derived information.
Cause . . . . . : Information was derived from a Query/400 query definition
                  and used instead of at least one query management object that could not be
                  found. The Query/400 query definition was found using the names specified
                  for locating the query management object.
Recovery . . . . . : Refer to the job log for the name and object type specific
                  message for each instance in which information had to be derived.

                                                                    Bottom

Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details
F10=Display messages in job log   F12=Cancel   F21=Select assistance level
```

Figure 7-71 Message QWM2704

4. At this point, if you are finished collecting data, you can end the Database Monitor by using the CL command:

```
ENDDBMON
```

5. Now you can use the SQL Performance Monitor tool in Operations Navigator to import and analyze the data collected. To do so, select **Database->SQL Performance Monitor**. Right-click **SQL Performance Monitor** and select **Import**. You see the dialog shown in Figure 7-72.

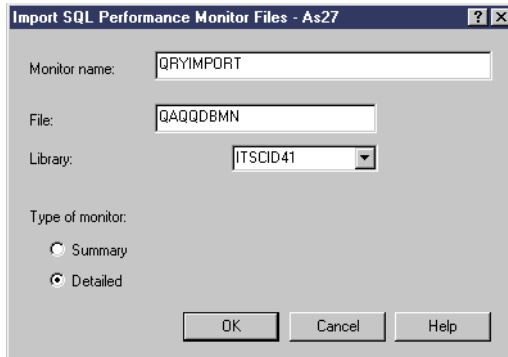


Figure 7-72 Importing data in SQL Performance Monitor

Here specify a name for the new Monitor you are creating and the file and library containing the previously collected data.

- The new monitor name is added to the list shown at the SQL Performance Monitor item. Right-click it and select **List Explainable Statements**, as shown in Figure 7-73.

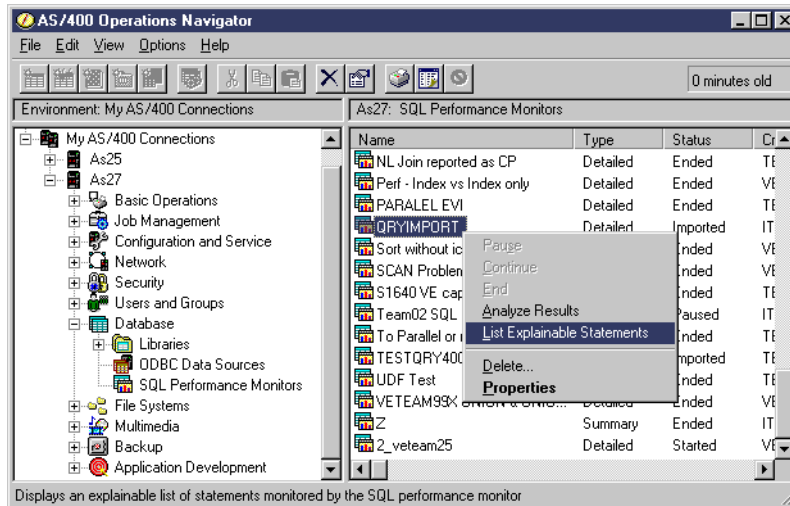


Figure 7-73 SQL Performance Monitor - List Explainable Statement

After you select this option, you are presented with the List Explainable Statements dialog (Figure 7-74). On this display, you can see the actual SQL statement generated by Query Manager to resolve the original Query/400 request.

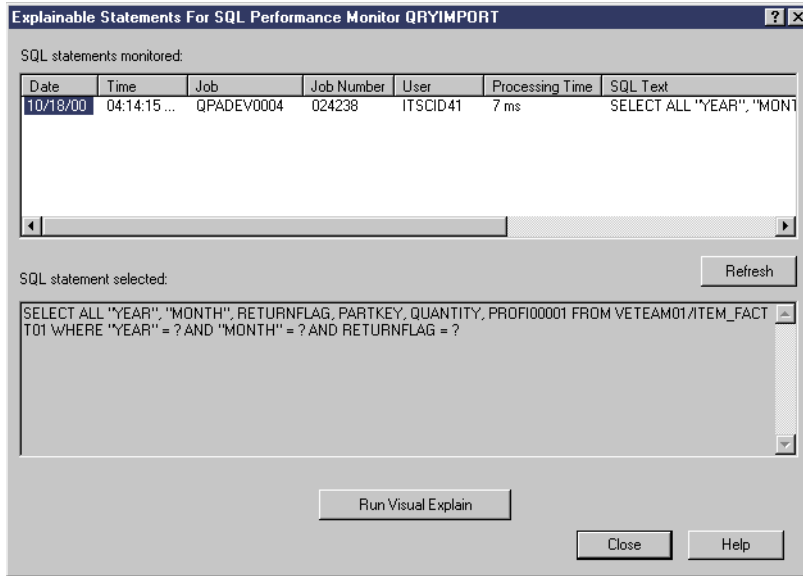


Figure 7-74 List Explainable Statements dialog

From here, you can link into Visual Explain to analyze the database behavior during the execution of this query. For more information on this subject, refer to Chapter 10, "Visual Explain" on page 301.



# Database Navigator

This chapter introduces you to the DB2 Universal Database for iSeries Database Navigator feature and its capabilities. It covers the following topics:

- ▶ Finding Database Navigator
- ▶ Finding database relationships prior to V5R1M0
- ▶ Database Navigator maps
- ▶ Database Navigator map interface
- ▶ Objects to Display window
- ▶ Database Navigator map display
- ▶ Available options on each active icon on a map
- ▶ Creating a Database Navigator map
- ▶ Adding new objects to a map
- ▶ Changing the objects to include in a map
- ▶ Changing object placement and arranging object in a map
- ▶ Creating a user-defined relationship

## 8.1 Introduction

The launch of DB2 Universal Database for iSeries Database Navigator, which is part of Operations Navigator in V5R1M0 Client Access Express, allows database administrators to map the complex relationships between database objects.

The database component of Operations Navigator at V5R1M0 provides additional graphical interfaces for new functions that include:

- ▶ The ability to create and manage tables, views, indexes, constraints, journals, journal receivers, and system (external) and SQL Triggers
- ▶ The ability to graphically view the relationships between the various parts of an existing DB2 UDB database and save and update these visual maps with the push of a button
- ▶ The ability to reverse engineer an existing database so that the database administrator can port the database to other iSeries servers as well as other platforms

The relationships that you see on the Database Navigator map are the relationships between:

- ▶ Tables (for example, referential integrity constraints)
- ▶ Any indexes over the tables
- ▶ Any constraints, such as primary, foreign, unique, and check
- ▶ Any views over the tables
- ▶ Any aliases for the tables, etc.

**Note:** Database Navigator is not intended to be a data modeling tool like some existing products in the industry.

### 8.1.1 System requirements and planning

Be sure you have an iSeries server with OS/400 V5R1M0, or higher, with:

- ▶ **5722-SS1:** Option 12 - Host Servers
- ▶ **5722-TC1:** TCP/IP Connectivity Utilities
- ▶ **5722-XE1:** Client Access Express V5R1M0

## 8.2 Finding Database Navigator

Database Navigator resides under the Database icon of Operations Navigator. Open the Operations Navigator window and click the desired iSeries server. Click the (+) sign next to the Database icon. Now, click the **Database Navigator** icon. The Database Navigator maps that are available appear.

Database Navigator is part of the Database icon within Operations Navigator. There are three functions beneath the Database icon (Figure 8-1):

- ▶ Libraries
- ▶ Database Navigator
- ▶ SQL Performance Monitors

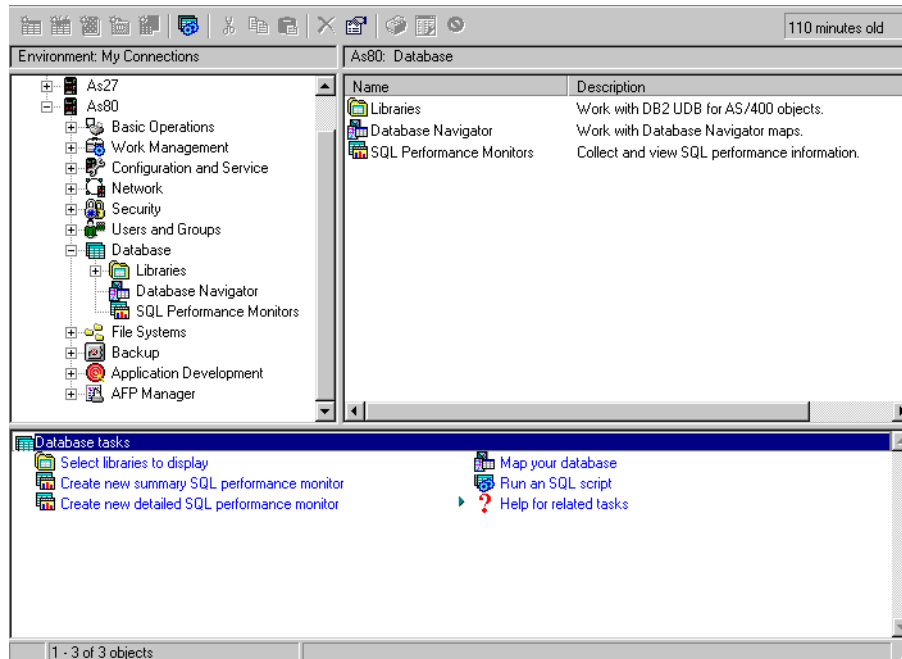


Figure 8-1 Database options within Operations Navigator

The new Database Navigator feature allows you to perform many tasks, including:

- ▶ Create a table
- ▶ Create a view
- ▶ Create an alias
- ▶ Create a journal
- ▶ Create an index
- ▶ Create a map of your database
- ▶ Create new SQL objects to be displayed in the map
- ▶ View the properties of a map
- ▶ View the properties of an object within a map
- ▶ Generate SQL for an object within a map
- ▶ Generate SQL for all objects within a map
- ▶ Generate SQL for selected objects within a map
- ▶ Generate SQL for visible objects in a map
- ▶ Expand a table in a map
- ▶ Collapse a table in a map
- ▶ Add a referential constraint
- ▶ Add a check constraint
- ▶ Add user defined relationships to a map
- ▶ Add a key constraint

An option also exists for removing and adding objects to this relationship, such as journals and receivers. These are not selected as a default for the map because they may cause the map to be very complicated. To add these extra objects, click the **Options** menu as shown in Figure 8-2.

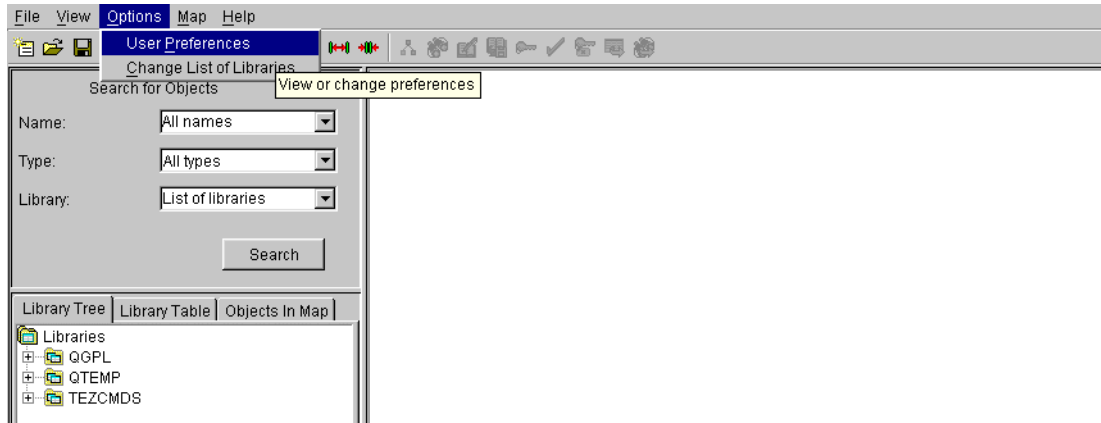


Figure 8-2 Viewing or changing the user preferences for the Database Navigator map

After you select the user preferences option, you can see the default user preferences for creating a Database Navigator map.

Figure 8-3 shows you the various objects that may be included on the map. From the Database Navigator map, you can directly affect the relationships on the database by adding or removing indexes, files, views, etc.

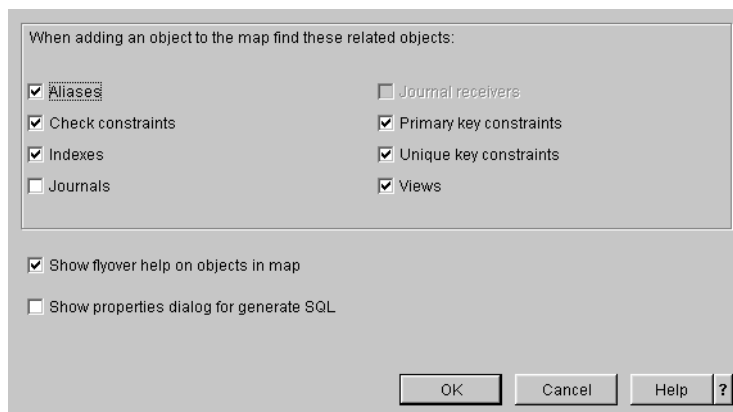


Figure 8-3 Database Navigator user preferences

## 8.3 Finding database relationships prior to V5R1M0

To see the benefits of Database Navigator, you must find the relationship between database objects on an iSeries server that is not at Client Access Express V5R1M0. You must use several commands to achieve the same results that you get with Database Navigator.

Some of the commands that are needed are:

- ▶ `DSPDBR FILE(SAMPLEDB16/ACT) OUTPUT(*PRINT)`

This command (Figure 8-4) shows the indexes, views, and constraints related to the selected table.

- ▶ `DSPFD FILE(SAMPLEDB16/ACT) TYPE(*CST) OUTPUT(*PRINT)`

This command (Figure 8-5) shows the details of the constraints built over the selected table.



- ▶ DSPFD FILE(SAMPLEDB16/ACT) TYPE(\*ACCPH) OUTPUT(\*PRINT)

This command (Figure 8-6) shows you the access path that is built over the selected table.

You also need to use the WRKJRNA command to determine which files are journaled to other journals. Although the DSPFD command also shows this, you cannot obtain an overview without using these commands.

It is possible to build a relationship map. However, it takes time and a great deal of effort. It is also very difficult for a new database administrator to envision the layout of an existing or new database and the effect of removing an index or constraint on other files.

This can result in unnecessary resources allocated to files, indexes, and constraints that may not be needed. This is because the referential integrity map is only as good as the last time the database administrator actually checked the authenticity of the map that was previously created.

The AFP viewer screens show the commands mentioned in the previous list (Figure 8-4 through Figure 8-6).

```

57168S1 V5R1M0 010525          Display Data Base Relations

DSPDBR Command Input
File . . . . . : FILE          ACT
Library . . . . . :             SAMPLEDB16
Member . . . . . :             *NONE
Record format . . . . . : RCDfmt *NONE
Output . . . . . :             *PRINT

Specifications
Type of file . . . . . :             Physical
File . . . . . :             ACT
Library . . . . . :             SAMPLEDB16
Member . . . . . :             *NONE
Record format . . . . . :             *NONE
Number of dependent files . . . . . :             6

Files Dependent On Specified File
Dependent File      Library      Dependency  JREF      Constraint
XACT1               SAMPLEDB16  Data
XACT2               SAMPLEDB16  Data
VACT                SAMPLEDB16  Data
VSTAFAC1           SAMPLEDB16  Data
VSTAFAC2           SAMPLEDB16  Data
PROJACT            SAMPLEDB16  Constraint  RPAA
  
```

Figure 8-4 Display Database Relations display

```

57168S1 V5R1M0 010525          Display File Description

File . . . . . : ACT
Library . . . . . : SAMPLEDB16
Type of information . . . . . : *CST
File attributes . . . . . : *ALL
System . . . . . : *LCL
Processor . . . . . : IBM AS/400 Display File Description Processor

File . : ACT      Library . : SAMPLEDB16  Type of file . : Physical *DATA

Constraint Description

Primary Key Constraint
Constraint . . . . . : CST          QSYS ACT 00001
Type . . . . . : TYPE          *PRIMARY
Key . . . . . : KEY          ACTNO
Number of fields in key . . . . . :             1
Key length . . . . . :             2
  
```

Figure 8-5 Display File Description display showing constraints on a particular file

```

5716SS1 V5R1M0 010525 Display File Description
File . . . . . : ACT
Library . . . . . : SAMPLEDB16
Type of information . . . . . : *ACCPH
File attributes . . . . . : *ALL
System . . . . . : *LCL
Processor . . . . . : IBM AS/400 Display File Description Processor
File . : ACT Library . : SAMPLEDB16 Type of file . : Physical *DATA

Access Path Description
Access path maintenance . . . . . : MAINT *IMMED
Unique key values required . . . . . : UNIQUE Yes
Access path journaled . . . . . : No
Access path . . . . . : Keyed
Constraint Type . . . . . : PRIMARY
Number of key fields . . . . . : 1
Record format . . . . . : ACT
Key field . . . . . : ACTNO
Sequence . . . . . : Ascending
Sign specified . . . . . : SIGNED
Zone/digit specified . . . . . : *NONE
Alternative collating sequence . . . . . : No

```

Figure 8-6 Display File Description display showing access paths

The entire process for creating a physical or mental picture of which table is related to which index is very difficult to administer. The practical difficulties of keeping this picture up-to-date requires time and effort on the part of the database administrator.

It is also difficult to explain the entire picture when doing training for new staff, and it requires valuable time and effort on the part of the database administrator.

The process is simplified with the new Database Navigator feature of Client Access Express V5R1M0.

## 8.4 Database Navigator maps

Database Navigator enables you to visually depict the relationships of database objects on your iSeries server. The visual depiction you create for your database is called a *Database Navigator map*. In essence, the Database Navigator map is a snapshot of your database and the relationships that exist between all of the objects in the map.

Click the **Database Navigator** icon to bring up the list of Database Navigator maps that are available in the system. The maps appear in the right-hand side of the Operations Navigator window as shown in Figure 8-7.

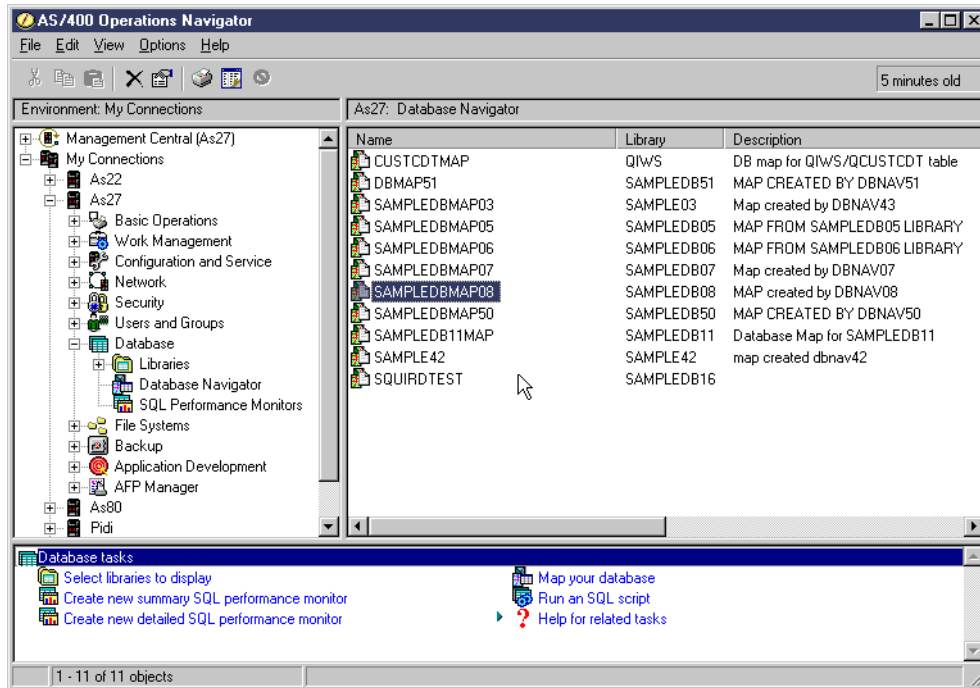


Figure 8-7 Database Navigator maps

Double-click the database map you want to view. The Database Navigator map window with the selected map appears as shown in Figure 8-8.

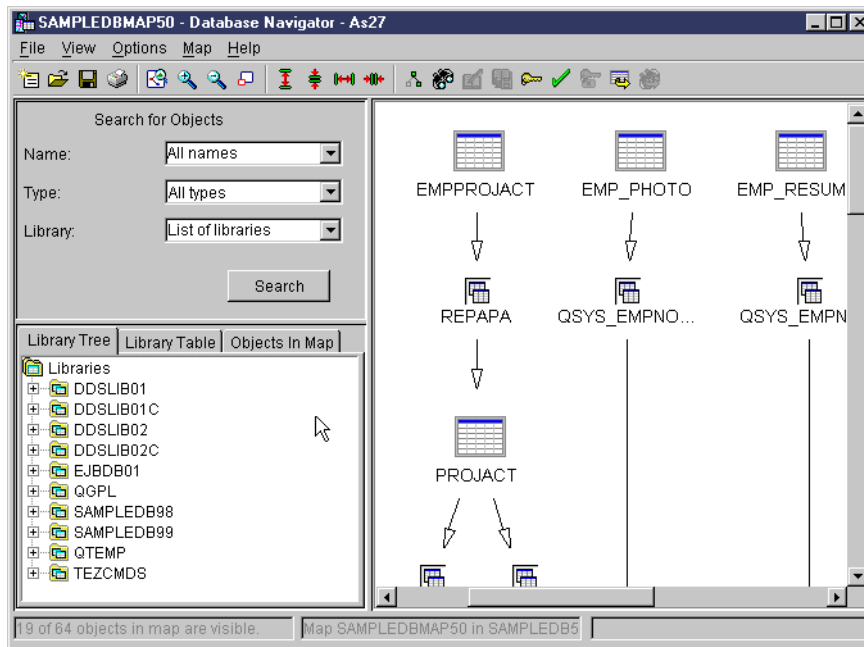


Figure 8-8 Database Navigator map

Be aware that the map shown is the Database Navigator map at the time that it was saved. This means that things may have changed on the system since the map was created and saved. To view an up-to-date picture of the database, refresh the map by clicking the **View** menu and selecting the **Refresh** option as shown in Figure 8-9.

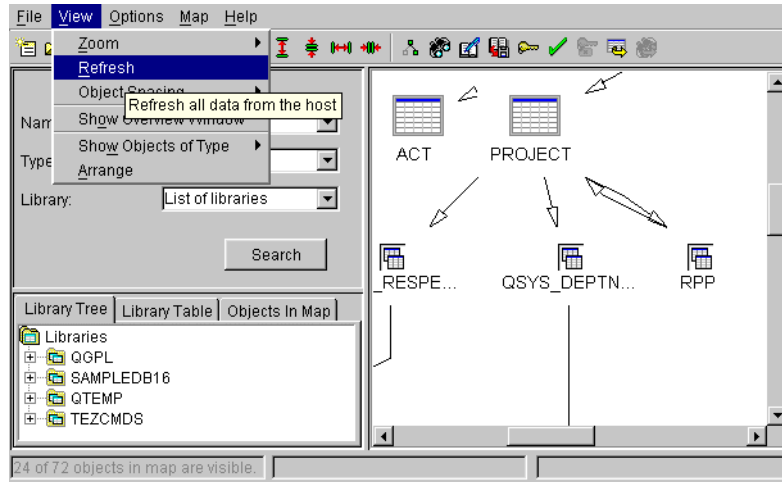


Figure 8-9 Refreshing a map

The Database Navigator maps are stored on the iSeries server. Only one person at a time can work on the map to ensure integrity. The maps are locked when they are being used. Because they are stored on the iSeries server, you must have a connection to the system to be able to open a map.

You can save multiple maps of the same database reflecting the database design at certain points in time. For this, you use different names. Whenever you want to compare how the database design has changed, you open and print the appropriate maps. Using the printouts, you can compare how the database design has changed over time.

**Note:** For a complete explanation of the icons, refer to 8.8, “The Database Navigator map icons” on page 269.

## 8.5 The Database Navigator map interface

As stated previously, the Database Navigator map provides a graphical interface that allows the database administrator to see the layout of the various objects in the database. One of the new functions added for V5R1M0 is the *task pad*. This is located in the lower part of the Operations Navigator window.

If you click on the various higher level options, such as Security, Users and Groups, Database, etc., the task pad changes accordingly to present you with the options that are available when the database functions is selected as shown in Figure 8-10.

The options available on the Database task pad are:

- ▶ Select libraries to display
- ▶ Create new summary SQL performance monitor
- ▶ Create new detailed SQL performance monitor
- ▶ Map your database
- ▶ Run an SQL script

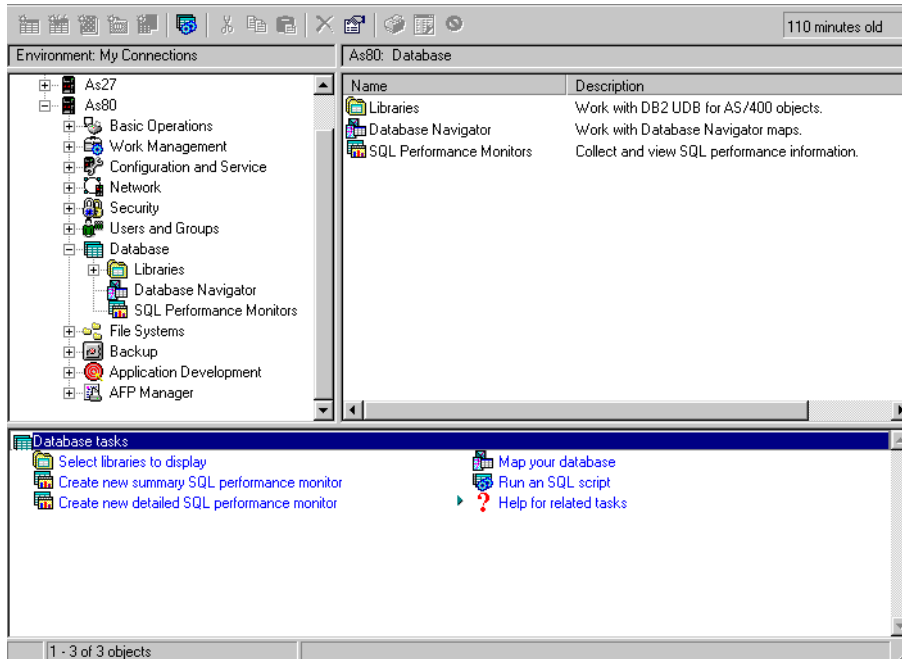


Figure 8-10 Database task pad options

Click the option in the task pad to create a map of your database. The window shown in Figure 8-11 appears.

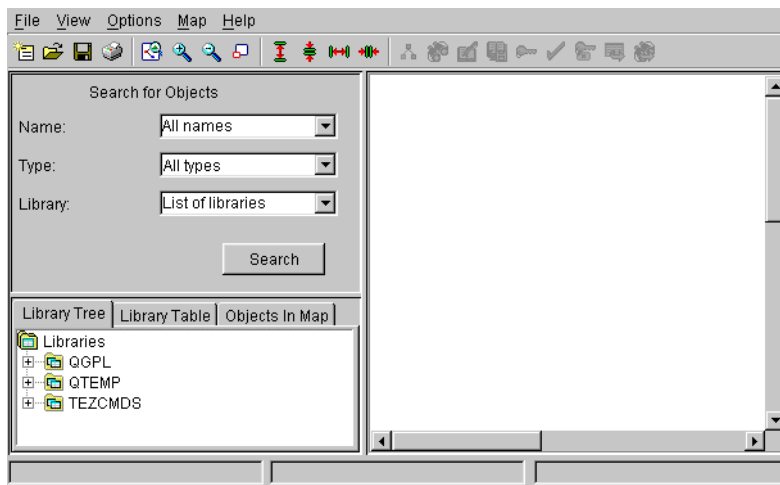


Figure 8-11 Default map display

The primary workspace for Database Navigator is a window that is divided into several main areas as shown in Figure 8-12. These areas allow you to find the objects to include in a map, show and hide items in a map, view the map, and check the status of changes pending for a map. The following list describes the main areas of the Database Navigator window:

► **Locator pane**

The locator pane, on the left side of the Database Navigator window, is used to find the objects that you want to include in your new map, or to locate objects that are part of an open map. The upper Locator Pane is a search facility that can be used to specify the Name, Type, and Library of the objects that you want to include in the map. The results of the search are displayed in the lower Locator Pane under the Library Tree and Library

Table tabs. When the results are displayed under these tabs, you can add objects to the map by right-clicking an object and selecting **Add to Map** or double-clicking the object name. Then, when the map is created, you can see a list of the objects in the map by clicking the **Objects In Map** tab.

The Locator Pane is divided in two parts:

– *The upper locator window*

This window allows you to search for database objects on the iSeries server. When an object is found, it is placed in the object window:

- On the search criteria, you can specify single objects or search for generic names using the \* (for example, EMPLE\*).
- You can specify all object types or indexes, tables, and views.
- You can specify one library from your library list or all libraries to search on.

– *The lower locator*

This window has three parts:

- *The library tree:* This can either show individual libraries, libraries in your list, or all libraries on the system.
- *The library table:* This shows the tables, indexes, or views of the libraries in the library tree.
- *Objects in map:* This shows all of the objects in the map, whether they are hidden or not. Within this display, you can select or deselect objects to be placed in the map.

**Note:** Any changes that are made using the search criteria require that you click the **Search** button to change the library tree or the library table displays.

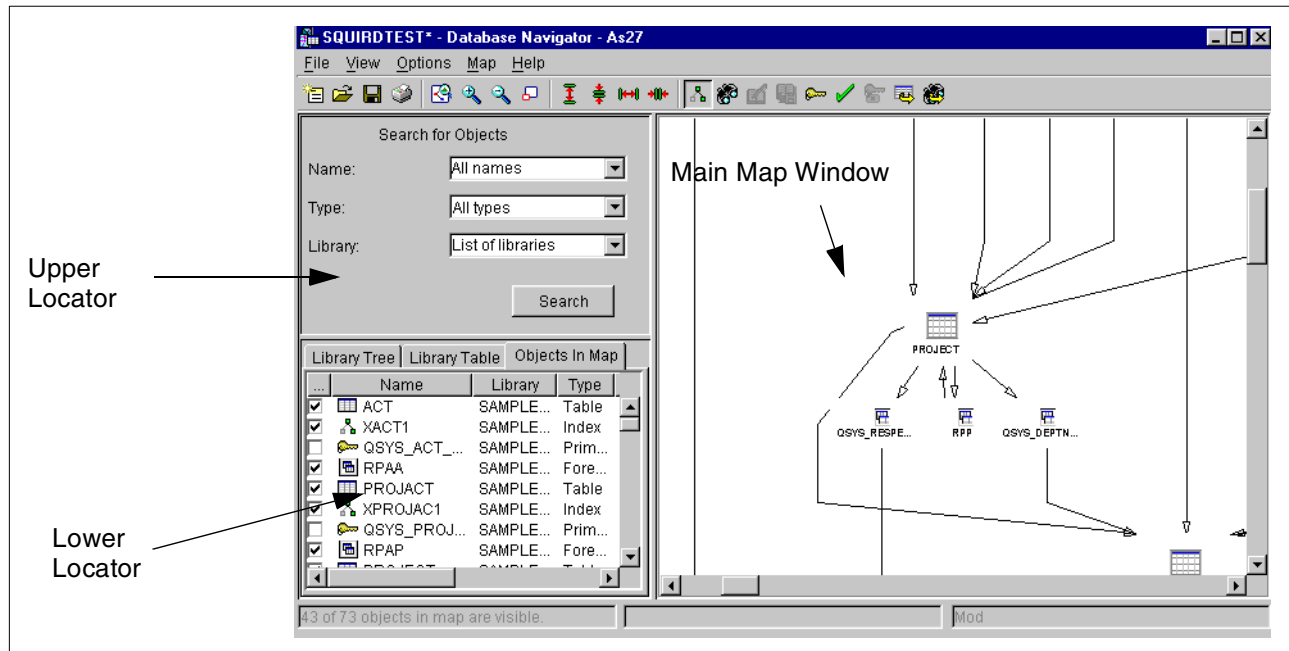


Figure 8-12 Three main windows in the Navigator map display

► **Map pane**

The map pane, on the right side of the Database Navigator window, graphically displays the database objects and their relationships. In the Map pane, you can:

- Add tables and views that exist on the system, but were not originally included in the current instance of the map
- Remove objects from the map
- Change object placement
- Zoom in or out on an object
- Make changes to objects in the map
- Generate the SQL for all objects in the map

These windows are the main interface that allow you to change what you see in the main map window, search for other objects to add to the map, and move the objects around within the map to make it easy to read.

► **Object status bar:** This part of the window consists of three parts (Figure 8-13):

- **Object Status Bar:** This displays the number of objects that are visible in the Database Navigator map and how many are eligible to be added to the map.
- **Action Status Bar:** This provides a clear description of the actions taken that affect the map and any modifications that are pending.
- **Modification Status Bar:** This indicates whether a modification has been made or is pending.

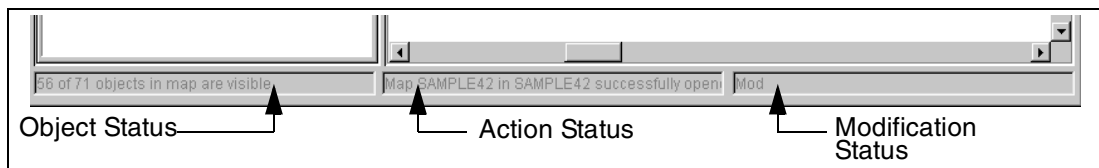


Figure 8-13 The status bar

The Database Navigator map display also supports the following menu options:

► **File menu:** You can select a number of options, including:

- **New:** Allows you to create a new map
- **Open:** Allows you to open a previously saved map
- **Close:** This closes the currently open map
- **Save:** Allows you to save the current map with which you are working
- **Save as:** Allows you to save the current map you are working with and change the name and location if the map has previously been saved
- **Print:** This option allows you to print to a previously defined printer
- **Exit:** This option closes the map of your database screen

**Note:** If changes are made to the map, or if this is a new map, you are prompted as to whether you want to save the map.

- ▶ **View menu:** The following options are available:
  - **Zoom**
    - *In:* Allows you to zoom in on the map
    - *Out:* Allows you to zoom out on the map
    - *Fit to Window:* Allows you to fit the map to the current window size
    - *To Selected Objects:* Positions the window to the object that has been selected
  - **Refresh:** This updates the database map with any changes that are made.
  - **Object Spacing:** This allows you to increase or decrease the vertical and horizontal spacings of the objects in the map.
  - **Show Overview Window:** This brings up a window (Figure 8-14) that allows you to see an overview of the map currently open. This overview allows you to position the main screen to any part of the map. This is particularly useful on very large or complicated maps.
  - **Show objects of type:** This allows you to add objects to the map, such as aliases, journals, etc.
  - **Arrange:** This allows you to change the map back to the original settings.

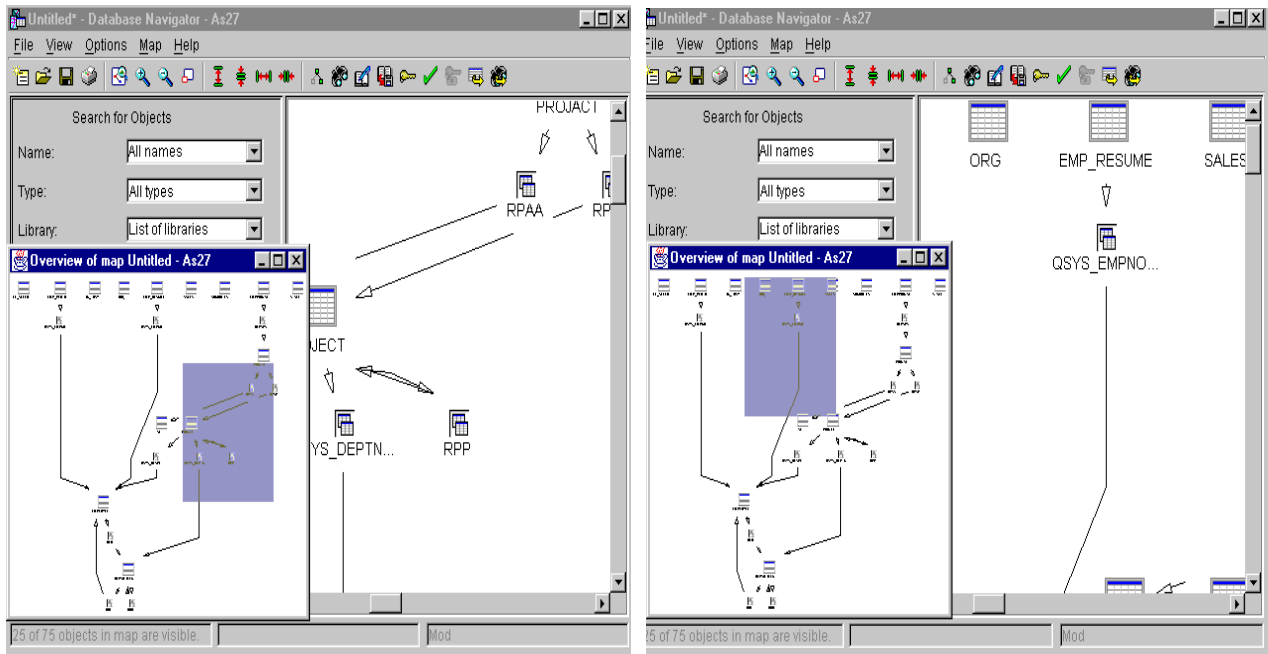


Figure 8-14 Overview windows showing how moving the overview box changes the main map display window

- ▶ **Options menu:** The following options are available:
  - **User Preferences:** This allows you to change the objects that appear on the map as it is created (Figure 8-15).



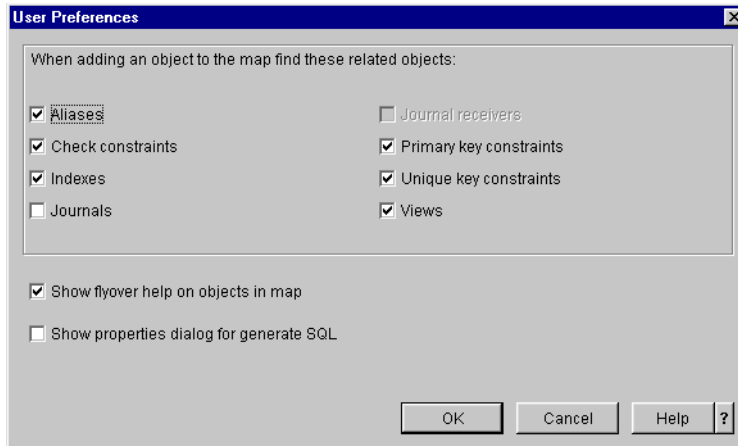


Figure 8-15 User Preferences display

- **Change List of Libraries:** This allows you to change the libraries that are displayed (Figure 8-16).

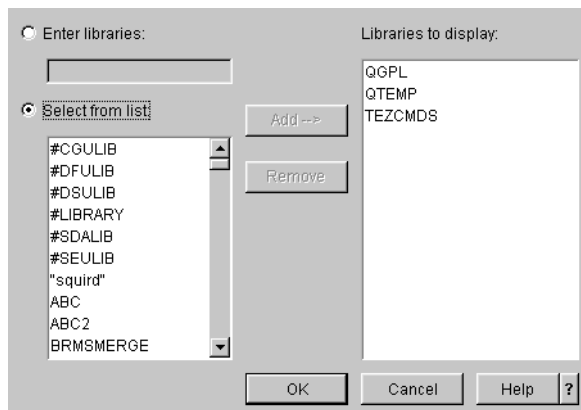


Figure 8-16 Change list of libraries display

If a new library is typed on the Enter Libraries box, instead of selecting from a list, the system ensures that the library exists before allowing the object to be added to your list.

- ▶ **Map menu:** The Generate SQL option appears with the following sub-options:
  - For all objects
  - Selected objects
  - Visible objects

For each of these options, the system creates the SQL script used to generate the objects, and it prompts the Run SQL Scripts window to appear as shown in Figure 8-17.

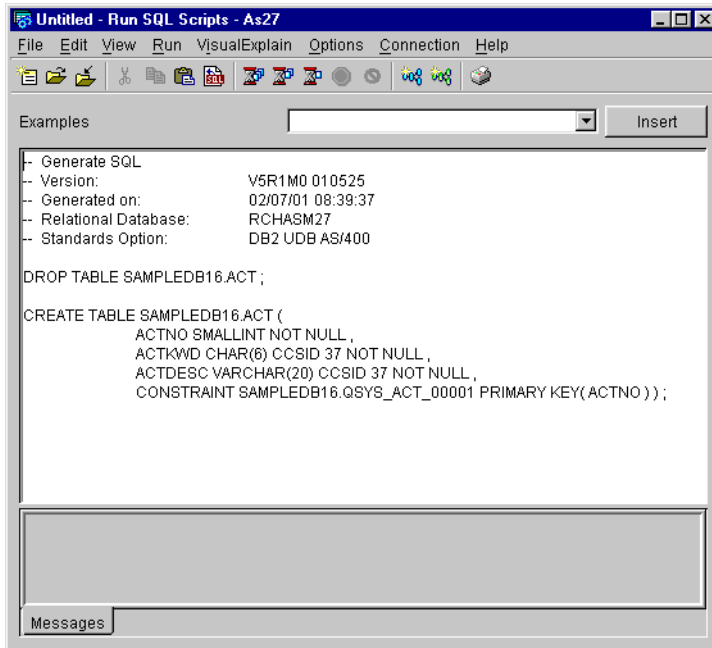


Figure 8-17 Generate SQL window

The Run SQL Scripts window allows you to see the SQL statement that was used to create the object and it allows you to take individual tables, whole databases, or entire maps of objects to other iSeries servers or SQL platforms.

The *Create* option includes the following sub-options:

- ▶ **Journal:** This allows you to create a new journal.
- ▶ **Table:** This allows you to create a new table.
- ▶ **View:** This allows you to create a new view.
- ▶ **User Defined Relationship:** This allows you to create a user-defined relationship. This helps by allowing the database administrator to add relationships of important tables, of the database, and so on. This function is likely used to illustrate a referential integrity constraint that is implemented on the application logic and is not defined in the database. This can also be used to illustrate relationships that are not physical in the map for debug or education purposes.

A tool bar exists that has a lot of the functionality previously mentioned. It includes the following features:

- ▶ Show or hide Indexes
- ▶ Show or hide views
- ▶ Show or hide journals
- ▶ Show or hide journal receivers
- ▶ Show or hide primary key constraints
- ▶ Show or hide check constraints
- ▶ Show or hide unique key constraints
- ▶ Show or hide table aliases
- ▶ Show or hide view aliases

**Note:** If the objects are not available to hide or view, the button is grayed out.

## 8.5.1 Objects to Display window

Within the Objects to Display window, only one option is available – the Find in Map option. This option allows you to find a specific object in the map. When this option is selected, the chosen object appears in the selected map window.

## 8.5.2 Database Navigator map display

The Database Navigator map main display is another interface for managing and changing your database using Operations Navigator. Each object on the Database Navigator map is active, and various options are available.

From the main display, you can add objects to a map, create new objects, etc., as previously described. This section explains the various functions available to you from this display.

Right-click the main window to view the following menus (Figure 8-18):

- ▶ **Create:** You can create journal, tables, views, and user-defined relationships by choosing this option.
- ▶ **Zoom:** You can zoom in or out and make the map fit the window by selecting this option.
- ▶ **Generate SQL:** You can generate the SQL for all objects or only the visible options by selecting this option.
- ▶ **Remove all line bends:** This removes all bends in the database map joining arrows.
- ▶ **Arrange:** Returns the objects within the map to their position at creation even if the map was saved previously.
- ▶ **Properties:** This shows you a properties display of the map itself.

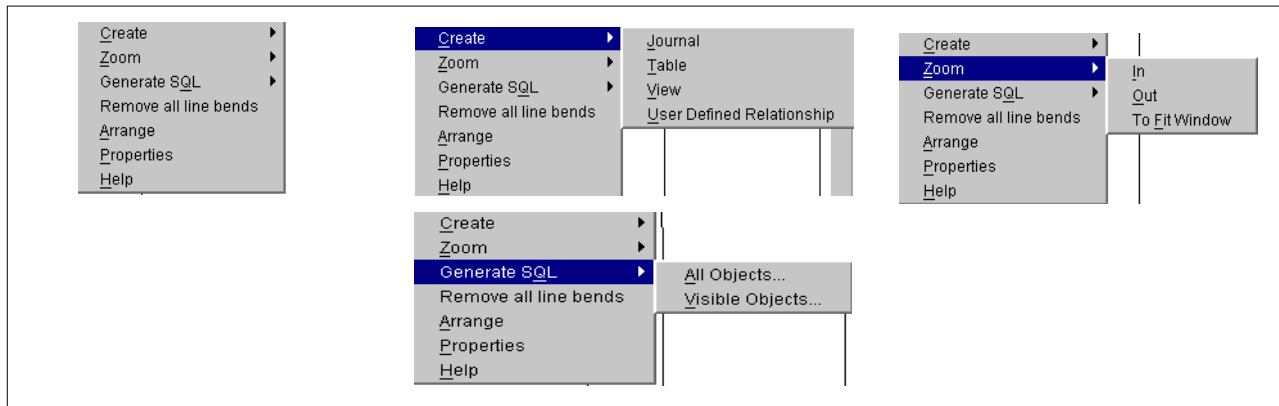


Figure 8-18 Options that are available when you right-click the map

**Note:** A map is saved on the iSeries server as an object type of \*FILE.

As previously stated, each object type is active. By right-clicking the objects, you can access several different options.

## Flyover

Because each object is active, there is a new function that allows you to view a brief description of an object within the map simply by placing the cursor over the object. This is called a *flyover*. Depending on the type of object, different information types appear. The basic display for each object shows the object name, the system name on which the object resides, the library, and the type of object as shown in Figure 8-19.

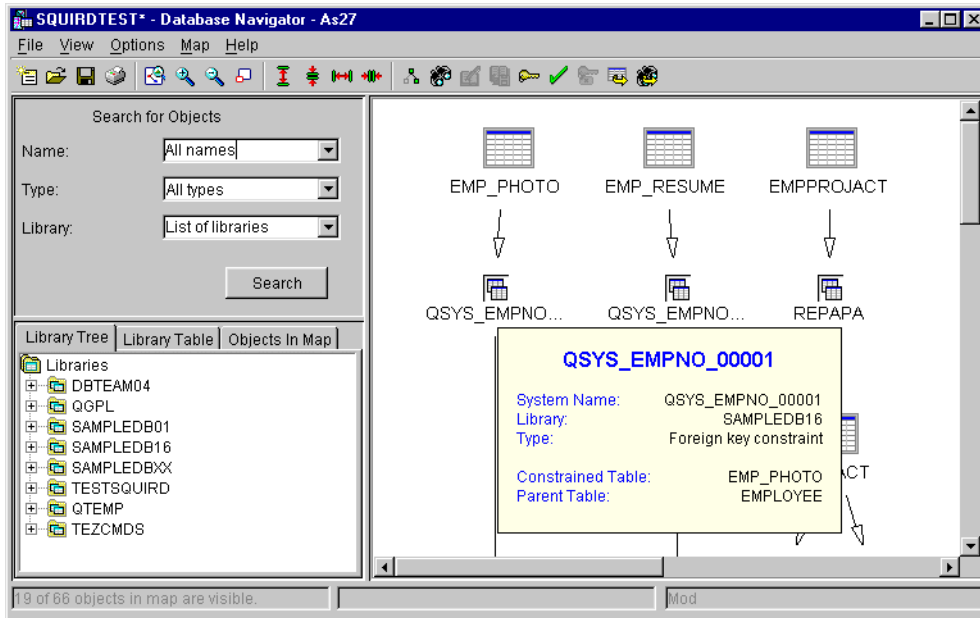


Figure 8-19 Example flyover display

After you refresh the display, a window, like the example in Figure 8-20, appears while the refresh runs.

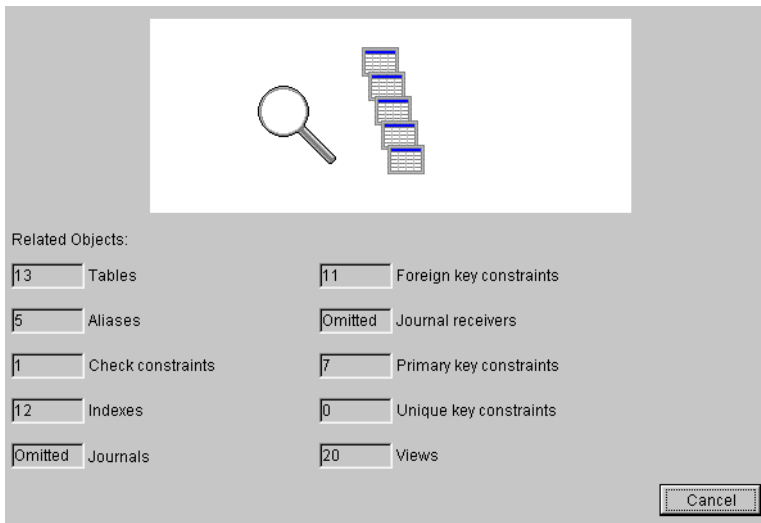


Figure 8-20 Refresh on database in progress

After the map is built or refreshed, you can manipulate the objects however you want. From within the map display, you can actually move the icons around to suit your requirements.

## 8.6 Available options on each active icon on a map

This section discusses the options that are available to you from within the map display. These options are available by right-clicking each of the different objects in the map.

### 8.6.1 Table options

Figure 8-21 shows the various options available to you when you are using the active icon for a table within a Database Navigator map. All objects on a map are active, and they enable you to manipulate the object without leaving the map.

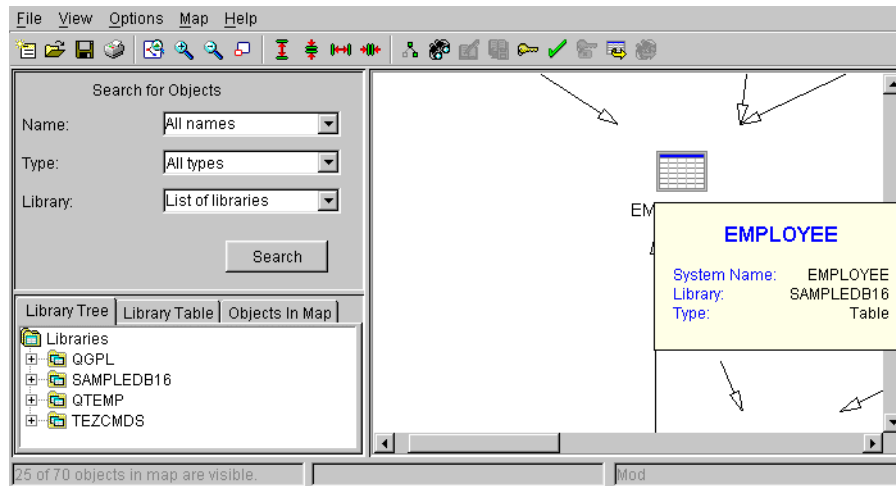


Figure 8-21 Flyover display of a table within a Database Navigator map

Right-click a table within the map. A window appears as shown in Figure 8-22.

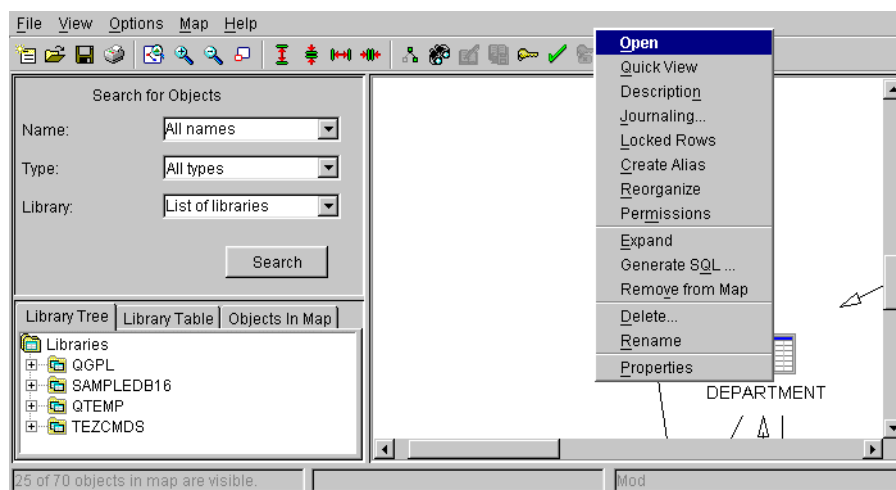


Figure 8-22 Right-clicking a table within a Database Navigator map

The options that appear include:

- ▶ **Open:** This allows you to open the file for update.
- ▶ **Quick view:** This shows you the file and its contents (read only).

- ▶ **Description:** From within this option, there are six tabs:
  - **General:** This shows the size of the object, the current number of rows, the number of deleted rows, and whether the table reuses deleted records.
  - **Allocation:** This shows the settings for the maximum number of rows, the initial number of rows, the increment of the number of rows, the maximum number of increments, and other options.
  - **Access Path:** This shows the current size of the access path, the maximum size, the maximum key length, whether the access path is valid or shared, whether it is journaled, what the maintenance and recovery of the access path is set to, and other options (Figure 8-23).

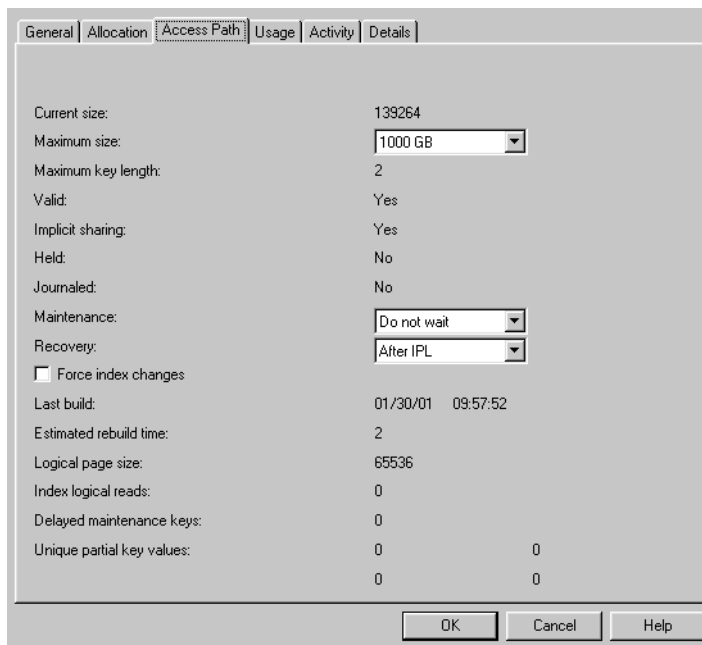


Figure 8-23 Access Path display

- **Usage:** This shows you the creation date of the table, the last used date, the last changed date, and other details of the table.
- **Activity:** This shows the latest activity on the table since the last machine restart.
- **Details:** This shows the creation date of the table, the maximum row length, and more.
- ▶ **Journaling:** This specifies whether journaling is on.
- ▶ **Locked Rows:** This shows any rows that are locked on the table.
- ▶ **Create Alias:** This allows you to create an alias for the table.
- ▶ **Reorganize:** This allows you to reorganize the file by compressing deleted records (by table key or by a selected index).
- ▶ **Permissions:** This allows you to set security for a table.
- ▶ **Expand (new function):** This shows additional details of the table, such as the columns and indexes built over the table.
- ▶ **Collapse (new function):** This returns the display to the default setting for the table.
- ▶ **Generate SQL (new function):** This creates an SQL script window (Figure 8-24) that allows you to recreate the table or multiple objects depending on the option selected from the Generate SQL screen. The Generate SQL function is new for V5R1M0 and is available

for individual objects or entire databases. This option is available through the Database Navigator map and through the library display within the database option in Operations Navigator. This option is discussed later in more detail.

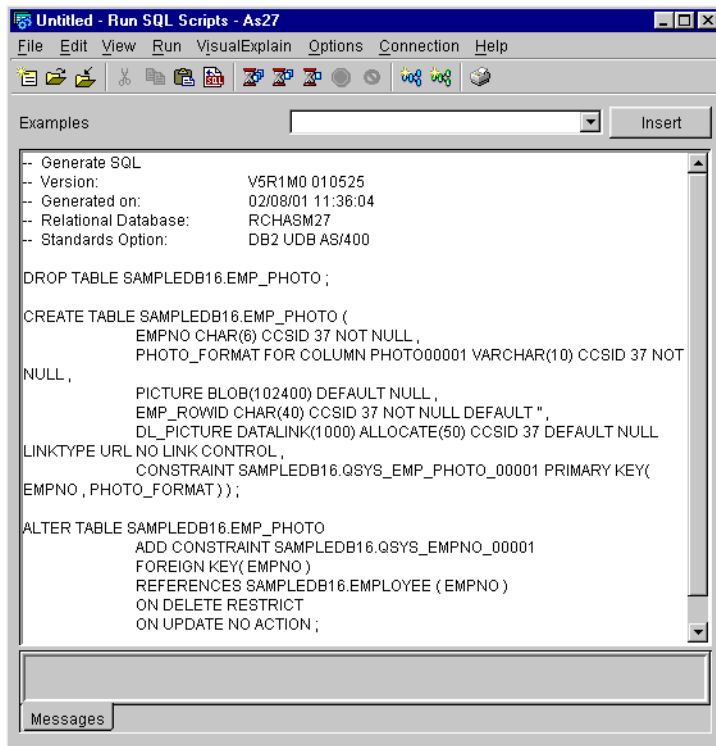


Figure 8-24 Generate SQL Script window

- ▶ **Remove From Map** (new function): This removes a particular view from the map. If the object is not included in the map, you see the Add to map function highlighted.
- ▶ **Delete**: This allows you to delete a particular table.
- ▶ **Rename**: This allows you to rename the table.
- ▶ **Properties**: This shows you a display of the properties of a table.

## 8.6.2 Index options

Right-click an index to access the options shown in Figure 8-25.

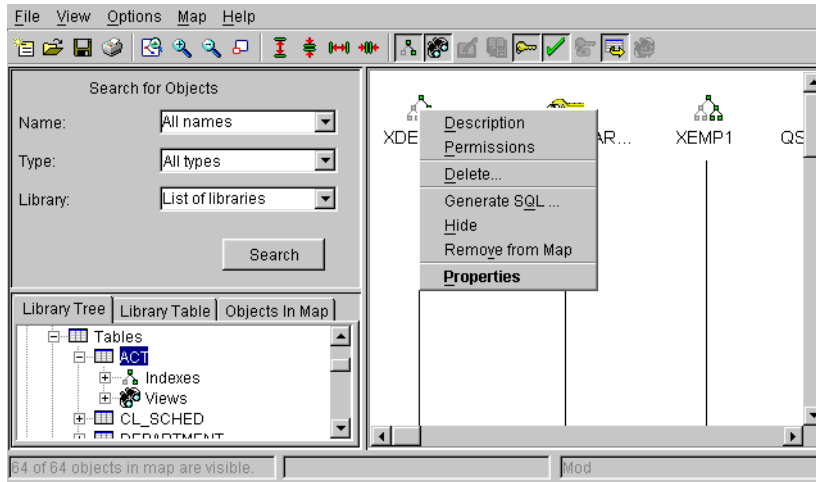


Figure 8-25 Right-clicking an index

The primary options are:

- ▶ **Description:** In this option, there are three views:
  - *Access path:* See Figure 8-23 for more details on this screen
  - *Usage:* This shows you the creation date of the index, the last used date, the last changed date, and other details of the index.
  - *Details:* This shows the creation date of the index, the maximum row length, and more.
- ▶ **Permissions:** This allows you to set security for the object.

### 8.6.3 Constraint options

If you right-click any of the constraints on the map, the following options appear:

- ▶ **Generate SQL** (new function): This creates an SQL script window (Figure 8-24).
- ▶ **Remove from map** (new function): This removes a particular constraint from the map. If the object is not included in the map, the Add to map function appears highlighted.
- ▶ **Properties:** This shows you the properties of the table over which the constraint is defined as shown in Figure 8-26.



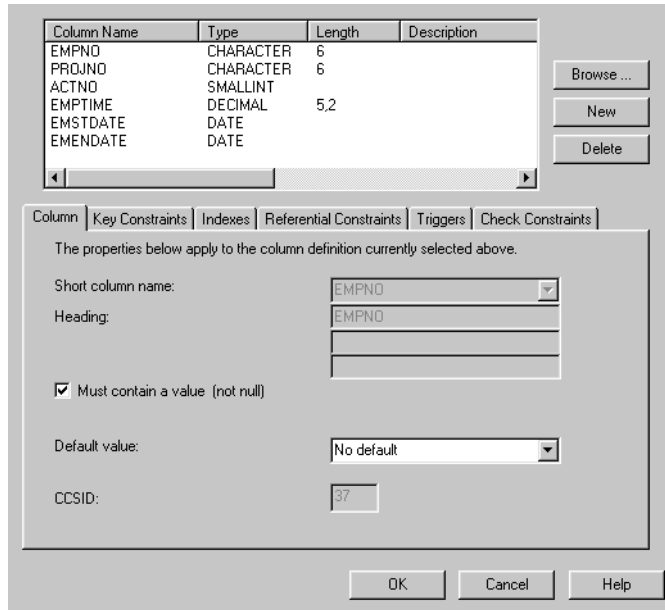


Figure 8-26 File properties window showing constraints

## 8.6.4 View options

Right-click any view on the map to see the following options:

- ▶ **Quick View:** This shows you the view contents (read only).
- ▶ **Description:**
  - *Usage:* This shows you the creation date of the view, the last used date, the last changed date, and other view details.
  - *Details:* This shows the creation date of the view, the maximum row length, and additional details.
- ▶ **Create Alias:** This allows you to create an alias for the view.
- ▶ **Permissions:** This allows you to set security for the view.
- ▶ **Generate SQL** (new function): This creates an SQL script window that allows you to recreate the table or multiple objects depending on the option selected from the generate SQL screen. The Generate SQL function is new for V5R1M0 and is available for individual objects or entire databases. This option is available through the Database Navigator map and also through the library display within the database option in Operations Navigator.
- ▶ **Remove** (new function): This removes a particular view from the map. If the object is not included in the map, the Add to map function appears highlighted.
- ▶ **Properties:** This shows you the properties of the view. If it is an SQL view, it shows the SQL statement used to create the view. If it is a logical file, a message appears stating that it was not created in SQL and, therefore, it cannot be shown.
- ▶ **Hide** (new function): This allows you to remove the view from the map only.

## 8.6.5 Journal options

If you right-click a journal, the options shown in Figure 8-27 appear.

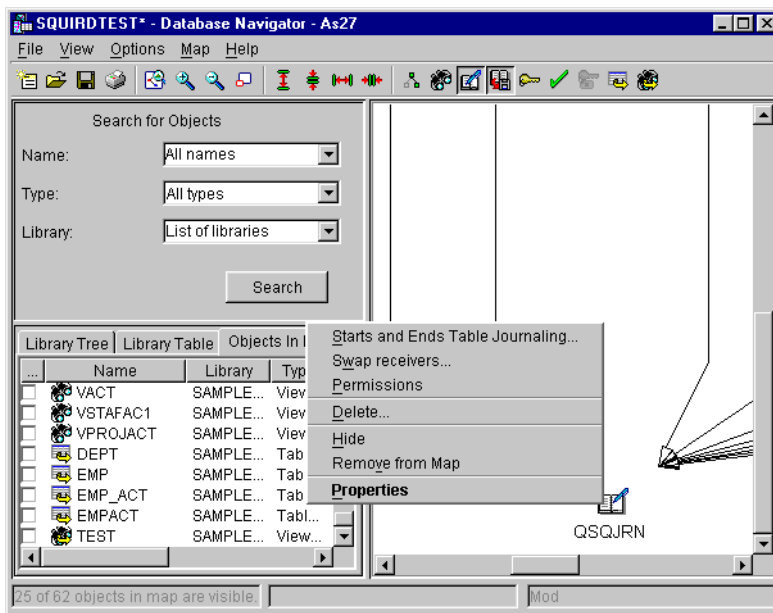


Figure 8-27 Right-clicking a journal

The various options include:

- ▶ **Start or End Table Journaling:** This allows you to end or start journaling on any table on the system to the selected journal.
- ▶ **Swap receivers:** This allows you to perform the equivalent of a CHGJRN \*GEN from a normal green-screen command.
- ▶ **Permissions:** This allows you to set security for the journal.
- ▶ **Delete:** This allows you to delete a particular journal.
- ▶ **Remove from map** (new function): This allows you to remove a particular journal from the map. If the object is not included in the map, the Add to map function appears highlighted.
- ▶ **Hide** (new function): This allows you to remove the journal from the map only.
- ▶ **Properties:** This shows you the properties of the journal.

## 8.6.6 Journal receiver options

The various Journal receiver options include:

- ▶ **Permissions:** This allows you to set security for the journal receiver.
- ▶ **Delete:** This allows you to delete a particular journal receiver.
- ▶ **Remove from map** (new function): This allows you to remove a particular journal receiver from the map. If the object is not included in the map, the Add to map function appears highlighted.
- ▶ **Hide** (new function): This allows you to remove the journal receiver from the map only.
- ▶ **Properties:** This shows you the properties of the journal receiver.

## 8.7 Creating a Database Navigator map

The visual depiction that you create of your database is called a Database Navigator map. To create a Database Navigator map, you need to follow these steps:

1. In the Operations Navigator window, expand your server **Database**.
2. Right-click **Database Navigator** and select **New** from the pull-down menu to create your Map as shown in Figure 8-28.

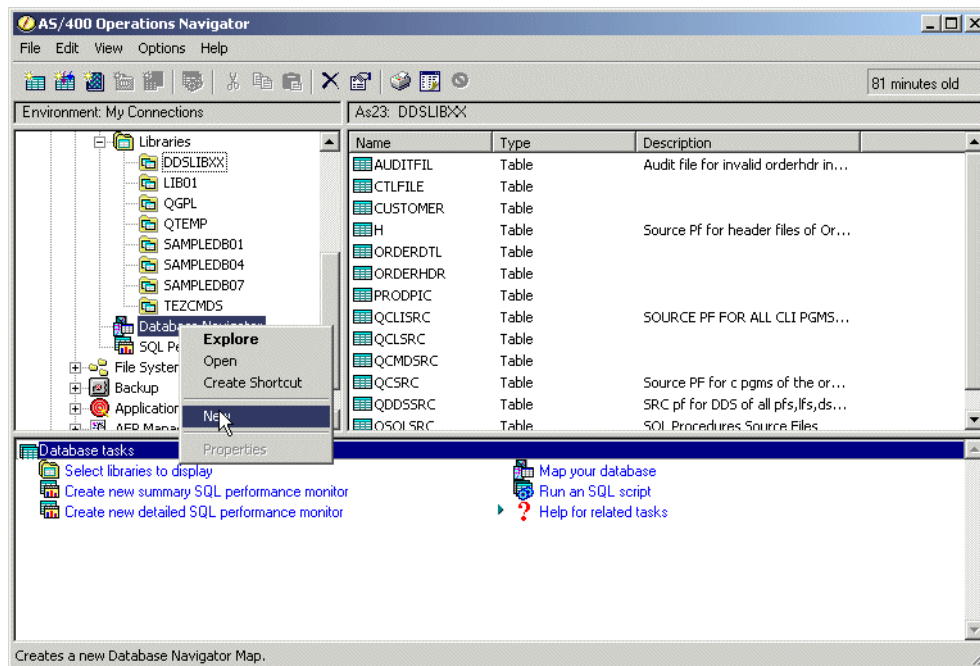


Figure 8-28 Database Navigator option

3. The Operations Navigator library list appears in the left side of the Database navigator window. Double-click the **SAMPLEDB04 library** to expand the objects.
4. Double-click **Tables** in the Locator Pane to expand all the tables in a database.
5. Double-click the **EMPLOYEE table** on the lower Locator Pane to start building a map. This table is added to the map and all related objects, as shown in Figure 8-29.

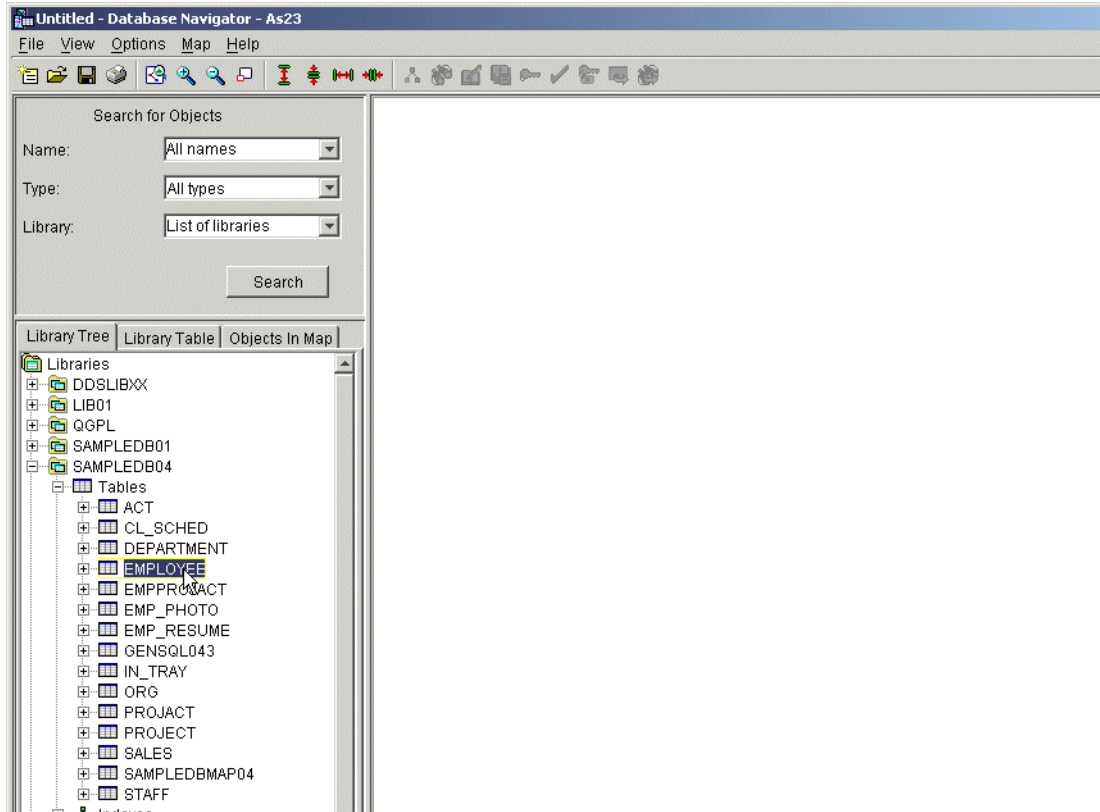


Figure 8-29 Selecting a database to build a map

- The map is built from the cross reference files (XREF) on the iSeries server. The relationship and statistics are based from the table that you selected to generate a map as show in Figure 8-30.

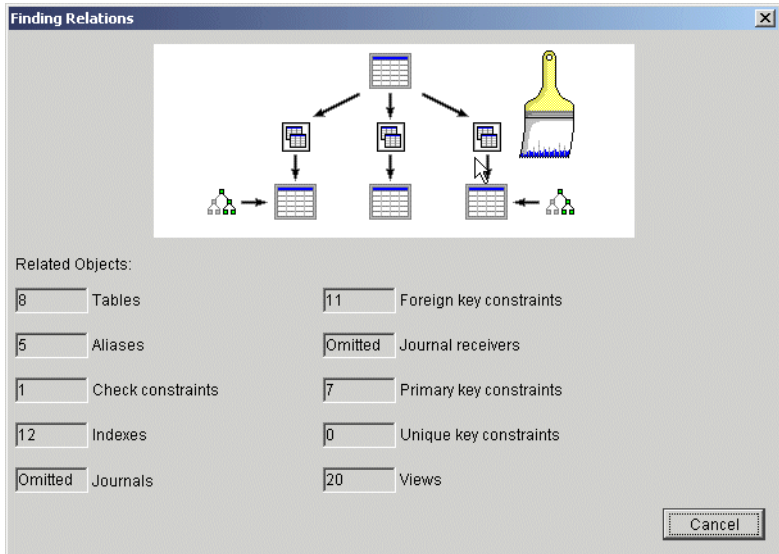


Figure 8-30 Building a Database Navigator map

- Click the minus (-) sign next to the **SAMPLEDB04** database object to collapse the tree view.

- Use the vertical and horizontal scroll bars to navigate the map on the Database Navigator window as shown in Figure 8-31.

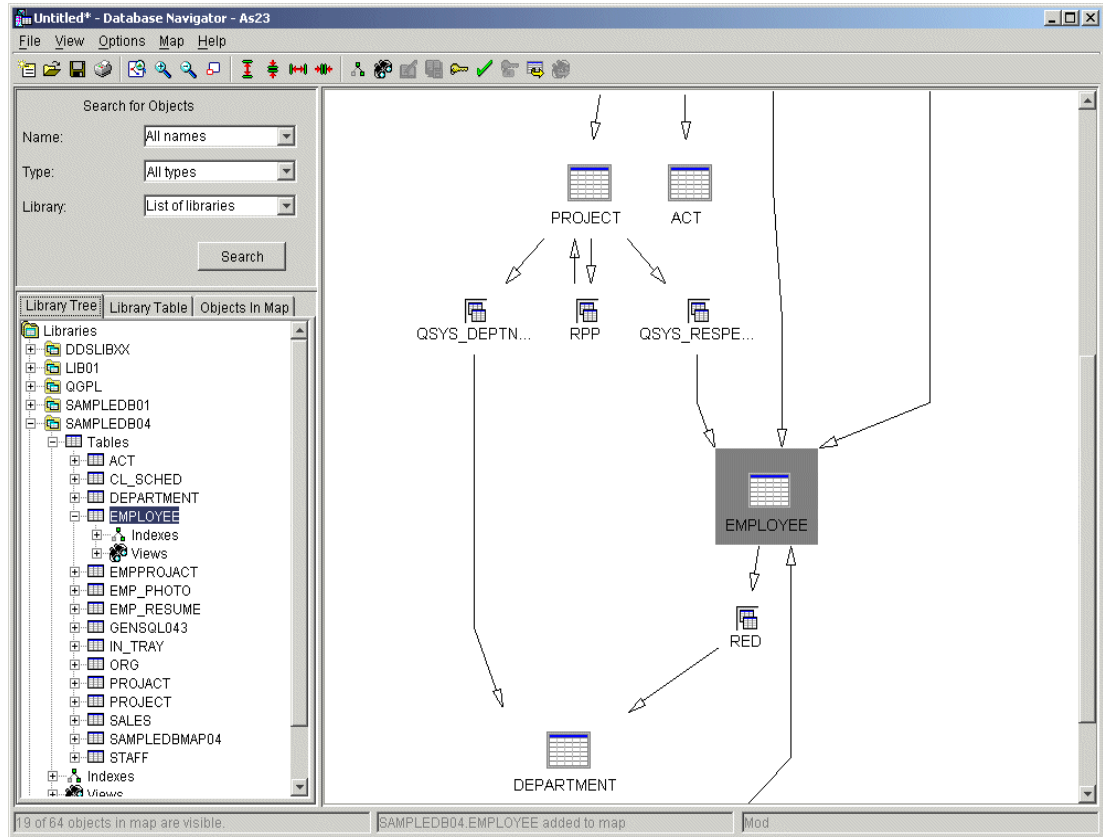


Figure 8-31 Database Navigator map

**Note:** You can also click the **Map your database** task on the task pad at the bottom of the Operations Navigator window to create a map.

- You can save this map by selecting **File-> Exit** from. Then, if changes are pending, select **Yes** on the Save Changes To dialog. This map can be reopened at a later time.

Once you create the map of your database, you can:

- ▶ Add new objects to a map
- ▶ Change the objects to include in a map
- ▶ Create a user-defined relationship

### 8.7.1 Adding new objects to a map

With Database Navigator, you can create new SQL objects to add to your map. Among the objects that can be created are:

- ▶ Tables
- ▶ Journals
- ▶ Views

To create new SQL objects to be displayed in a map, follow these steps:

1. Open a Database Navigator map.
2. Click the **View** menu. From the pull-down menu, select **Show Objects of Type -> Views** to include all Views in the map. The Object Status Bar that was updated with the new objects included in the map appears.

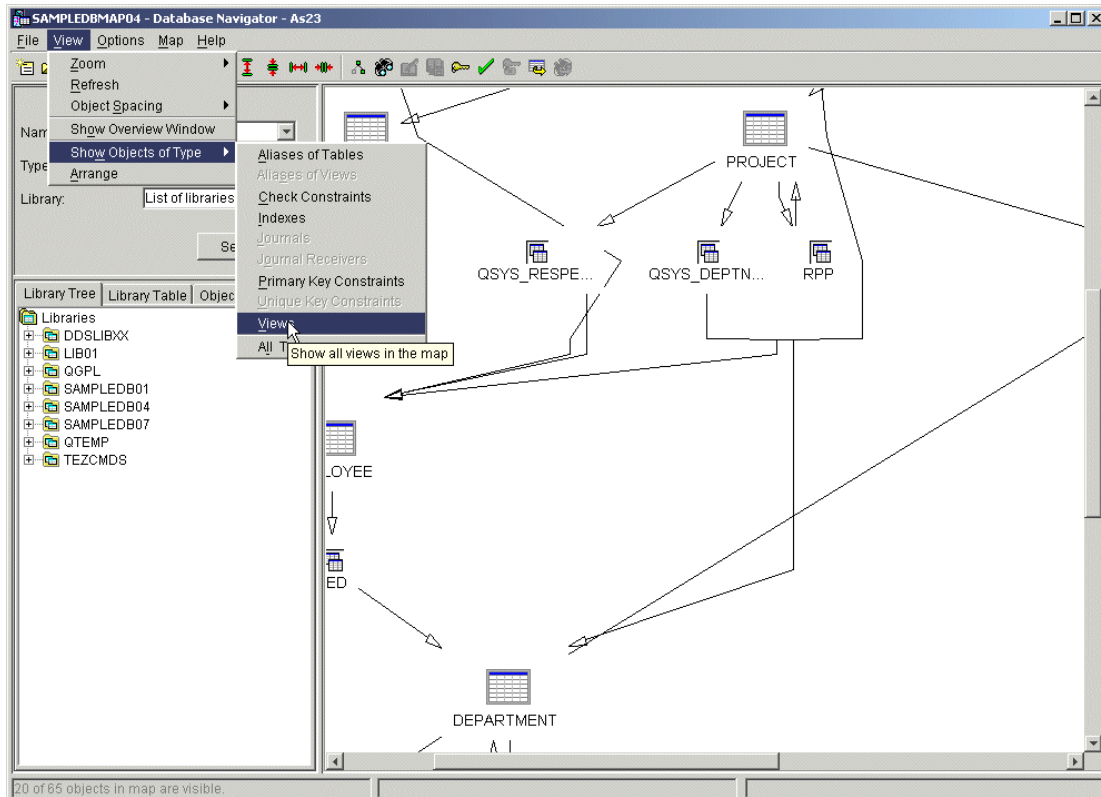


Figure 8-32 Adding Views objects in the map

3. Use the vertical and horizontal scroll bars to navigate to the top of the map.

**Important:** You can change the zoom level of the Database Navigator map to manage how much of the map you can see in the map pane on the Database Navigator Window.

## 8.7.2 Changing the objects to include in a map

By default, Database Navigator searches for and includes all objects in your map. To limit the number of objects that are searched for, you can change the user preferences.

To change which objects to include in the map, follow these steps:

1. Open a Database Navigator map.
2. From the **Options** menu, select **User Preferences**.
3. On the User Preferences dialog, in the *When adding an object to the map find these related objects* group box, select the objects you want to include, or deselect the objects you do not want to include.
4. Click **OK**.
5. If you want to refresh the map with the new preferences, click **Yes** in the Information box.

### 8.7.3 Changing object placement and arranging object in a map

When you have a map, it is possible to arrange and move objects in the map. You also learn how to remove the bends that appear on the relationship lines after the objects is moved to the new location.

1. Double-click the **EMPLOYEE** table from the list of tables to find this table in the map.
2. Drag-and-drop the **EMPLOYEE** table to the left as shown in Figure 8-33.
3. Right-click every relationship line and select **Remove Bends** to remove all bends.

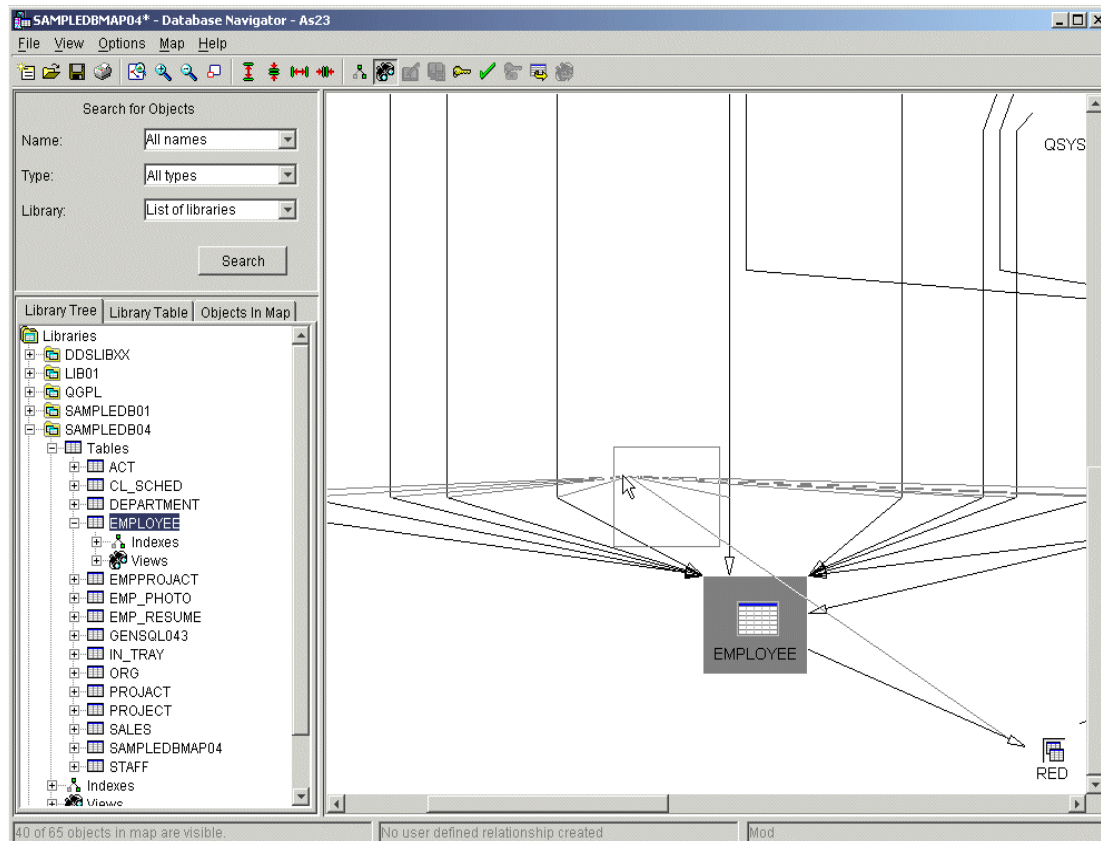


Figure 8-33 Changing object placement

**Important:** When you use the arrange option, it removes any customized object position or relationship line that you have created. The Arrange option puts the map back in a Default state.

4. Right-click in a free space in the map pane in the Database Navigator window. The Arrange function appears.

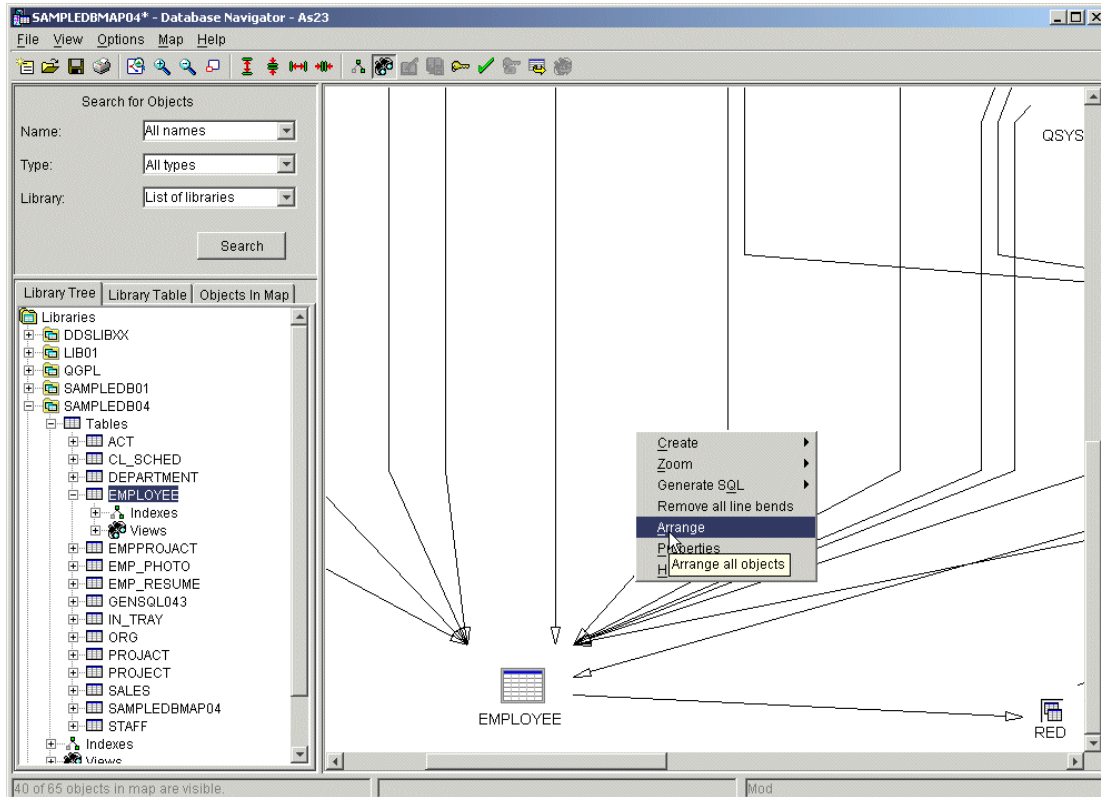


Figure 8-34 Arrange objects in the map

5. Select **Arrange** to minimize the line crossing the map.

## 8.7.4 Creating a user-defined relationship

As explained previously, when you have relationships that are defined by your programs, you can create a user-defined relationship in Database Navigator so that your relationship is displayed in the map. An example of this may be creating a user-defined relationship to remind programmers of an important join between two tables.

To add a user-defined relationship to your map, complete these steps:

1. Open a Database Navigator map.
2. Right-click in a free space on the map pane in the Database Navigator window. Select the **Create** function as shown in Figure 8-35.
3. Select **Create**, and then select **User-Defined Relationship** to create the new object (UDR).



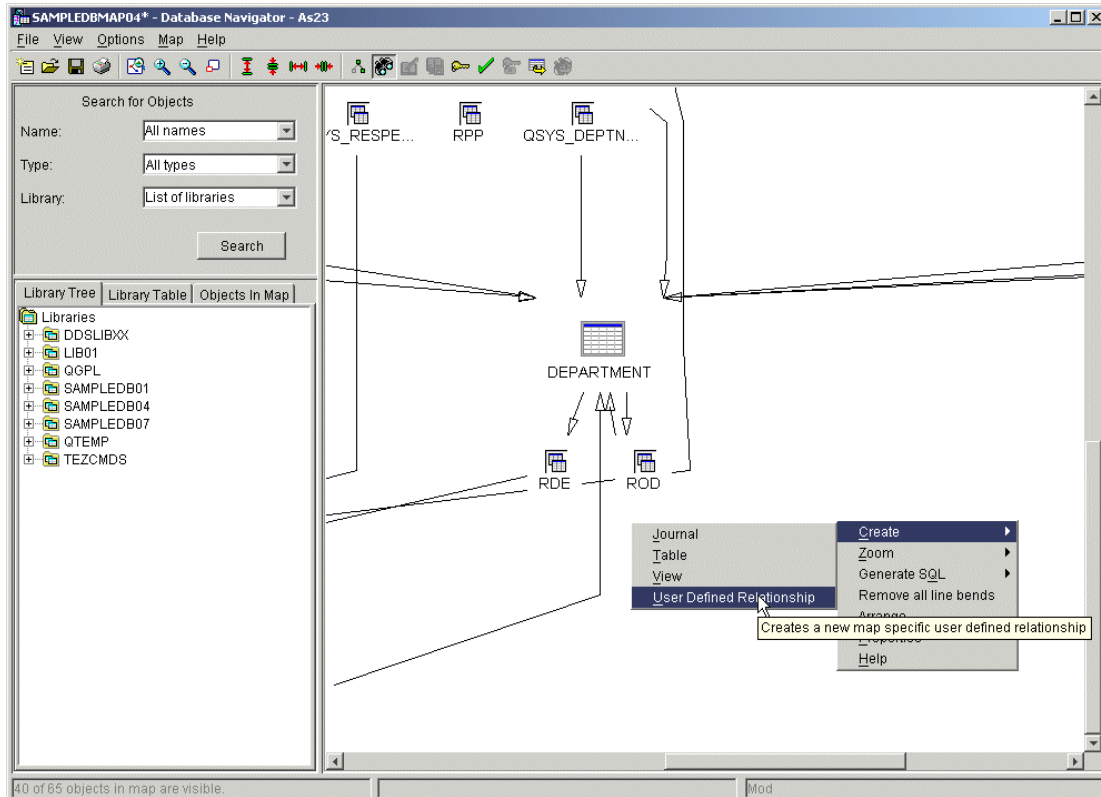


Figure 8-35 Selecting the function to create a user-defined relationship

4. Specify a name and a description for the user-defined relationship. Unlike some Operations Navigator functions where the description is optional, it is important to provide a meaningful description for your user-defined relationship because it is the only way for you to indicate what the user-defined relationship represents as shown in Figure 8-36.
5. Select the objects that you want to include in the relationship by selecting from the list of objects (Figure 8-36)
6. Choose the shape and color you want for the object (Figure 8-36).

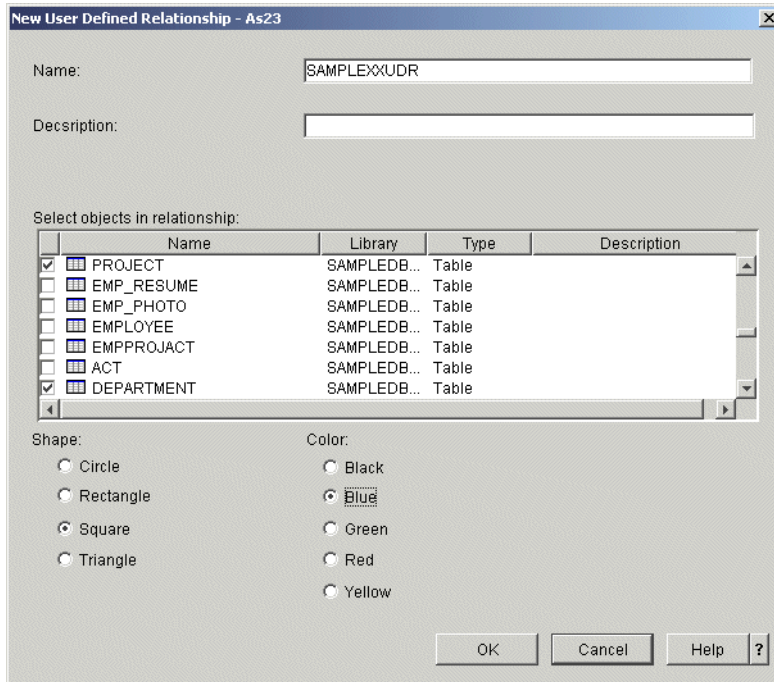


Figure 8-36 Creating a user-defined relationship

- Click **OK** to create the user-defined relationship. The map should show a user-defined relationship (UDR) as shown in Figure 8-37.

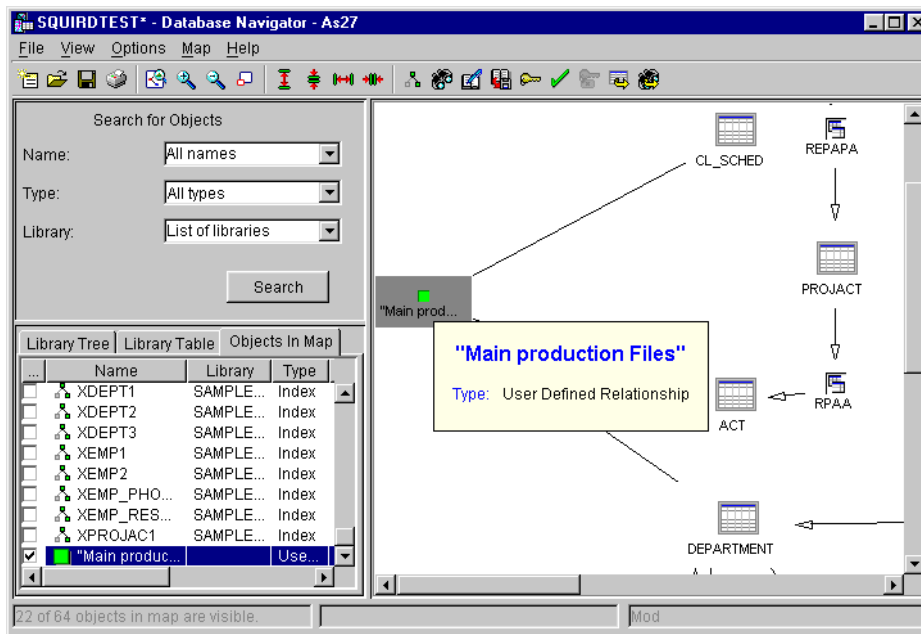













Figure 8-37 Flyover view of a user-defined relationship

## 8.8 The Database Navigator map icons

The icons that you may encounter on the Database Navigator map are shown in Table 8-1.

Table 8-1 Database Navigator map icons

---

	The Library icon is used in the Database Navigator map display to show a library.
	The Table icon is used in the Database Navigator map to show a table.
	The Table Alias icon is used in the Database Navigator map to show table aliases. It also is used as a toolbar icon for adding or removing a table alias from the Database Navigator map.
	The Index icon is used in the Database Navigator map to show an index.
	The Journal icon is used in the Database Navigator map to show a journal. It is also used as a toolbar icon for adding or removing a journal from the Database Navigator map.
	The Journal Receiver icon is used in the Database Navigator map to show a journal receiver. It is also used as a toolbar icon for adding or removing a journal receiver from the Database Navigator map.
	The Primary Key Constraint icon is used in the Database Navigator map to show a primary key constraint. It is also used as a toolbar icon for adding or removing a primary key constraint from the Database Navigator map.
	The Check Key Constraint icon is used in the Database Navigator map to show a check key constraint. It is also used as a toolbar icon for adding or removing a check key constraint from the Database Navigator map.
	The Unique Constraint icon is used in the Database Navigator map to show a unique constraint. It is also used as a toolbar icon for adding or removing a unique constraint from the Database Navigator map.
	The Foreign Key Constraint icon is used in the Database Navigator map to show a foreign key constraint.
	The View icon is used in the Database Navigator map to show a view. It is also used as a toolbar icon for adding or removing a view from the Database Navigator map.

---



The Show/Hide Index icon is used on the toolbar to add or remove an index from the Database Navigator map.



The Show/Hide Alias icon is used on the toolbar to add or remove an alias from the Database Navigator map.



Left-click this icon to set the zoom on the map so that it fits the current window size.



Left-click this icon to increase the level of zoom on the map at the position of the cursor.



Left-click this icon to decrease the level of zoom on the map at the position of the cursor.



Left-click this icon to invoke the Overview window function. This allows you to position your Database Navigator map panel to any part of a map.



Left-click this icon to decrease the horizontal level of spacing between objects on the map.



Left-click this icon to increase the horizontal level of spacing between objects on the map.



Left-click this icon to decrease the vertical level of spacing between objects on the map.



Left-click this icon to increase the vertical level of spacing between objects on the map.

---



## Reverse engineering and Generate SQL

Reverse engineering is one of the major changes that have been included in V5R1M0. This function allows you to create SQL for a given schema, table, index, view, etc., and all related objects to them if that option is selected. This enables database administrators to recreate, create duplicates, and port to other iSeries servers entire databases or particular parts of a database.

This chapter includes:

- ▶ What Generate SQL is
- ▶ Reverse engineering an existing database
- ▶ Generating SQL DDL statements from a DDS created database

## 9.1 Introduction

The new Generate SQL function is often referred to as “reverse engineering for Operations Navigator” because it provides a GUI interface that allows you to reverse engineer several types of database objects. The results are SQL create statements (often referred as DDL statements).

The Generate SQL function of Operations Navigator allows you to reconstruct SQL statements used to create existing database objects. With this function, you can reverse engineer database objects and then have the option to display the resulting SQL in the Run SQL Scripts window or saving the output to a file. Using the existing Run SQL Scripts functions, you can then edit, run, and save the SQL statement to a file on the PC.

The new Generate SQL Database Objects support the following objects:

- ▶ Aliases
- ▶ Distinct types
- ▶ Functions
- ▶ Indexes
- ▶ Procedures
- ▶ Schemas (collections) and libraries
- ▶ Tables and physical files
- ▶ Views and logical files

### 9.1.1 System requirements and planning

Before you Generate SQL, be sure the following prerequisites are available:

- ▶ 5722-SS1: Option 12 - Host Servers
- ▶ 5722-TC1: TCP/IP Connectivity Utilities
- ▶ 5722-XE1: Client Access Express, V5R1M0, with the latest Service Pack applied

### 9.1.2 Generate SQL

Reverse engineering (Generate SQL) allows you, through the Database Navigator map and the Libraries display of Operations Navigator, to re-engineer an SQL database or an iSeries database that were not created using SQL.

One of the uses of Generate SQL is to generate the SQL statements of tables, views, indexes, and constraints that were created using the Operations Navigator interface. For example, when you create a table using Operations Navigator, there is no method for saving the SQL statement that is behind the interface. In this case, Generate SQL provides a way to reverse engineer this object and obtain the SQL statement.

The Generate SQL function of Database Navigator also creates the SQL statements of databases created by DDS (physical and logical files). You must be aware that keyed-logical files are converted to SQL views.

When the Generate SQL process creates the Run SQL script for the selected object, it either marks any problem objects with SQL messages or it does not create the SQL for the object if it is not supported. You can create a Run SQL Script from object context or from schema context.

The object context can be invoked from either the Database Navigator map or the Operations Navigator Library display. To do this, right-click the object and select the **Generate SQL** option. There is a difference between what appears when using the two methods. If the Generate the SQL option is selected from the Library display, the information shown in Figure 9-1 appears.

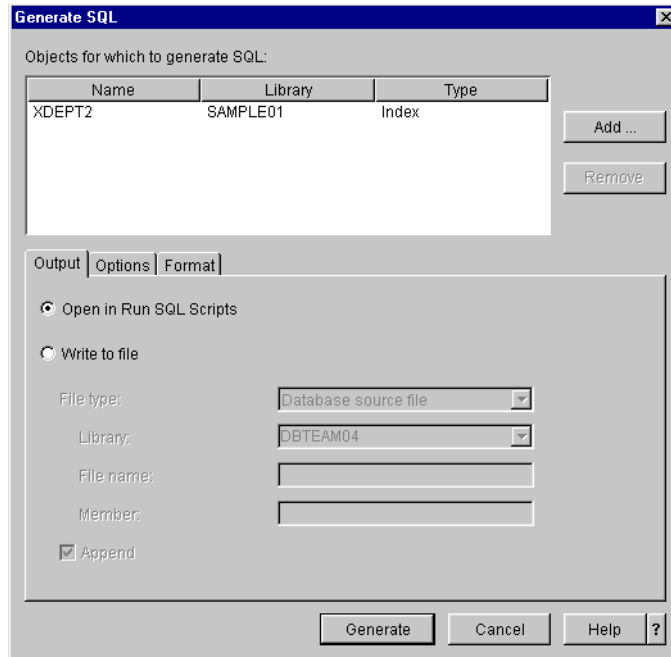


Figure 9-1 Operations Navigator Generate SQL display

The display shown in Figure 9-1 allows you to add or remove objects that will be re-engineered. This method allows you to change the objects that are selected and the standard by which they are generated, the format of the Run SQL script (Figure 9-2), and the options used to create the SQL script (Figure 9-3).

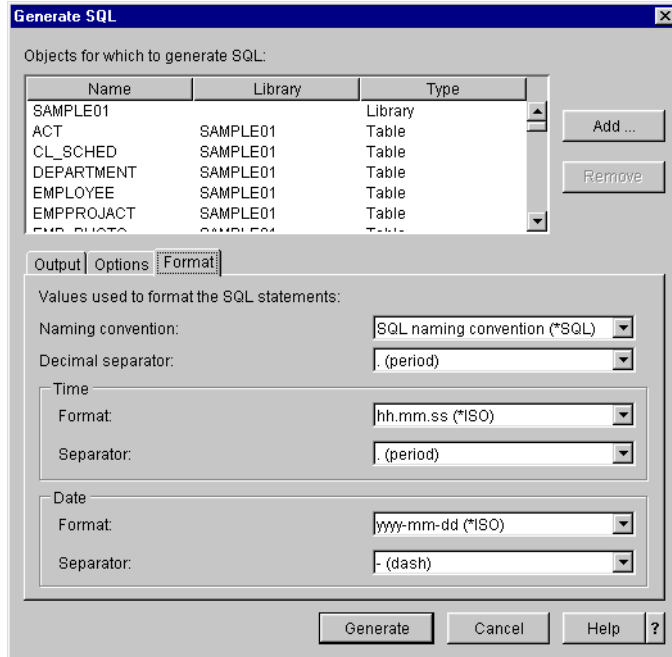


Figure 9-2 Generate SQL format options

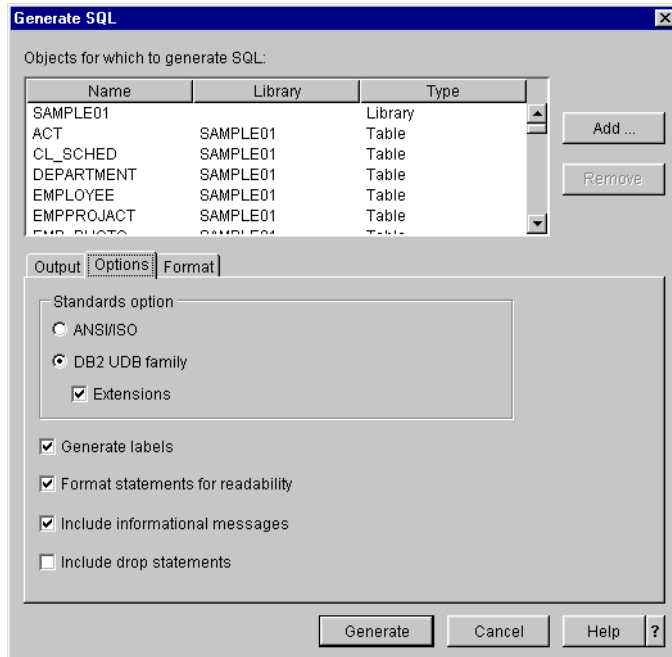


Figure 9-3 Generate SQL options

The options that you can define include:

- **Standards options:** This allows you to select which standards option you want for the generated SQL. The option that you choose affects the syntax of the generated SQL and ultimately how the Generate SQL runs. You may edit this value using the following sub-options:
  - **ANSI/ISO:** Select this option to allow the generation of SQL that can be executed on other ANSI/ISO SQL standard compliant databases.



- **DB2 UDB family:** Select this option to allow the generation of SQL for use on other DB2 family platforms.
- **DB2 UDB with iSeries extensions:** Select this option to allow the generation of SQL for use on other iSeries servers.

**Note:** As a general guideline, if you want to generate SQL that is run on other DB2 platforms, select DB2 UDB. In addition, if the platform is another iSeries server, choose to include iSeries extensions. The choice that you make for the standard can affect subsequent formatting choices.

- ▶ **Generate labels:** Select this option to include SQL labels and comments to be inserted into the generated SQL.
- ▶ **Format statements for readability:** Select this option to format the generated SQL statements with end-of-line characters, tab characters, and spaces.
- ▶ **Include informational message:** Select this option to include informational messages in your generated SQL. You should always include informational messages whenever you generate SQL for an object created using Data Description Specification (DDS). DDS is used to describe data attributes in file descriptions that are external to the application program that processes the data. You can then determine if you need to make changes to the generated SQL for it to run correctly. Once you make all the necessary changes, you may want to generate the SQL without the informational messages.

**Note:** If the object for which you are generating SQL was originally created using SQL, there should not be any informational messages.

- ▶ **Include drop statements:** Select this option to include drop statements for the objects for which you are generating SQL. The drop statements are inserted before the first Create SQL statement. This allows you to drop the object and then recreate it.

Click the **Generate** button to prompt the system to generate the SQL and bring up the Run SQL script window (Figure 9-4).

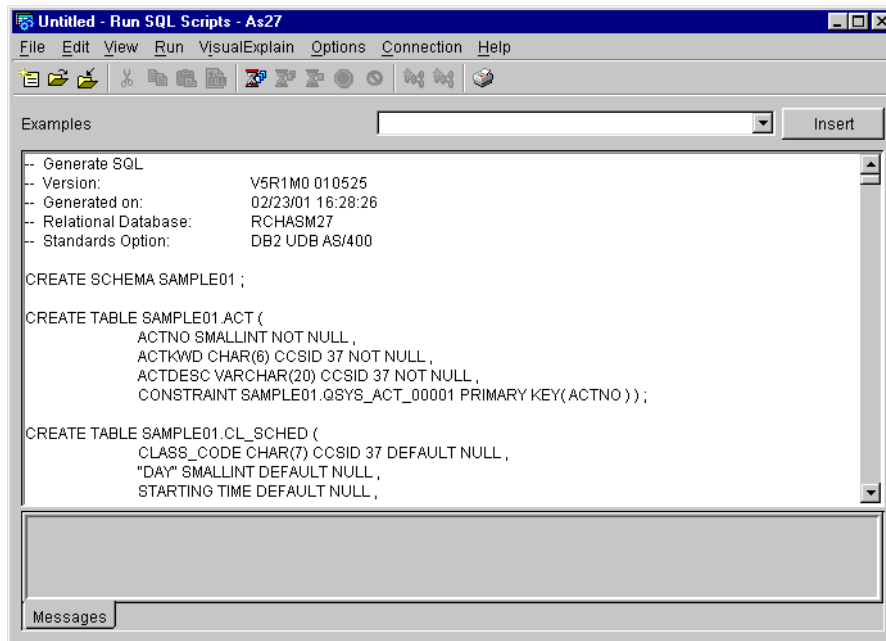


Figure 9-4 Generate SQL Run SQL Scripts window

One of the major advantages of the Generate SQL function is that the SQL can be ported to other iSeries servers and even to other platforms supporting SQL. This applies particularly to CASE tools that can use the Run SQL Script as input to recreate the database on other platforms.

## 9.2 Generating SQL from the library in Operations Navigator

With Generate SQL, there is an option from your library in the Operations Navigator window to generate the SQL DDL statement for some objects. To generate this statement, follow these steps:

1. Start Operations Navigator. Click the iSeries server that you want to access (Figure 9-5). Once you have entered your user ID and password, expand the **Database** option.

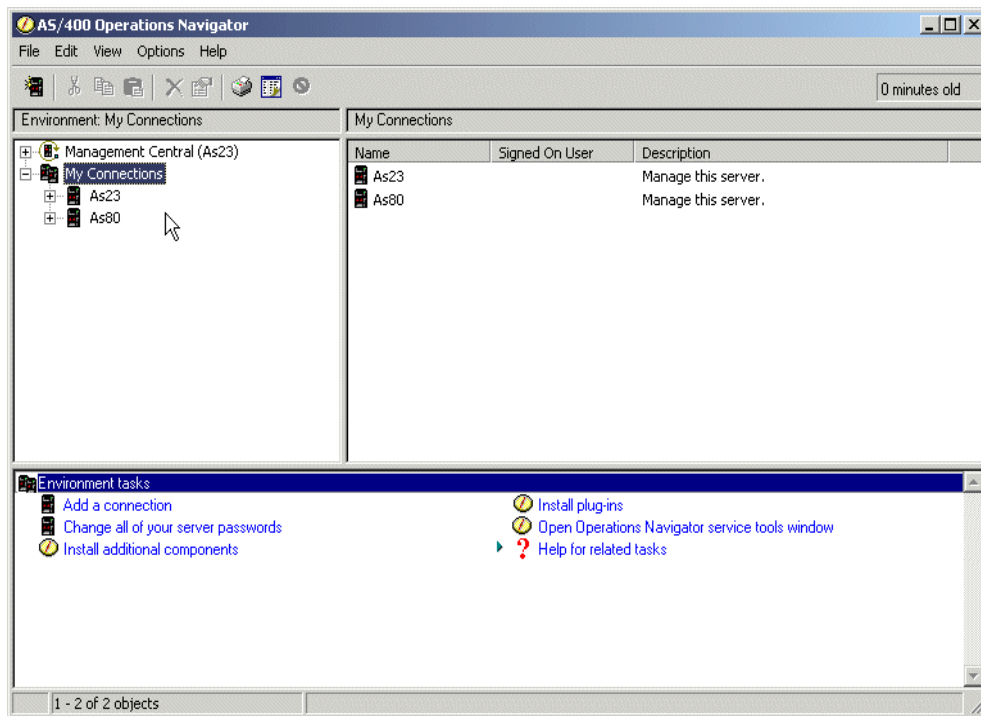


Figure 9-5 Operations Navigator

2. Under Database, click **Libraries**. Then select the library name, which in our case is SAMPLEDB04, for your iSeries server connection (Figure 9-6).

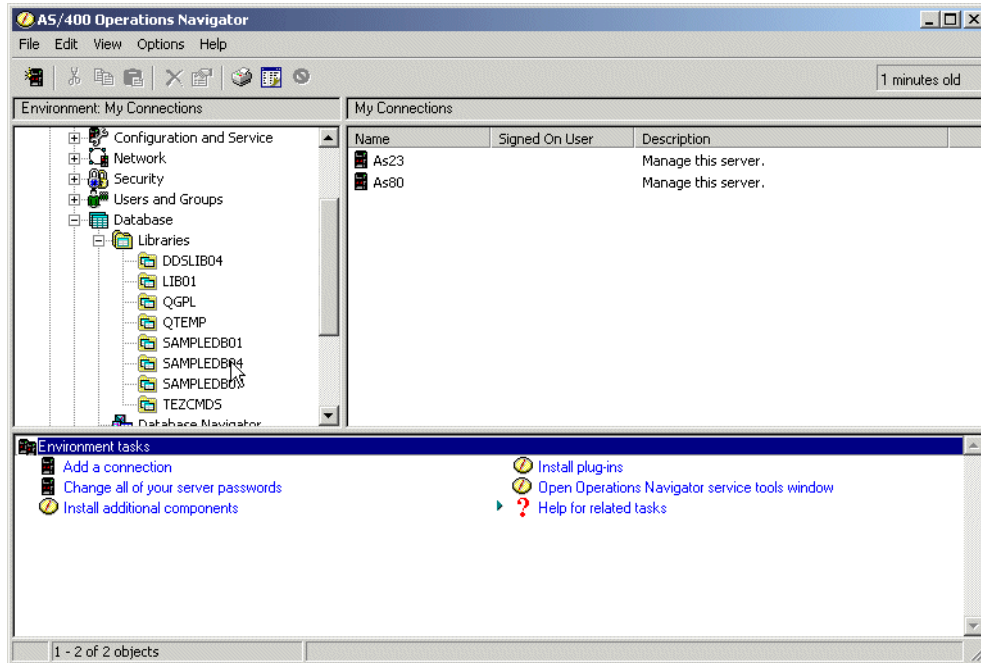


Figure 9-6 Find library

- Click the **SAMPLEDB04** library to display the current content in the right window panel. On the right panel, press the Ctrl key, and locate and select the following tables as shown in Figure 9-7:
  - ACT
  - CL\_SCHED
  - DEPARTMENT
  - EMP\_PHOTO
  - EMP\_RESUME
  - EMPLOYEE

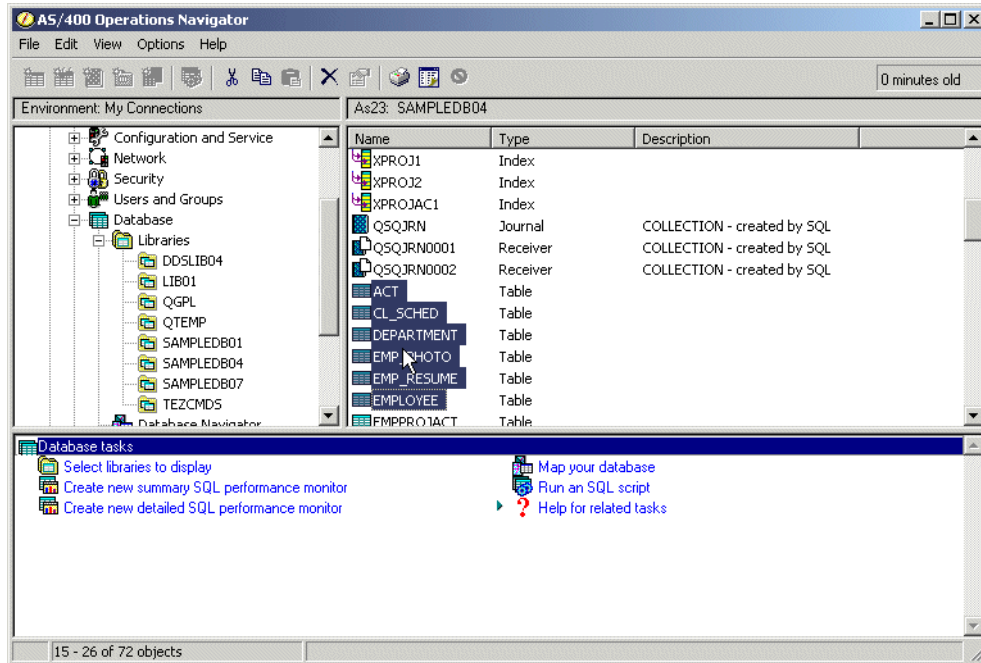


Figure 9-7 Selecting objects from the library to generate SQL

**Important:** When the Generate SQL function is invoked, the new Generate SQL window appears as shown in Figure 9-8. This window provides a list of the objects initially selected and three tabs that specify Output, Format, and Options that are used in the Generate SQL.

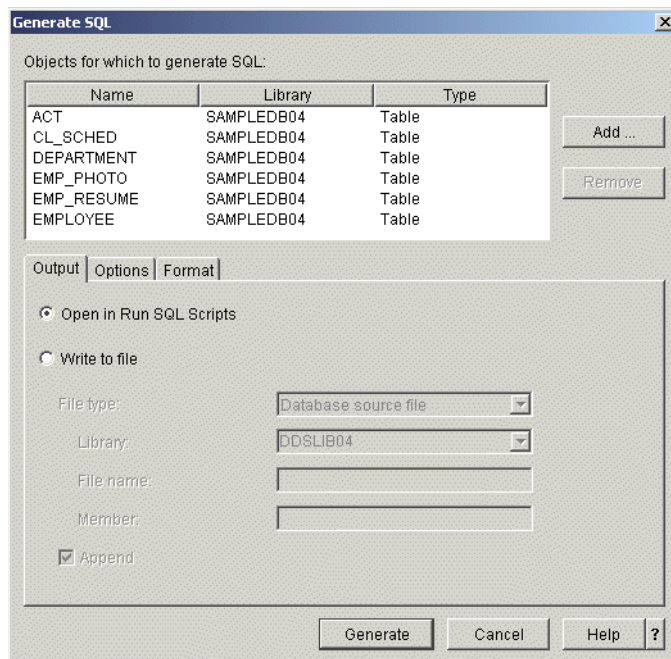


Figure 9-8 Generate SQL window

4. Click **Generate** to accept the default values as shown in Figure 9-9.

**Important:** The initial list of objects in the Generate SQL window could be modified using the Add and Remove buttons to add new objects or remove objects from the initial list.

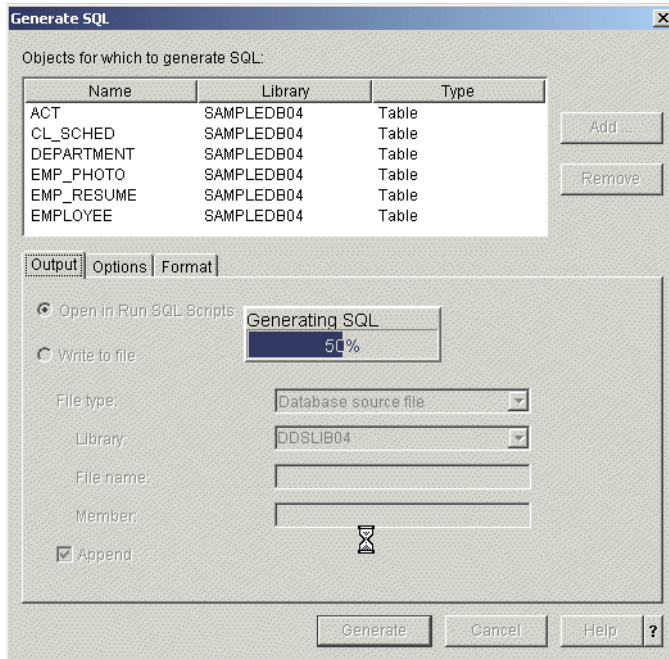


Figure 9-9 Generate SQL display

5. Switch to the new Run SQL Scripts window to see the generated SQL statement.

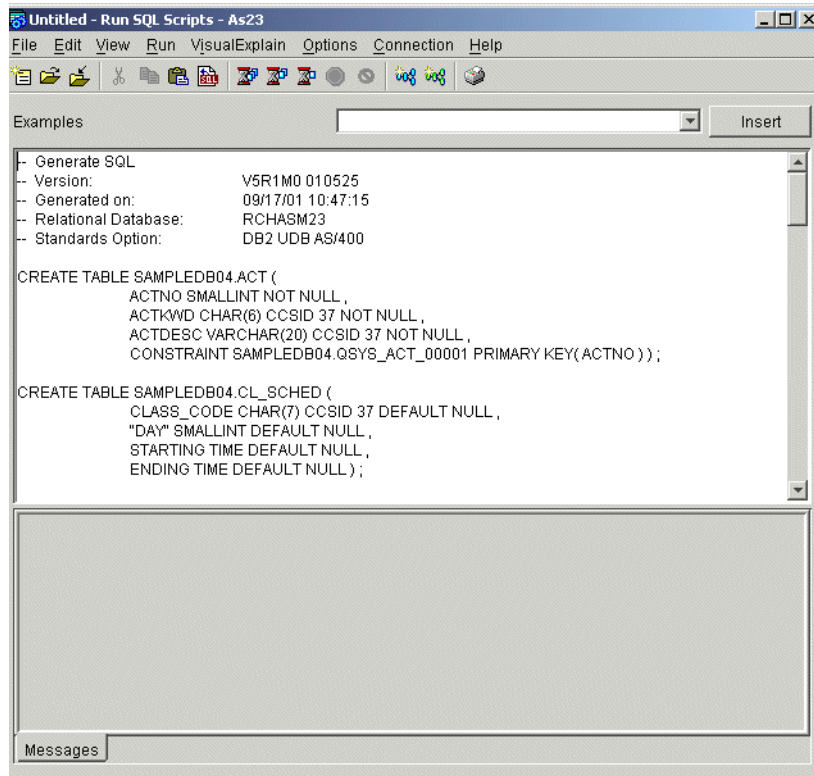


Figure 9-10 SQL generated in the Run SQL Scripts window

6. Click **File** and select **Save As...** from the pull-down menu to save the SQL script as shown in Figure 9-11.

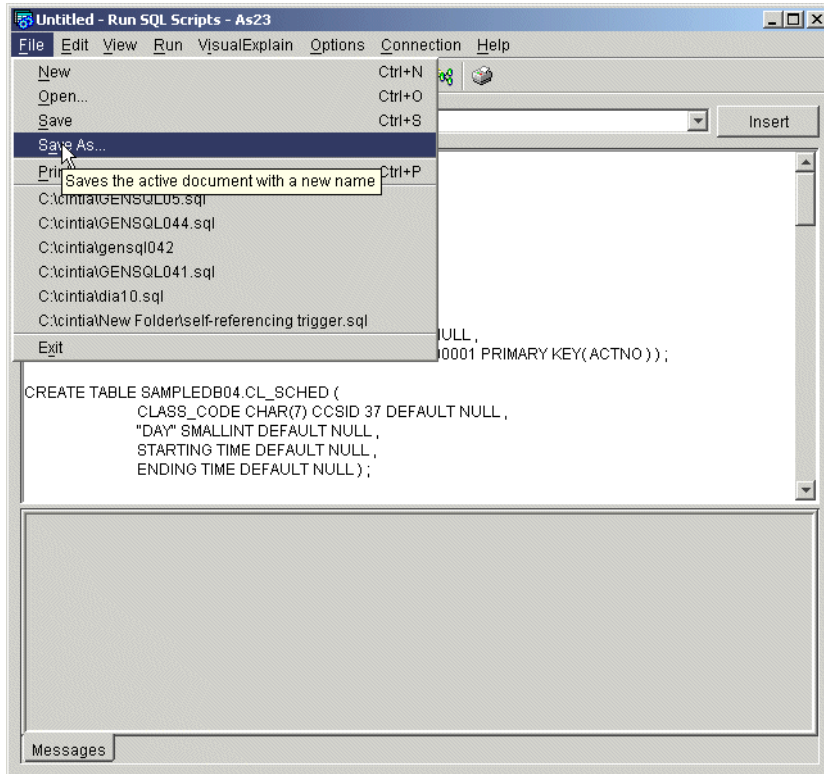


Figure 9-11 Saving the SQL Script

**Important:** You can use the SQL file to replicate your database files on another system (for example, a development system).

7. Click **Save** to save the SQL script file.

### 9.2.1 Generating SQL to PC and data source files on the iSeries server

You can generate the SQL statements to a PC file and to a source member on the iSeries server. Let's start by showing you how to generate the SQL statements of a group of objects to a PC file:

1. Start Operations Navigator.
2. Right-click the **SAMPLEDB04 library** (example in our case). Then select **Generate SQL** as shown in Figure 9-12.

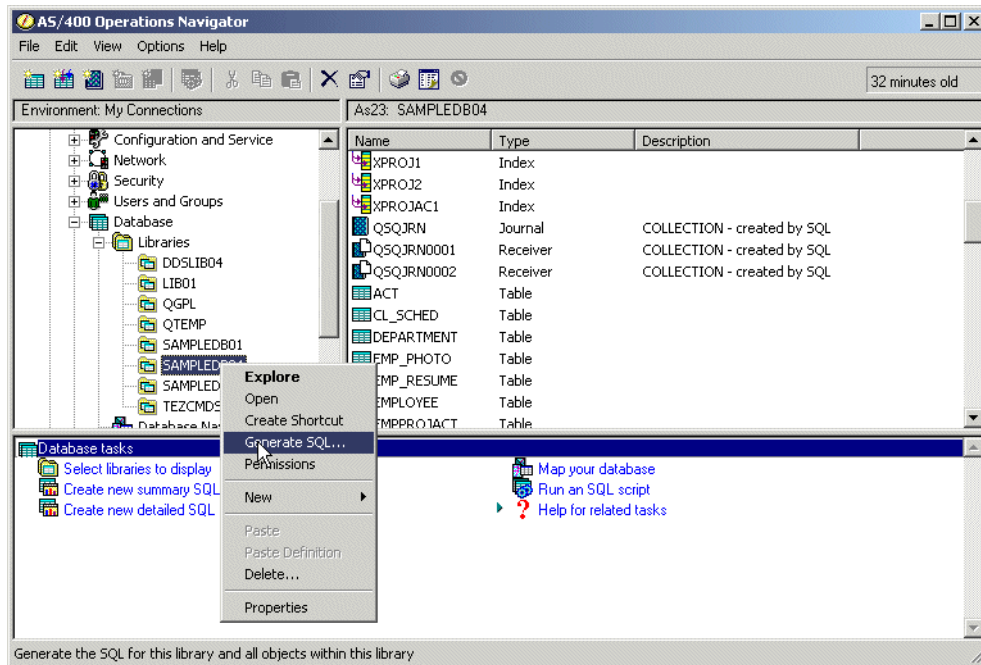


Figure 9-12 Generate SQL library in Operations Navigator

3. In the Generate SQL window, select the **Write to file** option on the **Output** tab as shown in Figure 9-13. The generated SQL is saved to a PC file.

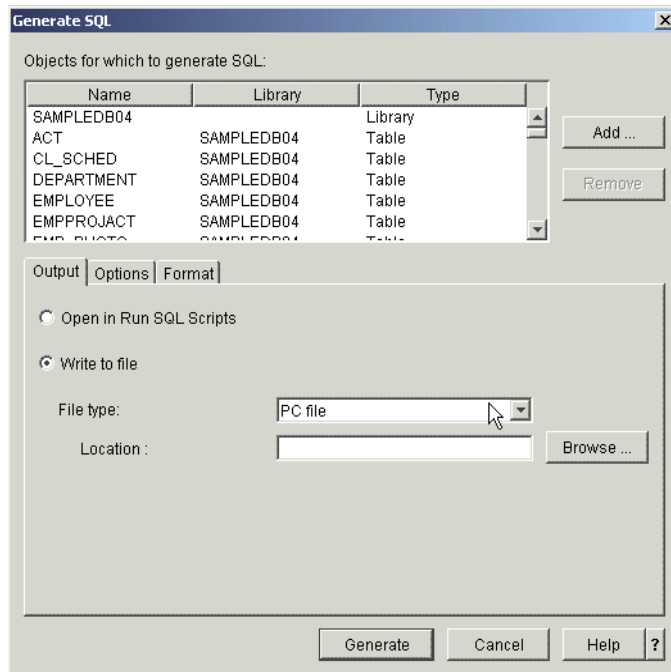


Figure 9-13 Selecting Generate SQL to PC

4. Click **File Type** and select the **PC file** option.
5. In the Location file, click **Browse**. Then select your directory (**c:\DB2NAVSQL**) from the pull-down menu to save your file.



- In the File name input field, type **GENSQL042.SQL**. In the Files of type input field, leave the default SQL files (.sql) as shown in Figure 9-14.

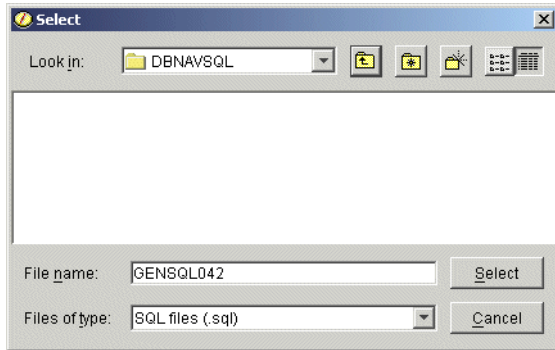


Figure 9-14 Saving the SQL script to a PC file

- Click the **Select** button to return to the Generate SQL tab.

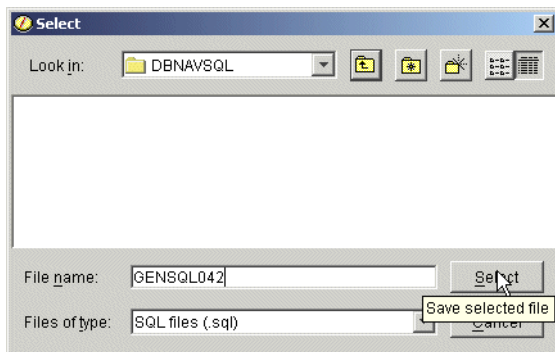


Figure 9-15 Select button

- Click the **Generate** button to start generating the SQL DDL statements for all the objects in the library. A status window appears showing the progress of the generate SQL process as a percentage (Figure 9-16).

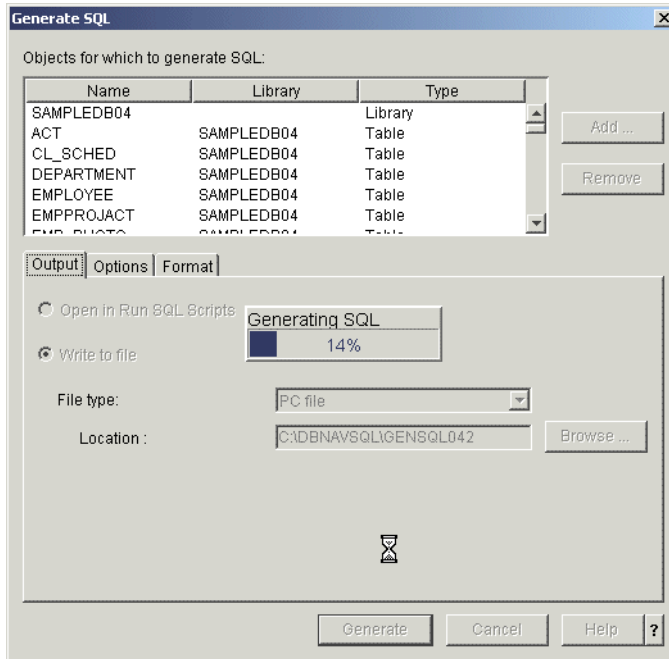


Figure 9-16 Generating SQL window

- In the Operations Navigator window, click the **Run SQL Script** icon in the database task pad as shown in Figure 9-17.

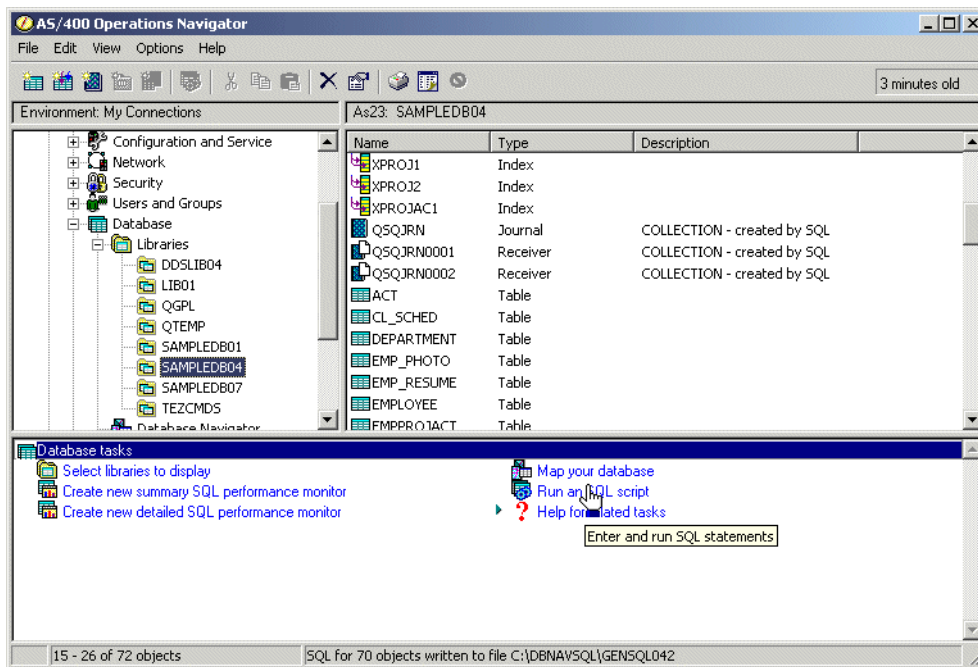


Figure 9-17 Selecting the Run SQL Script from the task pad option

**Important:** One of the new functions added in V5R1 is the task pad (located in the lower part of the Operations Navigator window). If you click the various higher level options, such as Security, Users and Groups, Database, etc., this task pad changes accordingly. One of the database tasks of the task pad is Run SQL Script.

10. In the Run SQL Scripts window, click **File** and select **Open** from the pull-down menu to open your SQL Script file (**GENSQL042**).
11. Click **Look in** and select your directory (**C:\DBNAVSQL**) from the pull-down menu to save your file.
12. Select your **GENSQL042** file and click **Open** as shown in Figure 9-18.

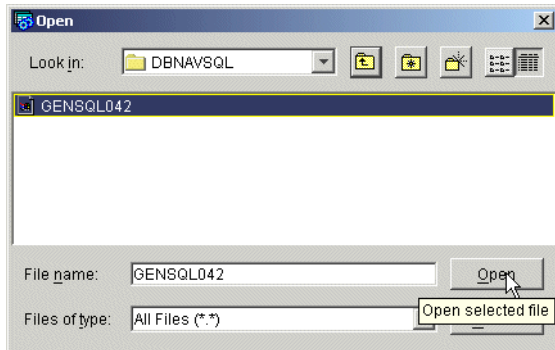


Figure 9-18 Restoring an SQL script file from a PC

13. View the SQL statements generated on the Run SQL Script window as shown in Figure 9-19. Take some time to analyze the order of the statements.

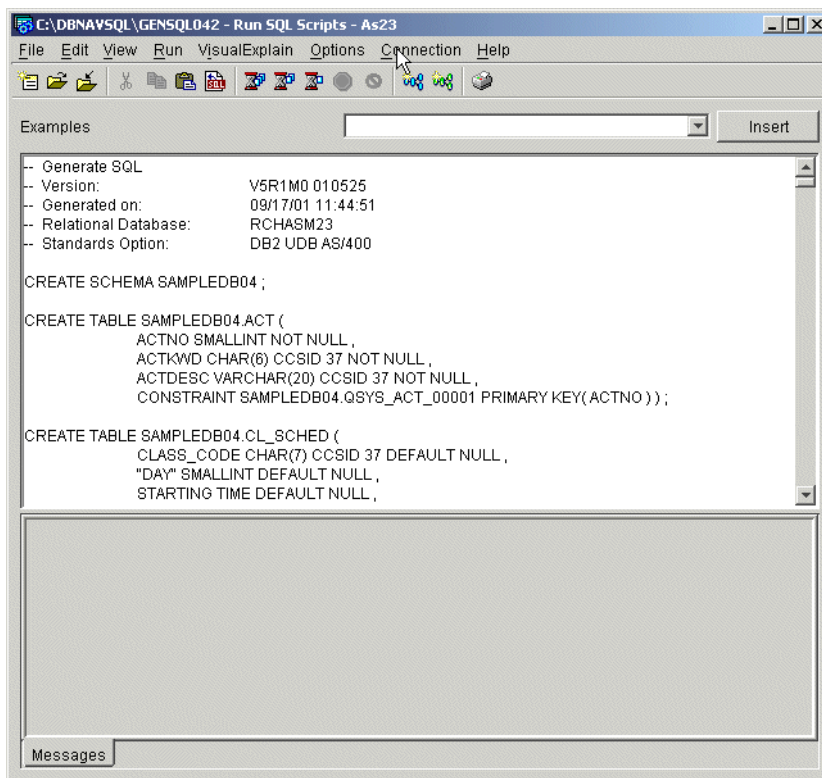


Figure 9-19 SQL Script statement generated

**Important:** Once the statements are generated, you can edit them to create a new copy in another library and optionally saved, or you can run them using the SQL Script facility. If multiple objects were selected to be SQL Generated, you have the option to run one, some, or all of the statements after any required editing.

Let's see now how to generate the SQL statements of a group of objects to a source physical file on the iSeries server:

1. Click the **SAMPLEDB04** library to display the content in the right window panel.
2. Click **File** and select **Generate SQL...** from the pull-down menu to view the Generate SQL window as shown in Figure 9-20. This is another way to generate SQL for a group of objects.

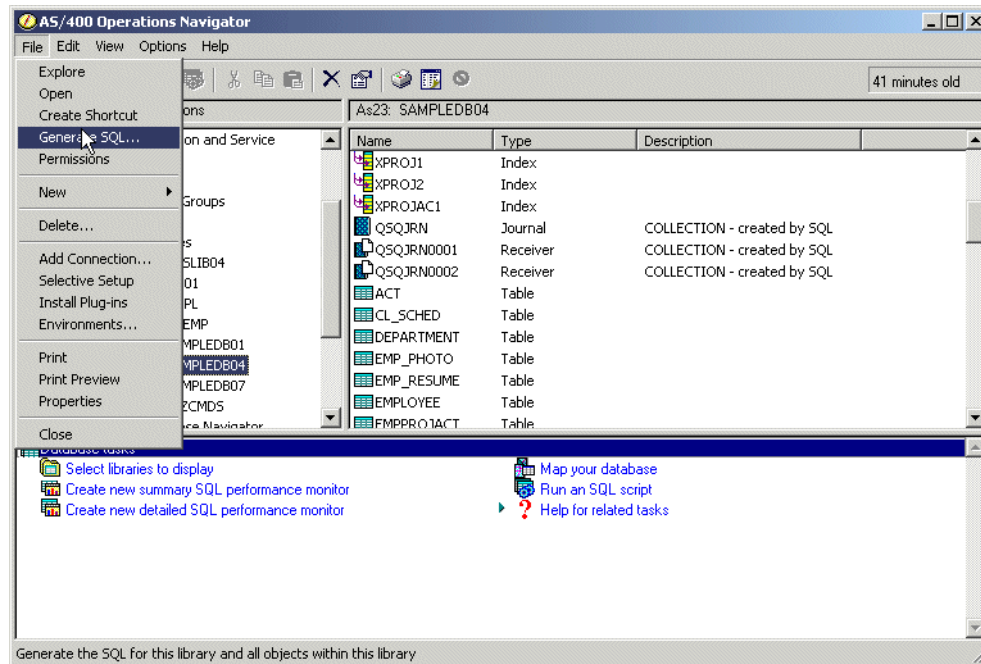


Figure 9-20 Selecting Generate SQL from the File menu

3. In the Generate SQL window, click the **Write to file** option in the **Output** tab as shown in Figure 9-21.

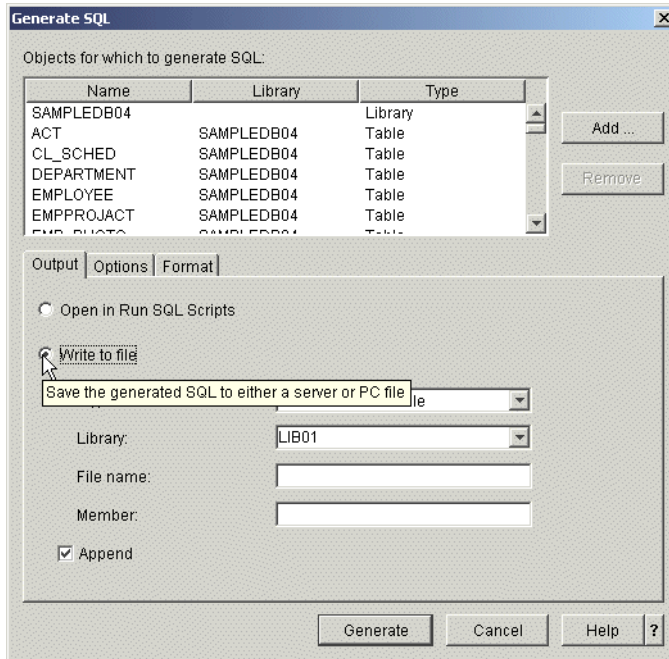


Figure 9-21 Generate SQL: Selecting Write to file

4. Click **File Type** and select the database source file.
5. Click **Library** and select the **SAMPLEDB04 library** in our case.
6. In the File Name input field, type **GENSQL043**. In the Member input field, type **GENSQL043**.
7. Click the **Generate** button to start the Generate SQL process on the iSeries server.

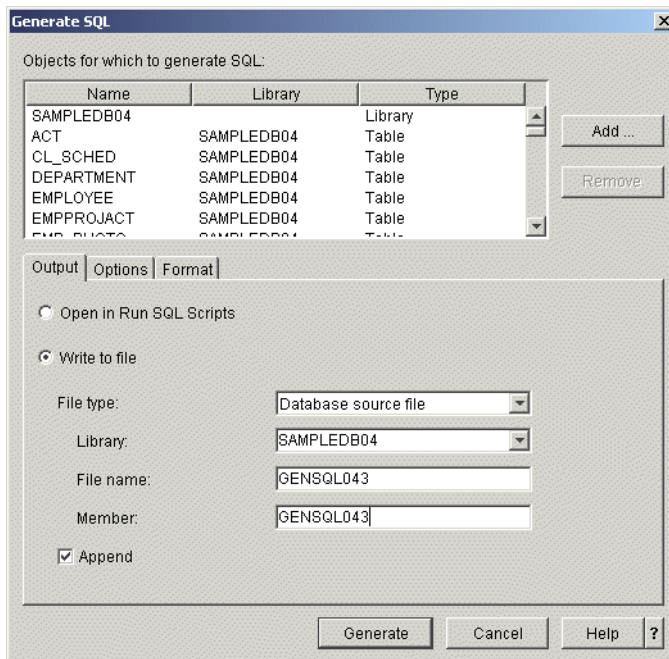


Figure 9-22 Starting the Generate SQL process on the iSeries server

**Note:** For existing files, the option to append to the file is provided. If an existing file is selected, and the append option is not chosen, you are asked if you want to overwrite the existing file.

8. Double-click **GENSQL043** to see the script on the Operations Navigator window as shown in Figure 9-23.

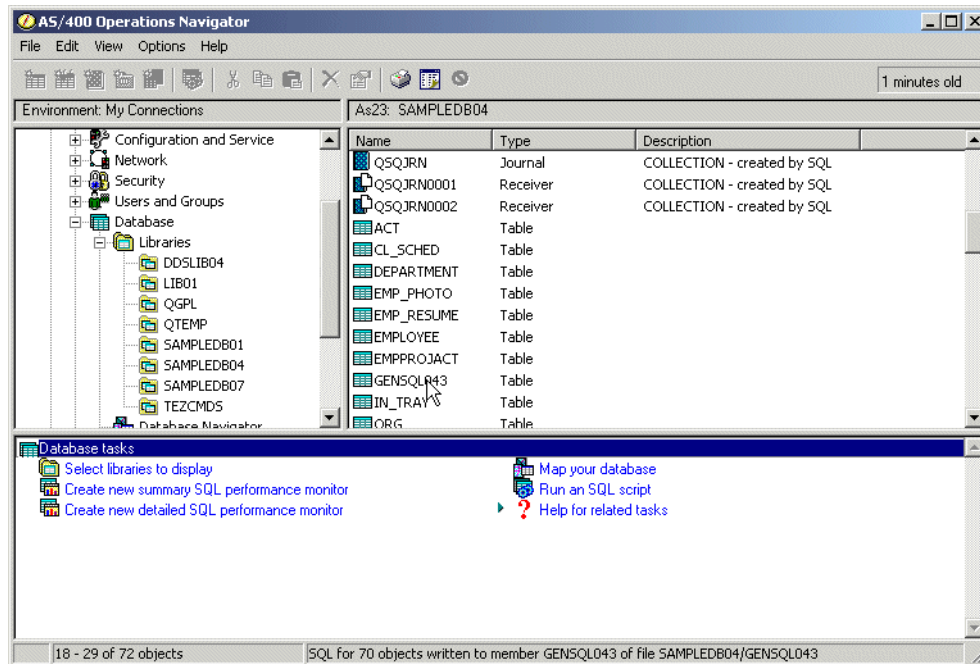


Figure 9-23 Selecting the source physical file to show the Generate SQL Script

9. Expand the window, and use the scroll bar to explore the script file as shown in Figure 9-24.

SRCSEQ	SRCDAT	SRCDTA
1.00	10917	- Generate SQL
2.00	10917	- Version: V5R1M0 010525
3.00	10917	- Generated on: 09/17/01 09:43:08
4.00	10917	- Relational Database: RCHASM23
5.00	10917	- Standards Option: DB2 UDB AS/400
6.00	10917	
7.00	10917	CREATE SCHEMA SAMPLEDB04;
8.00	10917	
9.00	10917	
10.00	10917	CREATE TABLE SAMPLEDB04.ACT (
11.00	10917	ACTNO SMALLINT NOT NULL,
12.00	10917	ACTKWD CHAR(6) CCSID 37 NOT NULL,
13.00	10917	ACTDESC VARCHAR(20) CCSID 37 NOT NULL,
14.00	10917	CONSTRAINT SAMPLEDB04.QSYS_ACT_00001 PRIMARY ...
15.00	10917	
16.00	10917	
17.00	10917	CREATE TABLE SAMPLEDB04.CL_SCHED (
18.00	10917	CLASS_CODE CHAR(7) CCSID 37 DEFAULT NULL,
19.00	10917	"DAY" SMALLINT DEFAULT NULL,
20.00	10917	STARTING TIME DEFAULT NULL,
21.00	10917	ENDING TIME DEFAULT NULL);
22.00	10917	
23.00	10917	
24.00	10917	CREATE TABLE SAMPLEDB04.DEPARTMENT (
25.00	10917	DEPTNO CHAR(3) CCSID 37 NOT NULL,
26.00	10917	DEPTNAME VARCHAR(36) CCSID 37 NOT NULL,
27.00	10917	MGRNO CHAR(6) CCSID 37 DEFAULT NULL,
28.00	10917	ADMRDEPT CHAR(3) CCSID 37 NOT NULL,
29.00	10917	LOCATION CHAR(16) CCSID 37 DEFAULT NULL,
30.00	10917	CONSTRAINT SAMPLEDB04.QSYS_DEPARTMENT_00001 ...
31.00	10917	
32.00	10917	ALTER TABLE SAMPLEDB04.DEPARTMENT

Figure 9-24 Exploring the SQL Script file from Operations Navigator

## 9.2.2 Generating SQL from the Database Navigator map

It is also possible to generate the SQL DDL statement from some or all objects in a map generated by the Database Navigator feature (Chapter 8, "Database Navigator" on page 239).

1. Click the Database Navigator icon to display the maps on the right that exist on the iSeries server as shown in Figure 9-25.

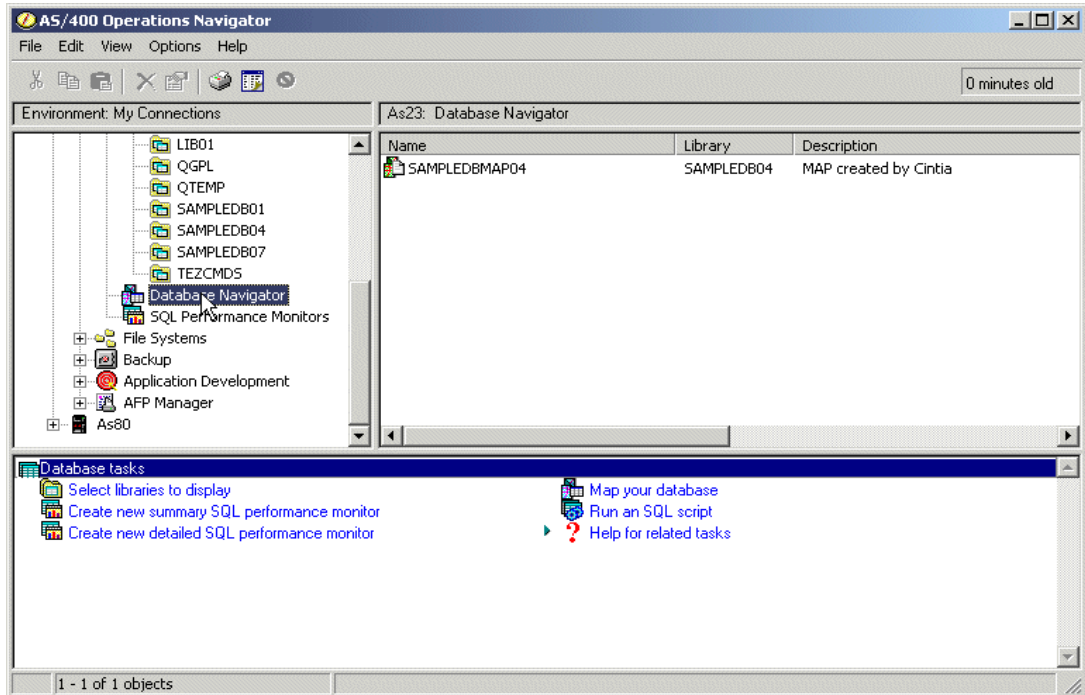


Figure 9-25 Opening the Database Navigator map

2. Double-click to open the database map that you created.
3. Click the **View** menu and select **Zoom-> To Fit Window** from the pull-down menu to fit all objects on the map in this window as shown in Figure 9-26.



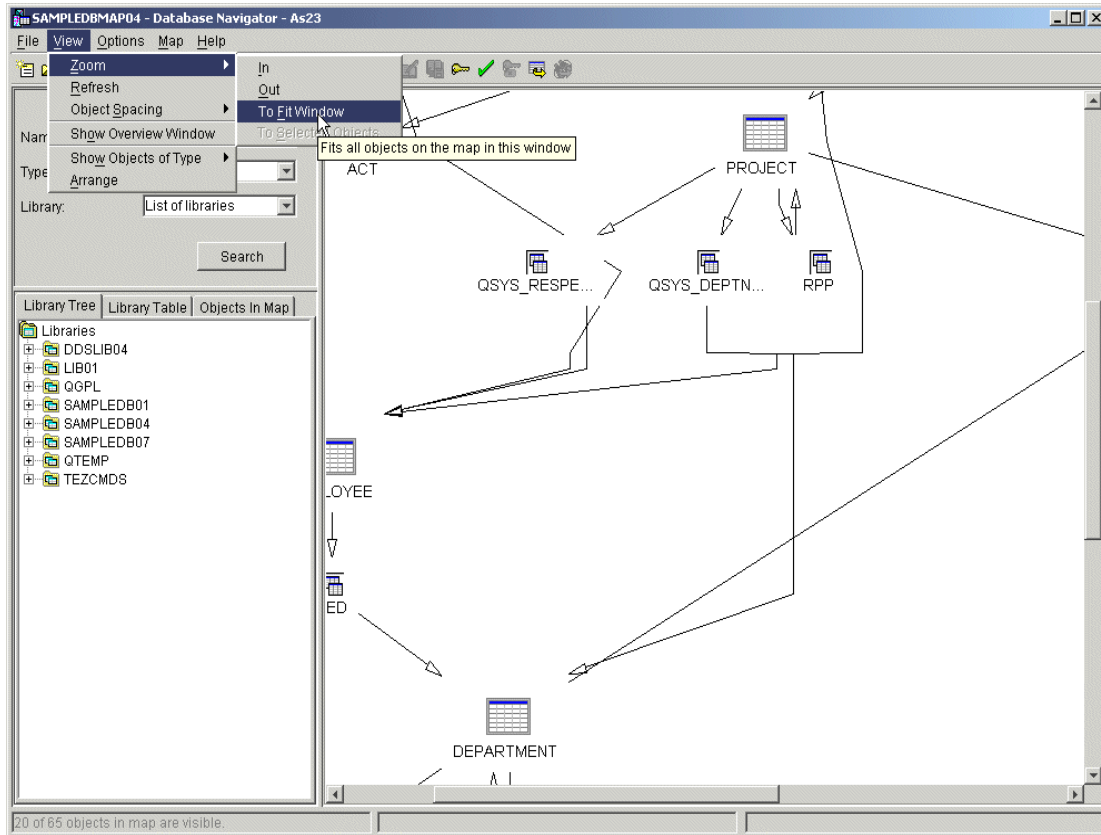


Figure 9-26 Fitting all objects in a map

4. Use the vertical and horizontal scroll bars to navigate to the top of the map as shown in Figure 9-27.

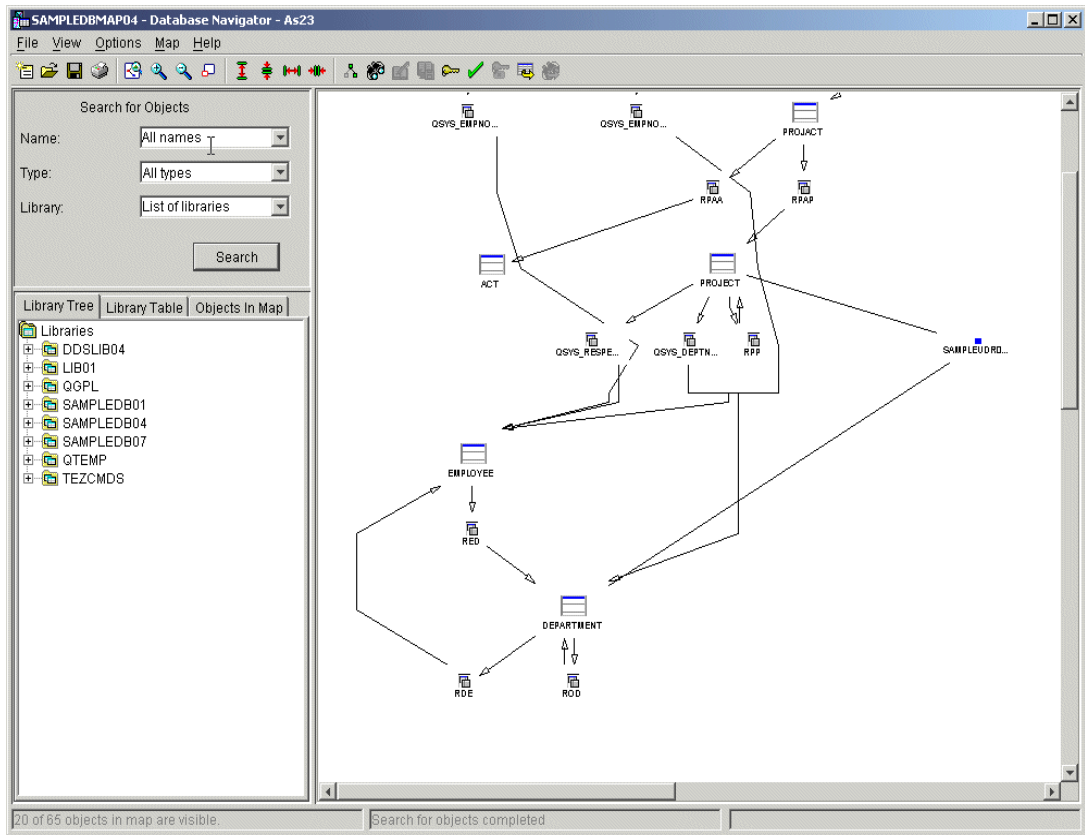


Figure 9-27 Viewing all objects includes in the map

5. Use the criteria selection in the locator pane and select only your **SAMPLEDB04** library. Click the **Library** parameter to select your library as shown in Figure 9-28.

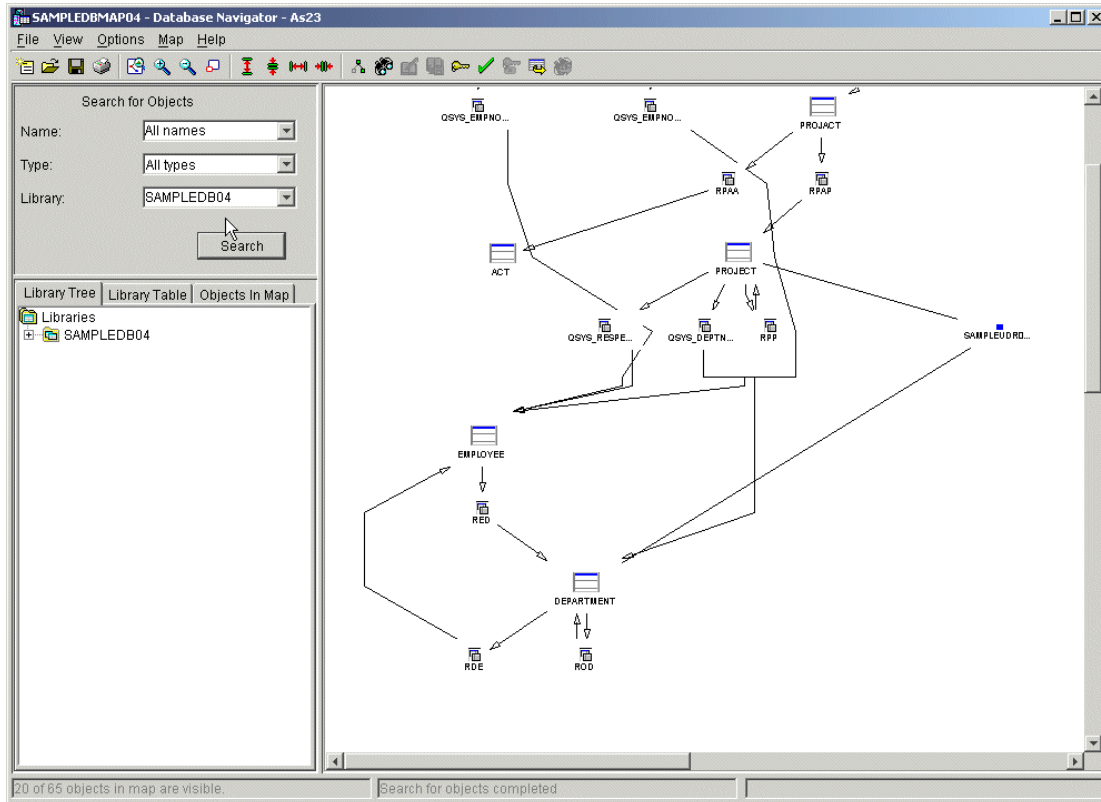


Figure 9-28 Selecting only your sample library to appear in the Database Navigator map

6. Click the plus (+) sign next your SAMPLEDB04 database to see the found objects, such as tables, indexes, and views.
7. Click the (+) sign next to the Tables database object to expand it.
8. Double-click the **EMPLOYEE** table in the list of tables to find this table in the map.
9. Right-click the **EMPLOYEE** table and select **Generate SQL...** as shown in Figure 9-29.

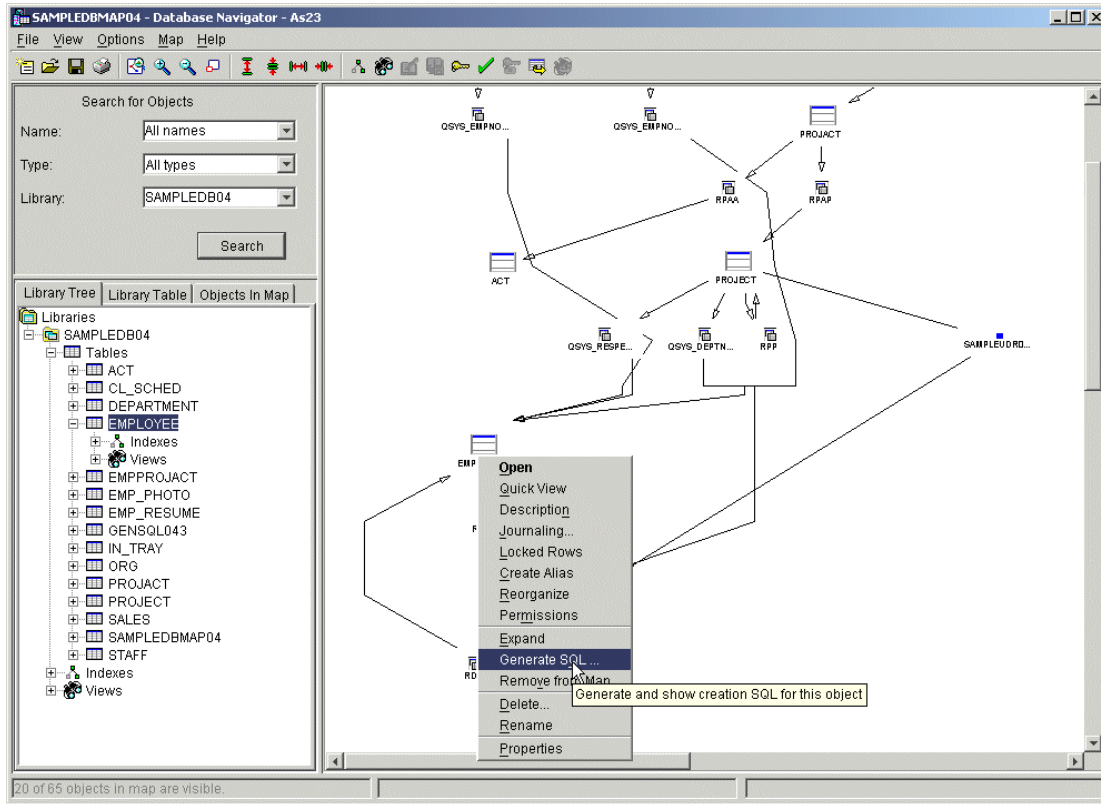


Figure 9-29 Generating SQL for a specific object from the map

10. In the Run SQL Script window, explore the Generated SQL statement, using the scroll bar to navigate as shown in Figure 9-30.

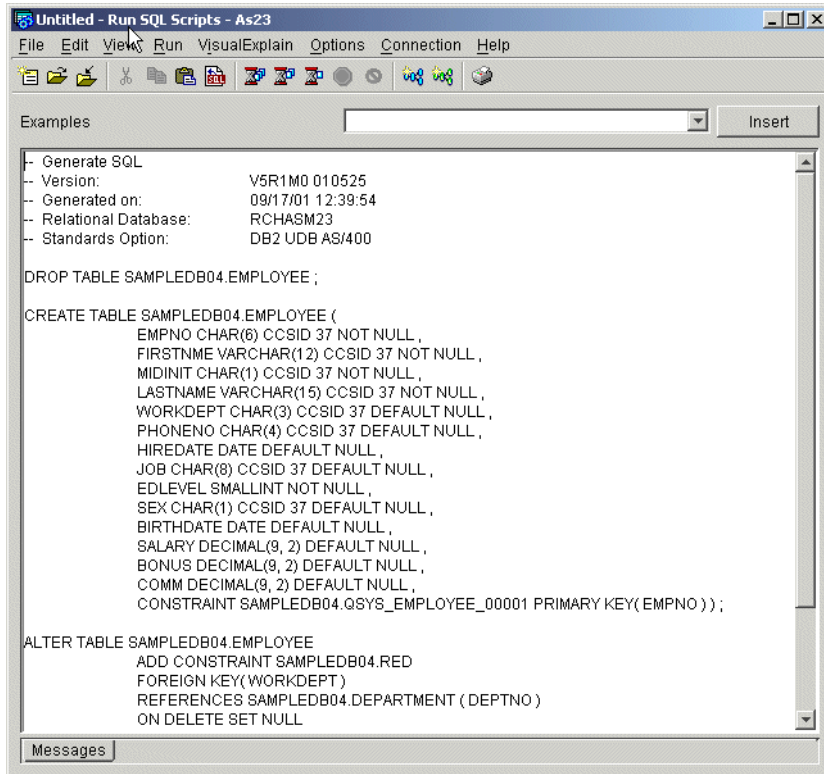


Figure 9-30 Generating SQL from the employee table

11. Click **File** and select **Save As...** from the pull-down menu to save the SQL script as shown in Figure 9-31.

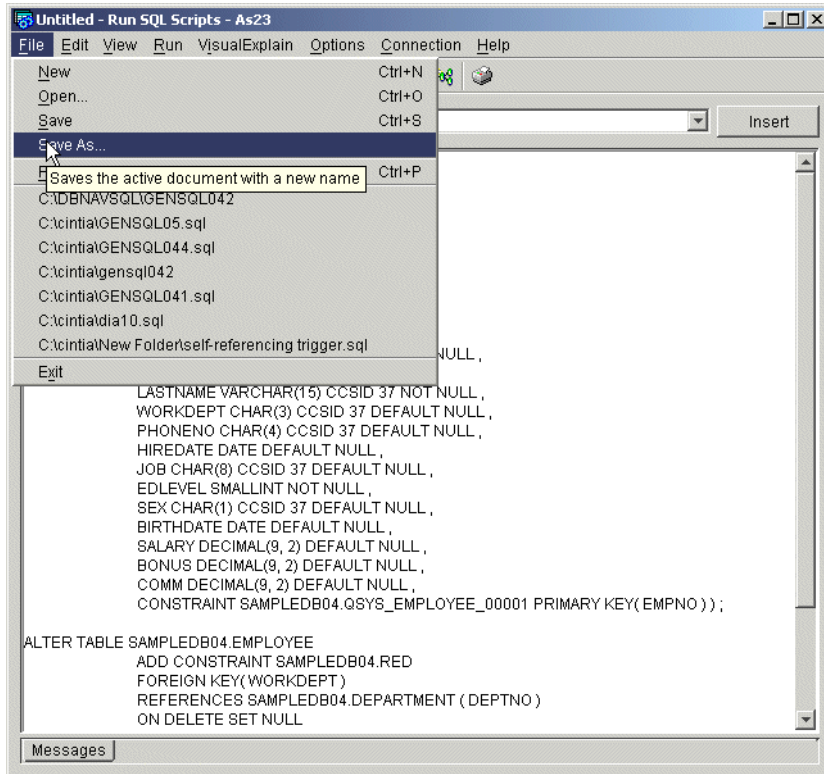


Figure 9-31 Saving the Script SQL

12. On the Save window, click your directory (**C:\DBNAVSQL**) from the pull-down menu to save your file.
13. In the Name input field, type **GENSQL044**. In the Type input field, leave the default SQL files (.SQL) as shown in Figure 9-32.
14. Click **Save** to save the SQL script file.

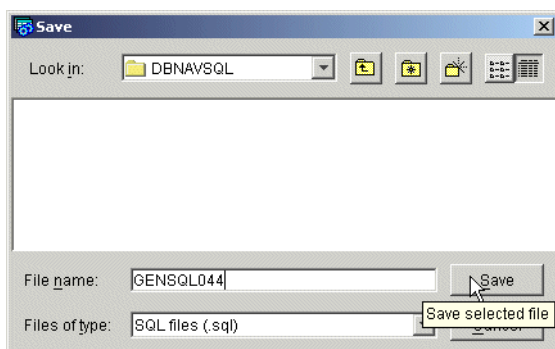


Figure 9-32 Saving the SQL Script file

Now let's see how to generate the SQL DDL statements for all the objects in a library.

1. Switch to the Database Navigator window.
2. Click the **Map** option and select **Generate SQL** from the pull-down menu. Click **All Objects...** to generate the SQL statement for all objects in your library as shown in Figure 9-33.

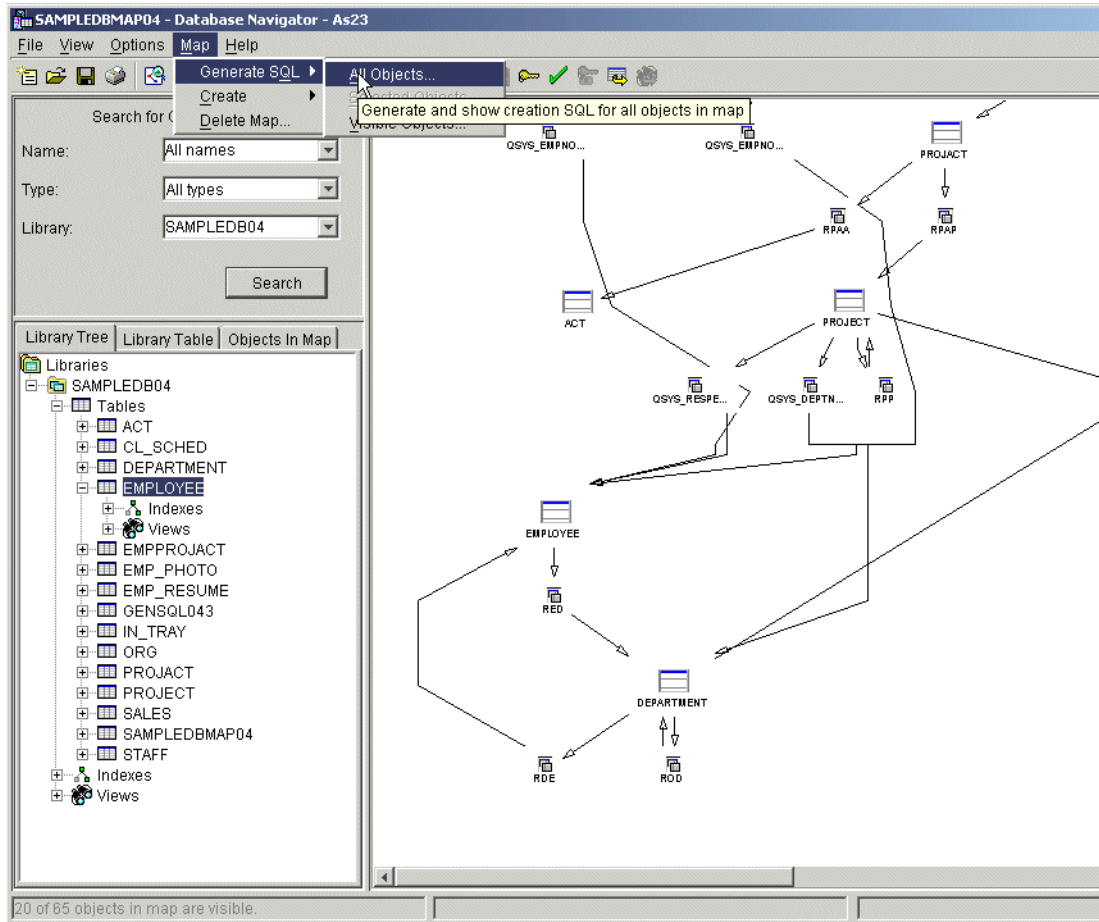


Figure 9-33 Generate SQL for all objects in a library

3. A status window appears showing the progress of the Generate SQL as a percentage.
4. Click **File** and select **Save As...** from the pull-down menu to save the map.
5. In the Save window, click to select your directory (**C:\DBNAVSQ**) from the pull-down menu to save your file.
6. In the File name input field, type **GENSQL045**. In the File of type input field, leave the default as SQL files (.SQL).
7. Click **Save** to save the SQL script file.
8. Click **File** and select **Exit** from the pull-down menu to close the Run SQL Script window.

### 9.2.3 Generating SQL from DDS

The Generate SQL function works with objects created using SQL and also with objects that were created using DDS. These objects can also be reverse engineered into an SQL create statement. This is a way to start migrating or changing existing DDS created databases to SQL.

Let's see how to reverse engineer an existing DDS created database:

1. Click the plus (+) sign next to the Libraries object to expand the list of libraries.
2. Change the list of libraries in Operations Navigator to include the library that has DDS created objects. For this example, let's say it is **DDSLIBXX**. Click **DDSLIBXX**.

3. Right-click the **DDSLIBXX** library and select **Generate SQL** as shown in Figure 9-34.

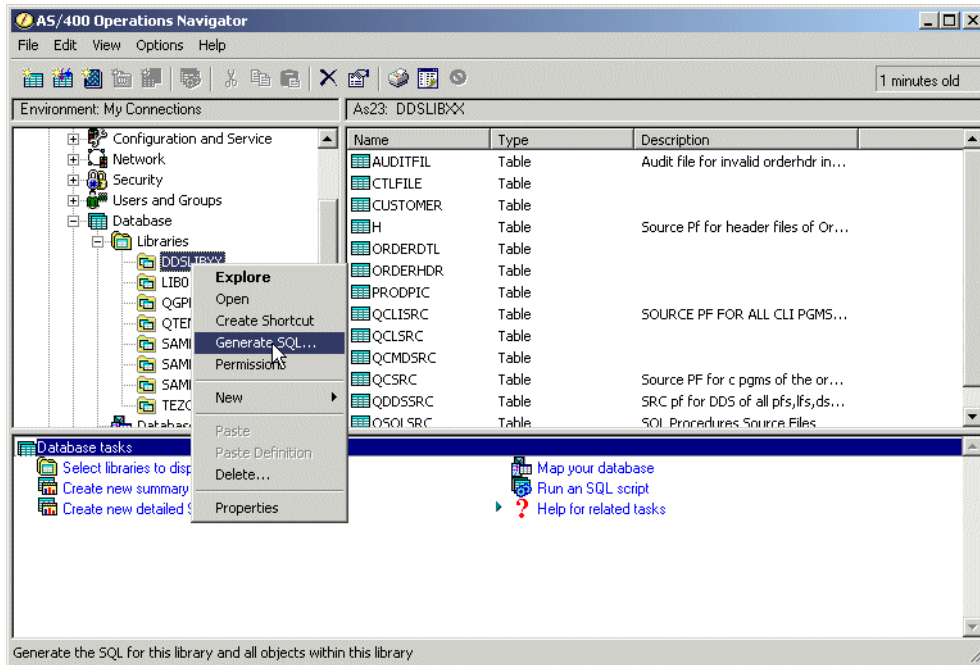


Figure 9-34 Selecting physical files to generate an SQL statement

4. Leave the default options. Click the **Generate** button in the Generate SQL window.
5. The SQL Script Center appears with the generated SQL DDL statements posted in the working area as shown in Figure 9-35.



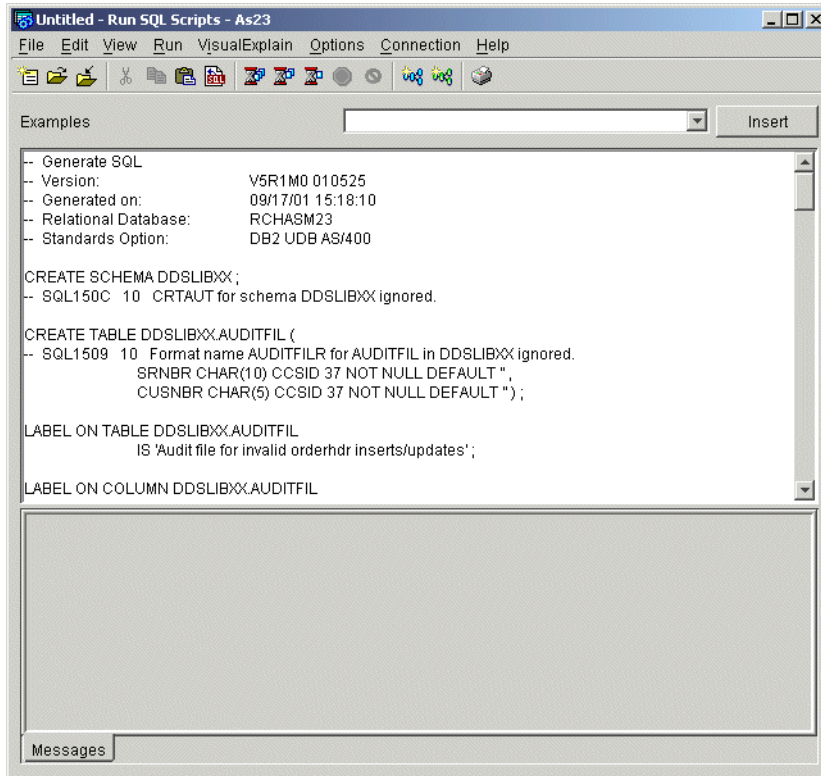


Figure 9-35 Exploring SQL script generated from physical files

**Important:** There are some DDS-specific keywords that *cannot* be converted to SQL. This appears in the code as messages SQL150C and SQL1509 (see Figure 9-35).





## Visual Explain

The launch of DB2 UDB for iSeries Visual Explain with Operations Navigator V4R5M0 was of great interest to database administrators working in an iSeries server environment. The product 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 figure out what is happening. Even better, if you currently have Operations 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 these questions:

- ▶ Where do I find Visual Explain?
- ▶ How do I use it?
- ▶ What can it be used for?
- ▶ Will it tune my SQL queries?
- ▶ What about green-screen queries and those slow running batch jobs?

**Note:** The Visual Explain tool is most effectively used when you have a firm understanding of the DB2 Universal Database for iSeries query optimizer and database engine. The recommended way to obtain this skill and build this understanding is to attend the classroom-based *S6140 - DB2 UDB for iSeries SQL & Query Performance Tuning and Monitoring Workshop* from IBM Learning Services at:  
<http://www-1.ibm.com/servers/eserver/iseries/education/>

## 10.1 A brief history of the database and SQL

If you look back in history, you will find that the database was actually “invented”. It rapidly gained widespread acceptance. So much so, that today, virtually all commercial applications are based on the concepts of a database.

Ever since this invention, programmers have been developing applications to use and maintain these databases in an organization. At the same time, the art of performance tuning databases has evolved.

In high-level languages, programmers optimized access to their database files with keyed access paths. Keyed access was then coded in the program to provide record-level access. Users were not involved at this time.

As standards evolved, databases became larger and were used for diverse purposes, including both transaction-based and data warehousing applications. Structured Query Language (SQL or query) has, at the same time, become the standard for database access.

The scope and power of SQL delivers a standard interface to any database that supports SQL standards. DB2 UDB for iSeries continues to adopt and support the SQL standards.

SQL can be used in pre-written applications on the iSeries server, as well as applications generated by users. Many interrogation tools running on PCs depend on the SQL interface to access data on the iSeries server.

The (anticipated) spread of e-commerce will lead to even more situations where SQL statements are executed on the iSeries server. The need to optimize data access has never been greater. The database is in the public domain now and not reserved only for programmers.

## 10.2 Database tuning so far

General performance tuning will always influence query performance. Therefore, general system usage, competition with other jobs, other queries, amount of memory, processor capacity, processor usage, and so on will always influence the performance of queries.

Assuming that the work environment can be controlled on any given system, the challenge is to apply similar levels of control to the database to optimize the queries. Queries running on the iSeries server are processed through a query optimizer, which creates an access plan based on the information it has available. This access plan includes information about the tables to be accessed and how the query will attempt to access those tables.

By reviewing this access plan, actions can be taken to influence the outcome, and therefore, the performance of the query. These actions can include the creation of indexes to support the query, or can involve changing the way that the query statements are structured to create a more efficient access plan.

### 10.2.1 Query optimizer debug messages

The earliest approach, and probably one of the most widely used, is the analysis of query optimizer debug messages. Running the query under the influence of debug causes the query optimizer to write additional informational messages to the job log.

By looking at the messages in the job log and reviewing the second-level text behind the messages, you can identify changes (for example, creating a new index) that could improve the performance of the query.

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, query optimizer debug messages can be generated in the job log without actually running the query. This means that a query that may take 16 hours to run can be analyzed in a few seconds. Some changes can be made to the query or the database and the effect can be modelled on the query in just a few minutes. The query would then be run when the optimum implementation has been achieved.

## 10.2.2 Database Monitor

More recently, query optimizer debug messages have been joined by the Database Monitor. The Database Monitor gathers query execution statistics from the iSeries server and records them in a database file. This database file is then analyzed to provide performance information to help tune the query or the database.

The Database Monitor is accessed directly from the database component of Operations Navigator. It can also be accessed from a 5250 device using the Start Database Monitor (STRDBMON) command or during the collection of performance data with the STRPFCOL command.

The analysis of the statistics gathered can be done through the SQL Performance Monitors in Operations Navigator. Operations Navigator provides many pre-defined reports to assist with the analysis of the performance data collected in this manner.

## 10.2.3 The PRTSQLINF command

For SQL embedded in program and package objects, the Print SQL Information (PRTSQLINF) CL command extracts the optimizer access method information out of the objects and places that information in a spooled file. The spooled file contents can then be analyzed to determine if any changes are needed to improve performance.

## 10.2.4 Iterative approach

The analysis of queries and tuning for query optimization is an ongoing iterative process. There is no easy solution for query performance and no precise table to which you can refer for the answers. Much depends on a “try it and see” approach.

With this approach, queries are analyzed, and changes are made to the environment. The query is run again, and the environment is adjusted. The process repeats until optimum performance is achieved.

The task of database tuning is complete only when the following statements are all true:

- ▶ Users and programmers are not generating any new queries.
- ▶ All existing queries have been completely tuned.
- ▶ Query selection, sort, and summarization do not change.
- ▶ The iSeries server workload is stable.
- ▶ The volume of data in the tables is stable.
- ▶ The content of the tables is not changing.

## 10.3 Introducing Visual Explain

In Client Access Express, the database component of Operations Navigator is a graphical way to manage the database. Visual Explain has been added to the database component in V4R5M0. Visual Explain provides a graphical way to identify and analyze database performance.

### 10.3.1 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 representing each unique component. Visual Explain also includes information on the database objects 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 you could try any of the changes suggested and see how they are likely to work, before you decide whether to implement them.

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 can be implemented.

### 10.3.2 Finding Visual Explain

Visual Explain is a component of Operations Navigator and is found in the Database section of Operations Navigator. To locate the database section of Operations Navigator, you need to establish a session on your selected iSeries server using the Operations Navigator icon.

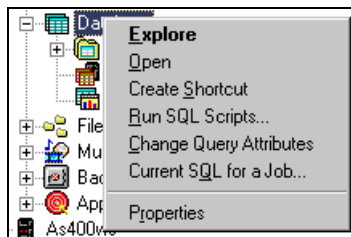


Figure 10-1 Database options under the Database icon

Many functions within Operations Navigator are obtained by right-clicking. For example, you can right-click the Database icon to gain access to several of the query functions (Figure 10-1).

Selecting *Run SQL Scripts* invokes the SQL Script Center. From the SQL Script Center, Visual Explain can be accessed directly, either from the menu or from the toolbar. This is explained in 10.4.1, “The SQL Script Center” on page 306.

Another way to access Visual Explain is through the *SQL Performance Monitor*. The SQL Performance Monitor is used to create Database Performance Monitor data and to analyze the monitor data with pre-defined reports.

Visual Explain works with the monitor data that is collected by the SQL Performance Monitor on that system or by the Database Performance Monitor (STRDBMON), which is discussed in 10.6, “Using Visual Explain with Database Monitor data” on page 318. Visual Explain can also analyze Database Performance Monitor data collected on other systems once that data has been restored on the iSeries server.

### 10.3.3 Data access methods and operations supported

Visual Explain was shipped for the first time in V4R5M0. Table 10-1 shows the methods and operations that are supported by Visual Explain.

Table 10-1 Query access functions supported

Optimizer access plan	Debug	Visual Explain
<b>Non-keyed access methods</b>		
Table Scan	✓	✓
Parallel Table Scan	✓	✓
Parallel Pre-fetch	✓	✓
Parallel Table Pre-load	✓	✓
Skip Sequential with dynamic bitmap	✓	✓
Parallel Skip Sequential	✓	✓
<b>Keyed Data Access Methods</b>		
Key Positioning and Parallel Key Positioning	✓	✓
Dynamic Bitmaps/Index ANDing ORing	✓	✓
Key Selection and Parallel Key Selection	✓	✓
Index-From-Index	✓	✓
Index-Only Access	✓	✓
Parallel Index Pre-load	✓	✓
<b>Joining, Grouping, Ordering</b>		
Nested Loop Join	✓	✓
Hash Join	✓	✓*
Index Grouping	✓	✓
Hash Grouping	✓	✓
Index Ordering	✓	✓
Sort	✓	✓
<b>Query Statements</b>		
Select	✓	✓
Update	✓	✓
Insert	✓	✓*
Delete	✓	✓

Optimizer access plan	Debug	Visual Explain
Sub Query	✓	✓*
Union	✓	✓*
View materialization	✓	✓*
<b>Operational Characteristics</b>		
Index Usage	✓	✓
Index Advice	✓	✓
Open Data Path Usage	✓	✓
Work Management details	✗	✓
<b>Notes:</b> ✓ Supported for analysis with this method. ✓* PTF # XXX is required for V4R5. You need to load the latest Database Group PTF to obtain the best functionality of Visual Explain.		

Each of the methods shown in this table (debug and Visual Explain) provide assistance with the task of debugging queries and the analysis of queries to optimize their performance. None of these will automatically perform the changes for you. The sole purpose is to provide you with the necessary information so you can make an informed choice.

The ease of use that Visual Explain offers can quickly disguise the fact that it is an advanced tool working for you in a highly technical area. Use Visual Explain to assist you with the task of enhancing query performance. Although Visual Explain cannot sort out problems for you, it can help you to identify and solve problems in a more effective way.

You still need to understand the process of query optimization, the different database access plans that can be implemented, and the effects of those plans on the system. You also need to understand the database that you are tuning, its use, and the impact of creating and changing indexes.

## 10.4 Using Visual Explain with the SQL Script Center

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 OS/400 through the use of CL commands.

### 10.4.1 The SQL Script Center

To access the SQL Script Center, right-click the **Database** option in Operations Navigator to see the Database menu. Select **Run SQL Scripts**. The Run SQL Script window appears with the toolbar as shown in Figure 10-2.

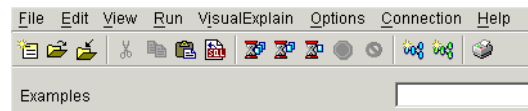


Figure 10-2 Toolbar from the SQL Script Center

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 (V5R1) statements within scripts.



The hour glass icons (green downward arrows in V4R5) indicate to run the statements in the Run SQL Scripts window. These options are also available under the Run menu (Figure 10-3). From left to right, they run all of the statements in the window (All), run all of the statements from the cursor to the end (From Selected), or run the single statement identified by the cursor position (Selected).

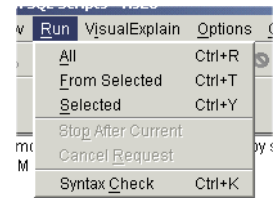


Figure 10-3 SQL Script Center Run options

To the right of the hour glasses in Figure 10-2 is a Stop button, which is colored red when a run is in progress. The final icon in the toolbar is the Print icon.

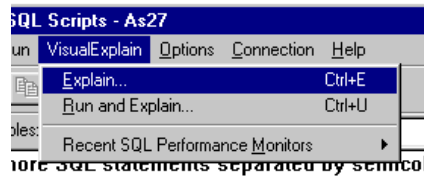


Figure 10-4 SQL Script Center Visual Explain options

This is followed by two Visual Explain icons, colored blue and green. The left Visual Explain icon (blue) is to explain the SQL statement. The right Visual Explain icon (green) is to run and explain the SQL statement. The actions that you will choose are explained in a moment. Both of these options are also available on the drop-down menu (Figure 10-4). You may choose either option to start Visual Explain.

Another option exists on the Visual Explain pull-down menu to show recent SQL Performance Monitors. SQL Performance Monitors can be used to record SQL statements that are explainable by Visual Explain. We recommend access via the SQL Performance Monitors icon, because this provides the full list of monitors.

An SQL script is defined as one or more statements from the Run SQL Script working area below the toolbar. An initial comment is provided. Each complete statement needs a delimiter to mark the end of statement. The SQL Script Center uses a semi-colon (;) for this purpose.

## 10.4.2 Visual 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. See Figure 10-5.



Figure 10-5 Visual Explain access

To optimize an SQL statement, the optimizer validates the statement and then 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 choosing Visual Explain Only, Visual Explain may not be able to explain some complex queries such as hash join, temp join results, etc. In this cases, users have to choose Run and Explain for the SQL statements to see the graphical representation.

## 10.4.3 Run and Explain

The Run and Explain option (Ctrl+U or the green toolbar icon) also 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. 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.

**Notes:**

- ▶ Visual Explain may show a representation that is different from the job or environment where the actual statement was run since it may 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 Run and Explain option.

## 10.5 Navigating Visual Explain

The Visual Explain graphics window (Figure 10-6) is presented in two parts. The left-hand 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-hand side of the display has the Query Attributes and Values. The display corresponds to the object that has been selected on the graph. Initially, 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.

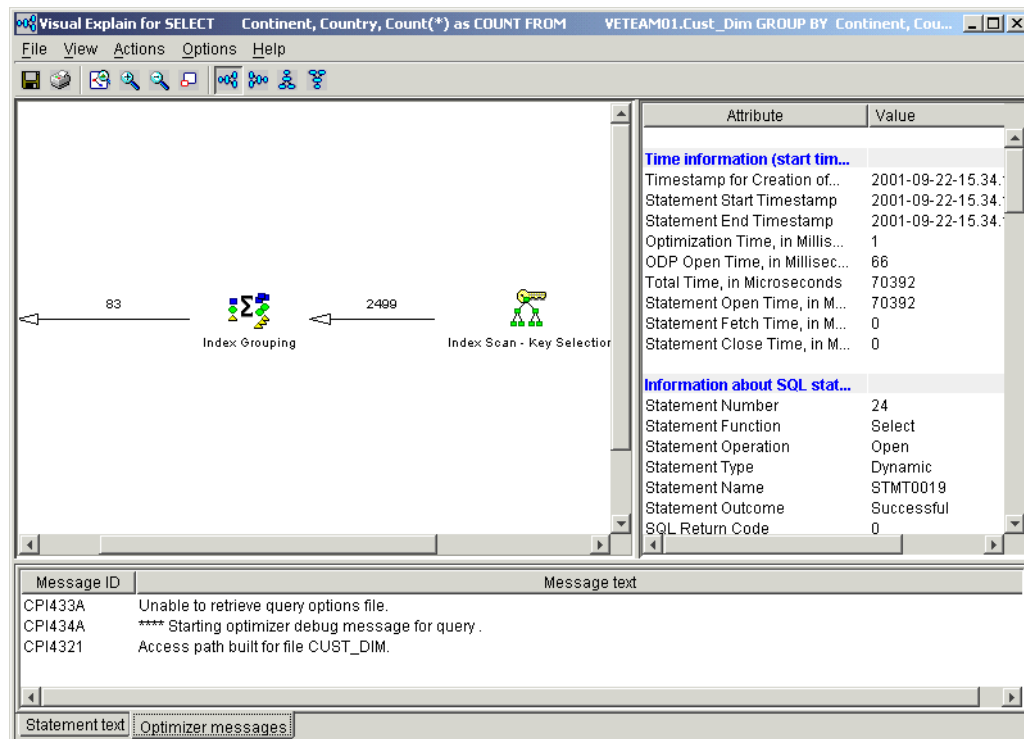


Figure 10-6 Visual Explain Query Implementation Graph and Query Attributes and Values

The default settings cause the display to be presented with the final result icon (a checkered flag) on the left of the display. Each of the icons 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 present the details that are known to the query for that part of the implementation. You may find it helpful to adjust the display to see more of the attributes and values. Query attributes and values are discussed further in 10.5.5, “Visual Explain query attributes and values” on page 315.

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 and can provide a short cut to table information to be shown in a separate window. More details are shown in 10.5.2, “Action menu items” on page 310.

The following action menu items may be found selectively on different icons:

- ▶ **Table Description:** Displays table information returned by Display File Description (DSPFD).
- ▶ **Index Description:** Displays index information returned by DSPFD.
- ▶ **Create Index:** Creates a permanent index on the iSeries server.
- ▶ **Table Properties:** Displays object properties.
- ▶ **Index Properties:** Displays object properties.
- ▶ **Display Query Environment:** Displays environment settings used during the processing of this query.
- ▶ **Additional fly-over panels:** These exist for many of the icons. By moving the mouse pointer over the icon, a window appears with summary information on the specific operation. See Figure 10-7.

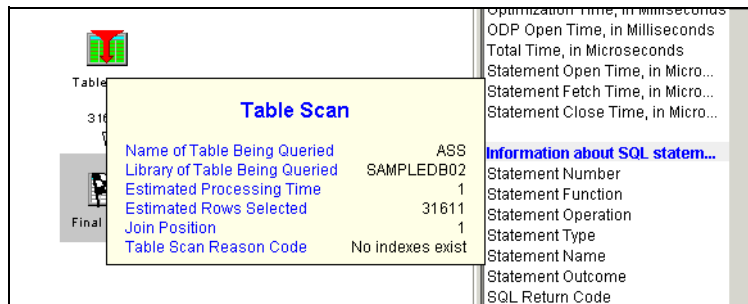


Figure 10-7 Table scan fly-over panel

The Visual Explain toolbar (Figure 10-8) helps you navigate the displays. The first four icons (from left to right) help you 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.

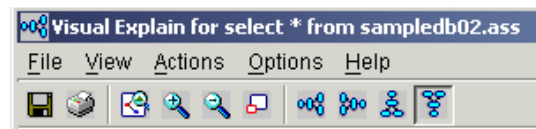


Figure 10-8 Visual Explain toolbar

The fourth icon (Overview) creates an additional window Figure 10-9 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.

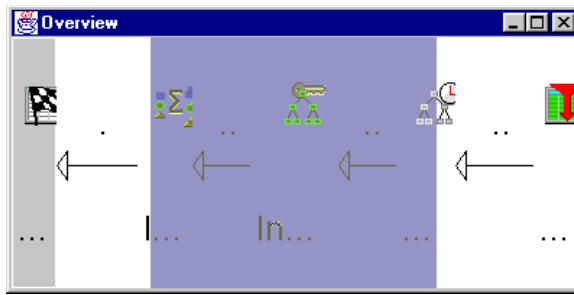


Figure 10-9 Visual Explain Overview window

In the Overview window (Figure 10-9), 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.

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 back. The

remaining 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.

## 10.5.1 Menu options

The menu options above the toolbar icons are File, View, Actions, and Help. The File option allows you to close the window. Starting on V5R1, the ability to either print or save the Visual Explain output as an SQL Performance Monitor file was added. The View options generally replicate the toolbar icons. The additional options are:

- ▶ Icon spacing (horizontal or vertical) changes the size of the arrows between the icons.
- ▶ Arrow labels allow you to show/hide the estimated number of rows that the query is processing at each stage of the implementation.
- ▶ Icon labels allow you to show/hide the description of the icons.
- ▶ Highlight expensive icons (new in V5R1) by number of returned rows.
- ▶ Highlight advised indexes (new in V5R1).

The Actions menu item replicates the features that are available on the display.

## 10.5.2 Action menu items

When you right-click a query implementation icon, a menu appears that offers further options. These options may include one of more of the following items.

### Table Description

The Table Description menu item (Figure 10-10) takes you into the graphical equivalent of the Display File Description (DSPFD) command. From here, you can find out 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.

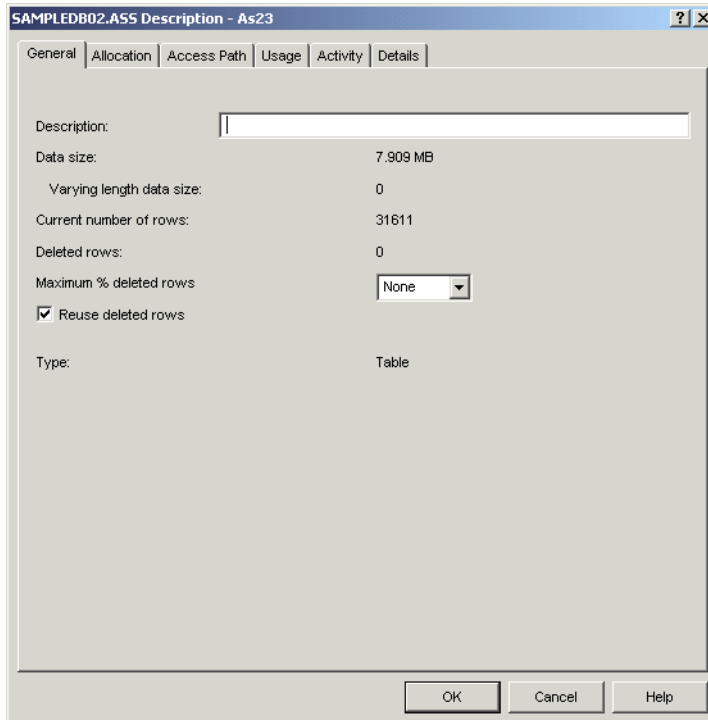


Figure 10-10 Table Description

## Table Properties

The Table Properties display (Figure 10-11) shows a list of the columns and their attributes from the table icons. A limited number of changes are allowed from the window.

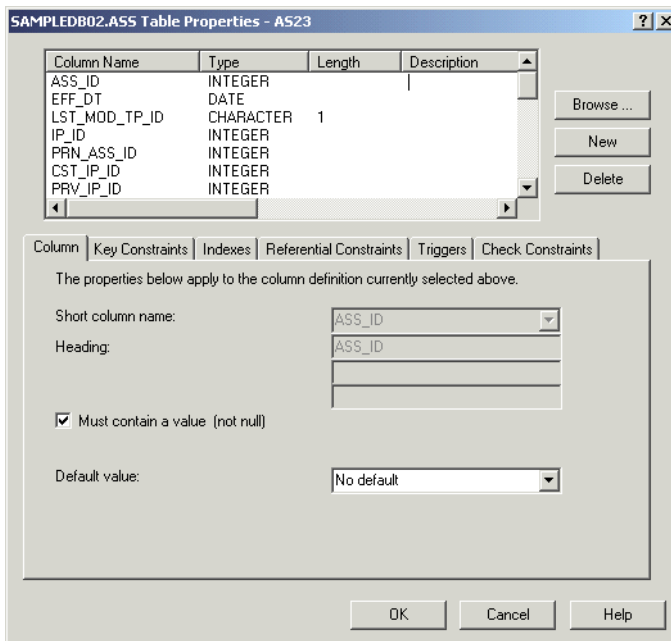


Figure 10-11 Table Properties

## 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. The Index Description display is shown in Figure 10-12.

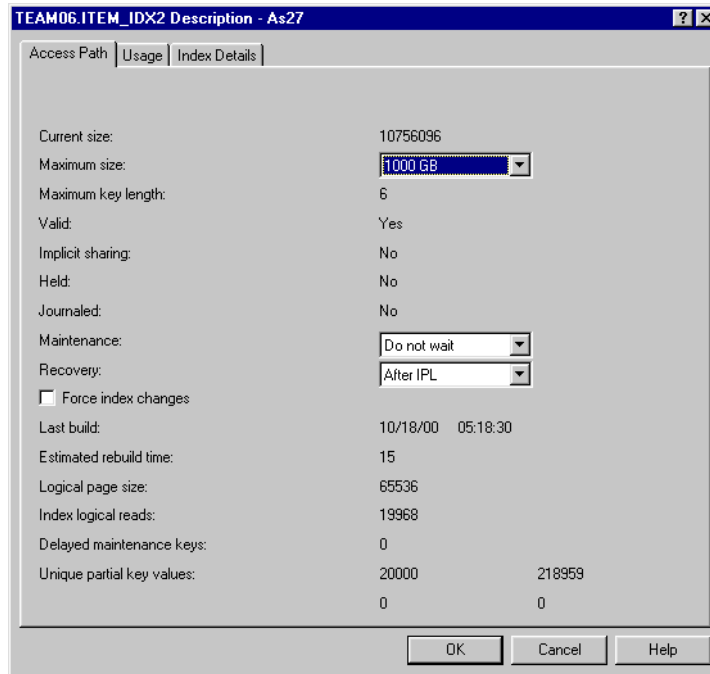


Figure 10-12 Index Description

## Index Properties

The Index Properties window (Figure 10-13) 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.

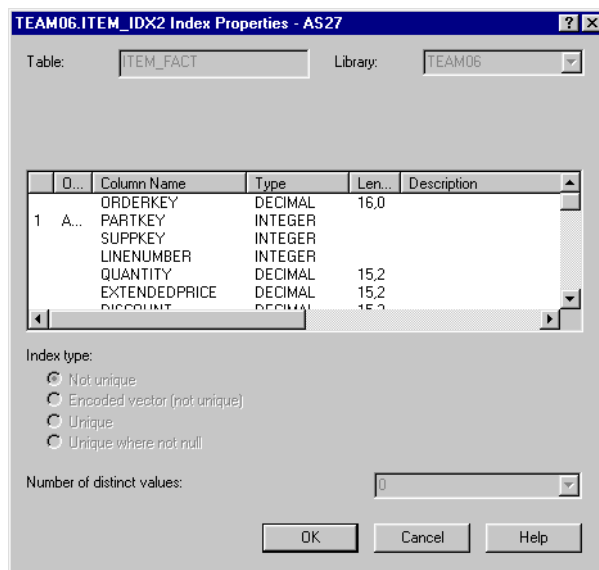


Figure 10-13 Index Properties

## Create Index

From the temporary index icon, the Create Index menu item takes you to a dialogue box where the attributes of the temporary index have been completed (Figure 10-14). Simply click OK to create a permanent index that mirrors the temporary index created by the query.

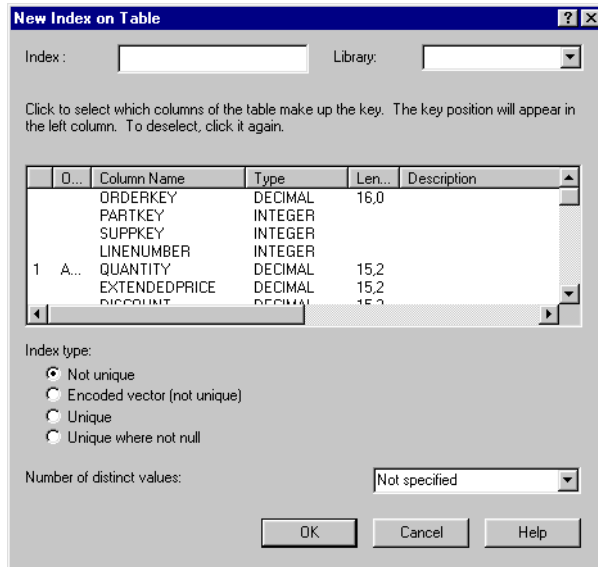


Figure 10-14 New Index on Table display

You need to enter an index 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 user-friendly features of Visual Explain, which gives you the ability to easily create an index that the optimizer has suggested.

### 10.5.3 Controlling diagram level of detail

Starting on V5R1, users can select how much detail they want to see on the Visual Explain graphs. The VISUAL\_EXPLAIN\_DIAGRAM row on the QAQQINI file lets you change the level of detail in Visual Explain. When it is set to \*BASIC or \*DEFAULT, it shows only the icons directly related to the query. When it is set to \*DETAIL, it also shows the icons that are indirectly related to the query, such as table scans performed to build temporary indexes. Figure 10-15 shows these two versions of explanation for the same sample query. Most users will be satisfied with the \*BASIC diagram while others, with more performance tuning experience, may prefer the \*DETAIL diagram.

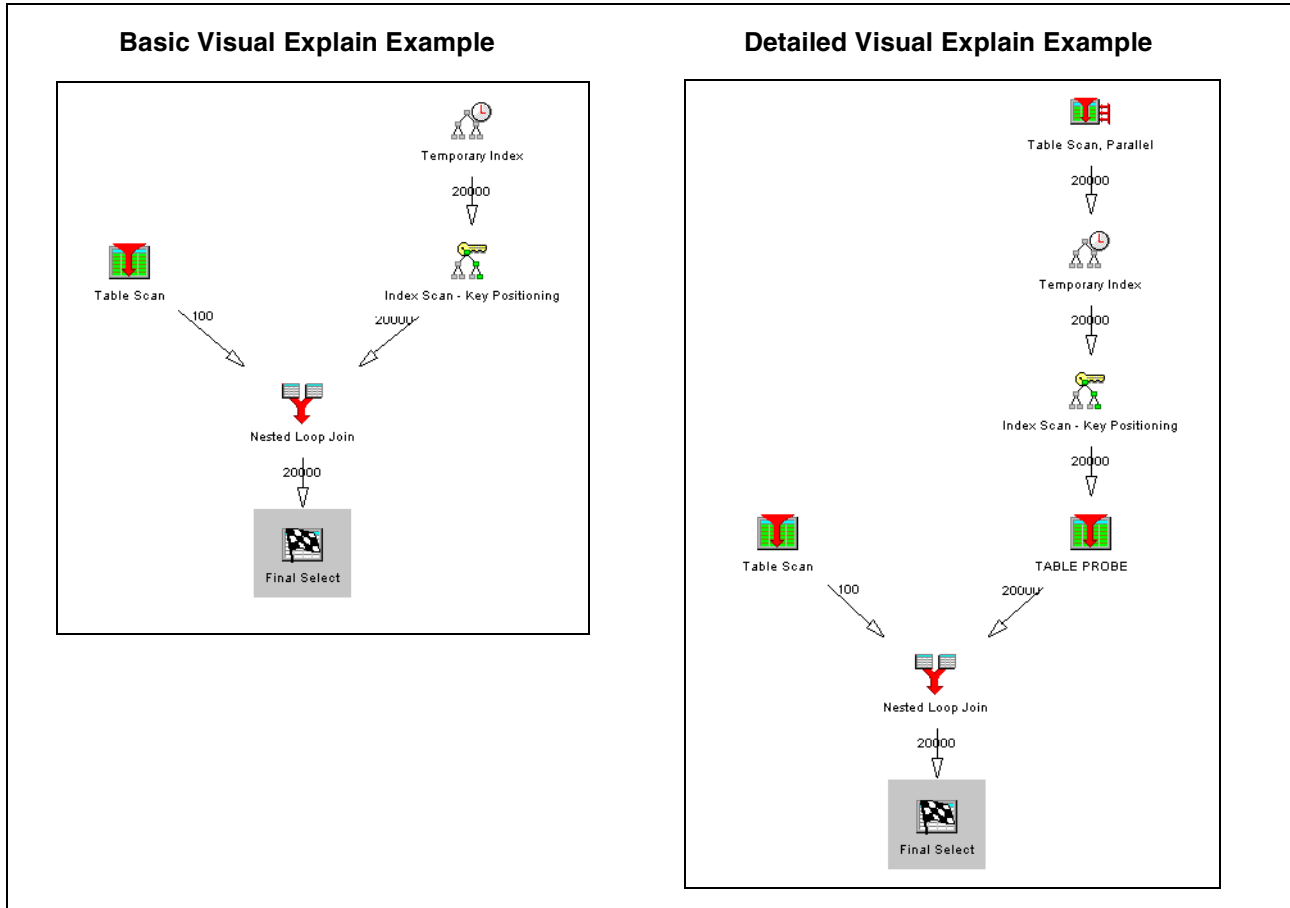


Figure 10-15 Basic and detailed Visual Explain comparison

In Figure 10-15, the table scan on the upper right of the detailed graph is part of the creation of the temporary index. Some other differences between \*BASIC and \*DETAIL are:

- ▶ If the single table query uses key positioning, key selection, and table scan, \*DETAIL would show an icon for key positioning, key selection, and table scan. \*BASIC would show only one icon – key positioning.
- ▶ If the single table query uses key positioning, \*DETAIL shows two icons – key positioning and table probe. \*BASIC would show only the key positioning icon.

### 10.5.4 Displaying the query environment

The query environment is available as a fast path from the Final Results icon and shows the work management environment (Figure 10-16) where the query was executed. This information can also be obtained from the Query Attributes and Values displays.



Attribute	Value
Memory Pool Size	18625142
Memory Pool ID	2
Date Format	ISO
Date Separator	-
Time Format	ISO
Time Separator	:
Decimal Point	.
Sort Sequence Table	*HEX
Sort Sequence Library	
Language ID	ENU
Query Options Table	
Query Options Library	*N
Query Time Limit	0
Parallel Degree Setting	NONE
Maximum Number of Tasks	0
Parameter Marker Conversion	N

Figure 10-16 Environment

### 10.5.5 Visual Explain query attributes and values

The query attributes and values show further 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

This section shows the name and library of the table being selected (Figure 10-17). If the table name is a long name (MASTERCUSTOMER), the name of the table being queried and the member of the table will be the short name (MASTE00001). The long name is in a separate line titled “Long Name of the Table being queried”.

Attribute	Value
<b>Table name, base table name, i...</b>	
Name of Table Being Queried	ACT
Library of Table Being Queried	SAMPLEDB02
Member of Table Being Queried	ACT
Name of Base Table	ACT
Library of Base Table	SAMPLEDB02
Member of Base Table	ACT
Name of Index Used	XIF12ACT
Library of Index Used	SAMPLEDB02
Member of Index Used	XIF12ACT

Figure 10-17 Table name

#### Estimated processing time and table info

The estimated processing time (Figure 10-18) shows the time the optimizer expects to take from this part of the query.

<b>Estimated processing time and ...</b>	
Estimated Processing Time	1
Estimated Cumulative Time	1
Total Rows in Table	49592
Table Size	22036480

Figure 10-18 Estimated processing time

## Estimated rows selected and query join info

The estimated rows selected (Figure 10-19) shows the number of rows 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 actually shows the number of rows selected. It also shows whether the query is CPU or I/O bound, which is information that was not accessible prior to Visual Explain.

Estimated rows selected and q...	
Estimated Rows Selected	1
Join Position	1
Table Data Space Number	1
Number of Tables Joined	1
I/O or CPU Bound	CPU Bound

Figure 10-19 Estimated rows selected

Queries can be very 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 SMP is installed on a multi-processor system, you should review the DEGREE parameter to ensure that you are using the systems resources effectively.

## Information about the index scan performed

This display shown in Figure 10-20 provides the essentials 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 the description of the different reason codes, refer to the manual *DB2 UDB for iSeries Database Performance and Query Optimization*.

Information about the index scan performed	
Index Scan - Key Positioning	Yes
Number of Columns Using Key Positioning	1
Estimated Entries Using Key Positioning	1
Index Scan - Key Selection	No
Index Only Access	No
Index Fits in Memory	Yes
Memory Pool Size	20143182
Memory Pool ID	2
Type of Index	Binary Radix
Index Usage	Primary Index
Number of Index Entries	49592
Size of Index, in Bytes	1318912
Page Size of Index, in Bytes	4096
Reason Code	Row selection
Index is a Constraint	No

Figure 10-20 Index scan

## SMP parallel information

The SMP information (Figure 10-21) shows the degree of parallelism that occurred on this particular step. It may appear for more than one icon, because multiple steps can be processed with differing degrees of parallelism. The display also shows whether either parallel pre-fetch or parallel pre-load was used as part of the parallel processing.

This information is only relevant when the DB2 SMP licensed feature is installed. The parallel degree requested is the number of parallel tasks that the optimizer used. This is a user setting defined with CHGQRYA, but the optimizer adjusts it based on the system resources.

SMP parallel information	
Parallel Pre-Fetch	No
Parallel Pre-Load	No
Parallel Degree Requested	2

Figure 10-21 SMP parallel information

## Index advised information

The Index advised section (Figure 10-22) 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, like base table name, library, and involved columns, will be easily identifiable, as shown in the Figure 10-22.

Index advised information	
Creation of an Index is Advised	Yes
Number of Primary Key Columns	2
List of Key Columns for Advised...	YEAR, MONTH
Type of Index Created	Binary Radix
Number of Unique Index Values	Not Available
ACS Table Name	*HEX
ACS Table Library	*N

Figure 10-22 Index advised

Note that it is possible for the query optimizer to not use the suggested index, if created. This suggestion is generated if the optimizer determines that a new index might improve the performance of the selected data by 1 microsecond.

## Information about temporary index created

This display provides information about the creation of a temporary index as part of the query optimizer implementation (Figure 10-23). The index created is reusable and specifies if a temporary index creation is allowing the associated ODP to be used. If the key column field names of the index are missing, this implies that derived fields were used.

Information about temporary index created	
Entries in Index Created	100029
Page Size of Index Created	65536
Row Size of Index Created	8
Alternate Collating Sequence Table	No
ACS Table Name	*HEX
ACS Table Library	
Index Created is Reusable	Yes
Index Created is Sparse Index	No
Type of Index Created	Binary Radix
Number of Unique Index Values	Not Available
Index Created is Permanent Object	No
Index Created From Existing Index	No
Parallel Degree Requested, New Index	0
Reason Code for Index Created	Ordering or Grouping
Key Columns of Index Created	QUANTITY ASCEND

Figure 10-23 Temporary index

## Additional information about SQL statement

The display in Figure 10-24 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 ODBC/JDBC driver settings.

The Statement is Explainable specifies if the SQL statement can be explained by the Visual Explain tool. In V4R5, not all statements are explainable. In this section, you will find the SQL statement if you selected the Final Select icon.

Additional information about S...	
CLOSECSR Value	
ALWCPYDTA Value	Optimize, optimizer may chose ...
Pseudo Open	No
Pseudo Close	No
Hard Close Reason Code	Not Available
ODP Implementation	Reusable
Dynamic Replan Reason Code	Access plan was not rebuilt
Dynamic Replan Subtype Code	
Timestamp of Last Replan	0001-01-01-00.00.00.000000
Parse Required	No
Data Conversion Reason Code	Not applicable
Level of Commitment Control	NC
Blocking Enabled	L
Delay Prep	No
Statement is Explainable	Yes
Naming Convention	SQL
Type of Dynamic Processing	System Wide Cache
Optimize LOB Data Types	Yes
User Profile for Compiled Prog...	Determined by Naming Convent...
User Profile for Dynamic	USRPRF(*USER)
Default Collection	ORDAPPLIB
Procedure Name on CALL	
Procedure Library on CALL	
SQL Path	*LIBL

Figure 10-24 Additional information

## Implementation summary for SQL statement

The Implementation summary for SQL statement (Figure 10-25) provides information about the type of SQL statement being processed. It also identifies functions that have a particular influence on the optimizer. It specifies the values of the variables used on the SQL statement (Host Variable Values). If the SQL Statement has an Order By, Group By, or a Join, it specifies which implementation was used. In the example in Figure 10-25, the SQL statement did not have an Order By, Group By, or join operation.

Implementation summary for SQL statement	
Host Variable Values	2000, 10
Host Variable Implementation	ISV, Interface supplied values
Ordering Implementation	Not Available
Grouping Implementation	Not Available
Join Implementation	Not Available
Union	No
Subquery	No
Distinct Query	No
Distributed Query	No
Implementation Summary	

Figure 10-25 Implementation summary

## 10.6 Using Visual Explain with Database Monitor data

Database Monitor data is query information that has been recorded by one of the DB2 UDB for iSeries performance monitors into a database table that can be analyzed later. Multiple Database Performance Monitors may 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.

The Database Performance Monitor can be started from Operations Navigator or with a CL command. With Operations 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 Database Performance Monitor in Operations Navigator.

The Start Database Monitor (STRDBMON) or Start Performance Monitor (STRPFRMON) (with STRDBMON(\*YES)) CL commands can also be used to collect Database Performance Monitor data. If you intend to use Visual Explain on the Database Monitor data collected with these CL commands, the data must be imported into Operations Navigator as detailed data. See 7.6, “SQL Performance Monitors” on page 220, for a detailed explanation on how to use SQL Performance Monitor and how to import DBMON data into Operations Navigator.

## Using Visual Explain

Click **Operations Navigator-> Database-> SQL Performance Monitors** to obtain a list of the SQL Performance Monitors that are currently on the system.

Right-click the **Performance Monitor**, and select **List Explainable Statements**. An “explainable” statement (Figure 10-26) is an SQL statement that can be explained by Visual Explain. Because Visual Explain does not process all SQL statements, it is possible that some statements will not be selected.

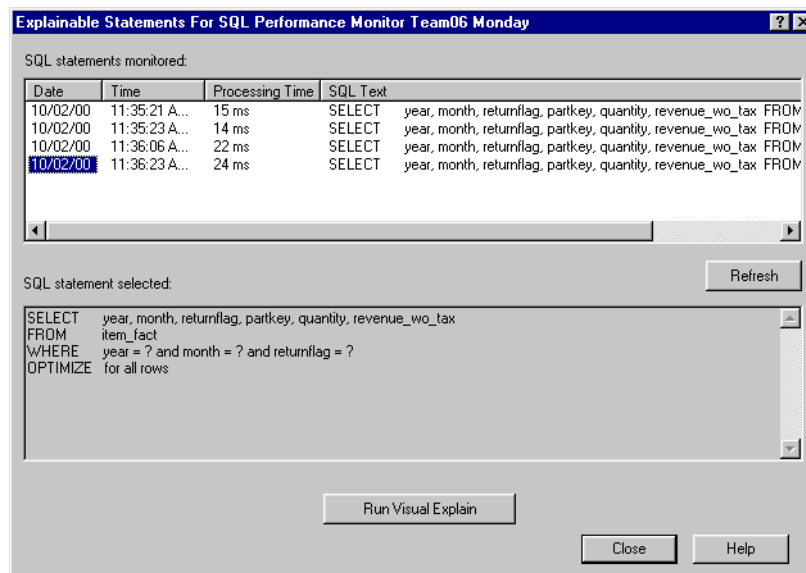


Figure 10-26 SQL explainable statements

The explainable SQL statements that have been optimized by the job are now listed. If you have been monitoring an SQL Script window, these will be the SQL statements that were entered.

**Note:** Query optimizer information is only generated for an SQL statement or query request when an ODP is created. When an SQL or query request is implemented with a Reusable ODP, then the query optimizer is not invoked. Therefore, there will be no feedback from the query optimizer in terms of monitor data or even debug messages and the statement will not be 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 10-26) 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 may either retain the performance data or remove it from the system at this time, depending on your requirements.

## 10.7 Non-SQL interface considerations

Obviously, the Database 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.

Some query interfaces on the AS/400 and iSeries servers are not SQL-based and, therefore, are not supported by Visual Explain. The interfaces not supported by Visual Explain include:

- ▶ Native database access from a high level language, such as Cobol, RPG, etc.
- ▶ Query
- ▶ OPNQRYF command
- ▶ OS/400 Create Query API (QQQQRY)

The query optimizer creates an access plan for all queries that run on the iSeries server. Most queries use the SQL interface, and generate an SQL statement, either directly (SQL Script Window, STRSQL command, SQL in high-level language (HLL) programs) or indirectly (Query Monitor/400).

Other queries do not generate identifiable SQL statements (Query, 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.

The statements that generate SQL and can be used with the Visual Explain via the SQL Performance Monitor include:

- ▶ SQL statements from the SQL Script Center
- ▶ SQL statements from the Start SQL (STRSQL) command
- ▶ SQL statements processed by the Run SQL Statement (RUNSQLSTM) command
- ▶ SQL statements embedded into a high level language program, such as Cobol, Java, or RPG
- ▶ SQL statements processed through an ODBC or JDBC interface

The statements that do not generate SQL and, therefore, cannot be used with Visual Explain via the SQL Performance Monitor include:

- ▶ Native database access from a high level language, for example, Cobol, RPG, etc.
- ▶ Query
- ▶ Open Query File (OPNQRYF) command
- ▶ OS/400 Create Query API (QQQQRY)

### 10.7.1 Query/400 and Visual Explain

Query/400, now renamed Query, is not supported by Visual Explain even though optimizer debug messages can be used with Query/400 queries since it does not generate SQL. Query/400 queries are often blamed for poor performance and sometimes even banned from execution during daylight hours. It is for this reason that some guidance has been provided to bring Query/400 queries into the scope of Visual Explain.

There is no direct Query/400 to SQL command. However, the Start Query Monitor Query (STRQMQR) CL command will run a query definition (object type \*QRYDFN) as an SQL statement, as long as the ALWQRYDFN parameter is set to either \*YES or \*ONLY.

If you are accessing a multi-member file, performance data is not collected for the second and subsequent members. Instead, you need to use an SQL supported interface, such as an alias for the members.

To use this SQL statement with Visual Explain, either start an SQL Performance Monitor for this job in advance of issuing the STRQMQR command, or use the native STRDBMON CL command to collect data for the job. See 7.6.1, “Starting the SQL Performance Monitor” on page 222.

## 10.7.2 The Visual Explain icons

The icons that you may encounter on the Visual Explain query implementation chart are shown here.



The *Final Result* icon displays the original SQL statement and summary information of how the query was implemented. It is the last icon on the chart.



The *Table Scan* icon indicates that all rows in the table were paged in, and selection criteria was applied against each row. Only those rows meeting the selection criteria were retrieved. To obtain the result in a particular sequence, you must specify the ORDER BY clause.



The *Parallel Table Scan* 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 would be 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 usually is related to the Dynamic Bitmap or Bitmap Merge icons.



The *Skip Sequential Parallel Table 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 provide for faster scans.



The *Parallel Encoded Vector Index* icon indicates that multiple tasks were used to perform the encoded vector index selection in parallel. This allows for faster scans that can be more easily processed in parallel.



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 *Incomplete Information* icon indicates that a query could not be displayed due to incomplete information.

---

## 10.8 SQL performance analysis using Visual Explain

This section presents a brief example on SQL performance analysis using Visual Explain. A complete explanation on performance analysis is beyond the scope of this redbook, but you can find extensive information on Redpapers and workshops at:

<http://www-1.ibm.com/servers/eserver/iserries/library/>

### 10.8.1 Database performance analysis methodology

There are many different methods to identify problems and tune troublesome database queries. One of the most common methods is to identify the most dominating, time-consuming queries and work on each of them individually. Another method is to leverage global information and to use this information to look for indexes that are “begging” to be created.

Operations Navigator SQL Performance Monitor provides you with tools for gathering and analyze SQL performance information. Once you have SQL performance data collected, you can use the predefined queries for looking for specific queries that have large table scans or that are evidencing some lack of indexes.

Those predefined queries can be reached by right-clicking the specific SQL Performance Monitor collected and selecting Analyze Results as shown in Figure 10-27.

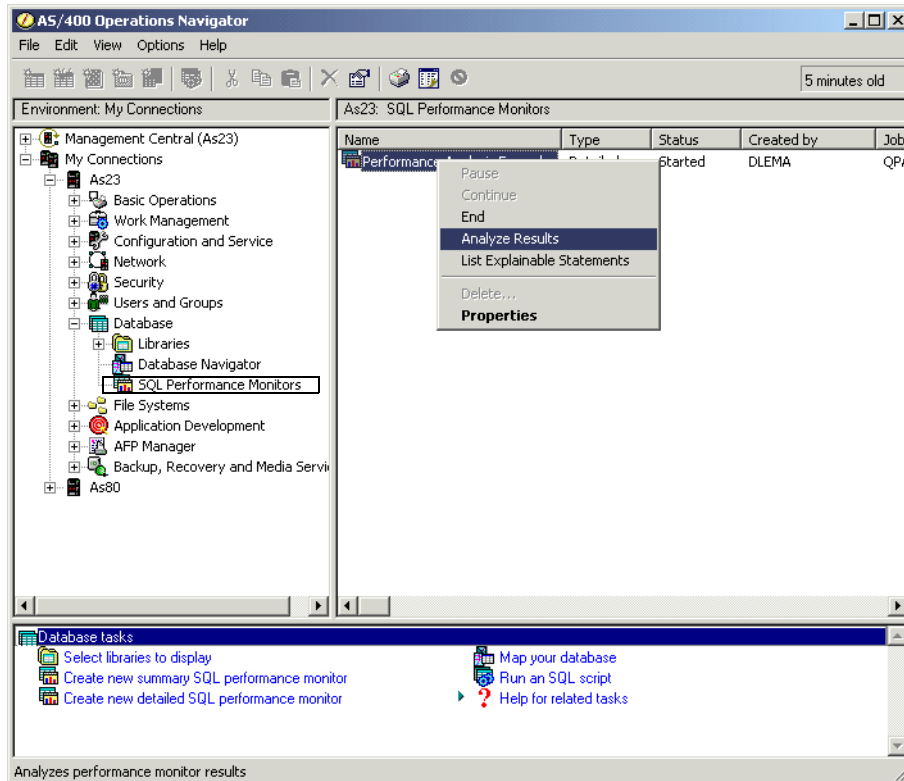


Figure 10-27 Analyzing SQL performance results

The Basic Statement Information predefined query gives you a very general idea of the queries being monitored, as well as the kind of access methods used by these queries. This reports provides you information related to execution time per each execution, total execution time, advised indexes, whether table scan or temporary index creation was used, and more.

Once you detect a query or set of queries that needs further analysis, you can use a detailed query analysis tool like Visual Explain to explore them in detail. Query analysis is iterative in nature. Try something running the job or the individual query to see if it worked. Try it again if it did not work. You can explain with Visual Explain the SQL statements contained in the SQL Performance Monitor Collected data by right-clicking the specific collection and selecting List Explainable Statements from the pop-up menu. A list of explainable statements appears and you can choose those in which you are interested.

As an example, Figure 10-28 shows a Visual Explain diagram that permits you to detect that for this query. It is performing a table scan and is not using parallelism. You can see that the SQL statement does not specify an OPTIMIZE FOR n ROWS portion, and the query degree is set to \*NONE.

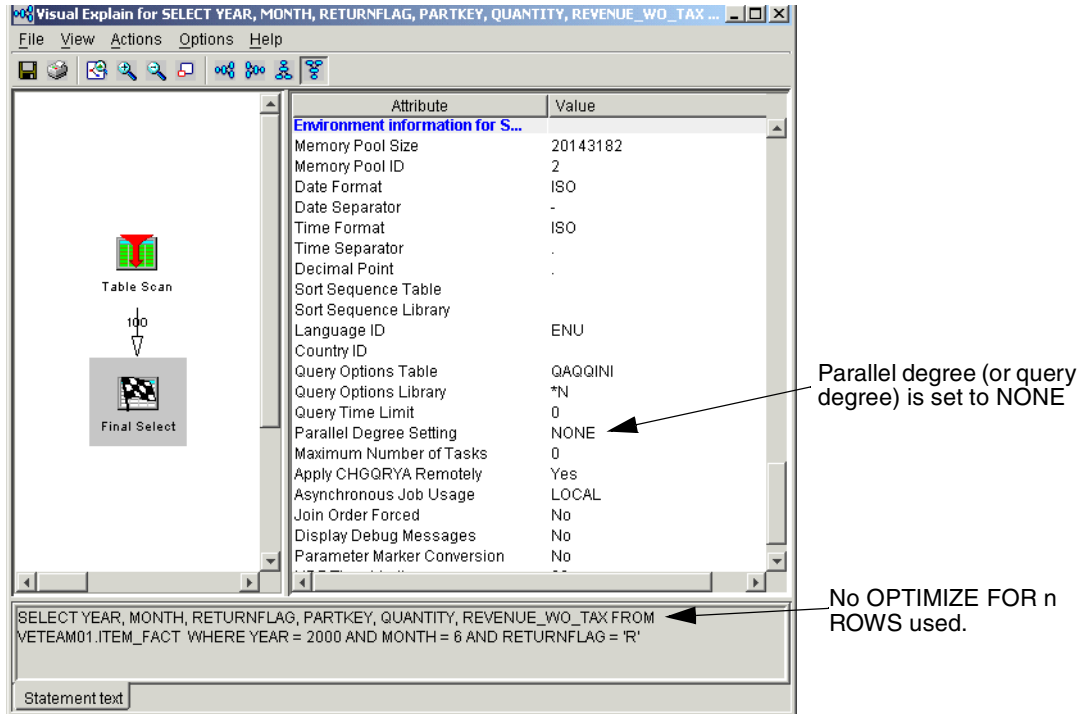


Figure 10-28 Analyzing a simple query: First iteration

Based on the information provided by Visual Explain, you change the statement including now an OPTIMIZE FOR ALL ROWS, and we change the parallel degree to \*OPTIMIZE. See Figure 10-29.

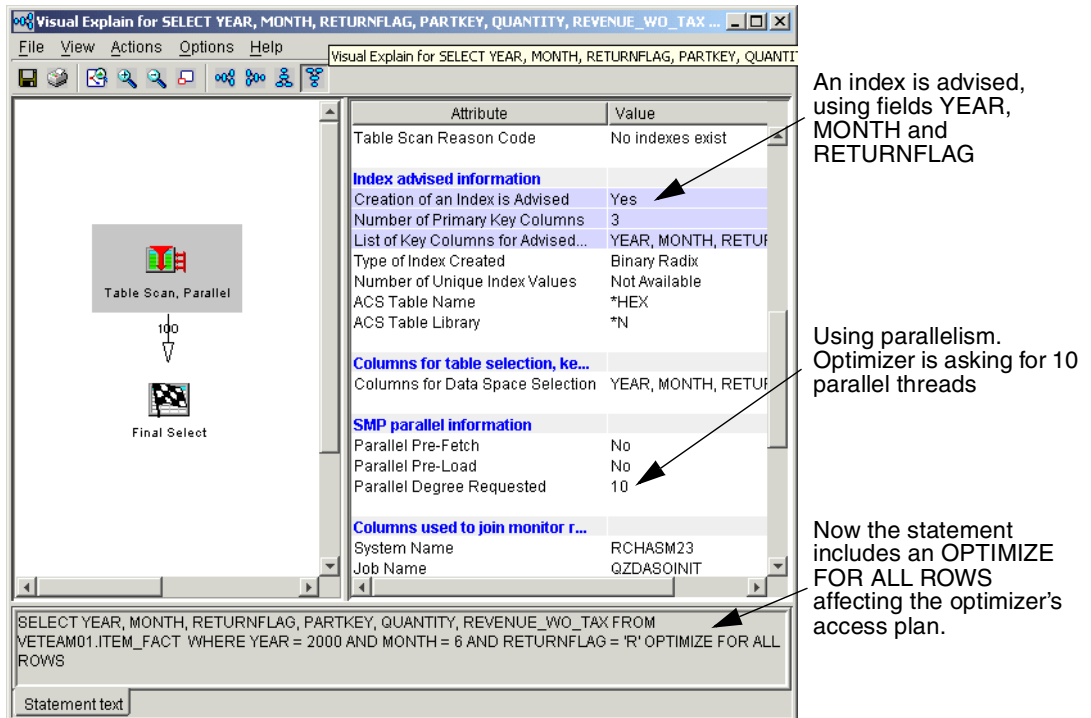


Figure 10-29 Analyzing a simple query: Second iteration

You can see the affect that the changes had over the analyzed query. You can go further and create the suggested index by right-clicking the Table Scan icon and selecting the Create Index option. A New Index on Table window appears (Figure 10-30), where the suggested fields are selected for you. You have to provide a name and library for the new index. You can also change the order of the fields and add new fields to the index if you consider that necessary.

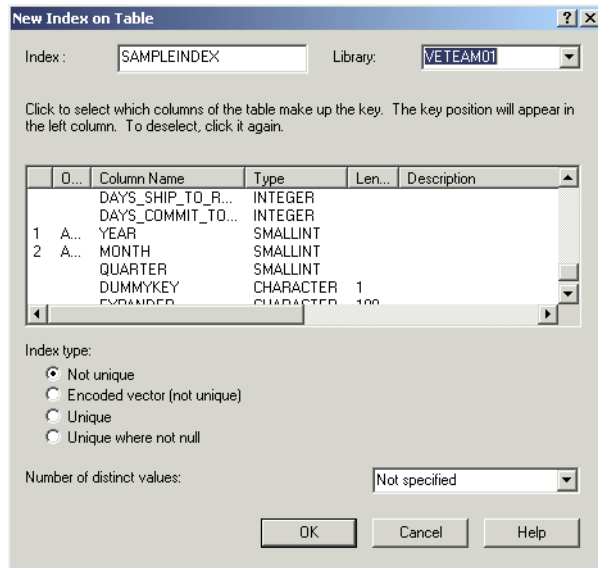


Figure 10-30 New Index on Table window

Now DB2 UDB for iSeries uses the suggested index, as shown in Figure 10-31. Note that it is possible that DB2 UDB for iSeries may not use the suggested index.

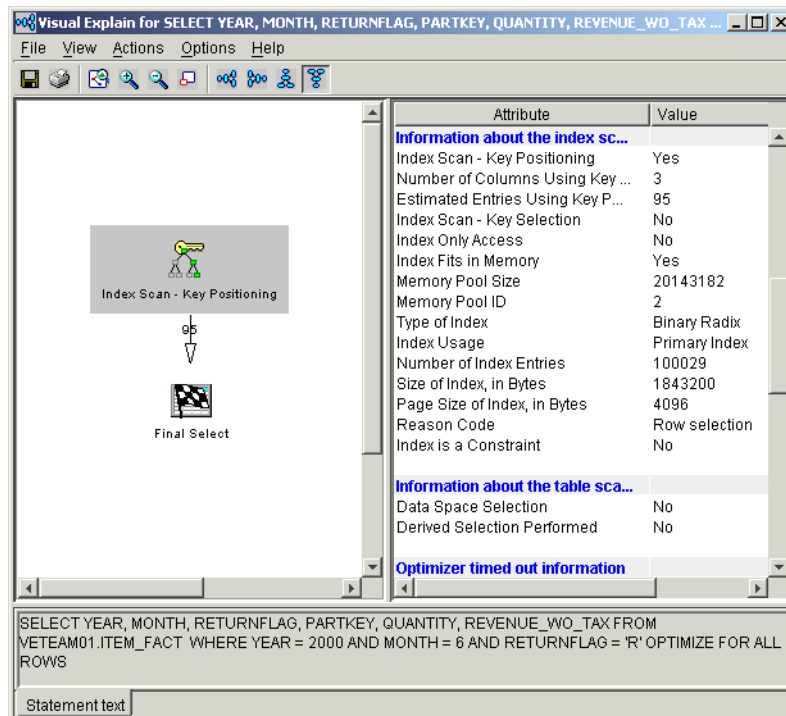


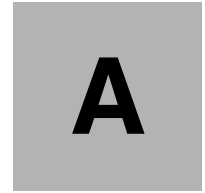
Figure 10-31 Analyzing a simple query: Third iteration

Is it really that simple? Tuning SQL statements and database performance can be a very demanding task, but with the new tools introduced in V4R5 and improved in V5R1, such as Visual Explain and SQL Performance Monitor predefined reports, it is becoming more accessible.

Performance tuning, particularly when dealing with database operations, is an iterative process but the availability and knowledge of powerful tools allow the performance analyst to find a solution quickly.

Knowledge and judicious usage of the OS/400 Database Monitor tool, its predefined queries, and particularly Visual Explain reduces significantly the time and effort required by performance analysts.





# **Order Entry application: Detailed flow**

This appendix provides detailed flow charts of each of the modules included in the Order Entry application scenario.

## Program flow for the Insert Order Header program

Figure A-1 shows a functional description of the various components of this application scenario. The DB2 UDB for iSeries functional highlights in this program include:

- ▶ Referential integrity constraints for the Order Header table
- ▶ Insert trigger on the Order Header file

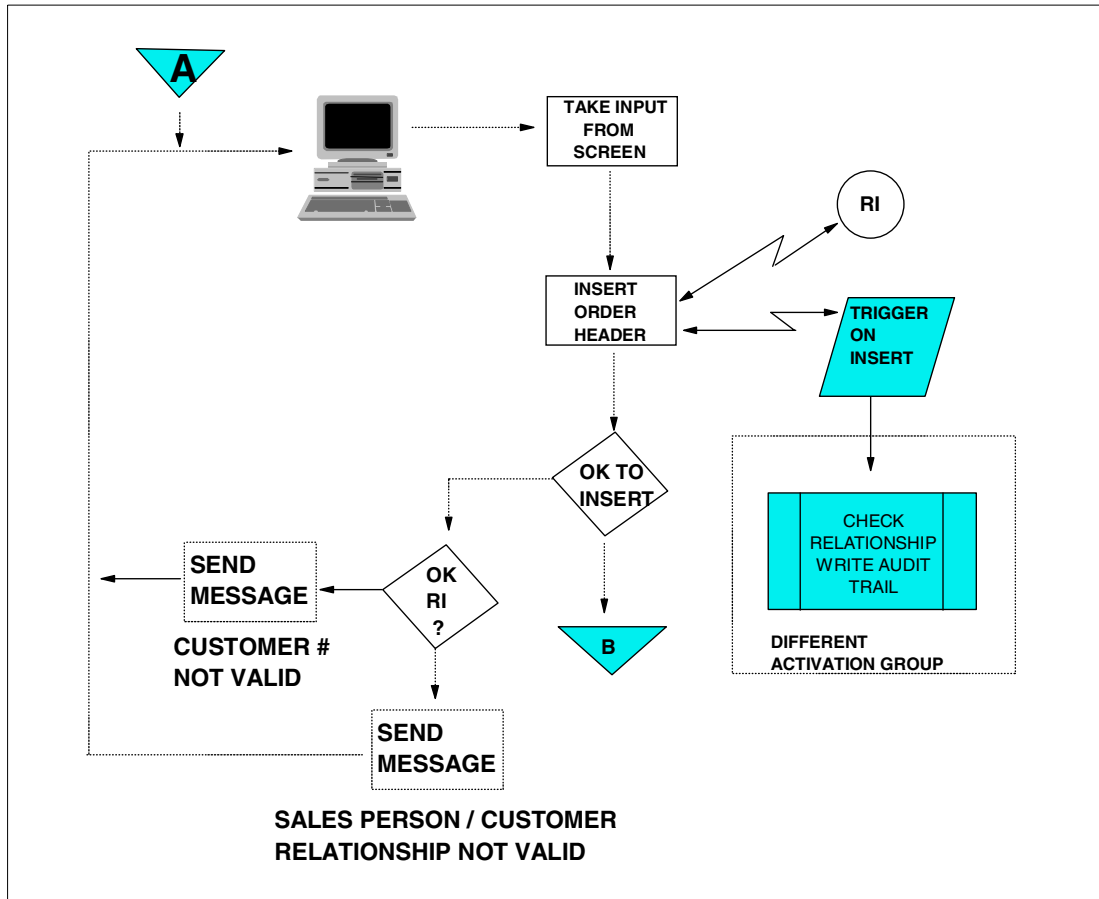


Figure A-1 Insert Order Header program flow

## Program description for the Insert Order Header program

The idea of this program is to show how to use the following new database functions in a real application:

- ▶ **Referential integrity:** When a record is inserted in the Order Header file, the system checks for an existing customer in the Customer table.
- ▶ **Database trigger:** Before the insert operation is completed, the database manager activates a program that can verify if the sales representative is assigned to the customer and log any violation attempt.
- ▶ **Program description:** The sales person periodically calls the customer over the phone and places an order. The sales person enters the customer number, the order and delivery date, and other general information. The application does not automatically generate an order number. For the sake of simplicity, this is entered by the sales representative.



A more detailed flow of this program is described as follows:

1. The program inserts a row into the Order Header table.
2. If the database referential constraint enforcement detects a customer number not defined in the Customer table, a program message is sent explaining that the customer number is invalid. A correct customer number must be entered.
3. The customer name is displayed at the terminal.
4. A row is inserted into the Order Header table.
5. Since an insert trigger is defined on this table, a program is automatically triggered by the database manager.
6. The trigger program checks if the current user profile is associated to the customer in the Sales/Customer table. If there is no match, the program writes an audit trail entry to an audit table.
7. If the insert is successful, the program returns a positive return code to the main program, which calls the Insert Order Detail program.

## **Program flow for the Insert Order Detail program**

DB2 Universal Database for iSeries functional highlights in this program include:

- ▶ Referential integrity constraints for the Order Detail table
- ▶ Referential integrity constraints for the Stock table (on remote system)
- ▶ Two-phase Commit and DRDA Level 2
- ▶ Remote stored procedure

The program flow for Insert Order Detail is shown in Figure A-2.

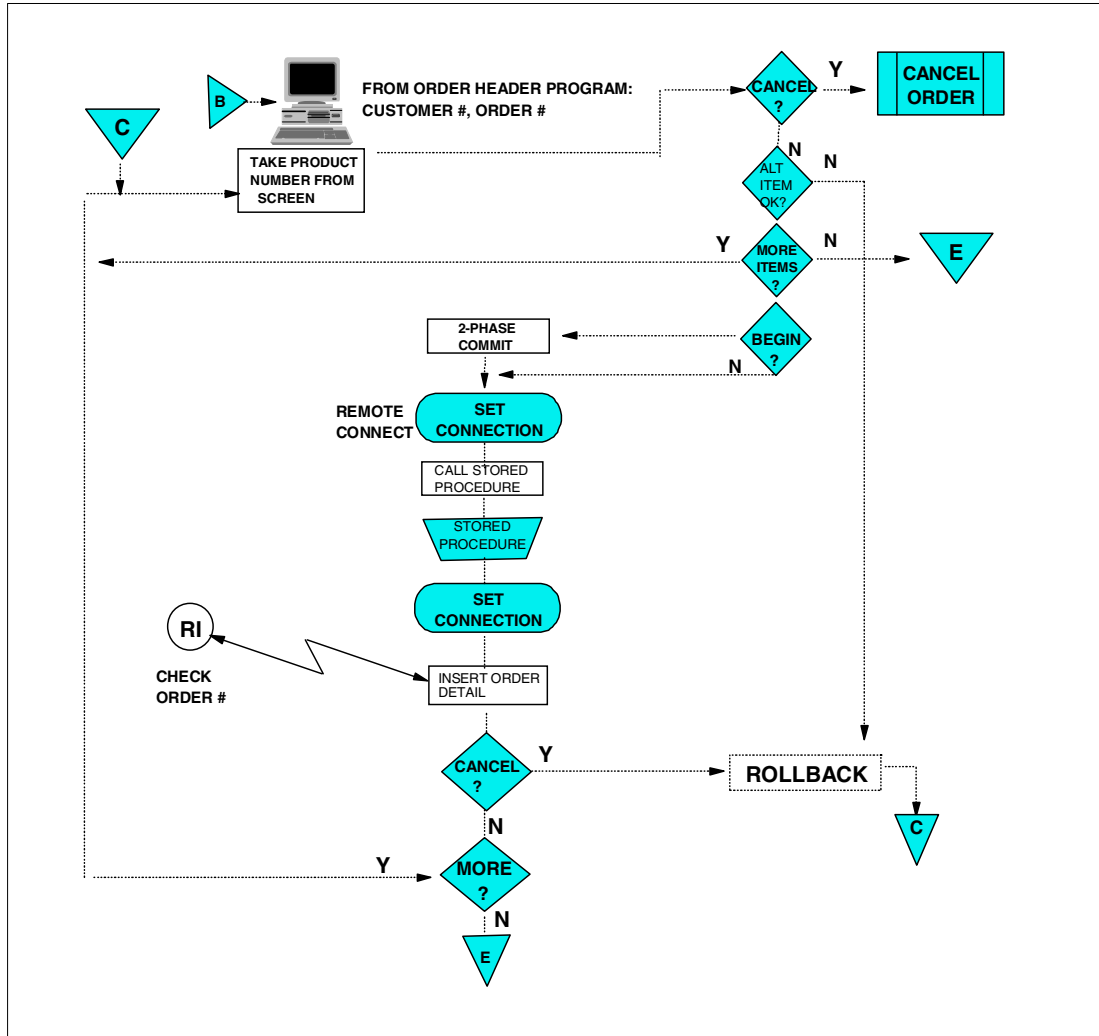


Figure A-2 Insert Order Detail program flow

## Program description for Insert Order Detail program

The idea of this program is to show how to use the following new database functions in a real application:

- ▶ **Referential integrity:** When a record is inserted into the Order Detail table for a new order item, the system checks for a matching order number in the Order Header table.
- ▶ **Two-phase commit with DRDA, Level 2:** This procedure inserts a record in a local file and updates the remote inventory file (STOCK file). At the end of this process, you want to release the locks on the inventory record and the transaction is committed. The two-phase commit support guarantees the integrity of this transaction.
- ▶ **Stored procedure:** To update the remote inventory file, this program calls a remote stored procedure. The stored procedure checks the availability of the product. If the product has low inventory levels, the stored procedure looks for an alternative and sends the new product code and description back to the calling application. The selected product information is displayed at the terminal and the user has the choice of accepting or rejecting the substitute item.

- ▶ **Program description:** This program can:
  - a. Get the customer number and the order number from the Insert Order Header program.
  - b. Get the product number and quantity for every single item from the display.
  - c. Issue a SET CONNECTION statement to the remote system. All the necessary CONNECT statements are performed by the main program.
  - d. Call a stored procedure at the remote system to:
    - Look for the product number in the remote inventory.
    - Update the Stock table, reducing the quantity on hand if the quantity available is sufficient.
    - Look for an alternative product if the requested one is out of stock, and update the corresponding quantity.
    - Pass the product information back to the calling program.
  - e. The stored procedure then passes control back to the calling program.
  - f. At this point, the program sets a connection to the local system and if the user accepts the record, the new item is inserted in the Order Detail file, and the whole transaction is committed. If the user rejects the item, a rollback brings the stock quantity on hand back to its original value.
  - g. A rollback is also performed if referential integrity checking on the Order Detail table fails. This happens if you insert the record with the wrong order number.
  - h. The user also has the option of cancelling the whole order. In this case, a Cancel Order program is called.
  - i. The program keeps a work field with the final totals of the whole order. When the entire order is completed, this value is passed to the next program – Finalize Order.

## Program flow for the Finalize Order program

The DB2 Universal Database for iSeries functional highlights in this program include the trigger on the Update Order Header row. See Figure A-3 for the program flow.

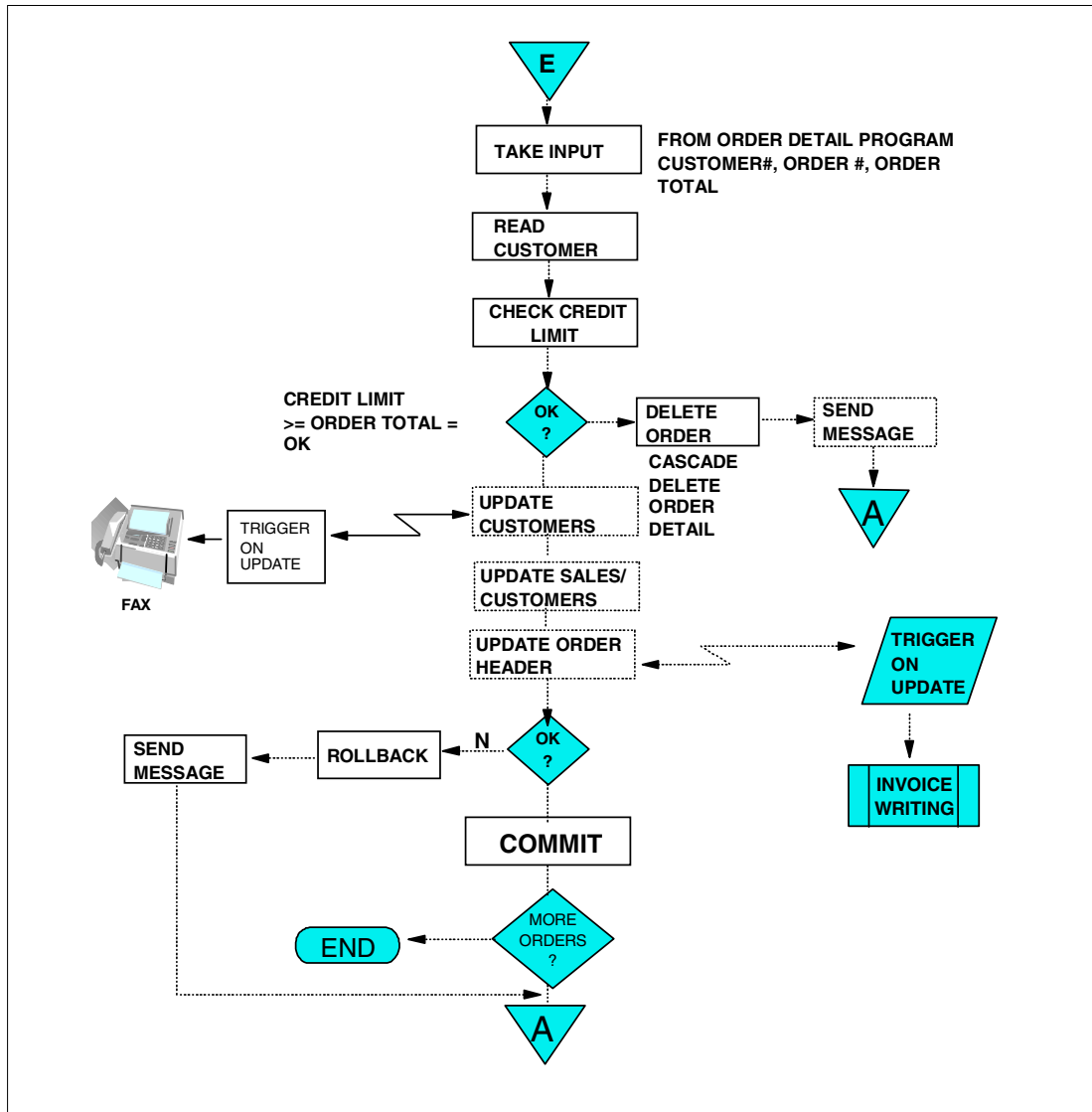


Figure A-3 Finalize Order program flow

## Program description for the Finalize Order program

The idea of this program is to show how to use the following new database functions in a real application:

- ▶ **Database triggers:** In this scenario, a program is triggered after the order header row is updated with the total amount of the order. This program prints the invoice at the branch office as soon as the order has been completed.

The program also updates the credit limit on the customer file. If the current balance exceeds 90% of the credit limit, a “warning” fax is automatically sent to the customer by a trigger program to allow the customer to take the appropriate actions (for example, applying for a credit limit increase, based on the credit history of the customer).

- ▶ **Program description:** This program can:
  - a. Get the customer number and the order number from the previous process along with the order grand total.

- b. Check the customer record. If the credit limit is exceeded, the order is cancelled. To delete the order, the detail is scanned, and the inventory quantity that is on hand for each item is updated by adding the amount reserved for this order. When this process is complete, the order header is deleted, and all the order detail disappears as a result of the \*CASCADE constraint on the order header file. The entire transaction is finally committed. Again, the two-phase commit support ensures that the local database and the remote stock file are kept synchronized.
- c. If the credit limit is OK, this program updates the following fields:
  - The total amount in the customer file to keep track of the customer balance
  - The total amount in the Sales Representative/Customer table to reflect the sales person's turnover with the customer
  - The total amount in the Order Header table items at invoice time
- d. Because an update trigger is specified on the Order Header table, an invoice program is started immediately. The invoice for the completed order is printed in the branch office. For more information about triggers, see *Stored Procedures and Triggers on DB2 Universal Database for iSeries*, SG24-6503.
- e. After the preceding updates have been done, COMMIT is executed.
- f. If there are more orders, the Insert Order Header program is started again.
- g. If there are no more orders, this Order Entry application has ended.





## Referential integrity: Error handling example

This appendix provides an example of a COBOL program that illustrates a coding example of the error handling when you use referential integrity.

In the following example, you can see a COBOL SQL implementation of this a procedure. The operation that activates the trigger and the referential integrity check is highlighted in bold. Immediately after the SQL insert, the application checks the SQLCODE for errors and reports the correct message to the user.

# Program code: Order Header entry program – T4249CINS

```
PROCESS OPTIONS.
IDENTIFICATION DIVISION.
PROGRAM-ID. T4249CINS.
AUTHOR. PROGRAMMER NAME.
INSTALLATION. ITSC LABORATORY.
DATE-WRITTEN. APRIL 2001.
DATE-COMPILED.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-AS400.
OBJECT-COMPUTER. IBM-AS400.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

        SELECT T42490HRD ASSIGN TO WORKSTATION-T42490HRD
               ORGANIZATION IS TRANSACTION
               FILE STATUS IS STATUS-ERR.
```

```
*****
DATA DIVISION.
FILE SECTION.
FD T42490HRD
   LABEL RECORD ARE STANDARD.
01 DSP01.
   COPY DDS-ALL-FORMATS OF T42490HRD.
```

```
*****
WORKING-STORAGE SECTION.

01 DSPFIL-INDICS.
   COPY DDS-ALL-FORMATS-INDIC OF T42490HRD.
```

```
77 IND-ON           PIC 1 VALUE B"1".
77 IND-OFF          PIC 1 VALUE B"0".
```

```
01 JOBA-AREA.
   03 BYTES-RTN      PIC 9(8) BINARY VALUE 0.
   03 BYTES-AVAIL    PIC 9(8) BINARY VALUE 0.
   03 JOBNAME        PIC X(10).
   03 USERNAME       PIC X(10).
   03 JOBNUMBER      PIC X(6).
```

```
*****
* Parameters for retrieve job attributes - USERID *
*****
```

```
01 RTV-JOBA.
   03 RTV-JOB-VAR    PIC X(50).
   03 RTV-JOB-LEN    PIC 9(8) BINARY VALUE 50.
   03 RTV-JOB-FMT    PIC X(8) VALUE "JOBIO400".
   03 RTV-JOB-NAME   PIC X(26) VALUE "*".
   03 RTV-JOB-ID     PIC X(16) VALUE " ".
```

```
01 STATUS-ERR      PIC XX.
01 ORDNUM          PIC X(5).
01 CUSTOMER        PIC X(5).
01 ODATE           PIC X(10).
01 ODLY            PIC X(10).
```



```

01 OTOTAL          PIC S9(9)V9(2) COMP-3.
01 INSERTOK        PIC 9.

```

```

EXEC SQL
  INCLUDE SQLCA
END-EXEC.
LINKAGE SECTION.

```

```

01 CUSTNBR          PIC X(5).
01 ORDNBR           PIC X(5).
01 RTCODE           PIC X.

```

```

*=====*
*This program has three output parameters: Customer numb.*
*Order number and Return code. The return code can be: *
*Rtcode = 0 - OK      Rtcode = 2 - F3      *
*=====*

```

```

PROCEDURE DIVISION USING CUSTNBR, ORDNBR, RTCODE.

```

```

DECLARATIVES.
TRANSACTION-ERROR SECTION.
  USE AFTER STANDARD ERROR PROCEDURE T4249OHRD.

```

```

WORK-STATION-ERROR-HANDLER.
  GOBACK.
END DECLARATIVES.

```

```

MAIN-LINE SECTION.
  OPEN I-O T4249OHRD.
  PERFORM INITIAZ-HEADER.

```

```

*=====*
* Call API to get job attributes and move the *
* output parameter into the work area *
*=====*

```

```

CALL "QUSRJOBI" USING RTV-JOB-VAR,
                      RTV-JOB-LEN,
                      RTV-JOB-FMT,
                      RTV-JOB-NAME,
                      RTV-JOB-ID.

```

```

MOVE RTV-JOB-VAR TO JOBA-AREA.
MOVE "0" TO RTCODE.
MOVE 0 TO INSERTOK.
MOVE IND-OFF TO IN15 IN ORDER-I-INDIC.
WRITE DSP01 FORMAT IS "EXITLINE".
PERFORM ORDER-ENTRY UNTIL
  IN15 IN ORDER-I-INDIC EQUAL IND-ON OR
  INSERTOK EQUAL 1.

```

```

IF IN15 IN ORDER-I-INDIC = IND-ON THEN
  MOVE "2" TO RTCODE
ELSE
  IF INSERTOK = 1 THEN
    MOVE "0" TO RTCODE.

```

```

*=====*
*We are not closing the file, because we are overlapping screens*
*=====*

```

```

* CLOSE T4249OHRD.
  GOBACK.

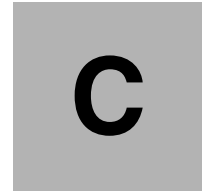
```

```

ORDER-ENTRY.
  PERFORM WRITE-READ-ORDER.
  MOVE ORHNBR OF ORDER-I TO ORDNUM.
  MOVE CUSNBR OF ORDER-I TO CUSTOMER.
  MOVE ORHDTE OF ORDER-I TO ODATE.
  MOVE ORHDLY OF ORDER-I TO ODLY.
  MOVE ZEROS TO OTOTAL.
  MOVE CUSTOMER TO CUSTNBR.
  MOVE ORDNUM TO ORDNBR.
  IF IN15 IN ORDER-I-INDIC NOT EQUAL IND-ON THEN
*
*   The programs inserts an order in ORDERHDR file.
*
  EXEC SQL
  INSERT INTO ORDENTL/ORDERHDR VALUES(:ORDNUM, :CUSTOMER,
  :ODATE, :ODLY, :OTOTAL, :USERNAME) :rk.4:erk.
  END-EXEC
  IF SQLCODE EQUAL 0 THEN
    MOVE 1 TO INSERTOK
  ELSE
*=====*
* After the insert operation, you should monitor the *
* following SQLCODEs: *
*   SQL0530(-530) - Referential Integrity violation *
*   SQL0803(-803) - Order Header already exists *
*   SQL0443(-443) - Trigger program signalled an exception *
*=====*
    IF SQLCODE EQUAL -530 THEN
      MOVE IND-ON TO IN98 OF ORDER-0-INDIC
      MOVE SPACES TO ORHNBR OF ORDER-0
      MOVE CUSTOMER TO CUSNBR OF ORDER-0
    ELSE
      IF SQLCODE EQUAL -803 THEN
        MOVE IND-ON TO IN99 OF ORDER-0-INDIC
      ELSE
        MOVE IND-ON TO IN97 OF ORDER-0-INDIC.
*****
  INITIAZ-HEADER.
  MOVE SPACES TO ORHNBR OF ORDER-0.
  MOVE SPACES TO CUSNBR OF ORDER-0.
  MOVE "0001-01-01" TO ORHDTE OF ORDER-0.
  MOVE "0001-01-01" TO ORHDLY OF ORDER-0.

WRITE-READ-ORDER.
  WRITE DSP01 FORMAT IS "ORDER" INDICATORS ARE ORDER-0-INDIC.
  MOVE IND-OFF TO ORDER-I-INDIC ORDER-0-INDIC.
  READ T42490HRD RECORD INDICATORS ARE ORDER-I-INDIC.

```



## Additional material

This redbook also contains additional material that is available on the Web. See the following sections for instructions on using or downloading the Web material.

### Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG244249>

Alternatively, you can go to the IBM Redbooks Web site at:

[ibm.com/redbooks](http://ibm.com/redbooks)

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG244249.

### Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
<b>dbadvfun.exe</b>	iSeries and client source code image
<b>readme.txt</b>	Readme documentation

### System requirements for downloading the Web material

The following list contains the most important requirements:

- ▶ **iSeries requirements**
  - OS/400 Version 5 Release 1
  - 5722-ST1 - DB2 Query Manager and SQL Development kit
  - 5722-SS1 - Host Servers

► **PC software**

- Windows 95/98, Windows NT, or Windows 2000
- Client Access Express for Windows
- PC5250 Emulation

## **How to use the Web material**

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

The readme.txt contains the instructions for restoring the iSeries libraries and directories, as well as installing the PC clients and run-time notes.

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

## IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 344.

- ▶ *DB2/400: Mastering Data Warehousing Functions*, SG24-5184
- ▶ *AS/400 Internet Security: Implementing AS/400 Virtual Private Networks*, SG24-5404
- ▶ *DB2 UDB for AS/400 Object Relational Support*, SG24-5409
- ▶ *Cross-Platform DB2 Stored Procedures: Building and Debugging*, SG24-5485
- ▶ *Managing AS/400 V4R4 with Operations Navigator*, SG24-5646
- ▶ *Stored Procedures and Triggers on DB2 Universal Database for iSeries*, SG24-6503

The following IBM Redbooks will be available in first quarter 2002:

- ▶ *Managing OS/400 with Operations Navigator V5R1 Volume 1: Basic Functions*, SG24-6226
- ▶ *Managing OS/400 with Operations Navigator V5R1 Volume 2: Advanced Functions*, SG24-6227

## Other resources

These publications are also relevant as further information sources:

- ▶ *DB2 Connect Personal Edition Quick Beginning*, GC09-2967
- ▶ *COBOL/400 User's Guide*, SC09-1812
- ▶ *COBOL/400 Reference*, SC09-1813
- ▶ *ILE RPG Programmer's Guide*, SC09-2074
- ▶ *ILE RPG Reference*, SC09-2077
- ▶ *DB2 UDB Application Development Guide V6*, SC09-2845
- ▶ *AS/400 National Language Support*, SC41-5101
- ▶ *Backup and Recovery*, SC41-5304
- ▶ *Work Management*, SC41-5306
- ▶ *Distributed Data Management*, SC41-5307
- ▶ *Client Access Express for Windows*, SC41-5509
- ▶ *ILE Concepts*, SC41-5606
- ▶ *SQL Programming Guide*, SC41-5611
- ▶ *SQL Reference*, SC41-5612
- ▶ *Database Programming*, SC41-5701
- ▶ *Distributed Database Programming*, SC41-5702

- ▶ *DDS Reference*, SC41-5712
- ▶ *Control Language Programming*, SC41-5721
- ▶ *CL Reference*, SC41-5722
- ▶ *System API Programming*, SC41-5800
- ▶ *SQL Call Level Interface*, SC41-5806
- ▶ *DB2 UDB for iSeries Database Performance and Query Optimization*:  
<http://submit.boulder.ibm.com/pubs/html/as400/bld/v5r1/ic2924/index.htm>

## Referenced Web sites

These Web sites are also relevant as further information sources:

- ▶ iSeries Information Center: <http://www.iseries.ibm.com/infocenter>
- ▶ DB2 Universal Database for iSeries main page:  
<http://www.iseries.ibm.com/db2/db2main.htm>
- ▶ PartnerWorld for Developer - iSeries site: <http://www.iseries.ibm.com/developer>
- ▶ Support Line Knowledge Base:  
<http://as400service.ibm.com/supporthome.nsf/document/10000051>
- ▶ Data Movement Utilities Guide and Reference:  
<http://www-4.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/document.d2w/report?fn=db2v7dmdb2dm07.htm#HDREXP0VW>
- ▶ iSeries Library: <http://www-1.ibm.com/servers/eserver/series/library>
- ▶ IBM Learning Services: <http://www-1.ibm.com/servers/eserver/series/education/>

## How to get IBM Redbooks

Search for additional Redbooks or Redpieces, view, download, or order hardcopy from the Redbooks Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

Also download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become Redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

## IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

# Special notices

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power To Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

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

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

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

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

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



# Index

## Symbols

\*DUW option 93  
\*FILE object 160  
\*RUW option 93  
\*SHRUPD 35

## Numerics

01222 status 51

## A

access path 5, 25, 181  
activation group 92, 93  
activation group ID 96  
active jobs 118  
add column 186  
Add Relational Database Directory Entry (ADDRDBDIRE) command 110  
Add Server Authentication Entry (ADDSVRAUTE) command 111  
adding  
    multiple constraints 32  
    referential constraint 28  
    relational database directory entry 110  
    server authentication entry 111  
ADDPFCST (Add Physical File Constraint) command 28  
ADDRDBDIRE (Add Relational Database Directory Entry) command 110  
ADDSVRAUTE (Add Server Authentication Entry) command 111  
administrative interface 124  
advanced functions 6, 11  
advanced journal attributes 180  
Advised Index 230  
alias 168, 182  
ALTER TABLE SQL statement 28, 188  
ALTER TABLE statement DROP clause 56  
Analyze Results panel 228  
analyzing SQL Performance Monitor results 228  
application design 101  
application example 12  
application flow using DRDA-2 94  
application integrity 22  
application message 78  
application requester (AR) 84, 110  
application server (AS) 84, 93  
apply journal changes 49, 53  
AR (application requester) 84  
AS (application server) 84  
ASP (auxiliary storage pool) 167  
authentication information 111  
automatic recovery 96  
auxiliary storage pool (ASP) 167

## B

breadth cascade 43  
business rules  
    referential integrity 22  
    translated to physical file constraints 32

## C

C ILE program referential integrity messages 52  
CASCADE delete rule 35  
CASCADE example 39  
cascade network 28  
CASCADE rule 23, 25  
catalog inquiry 63  
CCSID (Coded Character Set Identifier) 187, 205  
Change DDM TCP/IP Attributes (CHGDDMTCPA) command 109  
Change Physical File Constraint (CHGPFCST) command 53  
Change Query Attributes 164, 217  
Change Query Attributes (CHGQRYA) command 137  
Change Server Authentication Entry (CHGSVRAUTE) command 112  
check constraint 7, 24, 67, 192  
    application message 78  
    DB2 UDB for iSeries 69  
    defining 70  
    I/O message 77  
    integration into applications 77  
    management 79  
    state 80  
    tips and techniques 82  
check pending 53  
    condition 71  
    defined 24  
    protection from 49  
CHGDDMTCPA (Change DDM TCP/IP Attributes) command 109  
CHGPFCST (Change Physical File Constraint) command 53  
CHGQRYA (Change Query Attributes) command 137  
CHGSVRAUTE (Change Server Authentication Entry) command 112  
Client Access/400 6  
COBOL  
    deleting an order 103  
    DRDA-2 program example 102  
    ILE program, referential integrity messages 52  
Coded Character Set Identifier (CCSID) 187, 205  
coexistence for DRDA-1 and DRDA-2 95  
collection 166  
column 5  
command 181  
    CHGDDMTCPA 109  
    STRTCPSVR SERVER(\*DDM) 110

- command, CL 206
  - Add Physical File Constraint (ADDPFCST) 28
  - Add Relational Database Directory Entry (ADDRDB-DIRE) 110
  - Add Server Authentication Entry (ADDSVRAUTE) 111
  - ADDRDBDIRE (Add Relational Database Directory Entry) 110
  - ADDSVRAUTE (Add Server Authentication Entry) 111
  - Change Server Authentication Entry (CHGSVRAUTE) 112
  - Copy From Import File (CPYFRMIMPF) 126
  - Copy To Import File (CPYTOIMPF) 126
  - CRTSQLpgm 93
  - CRTSQLxxx 101
  - Print SQL Information (PRTSQLINF) 303
  - Remove Server Authentication Entry (RMVS-VRAUTE) 112
  - Start Debug (STRDBG) 118
  - Work with Active Jobs (WRKACTJOB) 118
- commit 204
- commit group 204
- commit mode 204
- COMMIT(\*NONE) 95
- commitment control 43, 204
  - requirements 25
- commitment definition 97
- condition clause 70
- condition clause of check constraint 75
- CONNECT 88
  - SQL statement 84, 92
- CONNECT (Type 1) 92
- CONNECT (Type 2) 93
- connection
  - current 88
  - dormant 88
  - held 88
  - multiple handling in DRDA-2 101
  - preserved 7
  - released 88
  - states 88
- connection DRDA 101
- connection management 86
  - methods 88
  - on DB2 UDB for iSeries 87
- consistency of data in multiple locations 90
- constraint 181, 188, 192
  - commands 53
  - displaying information 61
  - domain 68
  - enforcement 35
  - management 52
  - prerequisites 24
  - referential integrity network example 32
  - removing 56
  - self-referencing 34
  - states 52
  - table 68
  - tips 192
  - types 22
    - unique or primary key 29
- Control Center 149
- copy 183
- Copy From Import File (CPYFRMIMPF) command 126
- Copy To Import File (CPYTOIMPF) command 126, 138
- CPF502D notify message 51
- CPF502E notify message 51
- CPF503A notify message 51
- CPF523B escape message 51
- CPF523C escape message 51
- CPU bound 316
- CPYFRMIMPF (Copy From Import File) command 126
- CPYTOIMPF (Copy To Import File) command 126, 138
- create journal 177
- Create Physical File (CRTPF) command 170
- create SQL package 114
- CREATE TABLE SQL statement 28, 160
- CREATE VIEW 160, 162
- creating a referential constraint 28
- CRTPF (Create Physical File) command 170
- CRTSQLpgm 93
- CRTSQLpgm command 93
- CRTSQLxxx command 101
- current SQL statement 218
- current state 89
- cut 183
- cyclic constraint 24

## D

- data access
  - distributed 84
  - distributed environment 84
  - methods 305
- data consistency 90
- Data Description Specifications (DDS) 5
- data field 5
- data format 127, 139
- data inconsistencies 49
- data load
  - Data Definition Language example 133
  - file definition file example 130
- data loss 49
- data source translation 203, 205
- data, invalid check pending 53
- database administration 157
- Database functions 163
- database library functions 165
- Database Monitor 303
  - Visual Explain 318
- Database Navigator 8, 239
  - locator pane 247
  - map pane 249
  - menu options 249
  - system requirements and planning 240
- Database Navigator map 244
  - creating 261
  - display 253
  - generating SQL 289
  - icons 269

- interface 246
- table options 255
- database performance analysis methodology 323
- database relationship 242
- database synchronization on multiple systems 96
- Database task pad 246
- database tuning 302
- DB2 Connect 120
  - AS/400 port number 123
  - CCSID for user profile 121
  - over TCP/IP 120
- DB2 family 4
- DB2 UDB 7.2 data migration to DB2 UDB for iSeries 149
- DB2 UDB for iSeries 3
  - advanced functions 6
  - check constraint 69
  - distributed environment 5
  - DRDA-2 86
  - Import utility 126
  - journaling 124
  - moving data to DB2 UDB 7.2 152
  - Operations Navigator 161
  - overview 4
  - programming languages 272
  - sample schema 8
  - SQL support for connection management 92
- DDM (Distributed Data Management) 5
- DDM server job 109, 122
- DDS (Data Description Specifications) 5
- debug messages 302
- debug mode 220
- default libraries 204
- define check constraint 72
- DEFINED constraint state 52
- DEL (delimited ASCII file) 149
- delete 183, 193
- delete column 186
- delete constraint 56
- delete parent record example 36, 38
- delete rows 183
- delete rule
  - CASCADE, SET NULL, and SET DEFAULT 35
  - defined 23
- deleted record 170
- deleting an order 103
  - DRDA-2 and two-phase commitment control 103
  - COBOL example 103
- delimited ASCII file (DEL) 149
- delimited import file 128
- dependent file
  - defined 23
  - same file as parent file 34
- dependent table 68
- depth cascade 43
- detail rows 105
- DISABLED constraint state 52
- DISCONNECT 90
- DISCONNECT statement 93
- Display Check Pending Status (DSPCPCST) command 54

- Display Database Relations (DSPDBR) command 61
- Display Journal Entry Details display 45
- Display Physical File Description (DSPFD) command 61
- distributed data access 84
- Distributed Data Management (DDM) 5
- distributed database example 14
- distributed database network 87
- distributed environment 5
  - data access in 84
- Distributed Relational Database Architecture (DRDA) 6, 84, 201
- distributed relational database example 13
- Distributed Request (DR) 86
- Distributed Unit of Work (DUW) 7, 85
- DLTPCT parameter 170
- domain constraint 68
- dormant state 89
- DR (Distributed Request) 86
- DRDA 6, 83, 84
  - application server 108
  - COMMIT(\*NONE) 95
  - Distributed Unit of Work 7
  - initial connections 101
  - level 0 85
  - level 1 85
  - level 2 85
  - level 3 86
- DRDA (Distributed Relational Database Architecture) 6, 84, 201, 202
- DRDA over TCP/IP 108
  - troubleshooting 117
- DRDA-1 86
  - coexistence with DRDA-2 95
  - moving to DRDA-2 101
- DRDA-2 86
  - application flow example 94
  - Coexistence 95
  - coexistence with DRDA-1 95
  - CONNECT 92
  - connection management 87
  - connection management method 88
  - Connection Management on DB2 UDB for iSeries 87
  - DISCONNECT, DB2 UDB for iSeries 90, 93
  - performance 101
  - program example 102
  - protected conversation 90
  - RDB Connection Management Method 88
  - RELEASE, DB2 UDB for AS/400 93
  - SET CONNECTION 94
  - Synchronization Point Manager (SPM) 90
  - two-phase commit 90
  - unprotected conversation 90
- DRDA-2 and two-phase commitment control 105
- drop active connections 93
- DROP clause of ALTER TABLE statement 56
- DSPCPCST (Display Check Pending Status) command 54
- DSPDBR (Display Database Relations) command 61
- DSPFD (Display Physical File Description) command 61
- DUW (Distributed Unit of Work) 7, 85

## E

Edit Check Pending Constraints (EDTGPCST) command 58  
edit recovery for access path 181  
Edit SQL 176  
EDTGPCST (Edit Check Pending Constraints) command 58  
ENABLED constraint state 52  
error handling example 337  
escape message 51  
ESTABLISHED constraint state 52  
example 103  
    application flow using DRDA-2 94  
    CASCADE 39  
    delete parent record 36  
        no RESTRICT or NOACTION rule 38  
    Display Journal Entry Details display 45  
    distributed relational database 13  
    DRDA-2 program, COBOL 102  
    inserting the detail rows 105  
    logical consistency 13  
    multiple constraints 32  
    Order Entry application overview 12  
    referential integrity network 32  
    SQL CREATE TABLE 29  
    unmatched foreign key values 28  
Explainable Statement 231  
Export API 151  
Export command 151  
Export utility 125, 138, 149, 152

## F

failure recovery 96  
field definition file 127  
field level authority 162  
file availability 28  
Finalize Order program 333  
flyover 254  
foreign key 35  
    constraint prerequisites 24  
    defined 23  
    in same physical file as primary key 34  
foreign key value verification 28  
function, user defined 168

## G

Generate SQL 271, 272  
    from Database Navigator map 289  
    from DDS 297  
    Operations Navigator 276  
    to PC and data source files 281

## H

held state 89  
hierarchical structure 34

## I

I/O bound 316  
I/O messages 50, 77  
ILE C example 114  
ILE C programs 52  
ILE COBOL programs 52  
ILE program 102  
ILE RPG programs 51  
implicit primary key constraint 31  
import file 138  
Import utility 125, 126, 149, 153  
Include Debug Messages in Job Log 214  
Include Error Message Help in Run History 213  
index 168, 181, 188  
indexes for referential integrity 25  
Informix 86  
initial DRDA connection 101  
Insert Order Detail program 331  
Insert Order Header program 330  
insert rows 183  
inserting detail rows 105  
integrated exchange file (IXF) 149  
integrated relational database 4  
Interactive SQL 113  
invalid data check pending 53  
IXF (integrated exchange file) 149

## J

Java Database Connectivity (JDBC) 202  
Java stored procedures  
    See also stored procedures Java  
JDBC (Java Database Connectivity) 202  
job log 97  
JOIN statement 214  
journal 167, 177, 181, 192  
journal changes 49  
journal entries with referential integrity 45  
journal entry 177  
journal example 177  
journal receiver 177, 178, 181, 192  
journaling 43, 124  
journaling requirements 25  
journals 168

## K

key constraints 188  
key types defined 22  
keyed access path 26  
keyed logical file 5

## L

lab exercise 230  
Level Check (LVLCHK) parameter 186  
library 166, 167  
library name 208  
library-based functions 168  
like operating environments 84  
loader utility 126

- locator pane 247
- lock file 28
- locked rows 196
- locking files 35
- log not written 95
- logical consistency example 13
- logical file 5
- logical transaction 85
- Logical Unit of Work ID 96
- loss of data 49

## M

- manual recovery 97
- map pane 249
- mapping referential integrity messages 52
- maximum members 170
- member size 170
- messages
  - CPF502D 51
  - CPF502E 51
  - CPF503A 51
  - CPF523B 51
  - CPF523C 51
  - referential integrity 50
- Modify Selected Queries 229
- multiple connections 7
- multiple constraints 32
- multiple databases 7
- multiple locations, data consistency in 90

## N

- network
  - coexistence of DRDA-1/DRDA-2 95
  - referential integrity or cascade 28
- new journal receiver 193
  - attributes 181
- no RESTRICT or NOACTION rule 38
- NOACTION rule 35
  - defined 24
  - delete example without 38
  - enforcement 35
- non-SQL interface considerations 320
- notify messages 51
- NULLID collection 121

## O

- object-based function 181
- Objects to Display window 253
- ODBC (Open Database Connectivity) 6, 202
- Open 182
- Open Database Connectivity (ODBC) 6, 202
- openness 86
- Operations Navigator
  - Generate SQL 276
  - new V5R1 features 159
  - Visual Explain 301
- OPM programs 101
- Oracle 86

- Order Entry application 11, 12
  - advanced database functions 17
  - database 14
  - detailed flow 329
- Order Entry example 12
- Order Header entry program 338
- orphan foreign key values example 28
- OS/400 collection 166
- OS/400 library 166
- ownership of access path 25

## P

- parallel data load
  - data format 127, 139
  - delimited import file 128
  - field definition file 127
  - source file (FROMFILE) 127, 138
  - target file (TOFILE) 127, 139
- parallel data loader 137
- parent file
  - defined 23
  - same file as dependent file 34
- parent key
  - constraint prerequisites 24
  - defined 23
  - identifying 29
- parent record
  - delete example 36
  - no RESTRICT or NOACTION rule 38
- parent table 68
- PC user integrity 22
- performance 93
  - benefits of system provided referential integrity 22
  - DRDA-2 considerations 101
  - improved 25
  - referential integrity application impacts 50
  - when adding referential constraint 28
- performance collection files 205
- Permissions 193
- permissions 168, 181
- physical data 5
- physical file 5, 32, 170
  - add multiple constraints 32
  - constraints
    - referential integrity network example 32
- port number for DRDA connection 123
- predictive query governor 303
- primary key 23, 29
  - constraint 23, 29, 31
  - defined in SQL 29
  - in same physical file as foreign key 34
- properties 168, 193
- protected conversation 90, 93
- protocols 84
- PRTSQLINF (Print SQL Information) command 303

## Q

- QBATCH 207
- QRWTLSTN job 110, 117

- QRWTSRVR job 117
- query attributes and values 315
- query environment
  - Visual Explain
    - query environment 314
- query optimizer 220
  - debug messages 302
- Query/400 234, 320
- quick view 182

## R

- RDB Connection Management Method 93
- RDB parameter 101
- RDBCNNMTH parameter 93
- RDBCNNMTH(\*DUW) 88
- RDBCNNMTH(\*RUW) 88
- read lock 35
- record 5
- record field 5
- record selection 5
- recovery 96
  - automatic 96
  - from failure 96
  - manual 97
  - Work with Commitment Definitions (WRKCMTDFN)
    - command 98
- Redbooks Web site 344
  - Contact us xii
- Ref Constraint parameter 45
- referential constraint 28, 188
  - creating 29
  - defined 23
  - dependent file 31
  - enforcement 35
  - example 30
  - rules 23
- referential cycle 24
- referential integrity 17, 21, 25, 49
  - application considerations 50
  - check pending 53
  - concepts 22
  - constraint 18, 24, 68
    - concept 18
    - prerequisites 24
  - constraint management 52
  - constraint tips and techniques 82
  - defined 7, 23
  - error handling 337
  - example 13
  - I/O messages 50
  - introduction 22
  - journal entries 45
  - journaling and commitment control 43
  - message handling in applications 51
  - messages in RPG ILS programs 51
  - network 28
  - relationship 24
  - restoring data 49
  - rules ordering 36
  - SQLCODE values 52
  - verification queries 28
- referential integrity messages
  - in ILE C programs 52
  - in ILE COBOL programs 52
  - in ILE RPG programs 51
- relational database
  - directory 110
  - directory entry 110
  - integration overview 4
- RELEASE statement 93
- released state 89
- remote journal 177, 181, 193
- remote locations 84
- remote request 85
- Remote Request (RR) 85
- remote stored procedure 114
  - using DRDA over TCP/IP 113, 114
- Remote Unit of Work (RUW) 85, 93
- remove constraint 56
- remove internal entries 181
- remove journal changes 49, 53
- Remove Physical File Constraint (RMVPCST) command 56
- Remove Server Authentication Entry (RMVSVRAUTE)
  - command 112
- reorganize 182
- reorganize file/table 170
- restoring data 49
- RESTRICT rule 25, 35
  - defined 24
  - delete example without 38
  - enforcement 35
- retain server security data 111
- reused connections 93
- REUSEDLT parameter 170
- reverse engineering 271, 272
- RMVPCST (Remove Physical File Constraint) command 56
- RMVSVRAUTE (Remove Server Authentication Entry)
  - command 112
- RNQ1222 inquiry message 51
- RNX1222 escape message 51
- rollback 87, 204
- root value 34
- row 5
- RPG ILE program, referential integrity messages 51
- RR (Remote Request) 85
- rules 84
  - enforcement 35
  - ordering for referential integrity 36
  - referential integrity 22
- Run and Explain option 307
- Run History pane 200
- Run SQL Script
  - DDM/DRDA configuration summary 216
  - example using VPN journal 208
- Run SQL Scripts 164, 166, 182, 183, 197
  - Run option 210
- running CL in SQL Scripts 208

## S

- save and restore 53, 59, 81
- Screen Edit Utility (SEU) 131
- self study lab 160
- self-referencing constraint 34
- semantics 84
- SET CONNECTION 89
- SET CONNECTION statement 94
- SET DEFAULT delete rule 35
- SET DEFAULT rule 24
- SET NULL delete rule 35
- SET NULL rule 23
- SEU (Screen Edit Utility) 131
- shared access path for referential integrity 25
- shared lock 35
- side-effect journal entry 45
- SMAPP 181
- Smart Statement Selection 213
- SMP (Symmetric MultiProcessing) 137
- source file (FROMFILE) 127, 138
- span multiple databases 7
- SPM (Synchronization Point Manager) 86, 90
- SQL 84, 225
  - collection 167
  - connect statements 92
  - index 25
- SQL (Structured Query Language) 5
- SQL CREATE TABLE statement example 29
- SQL index 5
- SQL naming convention (operational difference) 206
- SQL performance analysis 323
- SQL Performance Monitor 213, 220, 228
  - analyzing summary results 228
  - detailed monitor analysis 230
  - reviewing results 226
- SQL procedure 168
- SQL script
  - running a CL command 206
  - tips for running CL 208
- SQL Script Center 306
- SQL statements
  - CONNECT 84
  - CREATE TABLE/ALTER TABLE 28
- SQL TABLE 170
- SQL table 5
- SQL Trigger 188
- SQL view 5
- SQL VIEW example 172
- SQL-92 standard 68
- SQLCODE values 52
- Start Debug (STRDBG) command 118
- starting and ending journaling 193
- starting the SQL Performance Monitor 222
- status 01222 51
- Stop on Error 213
- stored procedures 7
- STRDBG (Start Debug) command 118
- STRTCPSVR SERVER(\*DDM) command 110
- Structured Query Language (SQL) 5
- swap receivers 193

- Sybase 86
- Symmetric MultiProcessing (SMP) 137
- Synchronization Point Manager (SPM) 86, 90
- system failure check pending 53
- system naming convention, operational difference 206

## T

- table 162, 167, 168, 181, 182
- table constraint 68
- table options 255
- target file (TOFILE) 127, 139
- task pad 246
- TCP/IP 108
  - application requester 110
  - configuring on the application server 109
  - DB2 Connect access to iSeries 120
- transaction atomicity 43
- transaction isolation 95
- tree relationship among records in database 34
- triggers 7, 181, 188
- troubleshooting DRDA over TCP/IP 117
- two-phase commitment control 7, 83, 86, 90
  - needs assessment 18

## U

- unique constraint 23, 29
  - defined in SQL 29
- unique key 23
- unit of recovery (UR) 85
- unit of work (UoW) 85, 87
- unlike operating environments 84
- unprotected connections 93
- unprotected conversation 90
- UoW (unit of work) 85
- update lock 35
- update row 183
- update rule 24
- UR (unit of recovery) 85
- user defined function 168
- user defined type 168
- user-defined relationship 266

## V

- verification of foreign key value 28
- verification queries 28
- view 167, 168, 182
- view of physical data 5
- View Results button 229
- Visual Explain 212, 301, 304
  - data access methods 305
  - Database Monitor data 318
  - icons 321
  - navigation 308
  - query attributes and values 315
  - Query/400 320
  - SQL performance analysis 323
  - SQL Script Center 306
- Visual Explain Only option 307

VPN journal 208

## **W**

Work with Active Jobs (WRKACTJOB) command 118  
Work with Commitment Definition (WRKCMTDFN) command 98  
Work with Physical File Constraints (WRKPFCST) command 57  
worksheet format file (WSF) 149  
writing for DRDA programs 84  
WRKACTJOB (Work with Active Jobs) command 118  
WRKCMTDFN (Work with Commitment Definition) command 97, 98  
WRKPFCST (Work with Physical File Constraints) command 57  
WSF (worksheet format file) 149

## **X**

XDB Systems 86





Redbooks

## Advanced Functions and Administration on DB2 Universal Database for iSeries

(0.5" spine)

0.475" x 0.873"

250 <-> 459 pages







# Advanced Functions and Administration

## on DB2 Universal Database for iSeries



**Learn about  
referential integrity  
and constraints**

**See how Database  
Navigator maps your  
database**

**Discover the secrets  
of Visual Explain**

Dive into the details of DB2 Universal Database for iSeries advanced functions and database administration. This IBM Redbook equips programmers, analysts, and database administrators with all the skills and tools necessary to take advantage of the powerful features of the DB2 Universal Database for iSeries relational database system. It provides suggestions, guidelines, and practical examples about when and how to effectively use DB2 Universal Database for iSeries.

This redbook contains information that you may not find anywhere else, including programming techniques for the following functions:

- ▶ Referential integrity and check constraints
- ▶ DRDA over SNA, DRDA over TCP/IP, and two-phase commit
- ▶ DB2 Connect
- ▶ Import and Export utilities

This redbook also offers a detailed explanation of the new database administration features that are available with Operations Navigator in V5R1. Among the tools, you will find:

- ▶ Database Navigator
- ▶ Reverse engineering and Generate SQL
- ▶ Visual Explain
- ▶ Database administration using Operations Navigator

With the focus on advanced functions and administration in this fourth edition of the book, we moved the information about stored procedures and triggers into a new redbook – *Stored Procedures and Triggers on DB2 Universal Database for iSeries*, SG24-6503.

### **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](http://ibm.com/redbooks)

SG24-4249-03

ISBN 0738422320