DB2 Information Management Software

IBM

DB2 Universal Database for iSeries Administration

The Graphical Way on V5R3

- Unleash the new features of iSeries Navigator for database in V5R3
- Discover database maps, constraints, and Generate SQL
- Learn the secrets of Run SQL Scripts and Visual Explain

Hernando Bedoya
Nandoo Neerukonda
Edwin Perez

Redbooks

ibm.com/redbooks

International Technical Support Organization

DB2 Universal Database for iSeries Administration: The Graphical Way on V5R3

September 2004

SG24-6092-00

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

First Edition (September 2004)

This edition applies to Version 5, Release 3 of OS/400 and IBM i5/OS (product number 5722-SS1).

Contents

Notices

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

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

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

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

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

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

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

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

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

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

Trademarks

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

@server®	DRDA®	Language Environment®
AS/400®	ibm.com®	OS/400®
Distributed Relational Database Architecture™	iSeries™	Redbooks(logo) ™
DB2 Universal Database™	i5/OS™	Redbooks™
DB2®	Integrated Language Environment®	WebSphere®
	IBM®	

The following terms are trademarks of other companies:

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

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

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

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

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

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

Preface

iSeries™ Navigator for IBM i5/OS™ and OS/400® is *the* graphical interface to manage your IBM® @server iSeries runtime environment. Version 5 Release 3 (V5R3) of iSeries Navigator contains major functions and interface enhancements over previous releases.

This IBM Redbook presents an overview of all V5R3 iSeries Navigator functions to manage and administer DB2® Universal Database™ (UDB) for iSeries. It serves as a companion guide for the database administrator to use DB2 UDB for iSeries. This redbook includes the latest announced features in V5R3 that iSeries Navigator provides for the database administrator of DB2 UDB for iSeries.

This redbook discusses the new graphical interface provided for database in V5R3. In addition, it covers the following topics:

- Definition of tables, indexes, views, and other database objects
- Constraint definition and management
- Database Navigator maps
- Reverse engineering and Generate SQL
- Run SQL Script center
- Visual Explain

Note to reader: This redbook is based on OS/400 V5R3. All iSeries Navigator windows shown in the book, as well as references to the Information Center, are also based on V5R3 interfaces.

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 four years ago, he worked for IBM Colombia as an AS/400® IT Specialist doing presales support for the Andean countries. He has 20 years experience in the computing field and has taught database classes in Colombian universities. He holds a master's degree in computer science from EAFIT, Colombia. His areas of expertise are database technology, application development, and data warehousing.

Nandoo Neerukonda is a Software Technical Specialist at Countrywide Financial. He has over eight years of experience in various aspects of Application Design and Development, Systems Administration, and Support on the AS/400 and iSeries. His areas of expertise include DB2 UDB, SQL, RPG, Java™, WebSphere® MQ, XML, etc. He is a mentor and "go to guy" for many iSeries developers across multiple organizations within Countrywide. Nandoo holds a Bachelor of Technology degree in electronics and communications from Nagarjuna University in India.

Edwin Perez is the IT Coordinator at Cristaleria PELDAR, which is a top glass manufacturing company in Colombia and an affiliate of Owens-Illinois Inc. He is in charge of technological infrastructure in his company including the iSeries platform. He has seven years of experience in working with AS/400 and iSeries servers. Previously he worked as an RPG developer and as a database administrator for Bancolombia, the largest bank of Colombia. He holds a degree in computer science from the University of Antioquia in Colombia.

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

Christopher Brandt
Jim Flanagan
Kent Milligan
Jarek Miszczyk
IBM Rochester

Become a published author

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

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

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

`ibm.com/redbooks/residencies.html`

Comments welcome

Your comments are important to us!

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

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

  `ibm.com/redbooks`

- Send your comments in an Internet note to:

  `redbook@us.ibm.com`

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. JLU Building 107-2
  3605 Highway 52N
  Rochester, Minnesota 55901-7829

Part 1

Basic concepts

This part introduces the basics concepts of DB2 Universal Database (UDB) for iSeries. It also describes the sample schema used to illustrate many of the concepts in this book.

1

Introducing DB2 Universal Database for iSeries

This chapter begins with a general introduction to DB2 Universal Database (UDB) for iSeries. Then it provides an overview of its features as presented in this redbook. Finally, it defines the sample schema that is used in this book.

1.1 An integrated relational database

Integration has been a major element of differentiation of the iSeries server in the information technology (IT) 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, and backup and recovery are all vital components that 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 operating system (OS/400 and IBM i5/OS).

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 of the user data on the iSeries server is stored in a relational database. 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 performed by the hardware.

A recent survey indicates that a significant percentage of iSeries customers does not know that all of their business data is stored in a relational database. This may sound strange if you think that we consider the integrated database as one of the main technological advantages of the iSeries server. 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 database system in the marketplace.

During the last couple of years, each new release of OS/400 and i5/OS 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 includes DB2 for OS/390 and DB2 Universal Database. The DB2 UDB family is the IBM proposal in the marketplace of relational database systems. It 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: Features overview

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

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

As previously mentioned, the 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 implements other file systems in its design, the relational database on the iSeries server is the most commonly used database by customers. Your relational data is stored in the database, along with 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 the term *table* 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 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 in 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. Applications can use any database file to access DB2 UDB for iSeries data.

DB2 UDB for iSeries in a distributed environment

It is becoming 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 to operate 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. 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 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 and i5/OS.

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

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

- A sophisticated file transfer function that allows you to subset rows and columns
- Remote SQL application programming interfaces (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, which 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. Table 1-1 lists current the terms and its SQL equivalent.

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

SQL term	OS/400 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

1.3 DB2 UDB for iSeries sample schema

Within the code of OS/400 and i5/OS Version 5, Release 3, Modification 0 (V5R3M0), a stored procedure creates a fully functioning database. This database contains tables, indexes, views, aliases, constraints, journals, and journal receivers. It also contains data within these objects.

We use this database in this redbook to illustrate the new functions announced with iSeries Navigator V5R3M0. We can also use this database as a learning tool to explain the various functions available at V5R3M0 with iSeries Navigator. Furthermore, the database provides a method to teach application programmers or new database administrators how to build relationships 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 in V5R3M0. 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 SQL statement:

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

You can find this statement in the drop-down list in the Run SQL Script window (Figure 1-1).

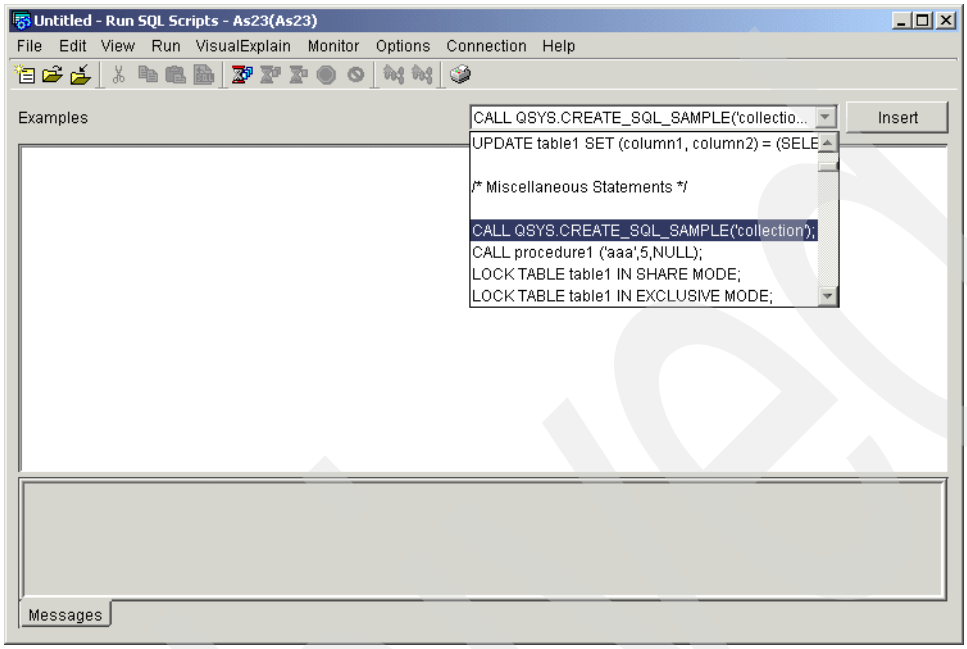


Figure 1-1 Example display showing the schema CREATE statement

Note: The schema name needs must be in uppercase. This sample schema will 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 constitutes a sample database that demonstrates some of the features of DB2 Universal Database for iSeries. Figure 1-2 shows an entity-relationship (ER) diagram of the database.

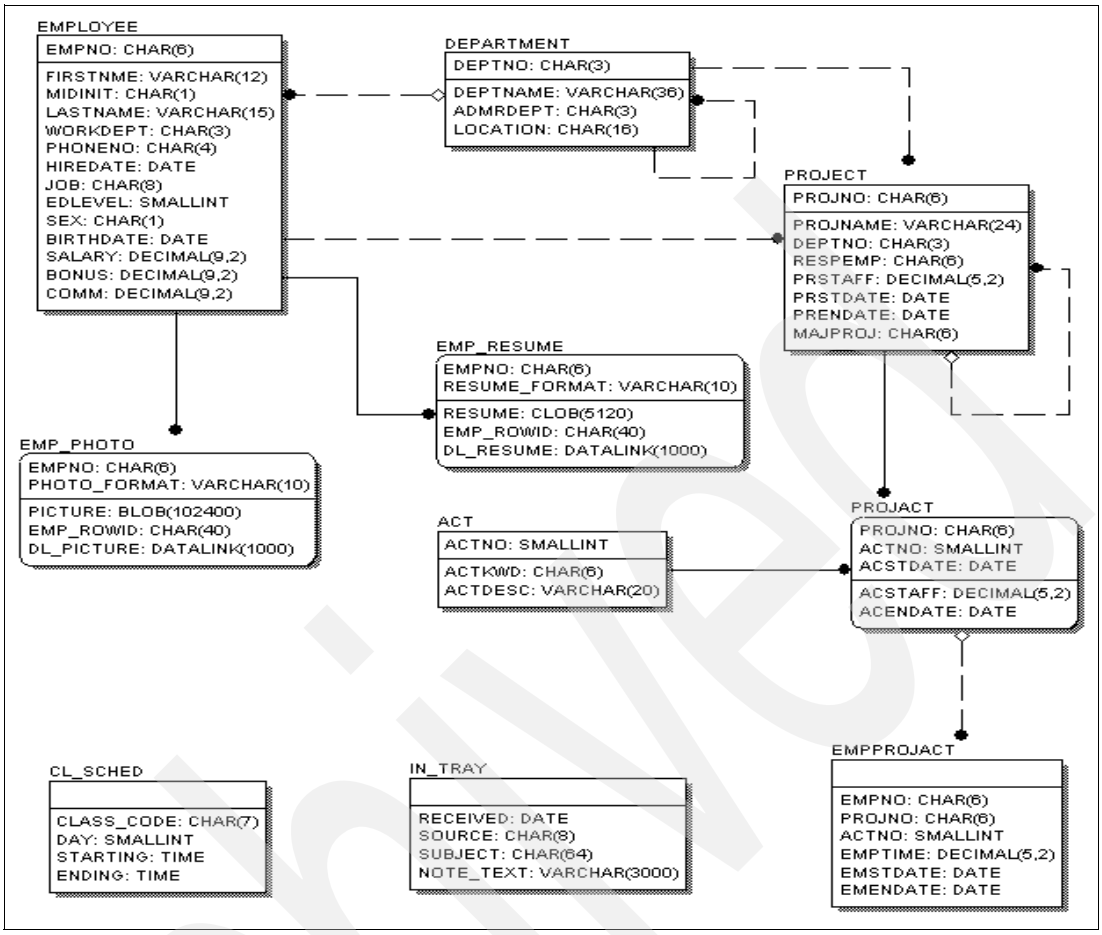


Figure 1-2 Sample schema: Entity-relationship diagram

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. Three other tables are 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 we just described.

2

Introducing iSeries Navigator

This chapter introduces you to iSeries Navigator and its capabilities to administer with a graphical interface the iSeries resources. It also lists the major iSeries Navigator components. In addition, this chapter explains, what can you do with iSeries Navigator and well as what's new in Version 5 Release 2 (V5R2) and in Version 5 Release 3 (V5R3).

2.1 Spotlight on iSeries Navigator

iSeries Navigator is a powerful graphical interface to manage your iSeries servers. Its Navigator functionality includes system navigation, configuration, planning capabilities, and online help to guide you through your tasks.

iSeries Navigator makes operation and administration of the server easier and more productive. It is the only user interface to the new, advanced features of the OS/400 operating system. And, by using Management Central, it can help manage multiple servers from a central system.

Figure 2-1 shows an example of the iSeries Navigator main window.

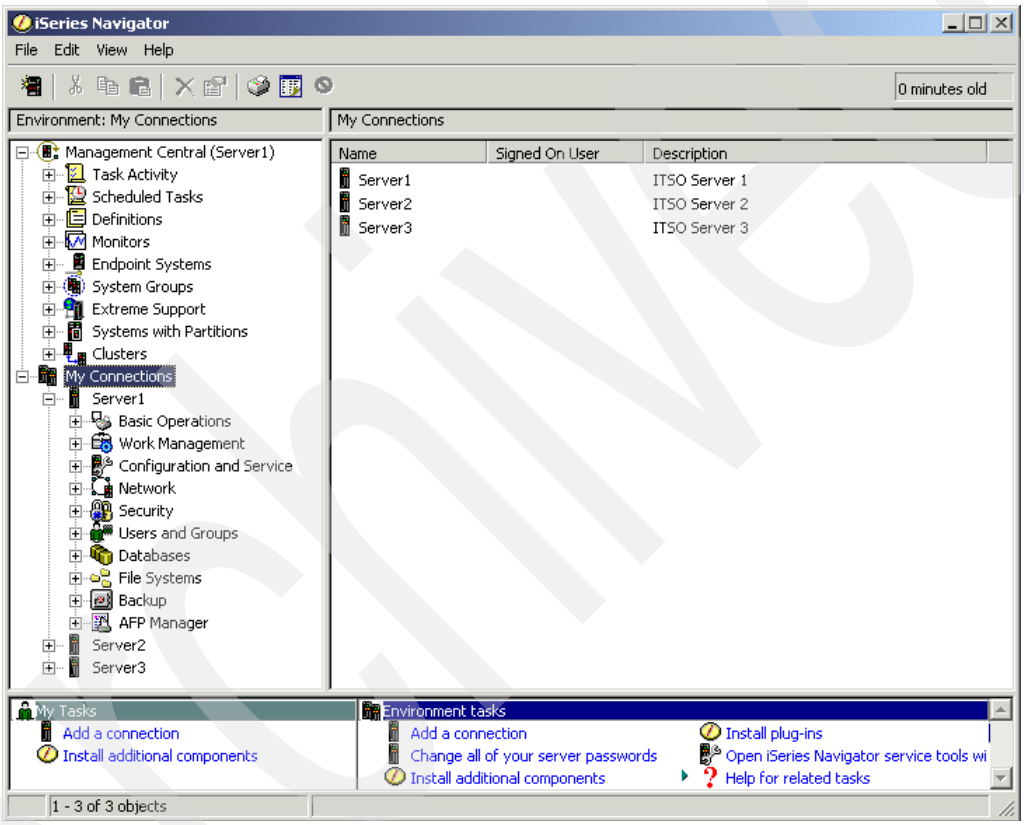


Figure 2-1 iSeries Navigator window

2.2 What you can do with iSeries Navigator

iSeries Navigator provides a powerful suite of functions to handle your system administration tasks. After you install iSeries Navigator, use the online help for step-by-step instructions for all the individual tasks that you can do “the GUI way” in iSeries Navigator.

The following sections explain some of the functions that are available in iSeries Navigator.

Manage multiple systems

Management Central is an integral part of iSeries Navigator. It allows you to manage tasks across one or more servers simultaneously. Simplify your system management by using task scheduling, real-time performance monitoring, managing fixes (or program temporary fixes

(PTFs)), distributing objects, managing users and groups, running commands from a central system, and much more.

Manage logical partitions

Use iSeries Navigator to access your logical partition (LPAR) information, change your LPAR configuration, manage security, and change processing resources without requiring a system restart. On an iSeries server, you can use iSeries Navigator, Dedicated Service Tools (DST), and System Service Tools (SST) to manage your LPARs.

Secure your network

Network security means that your customers, employees, and business partners can obtain the information that they need to do business with you in a secure environment. iSeries Navigator has several integrated features that help to build a strong defense against the security risks that you may encounter along the way. Use iSeries Navigator to configure everything from your basic system security policy to secure end-to-end virtual private network (VPN) connections, filter rules, address translation, Secure Sockets Layer (SSL), enterprise identity mapping, and much more.

Work with databases

You can perform many database tasks using iSeries Navigator. These tasks include creating and maintaining database objects, inserting or changing data, monitoring and analyzing query performance, and creating a map of your database. This option is the main focus of this book.

Connect your network

Use the Network component of iSeries Navigator to configure and manage network communications on your server. Find wizards to easily configure the interfaces, routes, and servers that are necessary for network communications. Verify connectivity using the PING and trace route utilities. Then, choose from a variety of functions to customize the network. For example, you may select IP policies to control and secure your network traffic using Quality of Service (QoS) or VPN. Or you may choose Remote Access Services to create point-to-point connections.

Work with system jobs

System jobs are created by the operating system to control system resources and perform system functions. System jobs run when the iSeries server starts, without user input. These jobs perform a variety of tasks from starting the operating system, to starting and ending subsystems, to scheduling jobs.

2.3 What's new for iSeries Navigator in V5R2

The new features in V5R2 include:

- Management Central
  - Business-to-business (B2B) activity monitors
  - File monitors
  - Synchronized date and time
  - Automatic monitor restart
- System values
  - New restore category
  - Lock function for security related system values

- Wireless
  - Detailed Management Central task summary information
  - Connection customization
- Application administration
  - Ability to centralize your application administration settings on a single iSeries server
- Work management
  - System status
  - Output queues
  - Threads
  - Transactions
  - Last SQL statement
- Hardware
  - Tape devices
  - Direct access storage device (DASD) parity sets
- Network
  - Support for IPv6 and a new wizard to configure IPv6
  - Lines folder to show all physical interfaces
  - Ability to create your own user-defined servers
- Security
  - Enterprise Identity Mapping (EIM)
  - Network authentication service wizard
  - Object signing and signature verification
- Database
  - SQL Assistant
- Backup and Recovery and Media Services
  - New move policy wizard
  - Global policy properties

A complete description of these new features is beyond the scope of this redbook. Consult the following resources for details about iSeries Navigator and its new features:

- iSeries Information Center

  http://www.iseries.ibm.com/infocenter
- iSeries Navigator

  http://www-1.ibm.com/servers/eserver/iseries/navigator/

2.4 What's new for iSeries Navigator in V5R3

The new features in V5R3 are:

- Management Central
  - Enhanced SSL and authentication (sign on to one system, be recognized by others)
  - High availability for Management Central when running in a cluster environment
  - Synchronize Functions wizard to duplicate EIM and Kerberos configurations from a model system to your endpoints
- Intelligent Agents
- Work Management
  - Open file system objects (view and work with open file system objects for an active job)
- Disk Management
  - Print the graphical view
  - Perform geographic mirroring of independent disk pools to systems at other sites
- Network
  - Create and manage mapping policies
  - Test EIM mappings
  - Support for x.509 and digital certificates
- Security
  - Network authentication service (create and manage keytab entries)
- Database

  All of the following new features are discussed in detail in Part 2, "Database administration" on page 15.
  - Sequence objects, partitioned tables, check pending constraints, materialized query tables, etc.
  - Objects organized by container
  - Stop, resume, and view progress of table reorganization
  - View the SQL statements used to perform database actions using Show SQL
  - Library support enhanced to support SQL object schema
  - Use iSeries graphical debugger to launch and run form SQL scripts
- Integrated file system (IFS)
  - Support for scanning files using third-party virus scanning software
- Backup and Recovery and Media Services (BRMS)
  - View and save Hypertext Markup Language (HTML)-based reports
  - Reclaim media
  - Manage devices
  - Backup policy omits of folders, libraries, and library objects
  - Restore printer output and archive objects

In addition to these new features, you can now access and perform a subset of iSeries Navigator tasks through an Internet Web browser. The interface may look slightly different, but the tasks that you can perform are the same. The following iSeries Navigator functions are available to work with on the Web:

- Messages
- User Jobs
- Printer Output (in Basic Operations)
- Active Jobs

- Server Jobs
- Output Queues
- Subsystems (in Work Management)
- System Values and Time Management (in Configuration and Service)
- Database Objects in (Databases)

For a complete description about iSeries Navigator, see the following IBM Redbooks:

- *Managing OS/400 with Operations Navigator V5R1 Volume 1: Overview and More*, SG24-6226
- *Managing OS/400 with Operations Navigator V5R1 Volume 2: Security*, SG24-6227
- *Managing OS/400 with Operations Navigator V5R1 Volume 3: Configuration and Service*, SG24-5951
- *Managing OS/400 with Operations Navigator V5R1 Volume 4: Packages and Products*, SG24-6564
- *Managing OS/400 with Operations Navigator V5R1 Volume 5: Performance Management*, SG24-6565

Part 2

Database administration

iSeries Navigator offers a window-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 (UDB) for iSeries database administration functions that are available through the iSeries Navigator graphical interface that come packaged with iSeries Access Express for Windows® Version 5 Release 3 (V5R3).

This part covers the following topics:

- Database functions using iSeries Navigator
- Using Database Navigator
- Reverse engineering and Generate SQL
- Visual Explain

3

Working with databases using iSeries Navigator

This chapter covers many of the common database administrator (DBA) tasks and functions that can you can perform by using iSeries Navigator. It provides helpful information about the following topics:

- DB2 Universal Database (UDB) for iSeries through iSeries Navigator
- Database objects in iSeries Navigator
- Schemas
- Tables
- Views
- Aliases
- Indexes
- Triggers, procedures, functions, and distinct types
- Sequences
- Packages
- Journal management
- Reorganizing tables
- Transactions

3.1 Getting started with iSeries Navigator

iSeries Navigator is a powerful graphical interface to the iSeries server. It provides an array of functions to handle your system administration tasks including database administration. Most of the iSeries Navigator database operations are based on Structured Query Language (SQL), but you do not need to fully understand SQL to perform the tasks.

> **Tip:** You can use either the Show SQL or Show Command buttons from most of the database windows in iSeries Navigator to see the actual SQL statement or CL command that is used to perform a task.

3.1.1 Overview of DB2 UDB for iSeries through iSeries Navigator

From the Database function, you can manage multiple databases. This includes the system database, user databases, and remote databases.

Using iSeries Navigator, you can perform the following database tasks among others:

- Create and manage various database objects through various containers provided in iSeries Navigator
- Create queries from scratch, or reverse engineer and generate queries, for your database objects and run them using Run SQL Scripts Center
- Monitor and tune your databases and queries using Database Monitors, Statistic Requests, and Visual Explain interfaces
- Map and explore your databases using Database Navigator maps
- Secure your database objects, set up referential integrity, and manage check pending constraints
- Work with transactions including both global and database transactions

> **Note:** The Database component of iSeries Navigator is not installed by default when you choose the *Typical* installation option of iSeries Access for Windows. If the Database component is not currently installed, you can run Selective setup to install it as discussed in the iSeries Information Center at:
>
> http://www.iseries.ibm.com/infocenter
>
> When you reach this Web site, click **Connecting to iSeries** →**iSeries Navigator** →**Getting to know iSeries Navigator** →**Installing iSeries Navigator**.

You access most of the database functions described in this and subsequent chapters from the context menus at various levels under the Databases tree of iSeries Navigator. Figure 3-1 shows the some of the options that available under Databases tree in iSeries Navigator.

This book emphasizes the actions that are most significant on all the menus. Refer to iSeries Navigator online help for common actions, such as Explore, Open, Shortcuts, Print, etc.

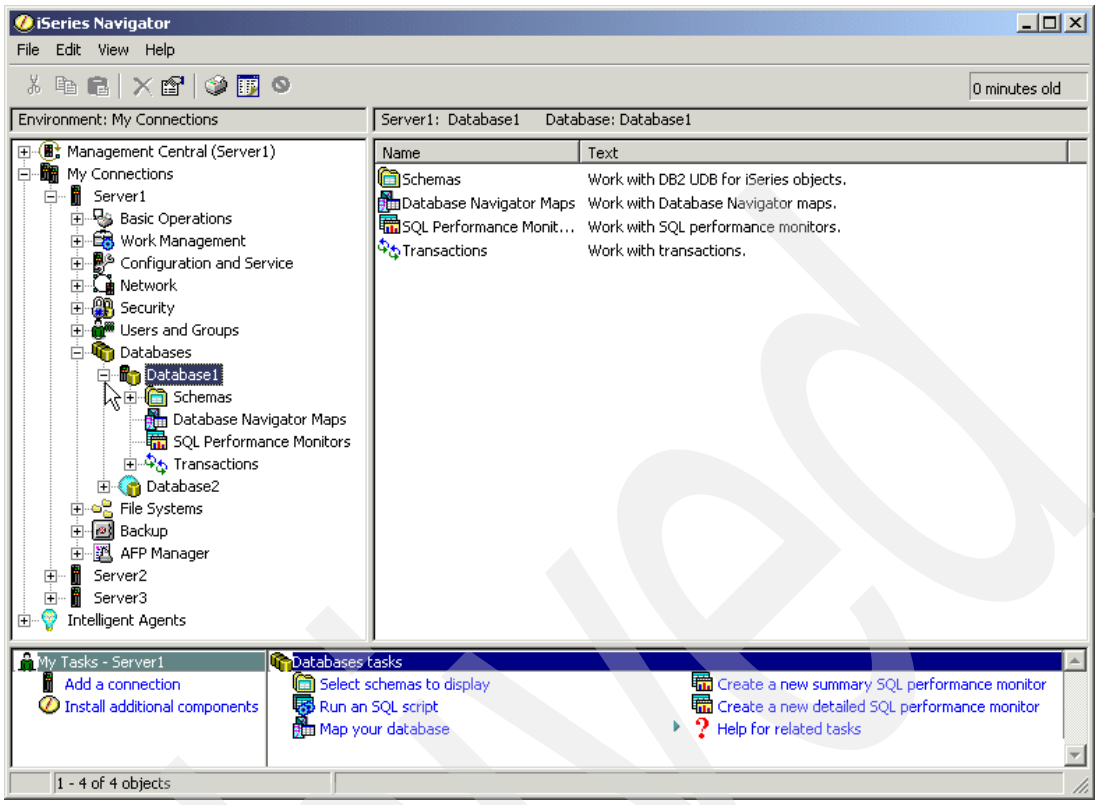


Figure 3-1 Database functions overview

3.1.2 Introduction to database objects in iSeries Navigator

You can create and manage many objects in iSeries Navigator. Among the objects that you can create and manage are:

- Schemas
- Tables (including materialized query tables and partitioned tables)
- Aliases
- Indexes
- Views
- User-defined distinct types (UDTs)
- User-defined functions (UDFs)
- Triggers
- Procedures
- Sequences
- SQL packages
- Journals
- Journal receivers
- Constraints

3.2 Working with database objects

Most database objects are created from the schema container object. Follow these steps to navigate to the containers under a schema as shown in Figure 3-2:

1. Under My Connections, expand ***your system name*** →**Databases** →**Schemas**.
2. Expand the schema that you want.
3. You see various containers underneath. Right-click any of these containers and the objects in them to see the actions that you can perform on them.

Tip: You can also create other database objects. Simply right-click the schema of preference and select **New**.

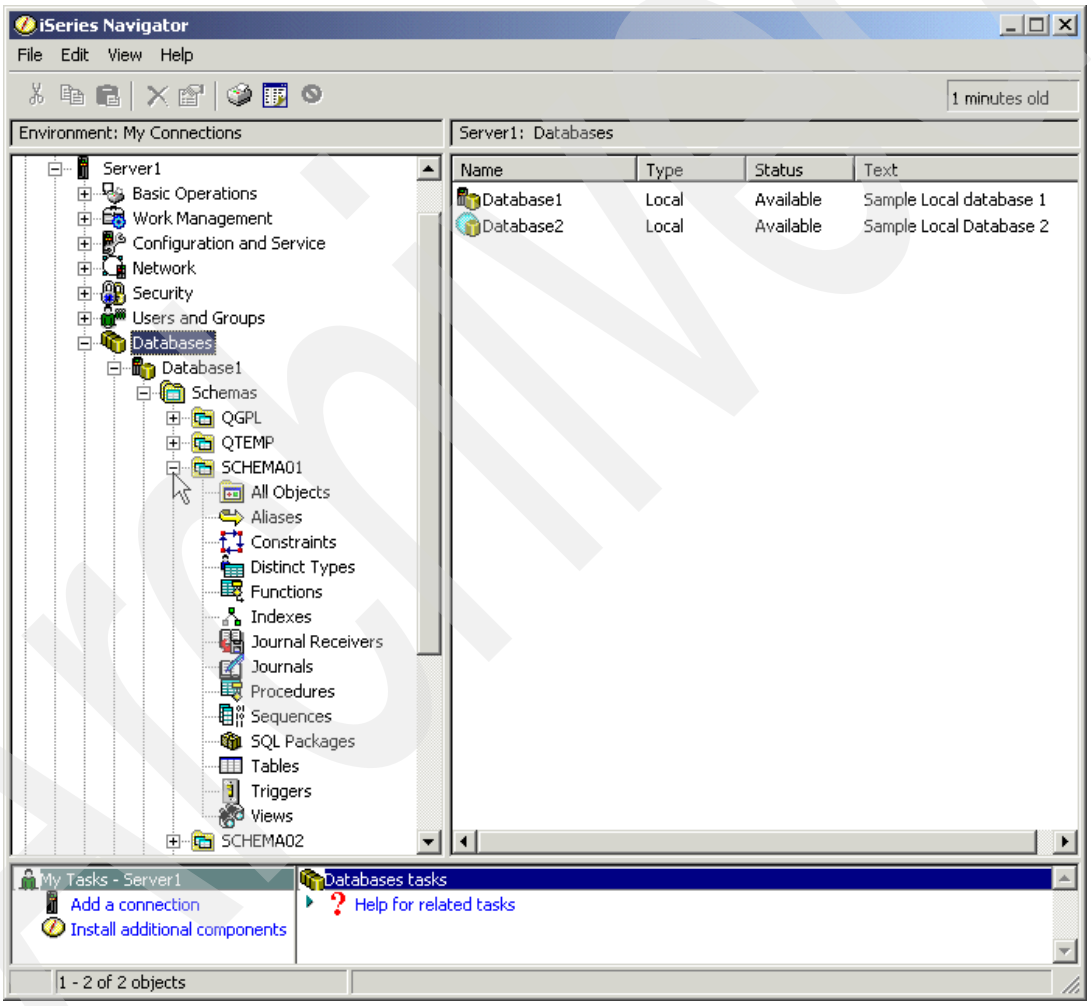


Figure 3-2 Navigating to the containers under a schema

3.2.1 Schemas

A *schema* is a database structure that contains other databases objects. Some of the objects that a schema may contain include tables, views, journals, and journal receivers. A schema is also called a *collection* or a *library*.

Note: A schema may contain any of the supported iSeries object types. However, under the iSeries Navigator's Database interface, you only work with objects that are related to database support.

Editing the list of schemas displayed

The list of schemas, under the Schemas tree, is built from the initial library list parameter of the job description. This description is associated with the iSeries user profile that is used to connect from iSeries Navigator. By default, only the schemas in the user portion of your OS/400 library list are included. The list includes any other schema that you specified to add here in previous iSeries Navigator sessions.

Note: The list of schemas displayed in iSeries Navigator is maintained separately on OS/400 in the QUSRSYS/QAUGDBLL table for all the users. Any changes that you make to this list do *not* cause your OS/400 library list to change. The list is persistent across future iSeries Navigator sessions of the same user.

The list of the schemas that is displayed can be modified in many ways in iSeries Navigator. The following sections discuss some of the ways to modify a list of schemas.

Adding to the list of schemas displayed

To add to the list of schemas, follow these steps:

1. Select **Schemas**, right-click, and click **Select Schemas to Display** as shown in Figure 3-3.

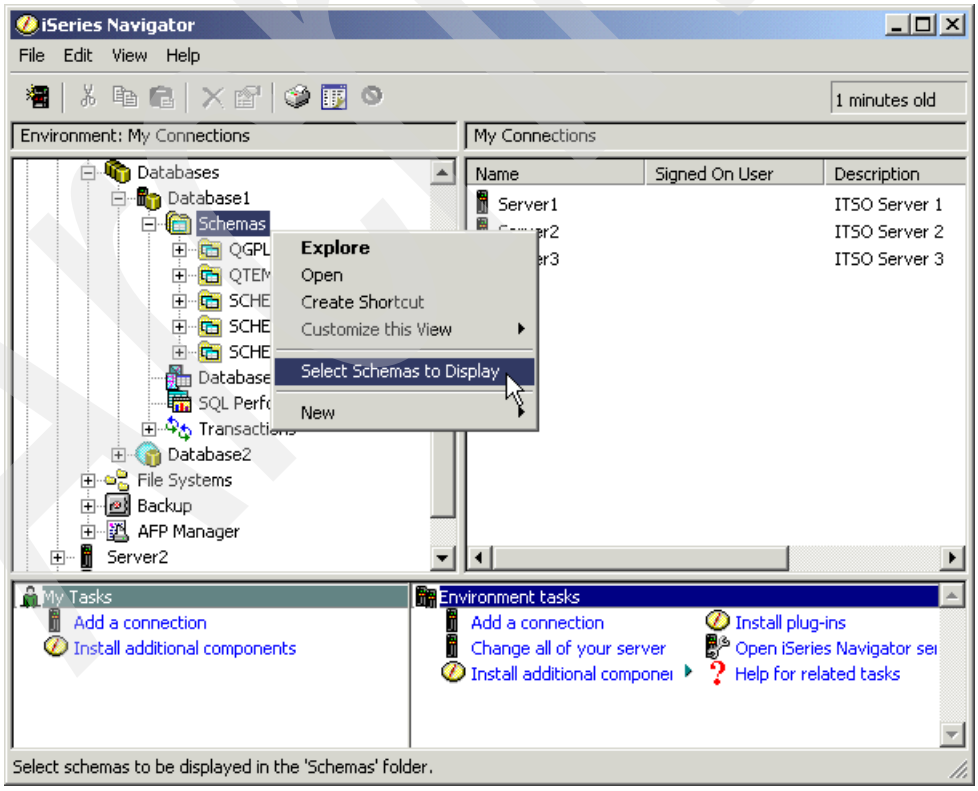


Figure 3-3 Navigating to Select Schemas to Display

2. In the Select Schemas to Display window (Figure 3-4), select the **Enter schema names** option and type a name. Click **Add->** and then click **OK** to add the name to your list of schemas.

> **Tip:** To add multiple schemas, type the name of each schema in the text box separated by a space.

3. Optionally, you can search for schemas:
   a. Select **Search for schemas**. Type either the full name or the generic name as shown in Figure 3-4 and click **Search**.
   b. When the matching schemas are displayed, select the schema that you want. Click **Add->** to add to your list of schemas displayed. To select multiple schemas, hold down the Shift key and click the schema names. Click **OK** after you add the necessary schema or schemas.

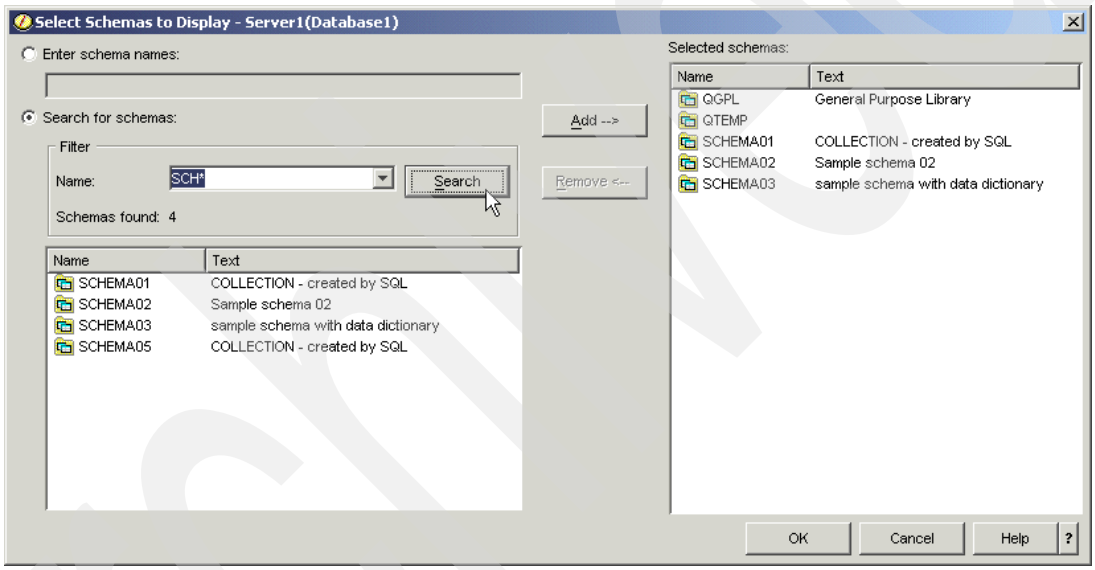


Figure 3-4 Select Schemas to Display window

Removing schemas from the displayed list

There are two ways to remove a schema from the list:

- From the Select Schemas to Display window

  The schemas that are currently in your display list are shown in the right pane under Selected schemas (Figure 3-5). Select the schema or schemas that you want to remove. Click **Remove<-** and then click **OK**.

> **Important:** In order for any changes that you make to the list to take effect, you must click **OK** in this window.

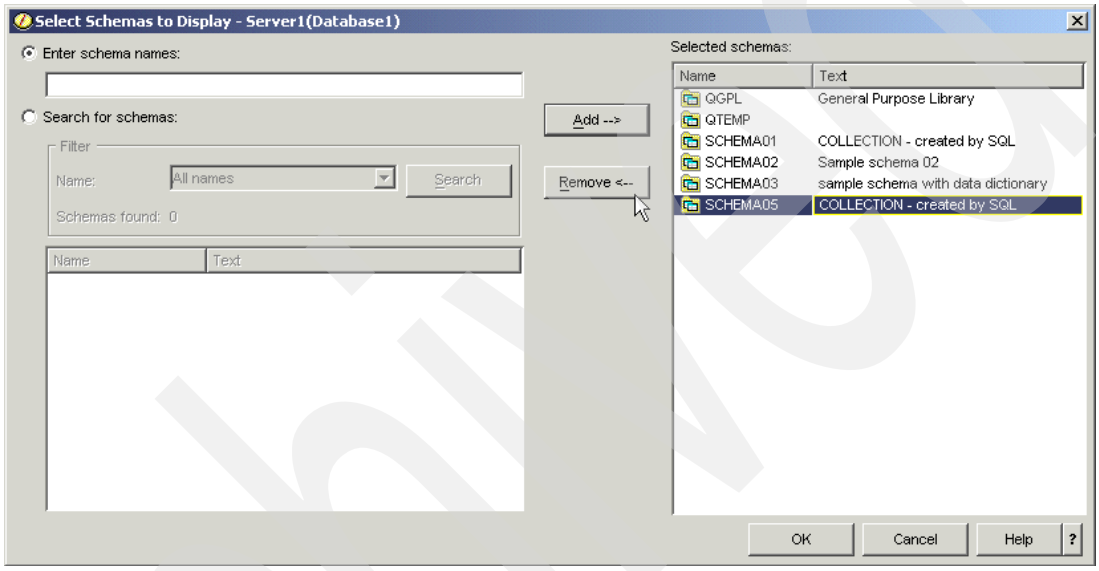


Figure 3-5 Removing a schema using the Select Schemas to Display window

- In the iSeries Navigator window, under the Schemas tree

  Under Schemas, right-click any schema in the list and select **Remove from List** as shown in Figure 3-6.

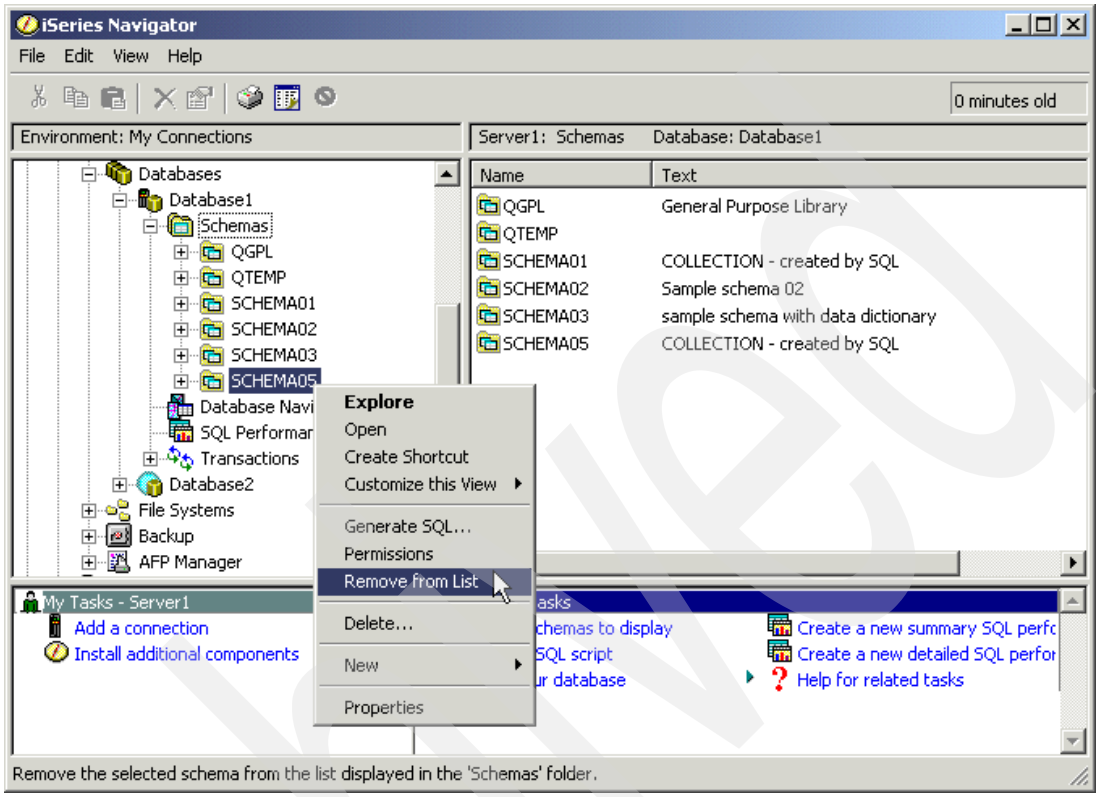


Figure 3-6 Removing a schema using the Schema context menu

Creating a schema using iSeries Navigator

You can create a new schema from iSeries Navigator as explained here:

1. Select **Schemas**, right-click, and select **New** →**Schema**. See Figure 3-7.

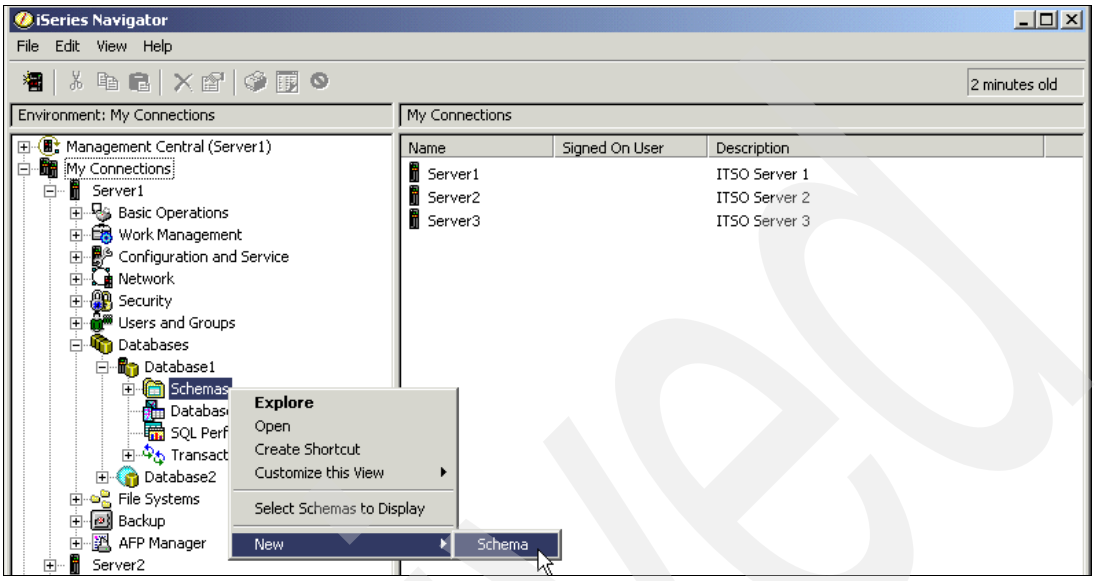


Figure 3-7 Creating a Schema using iSeries Navigator

2. The New Schema window (Figure 3-8) opens. Follow these steps:
   a. Select **Add to displayed list of schemas** to add this schema to the list of schemas displayed. This option is selected by default in the New Schema window.
   b. Optionally, create the schema as a standard library. Select **Create as a standard library**. When you select this option, default objects of a schema, such as catalog views, journal, and journal receiver, are not created.
   c. Optionally, create an IDDU data dictionary in the schema. Select **Create a data dictionary**.
   d. From the Create in list, select the disk pool to specify where you want to create the schema. The default for system database is **System disk pool**. For a user-defined database, the drop-down list shows any available independent disk pool groups.
   e. In the Text field, type the description for the schema.

   > **Tip:** Click the **Show SQL** button in the New Schema window to see which command or statement is executed for each of these options.

   f. Click **OK** to create the new schema.

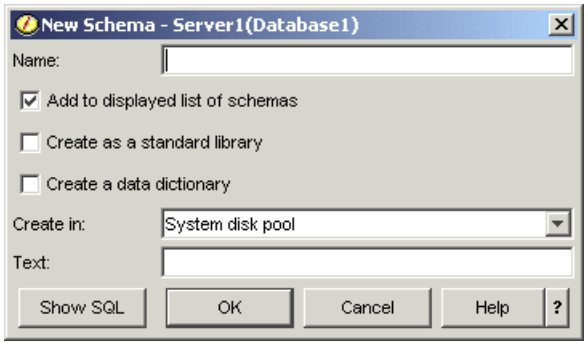


Figure 3-8 New Schema window

Deleting a schema

Using iSeries Navigator, you can delete database objects including schemas. However, when you delete a database object, it may affect other objects that depend on it. For example, if you delete a table that is used in a view definition, you are prompted to specify whether you also want to delete that view definition.

You can also delete a schema as explained here:

1. Right-click the schema that you want to delete and choose **Delete**.
2. In the Confirm Object Deletion window (Figure 3-9), choose one of the following options:
   - Select **Take default action on all dependents** to specify that dependent objects will assume the default action for the deleted object. For example, any journals that may be in the schema are detached from their journal receivers. Then both journals and journal receivers are deleted provided that they are not in use.
   - Select **Delete all dependents** to delete dependents that exist in other schemas.
   - Select the **Do not delete selected object** option if you do not want to delete a schema when dependent objects exist.

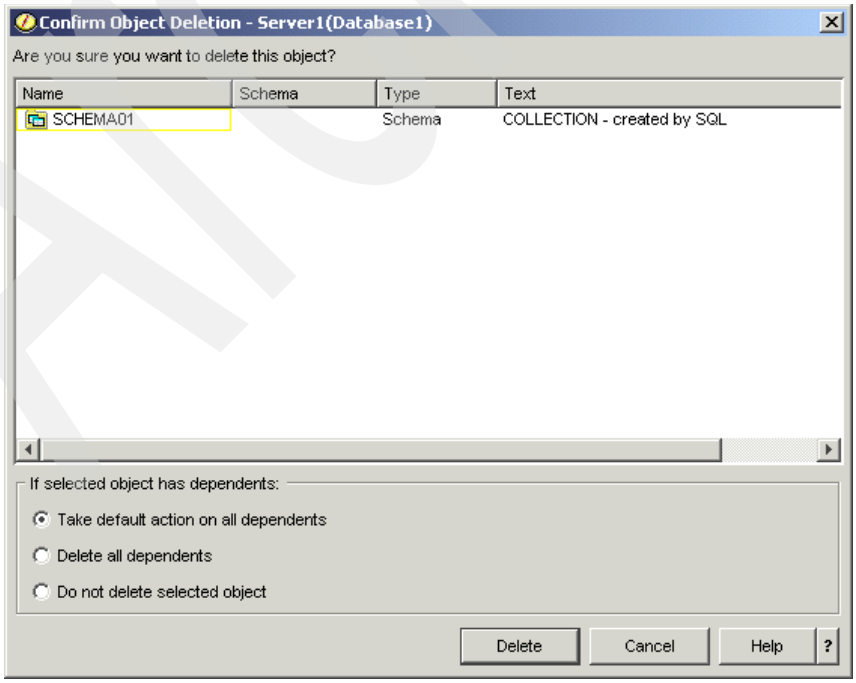


Figure 3-9 Confirm Object Deletion window

Generate SQL from a schema

To reverse engineer and generate SQL for database objects in a schema, select the schema that you want, right-click, and select **Generate SQL** (Figure 3-10). For a detailed discussion, see 6.2, “Generating SQL from the schema in iSeries Navigator” on page 146.

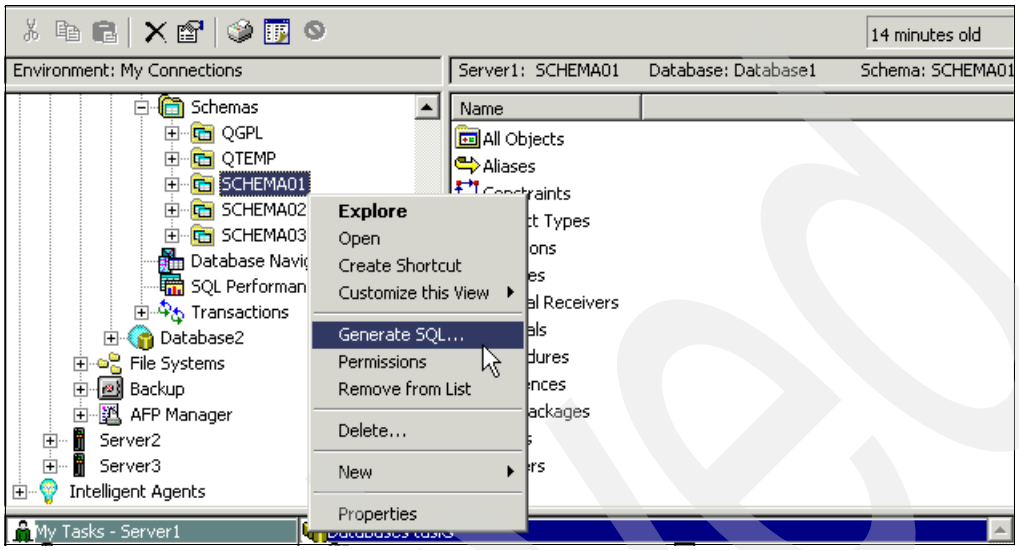


Figure 3-10 Generate SQL from a schema

Working with permissions for schemas

To set permissions for a schema and its objects, follow these steps:

1. Expand **Schemas**. Select the schema that you want, right-click, and select **Permissions** as shown in Figure 3-11.

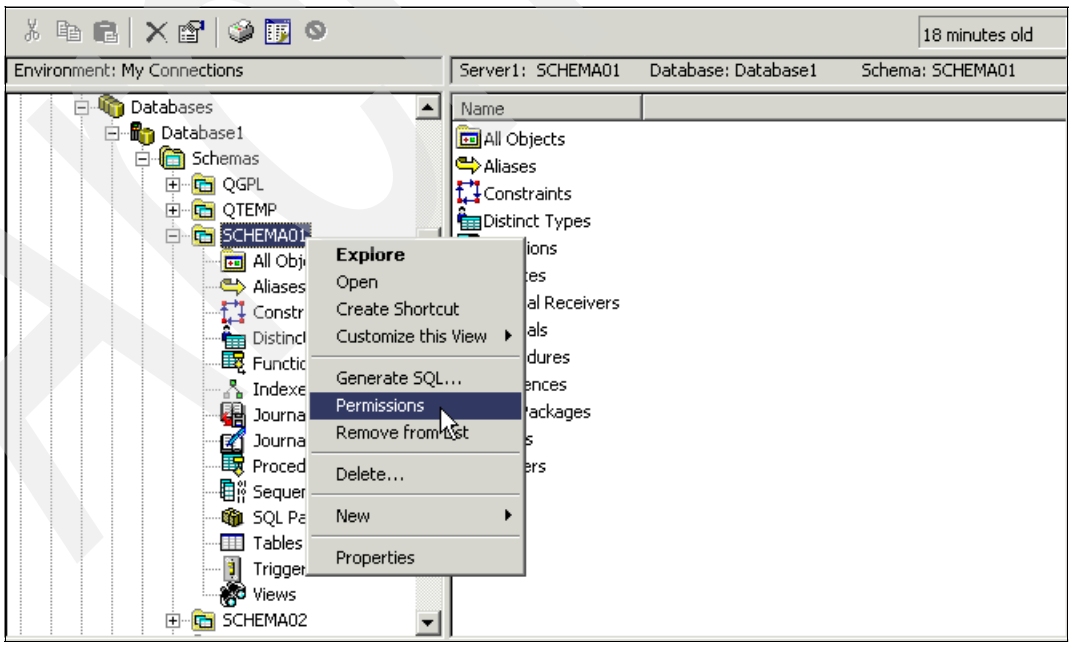


Figure 3-11 Navigating to Permissions for a schema

2. In the Schema Permissions window (Figure 3-12), complete the following tasks:
   a. Under Authorities view, select **Details** to set all types of permissions including data authorities.
   b. Click the **New Objects** button to set the default public authority for objects that may be created in the future.
   c. Click **Apply**.
   d. Click **OK**.

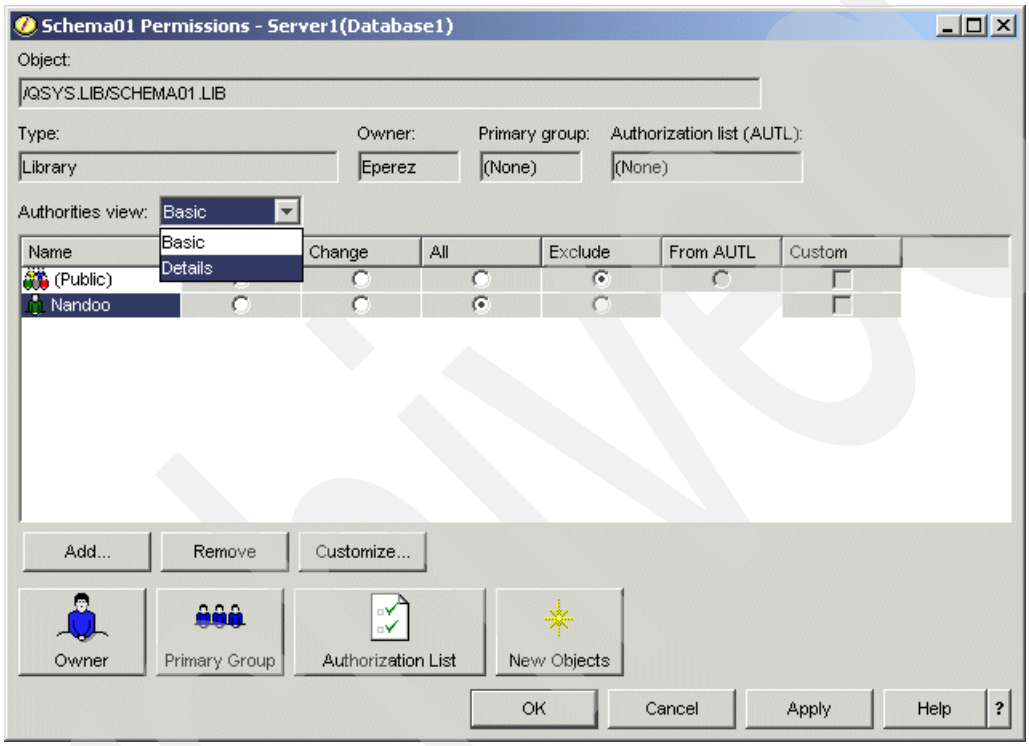


Figure 3-12 Schema Permissions window

For more information, visit the iSeries Information Center at:

http://www.ibm.com/eserver/iseries/infocenter

When you reach this Web site, select **Database** →**Programming** →**Database programming** →**Set up database files** →**Securing a database** →**Specifying public authority** →**Setting a default public authority for new files using iSeries Navigator**.

3.2.2 Tables

A table is a basic database object that is used to store information. It is a two-dimensional arrangement of data that consists of rows and columns.

Creating a table

In the New Table definition window of iSeries Navigator, you can define constraints and table partitions. When you create the table, you can also create indexes, aliases, and triggers from the context menu of the table.

To create a table, you can right-click a schema and choose the context menu option, or under a selected schema, select either the **All Objects** container or **Tables** container.

Follow these steps to create a sample table:

1. In the iSeries Navigator window, expand ***your system name*** →**Databases**. Select the database that you want to work with.
2. Expand **Schemas** and select the schema that you want to work with.
3. Under the schema that you selected, select the **Tables** container, right-click, and choose **New** →**Table**.
4. In the New Table definition window (Figure 3-13), follow these steps:
   a. Type a name and description for the table.

      Optionally, you can type a name of up to 10 characters in System table name field, if you do not want the System-generated name.

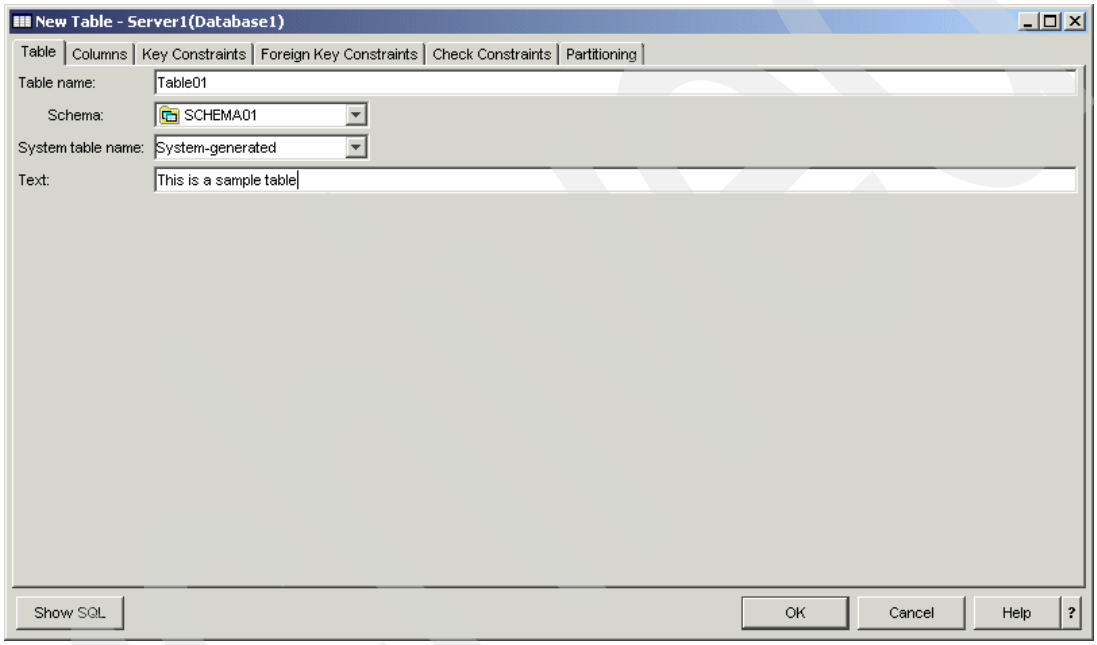


Figure 3-13 New Table definition window

   b. Click the **Columns** tab of the window.
   c. On the Columns page, click **Add**.
   d. The New Column definition window (Figure 3-14) opens. Enter a name and other details such as data type, length, description, etc. and click **Add**.

      Note: For more information about various data types and SQL special registers that you can use to define a column, visit the iSeries Information Center at:

      `http://www.iseries.ibm.com/infocenter`

      When you reach this Web site, select **Database** →**Reference** →**SQL Reference** →**Language Elements**.

   e. Repeat step d to add multiple columns and then click **Close**.

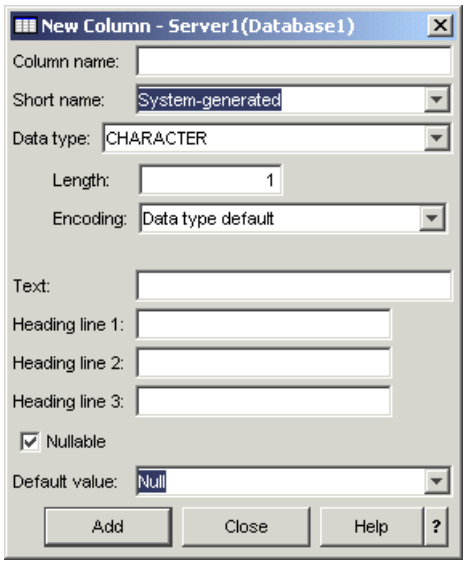


Figure 3-14 New Column definition window

5. Back in the New Table definition window, click **OK** to create the table.
6. When the table is created, you can secure the table and or its columns. Simply right-click the table, select **Permissions**, and set the appropriate permissions. Figure 3-15 illustrates a sample of column security for the Ssn column.

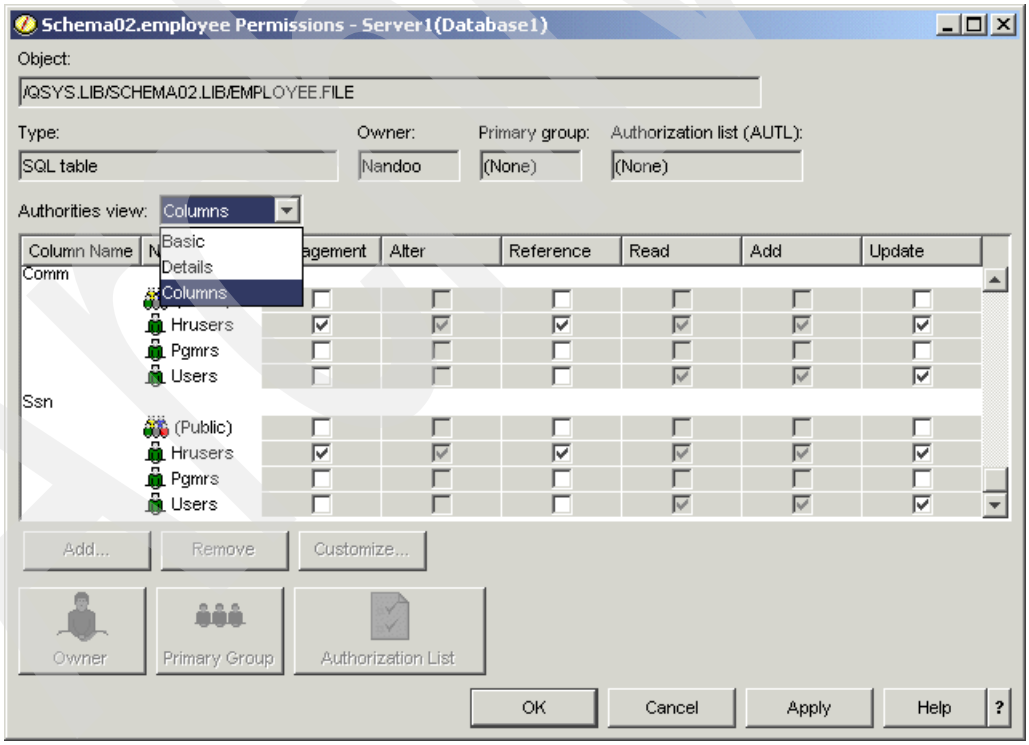


Figure 3-15 Setting permissions for the columns of a table

Important: For important information related to SQL naming, refer to the iSeries Information Center at:

http://www.iseries.ibm.com/infocenter

When you reach this Web site, select **Database** →**Reference** →**SQL Reference** → **Language Elements** →**Naming Conventions**. In addition, for a detailed explanation about the rules for system name generation, see **Database** →**Reference** →**SQL Reference** →**Statements** →**CREATE TABLE**.

Modifying a table definition

To modify a table definition, select a table, right-click, and select **Definition** (Figure 3-16).

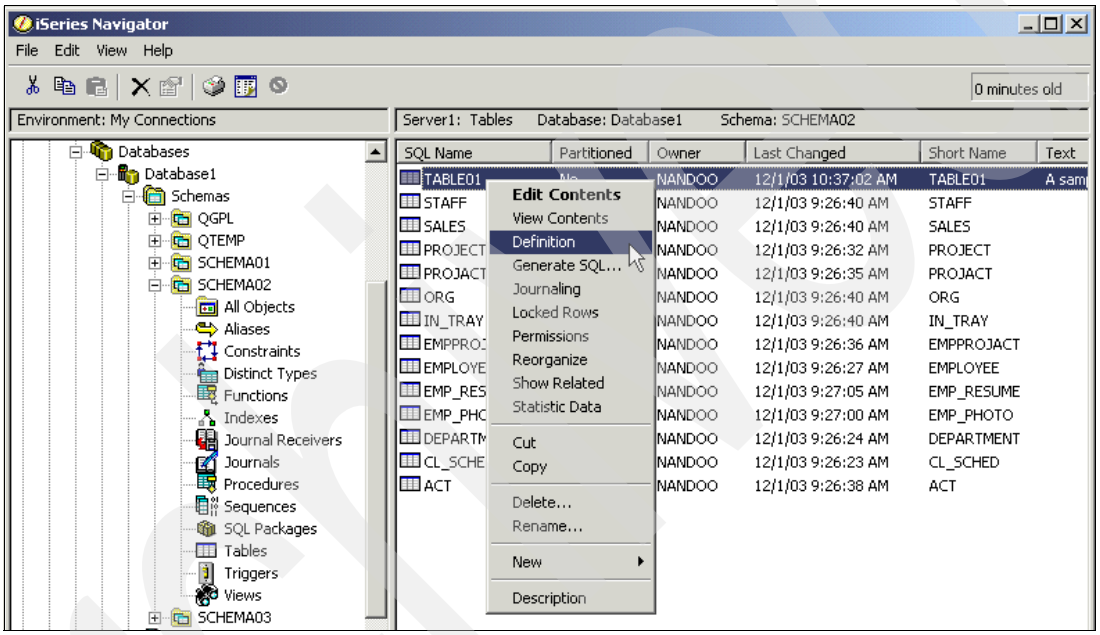


Figure 3-16 Modifying a table definition

Copying a column definition from another table or view

To copy a column definition, follow these steps:

1. In iSeries Navigator, select a table, right-click, and select **Definition** as explained in the previous section.
2. In the New Table window (Figure 3-13 on page 29) that opens, select the **Columns** tab.
3. On the Columns page, click **Browse**.
4. The Browse Columns window (Figure 3-17) opens. Complete these tasks:
   a. In the left pane, expand a schema and select a table.
   b. In the right pane, select a column. To select multiple columns, hold down the Shift key. Click **Add** to copy the definition or definitions.

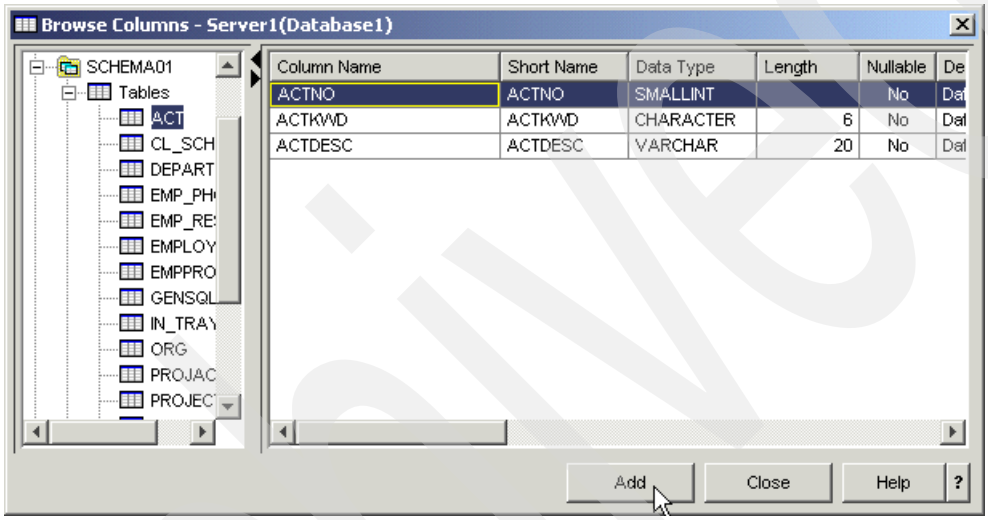


Figure 3-17 Browse Columns window

   c. Repeat step b to select columns from other tables and views.
   d. Click **Close** when you are done.
5. If you are done, click **OK**.
6. Click **Add** to continue adding new column definitions from the Column tab of the Table definition window.

Adding an identity column to a table

Identity columns provide a way to generate a sequence for every row in a table. Every time that a new row is added to a table with an identity column, the identity column value in the new row is incremented (or decremented) by the system. Refer to “Differences between identity columns and sequences” on page 67 for more information.

> **Restriction:** Only one identity column per table is allowed. When you alter a table definition, you can specify only a column that you are adding as an identity column. You cannot change an existing column to an identity column.

Follow these steps to set a column as an identity column:

1. In the Table definition window, select the **Columns** tab.
2. On the Columns page, click **Add**.
3. The New Column definition window (Figure 3-18) opens. Specify the column name, and select a short name. For Data type, select **NUMERIC**. When you select this option, the window changes instantly to appear like the example in Figure 3-19.

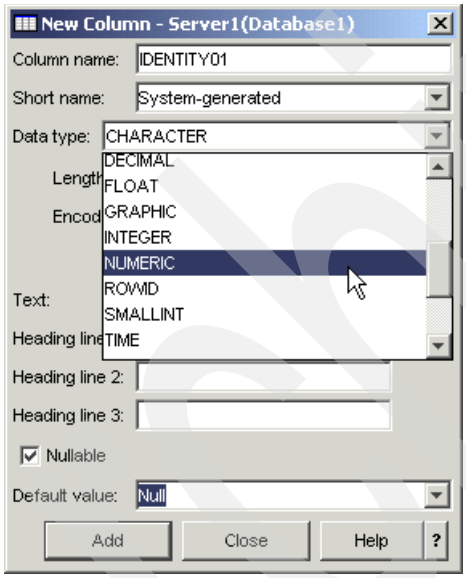


Figure 3-18 Selecting the data type for the new identity column

4. As shown in Figure 3-19, complete these tasks:
   a. Select the **Set as identity column** option.
   b. In the Identity characteristics box, specify the appropriate values for Increment, Starting value, Minimum value, and Maximum value.
   c. Select the **Cycle values when the maximum or minimum value is reached** option to restart the sequence when boundaries are reached.
   d. Select the **Generate values in the order requested** option to generate the sequence for all the requests in the order they are received by the system. To improve performance, we recommend that you leave this option as cleared.
   e. The Database manager generates column value option specifies that the database manager generates the value for the identity column *always*. For this option, from the drop-down list, select **By default** to specify that the database manager generates a value for the column when a row is inserted only if a value is not specified for the column. If a value is specified, the database manager uses that same value.
   f. The Values to cache option specifies the maximum number of values of the sequence that the database manager pre-allocates and keeps in memory. Pre-allocating and storing values in the cache improves the performance of inserting rows into a table.
   g. Click **OK**.

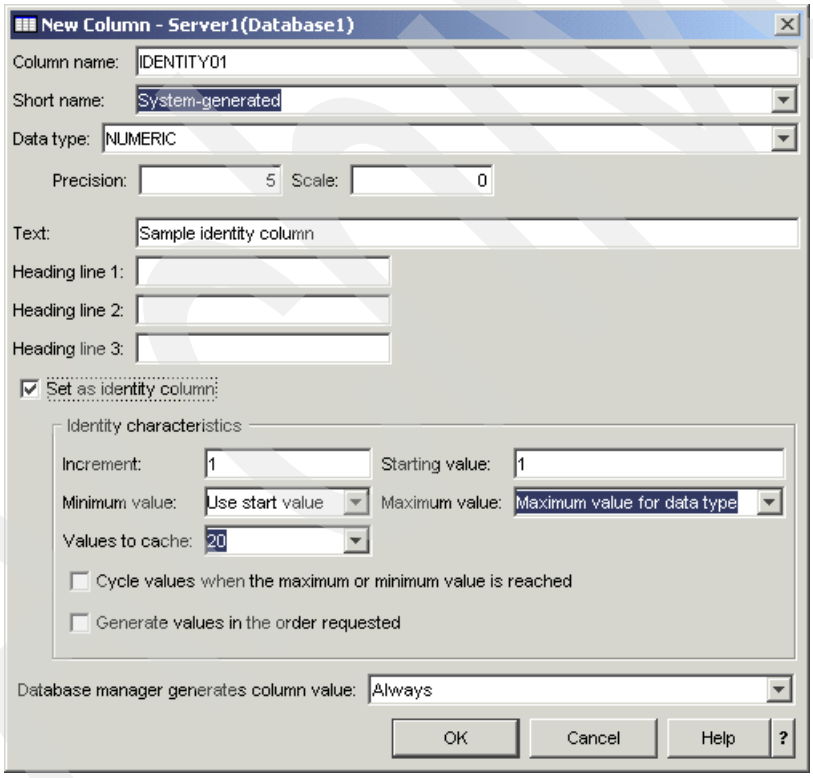


Figure 3-19 Identity characteristics

Important: During a system failure, all cached identity column values that are yet to be assigned are lost. Therefore, they will never be used. Also, if the Restart with value option is selected while altering an identity column, the cached values are lost. Therefore, the value specified for the cache also represents the maximum number of values that can be lost during a system failure or alteration of the identity column.

In iSeries Navigator, the options that define the characteristics of an identity column are similar to those in the Sequence object definition window. For more information about these options, refer to “Creating a sequence” on page 68.

Creating a partitioned table

The data in a partitioned table is contained in one or more local partitions (members). Partitions inherit the design characteristics of the table on which they are based. Each characteristic can contain the maximum number of rows for a table, which is approximately 4294967000. Therefore, partitioning allows your table to be much larger than a non-partitioned table. You can have up to a maximum of 256 partitions.

Partitioning may also enhance the recoverability and manageability of your database. Each partition can be saved, restored, exported from, imported to, dropped, or reorganized independently of the other partitions. In addition, dropping a partition provides significantly better performance than deleting the same rows from a non-partitioned table.

Note: You must have DB2 Multisystem installed on your iSeries server to take advantage of partitioned tables support.

Two mechanisms can be specified to determine, into which partition, a specific row is inserted. These mechanisms are range partitioning and hash partitioning.

Range partitioning allows a user to specify different ranges of values for each partition that determines where each partition will begin and end. When a row is inserted, the values specified in the row are compared to the specified ranges to determine which partition is appropriate.

Hash partitioning divides the table into specified number of equally sized partitions. It enables a user to specify a partitioning key on which a hash algorithm is used to determine which partition is appropriate while inserting a row. The partitioning key is a set of one or more columns in a partitioned table that are used to determine in which partition a row belongs. You can have up to a maximum of 120 columns for a partitioning key.

Restriction: A partition table cannot have an identity column or a ROWID type column. Also, if an existing table has referential constraints, you cannot change it to a partitioned table. For more information and other restrictions, see the *DB2 Multisystem* topic in the iSeries Information Center at:

http://www.iseries.ibm.com/infocenter

When you reach this Web site, navigate to **Database** →**Database overview** →**What’s new for V5R3** and click **Partitioned tables** on the right to see the restrictions.

Partition by hash

Hash partitioning places rows at random intervals across a user-specified number of partitions and key columns.

Follow these steps to create a partition table with hash partitioning:

1. In the Table definition window, click the **Partitions** tab.
2. On the Partitions page (Figure 3-20), complete the following tasks:
   a. Under Partitioning type, select **Partition by hash**. Then, in the Number of partitions field, type the quantity.
   b. Under Partition key columns, from the available columns, select the column that you want and click **Add**. To select multiple columns, hold down the Ctrl or Shift key and click the column names.

   > **Tip:** To see how many columns are displayed and how many are selected, view the status line (indicated at the bottom of the Available columns pane). To do this, in either the Available columns pane or Selected Columns pane, right-click anywhere on the title area for the columns and select **Status line** →**Show status line**. This option is available on all definition windows when applicable.

   c. Optionally, you can reorder the columns for the partition key from the Selected columns pane. You can also remove columns that you do not want in the partition key by selecting a column name and clicking the **Remove button**.
   d. After you have the columns that you want for the partition key in the Selected columns pane, in the Table definition window, click **OK**.

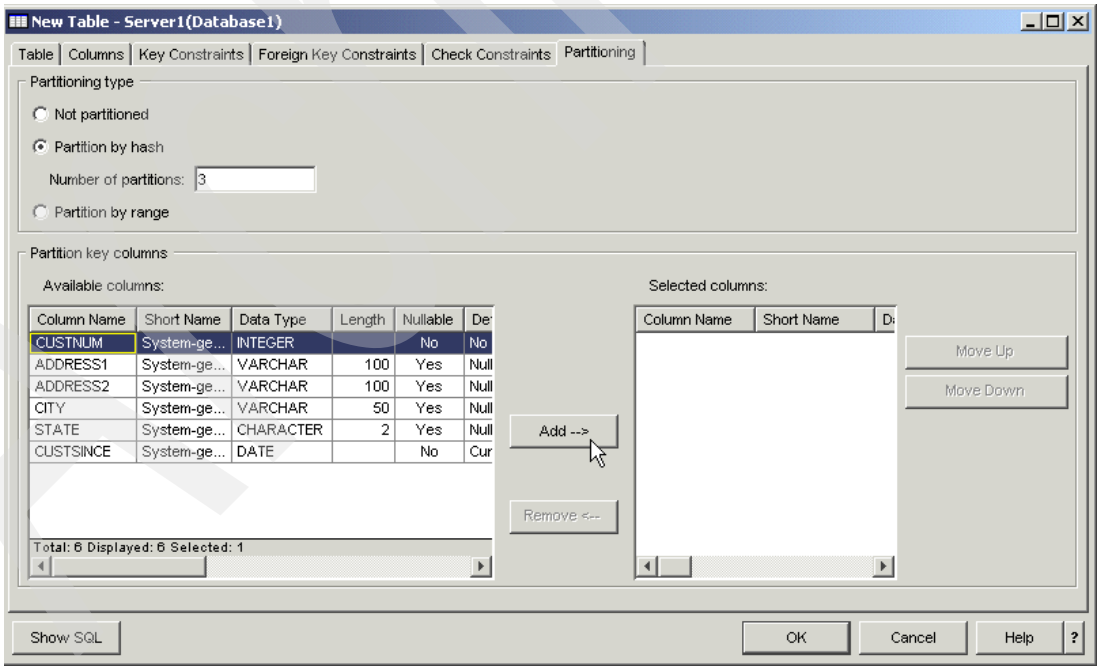


Figure 3-20 Partition by hash

3. To verify the partitions that you created, right-click the table and select **Show Partitions**, or select the definition and click the **Partitioning** tab.

Note: When a partitioned table is created, a system-supplied check constraint is added to each partition. This check constraint cannot be displayed, altered, or removed by the user. For hash partitioning, this check constraint validates that the data is based on the condition `Partition number = MOD(HASH(`*`Partition keys`*`), `*`Number of partitions`*`) + 1`, where the Hash function returns a value between 0 and 1023. The null values are always placed in the first partition.

Partition by range

Range partitioning divides the table based on user-specified ranges of column values. You can specify the ranges of column values that are used to determine the target data partition when inserting a row into the table. Complete the following steps to create a table with range partitioning:

1. In the Table definition window, click the **Partitioning** tab.
2. On the Partitioning page (Figure 3-21), complete these tasks:
   a. Select **Partition by range**.
   b. Under Partition key columns, select the column or columns that you want and click the **Add-->**.
   c. Optionally, click **Set Null Values** to handle null values. *Nulls First* places the null values *before* the first valid value for the datatype. *Nulls Last* places the null values *after* the last valid value for the datatype.
   d. After you select the Partition key columns, under the Partitions section, click **Add**.

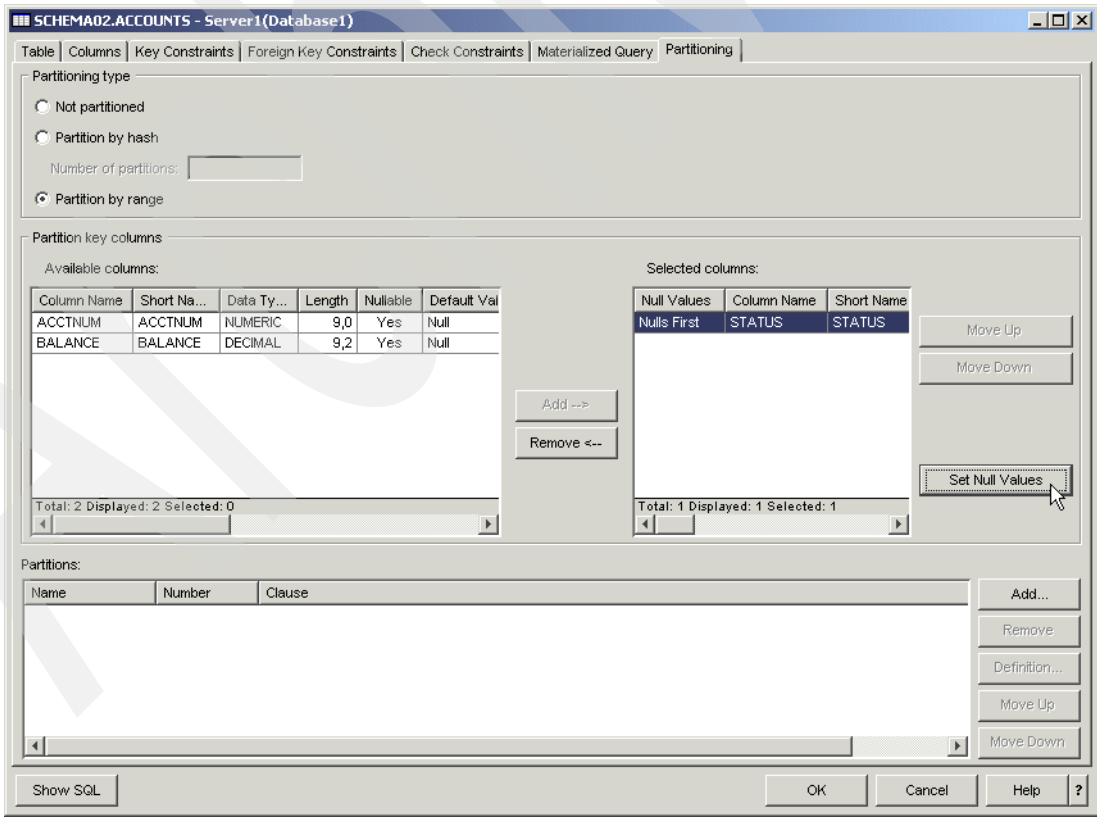


Figure 3-21 Partition by range

3. The New Table Partition window (Figure 3-22) opens. Complete these tasks:
   a. Specify either the Partition name or the Partition number, but not both, to identify each partition.
   b. Select a Start Value and End Value to set the boundaries for the partition.
   c. To make the start and end values inclusive, select the **Include range key start values** and **Include range key end values** check boxes. If you want to exclude the start or end values from the partition, deselect the appropriate option.
   d. Repeat steps a through c to define additional partitions. Make sure to specify the ranges for all the partitions in the correct order.

      Important: The boundaries for range key values must not overlap. In addition, if you leave any range of values when setting the boundaries of the Partition key, then any rows with column values outside the specified range are rejected due to the implicit check constraint enforced by the system. Also, an existing non-partitioned table cannot be partitioned if the column values of the partition key fall outside the range defined.

   e. After you add all the partitions that you want, click **Close**.

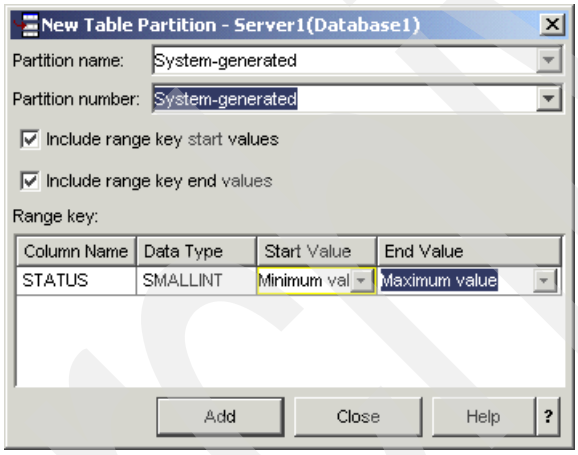


Figure 3-22 New Table Partition window

4. You return to the Table definition window. To make changes to any of the partitions that you defined, select the partition and click the **Definition** button as shown in Figure 3-23. If you are satisfied with the partitions that you defined, click **OK** to create them.

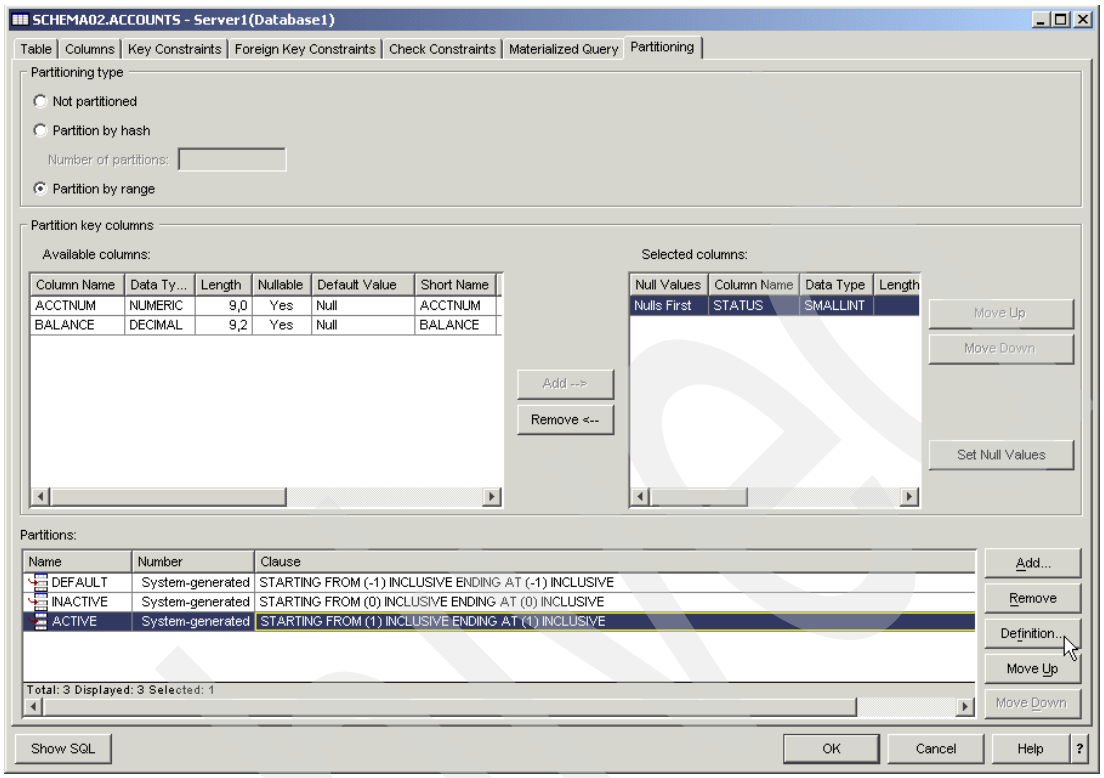


Figure 3-23 List of defined range partitions

After the table is partitioned, you can select the table, right-click, and select **Show Partitions** as shown in Figure 3-24.

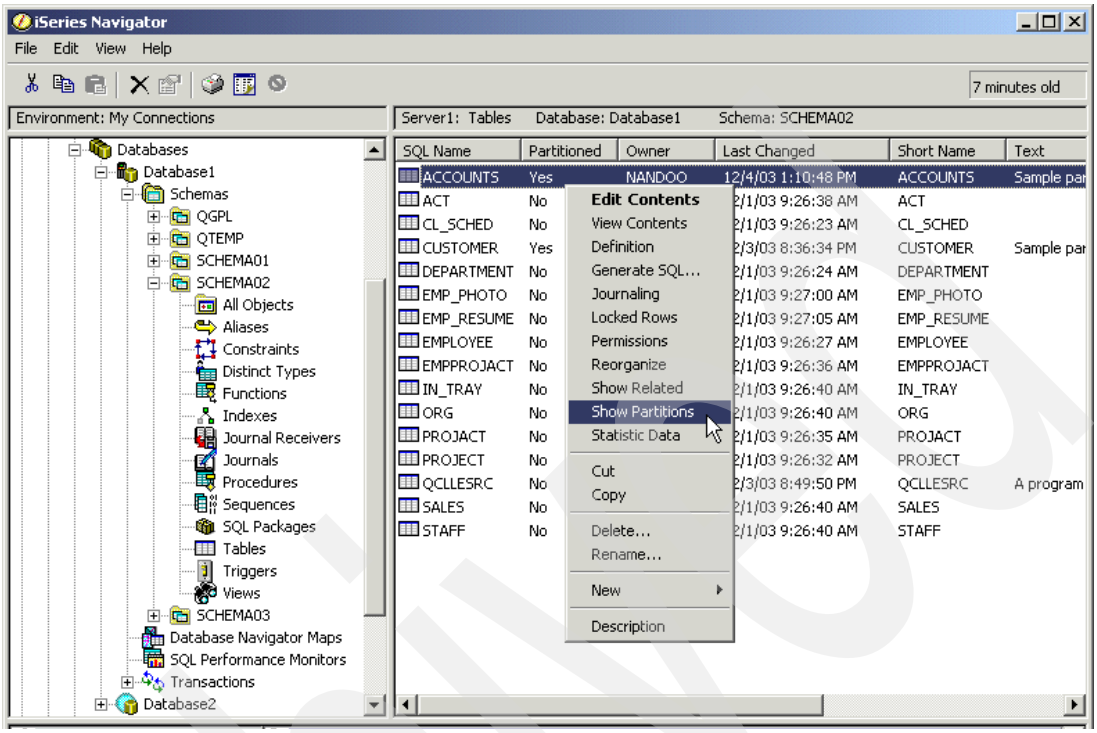


Figure 3-24 Show Partitions option

The Table Partitions window shows the list of partitions along with the partition properties. In this window, the options in the context menu of any partition enable you to work with that partition independently. You can choose to look at the definition or view the contents of a single partition, etc. Figure 3-25 shows the context menu of the partition.

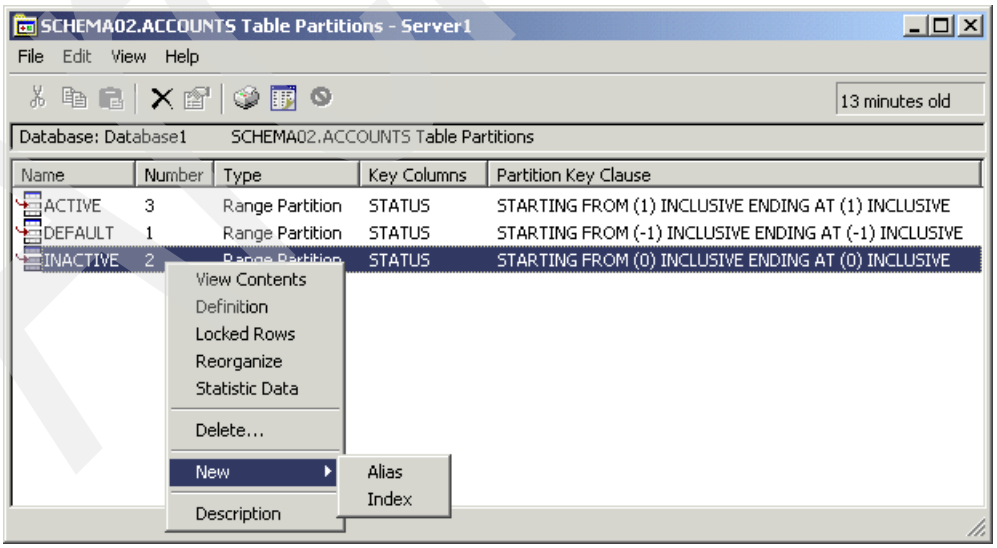


Figure 3-25 Table Partitions window

Dropping a partition

You can drop a partition from a partitioned table in many ways using iSeries Navigator. When a partition is dropped from a hash partitioned table, you are prompted to decide whether to preserve the rows or delete the rows. When a partition is dropped from a range partitioned table, all the rows are deleted from that partition.

The following steps illustrate one way to drop a partition:

1. From the Table Partitions window (Figure 3-26), select a partition that you want to drop, right-click, and select **Delete**.

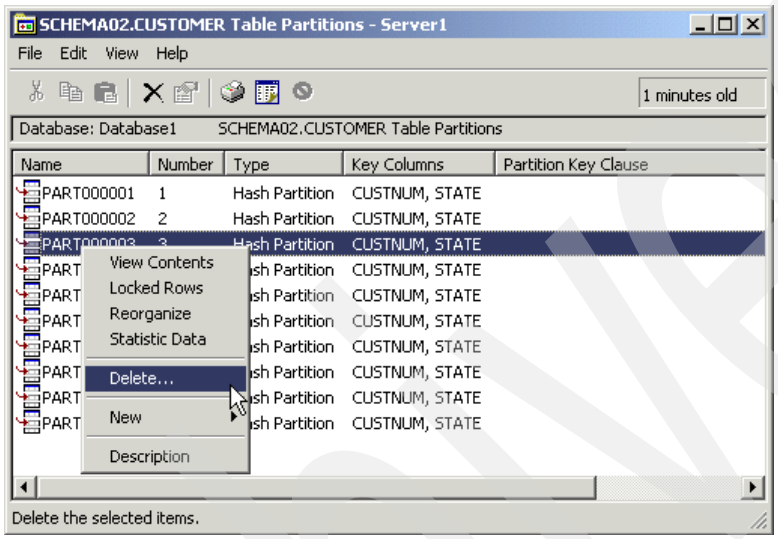


Figure 3-26 Dropping a partition

2. In the Confirm Object Deletion window (Figure 3-27), click **Delete**.

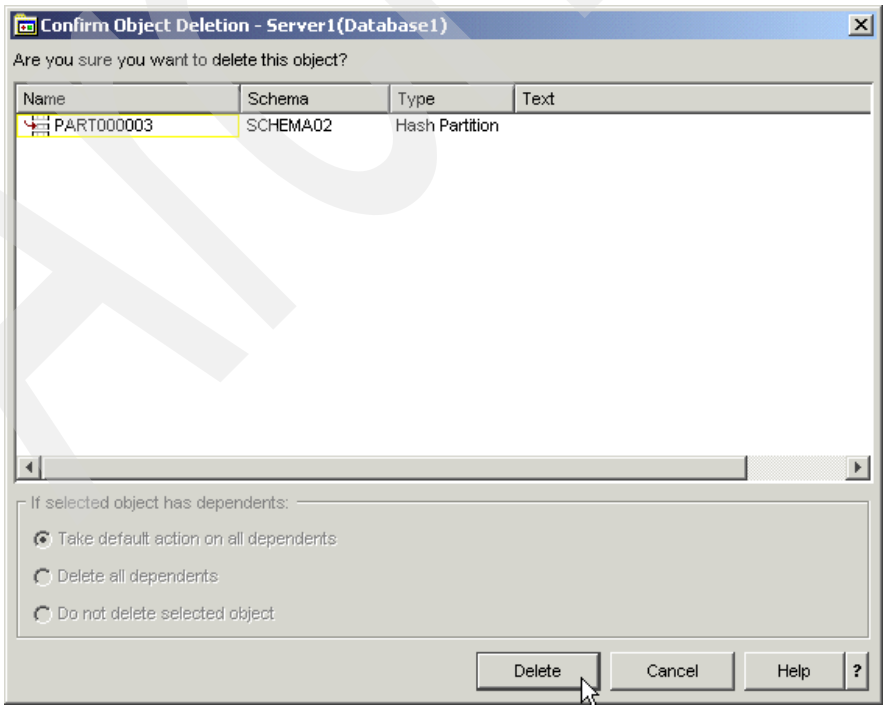


Figure 3-27 Confirm Object Deletion window for a partition

3. In the window that prompts you to confirm row deletion (Figure 3-28), select **No** if you do not want to delete the rows.

> **Note:** When you drop a partition from a range partitioned table, you are *not* prompted to confirm row deletion. All the rows are deleted automatically.

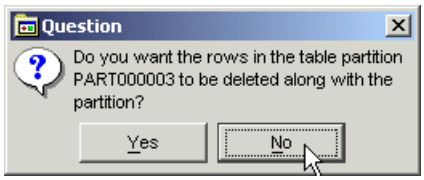


Figure 3-28 Confirming row deletion prompt

Changing a partitioned table to a non-partitioned table

Using iSeries Navigator, you can change a partitioned table to a non-partitioned table. The following steps explain how to do this using iSeries Navigator:

1. In the Table definition window, click the **Partitioning** tab.
2. On the Partitioning page (Figure 3-29), you see the list of partitions is displayed under the Partitions sections.
   a. Under Partitioning type, select **Not partitioned**.
   b. Click **OK**.

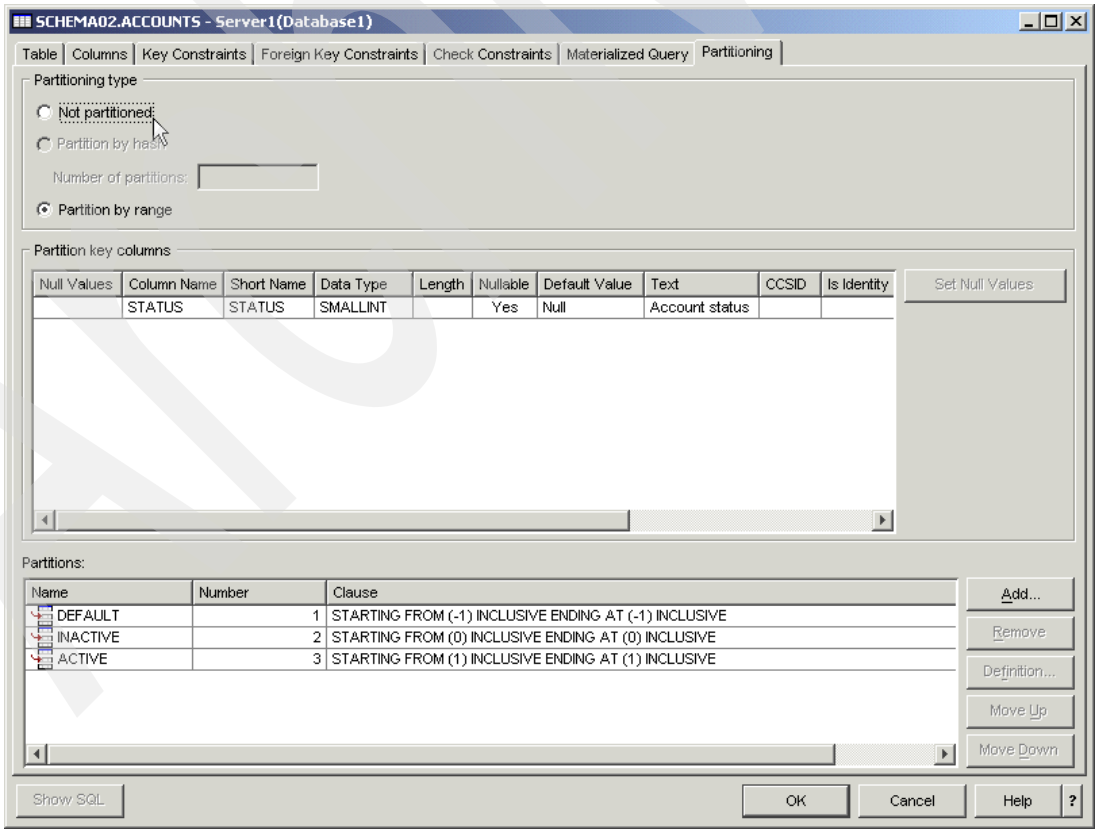


Figure 3-29 Partitioned table

The window changes to reflect the change from a partitioned table to a non-partitioned table as shown in Figure 3-30.

Note: Changing a partitioned table that contains data into a non-partitioned table requires data movement between the data partitions. You cannot change a partitioned table that is larger than the maximum non-partitioned table size to a non-partitioned table. For other considerations and related information, refer to the *DB2 Multisystem* topic in the iSeries Information Center on the Web at:

http://www.iseries.ibm.com/infocenter

When you reach this Web site, navigate to **Database** →**Reference** →**SQL Reference** →**Tables**. In the right pane, scroll down and click the **DB2 Multisystem** link.

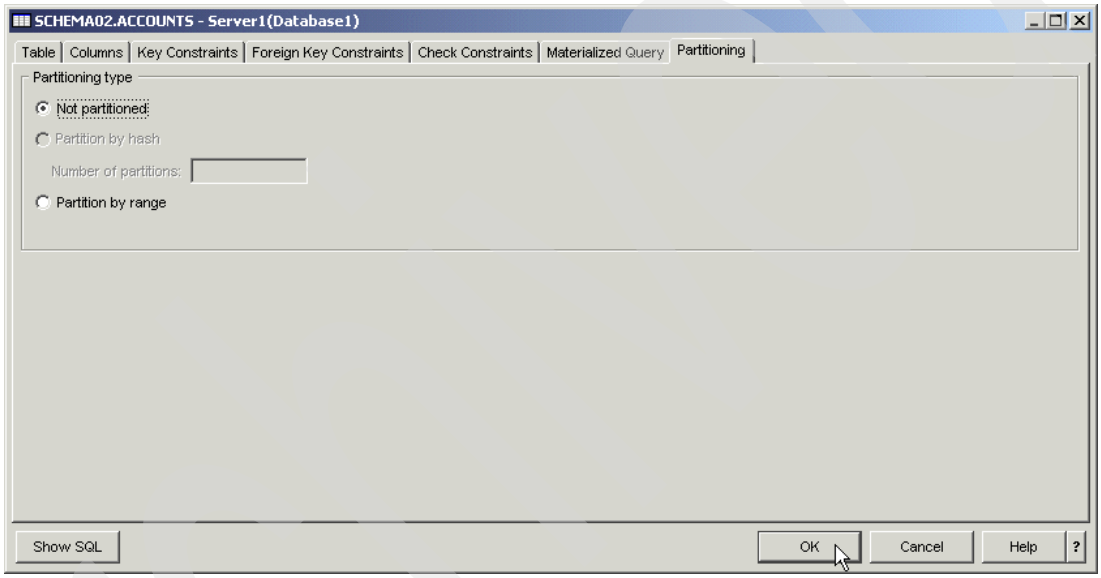


Figure 3-30 Changing the partitioned table to a non-partitioned table

Materialized query tables

A materialized query table is a query statement. It provides precomputed result sets for specified queries, which speed access to the larger source tables against which they run. It is a staging table for a large base table.

A materialized query table is used to contain materialized data that is derived from one or more source tables specified by a select-statement. A *source table* is a base table, view, table expression, or user-defined table function. The *select-statement* specifies the query that is used to refresh the data in the materialized query table.

Creating a new materialized query table

In iSeries Navigator, you can use the Materialized Query Table window to create a new materialized query table. Or, from the Table definition window, you can change an existing simple table to become a materialized query table.

The following steps explain how to create a materialized query table:

1. Select the database that you want to work with. Then expand **Schemas** and select the schema in which you want to create the materialized query table.
2. Select the **Tables** container, right-click, and choose **New** →**Materialized Query Table**.
3. On the Table page (Figure 3-31), for Table name, type the name and text description for the table.
4. Click the **Materialized Query** tab.

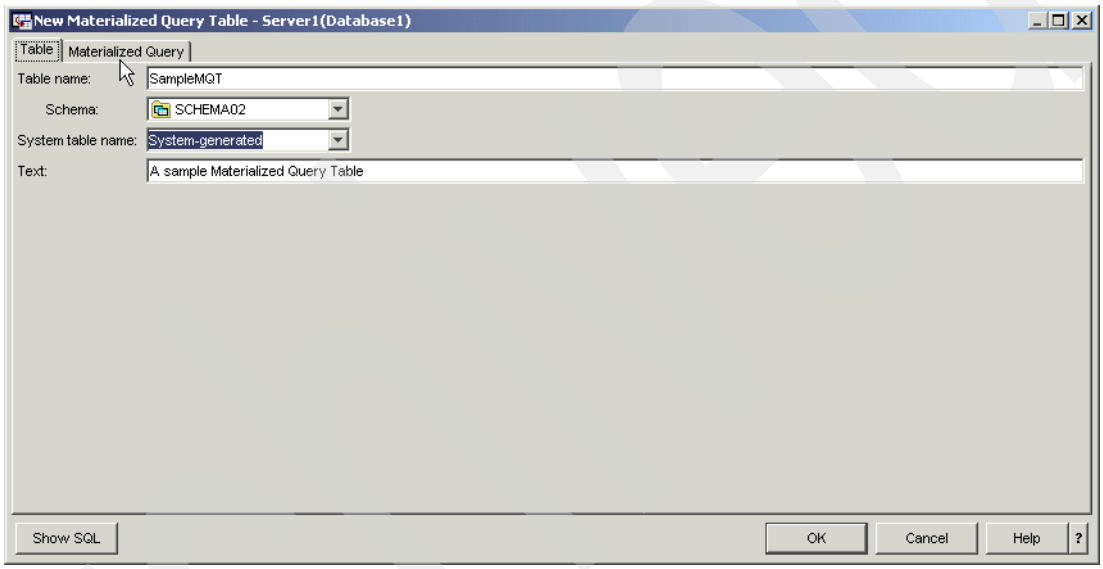


Figure 3-31 New Materialized Query Table window

5. On the Materialized Query page (Figure 3-32), complete these tasks:
   a. In the Full select statement box, type an SQL statement. Optionally click the **SQL Assist** button to build the SQL statement interactively. Refer to 7.1.4, “SQL Assist” on page 184, for more information about SQL Assist.
   b. Click the **Check Syntax** button to see if there are any syntax errors. You do not need to do this if you use the SQL Assist button to build your SQL statement.
   c. Optionally, click the **Preview Results** button if you want to see the results of executing your SQL statement.
   d. To pre-populate your materialized query table, select the **Populate table with select statement results** check box.
   e. Optionally, select the **Enable for query optimization** check box to make this materialized query table available to the optimizer to help improve the performance of the queries.
   f. Click **OK** to create the new materialized query table.

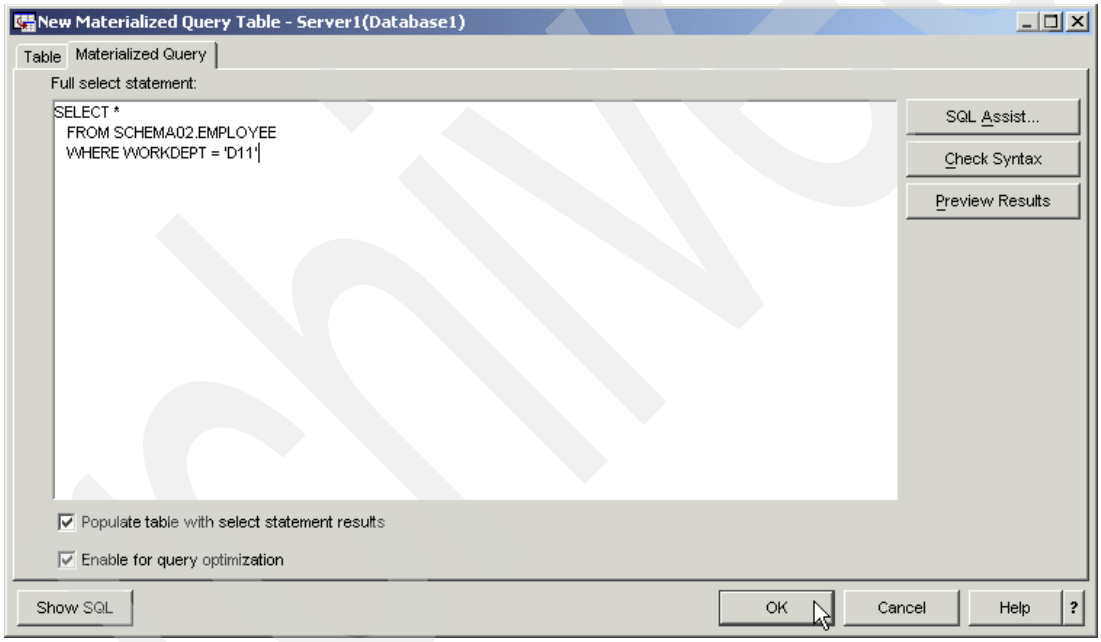


Figure 3-32 Materialized Query tab

Changing a simple table to a materialized query table

You can convert a simple table to a materialized query table by using the Table definition window in iSeries Navigator. Changing a base table into a materialized query table must meet the following characteristics:

- The number of columns in the table must be the same as the number of result columns in the select-statement.
- The column attributes of each column of the table must be compatible to the column attributes of the corresponding result column in the select-statement.

For additional restrictions and more information, visit the iSeries Information Center at:

`http://www.iseries.ibm.com/infocenter`

When you reach this site, select **Database** →**Reference** →**SQL Reference** →**Statements**. Then select the **ALTER TABLE** and **CREATE TABLE** topics.

Follow these steps to change a simple table to a materialized query table:

1. Open the Table definition window of the table that you want to change.
2. Select the **Materialized Query** tab.
3. Make sure that the base table meets all the requirements mentioned earlier.
4. On the Materialized Query page (Figure 3-33), complete these tasks:
   a. Select the **Set as materialized query table** check box.
   b. In the Full select statement box, enter the statement that you want to use. The columns of the result table from the query must match the base table definition.
   c. Select the appropriate options that you want on this window as explained in “Creating a new materialized query table” on page 44.
   d. Click **OK** to change the table to a materialized query table.

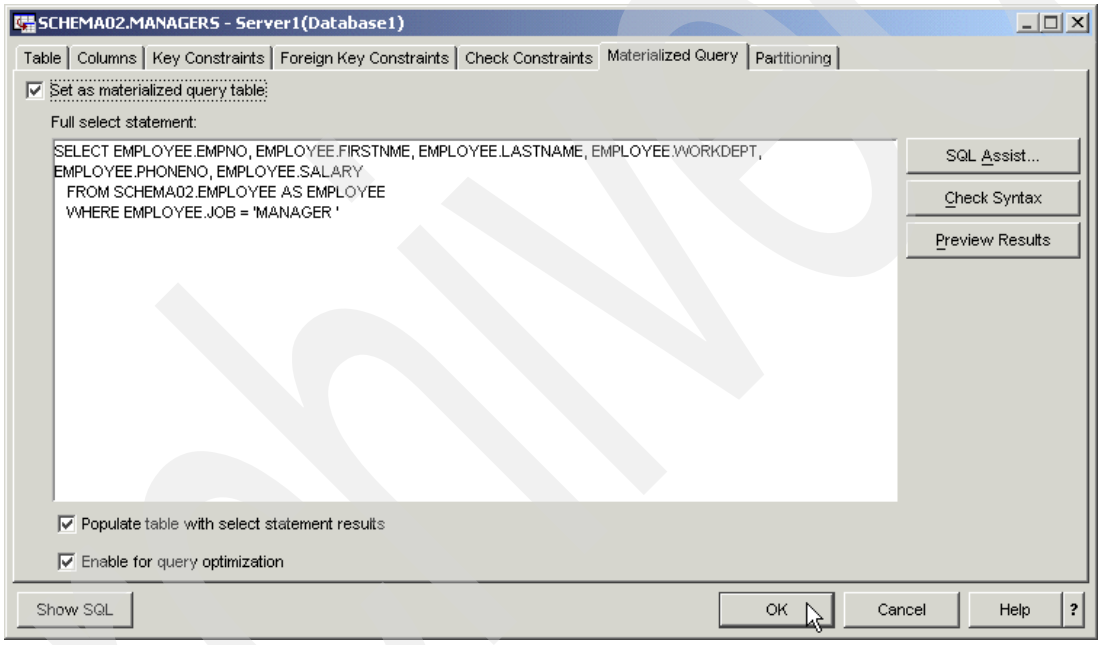


Figure 3-33 Changing a simple table to a materialized query table

Here are a few more tasks that you can perform on a materialized query table using iSeries Navigator:

- You can refresh the contents of a materialized query table at anytime. Select the materialized query table, right-click, and select **Refresh Table**. This option *replaces* the contents of the table.
- From the Table definition window, you can modify the definition of an existing materialized query table.
- You can redefine a materialized query table as a simple table. In the Table definition window, under the Materialized Query tab, clear the **Set as materialized query table** check box.

Working with the contents of a table

In iSeries Navigator, you can view or edit the contents of table by choosing the appropriate option from the context menu of the table that you want.

Viewing the contents of a table

To display the contents of your table, select the table that you want, right-click it, and choose **View Contents** as shown in Figure 3-34. The View Contents option enables you to access a datalink column and its associated Uniform Resource Locators (URLs), and then select a browser to launch.

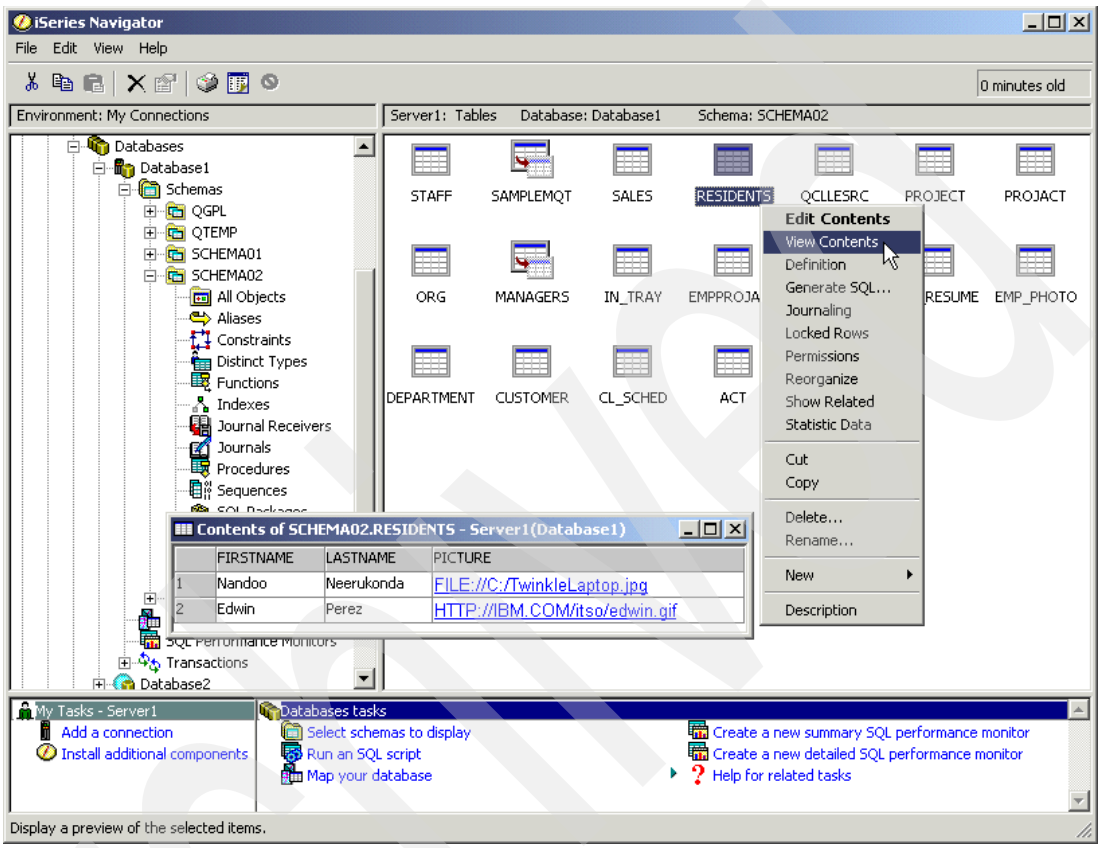


Figure 3-34 Viewing the contents of a table

Editing the contents of a table

With the appropriate authority, you can edit the contents of a table. This includes inserting and deleting rows. You can edit the contents of a simple table, a partitioned table, or a materialized query table or view using iSeries Navigator. The following steps explain how to edit the contents of a table:

1. Select the table that you want to edit, right-click, and select **Edit Contents**. Or you can double-click the table that you want to edit to open the Edit Contents window.
2. In the Edit Contents window (Figure 3-35), click the cell that you want to edit and type a value.

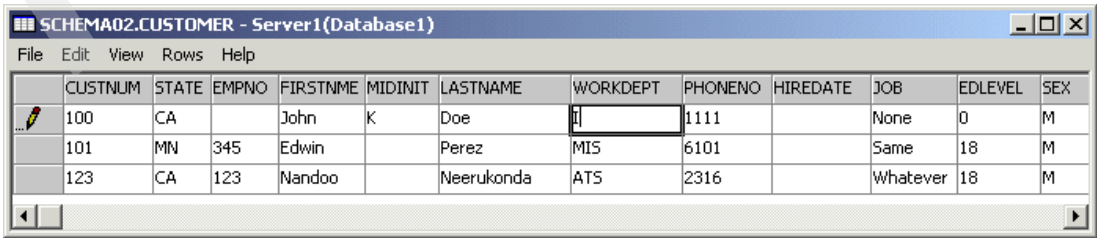


Figure 3-35 Editing the contents of a table

– To add a new row, from the menu bar, select **Rows** →**Insert**. Then enter the values as explained earlier.

> **Note:** The rows do not appear in any key order. The data that you enter must match the type of the column and satisfy all constraints. If you do not enter a value in a column, the default value is entered, if allowed.

– If you want to delete an existing row from the table, from the menu bar, select **Rows** →**Delete** as shown in Figure 3-36.

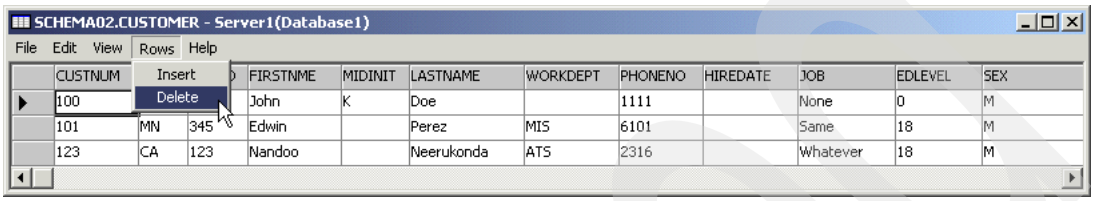


Figure 3-36 Deleting a row

3. When you are done editing, from the menu bar, select **File** →**Save** to permanently apply the changes to the table. Optionally, you can press the Esc key or choose **File** →**Exit** if you do not want to save the changes.

Displaying locked rows

To display the locked rows of a table, select the table that you want, right-click, and choose **Locked Rows**. From the Locked Rows window (Figure 3-37), you can either display the properties of the lock or view the lock holders.

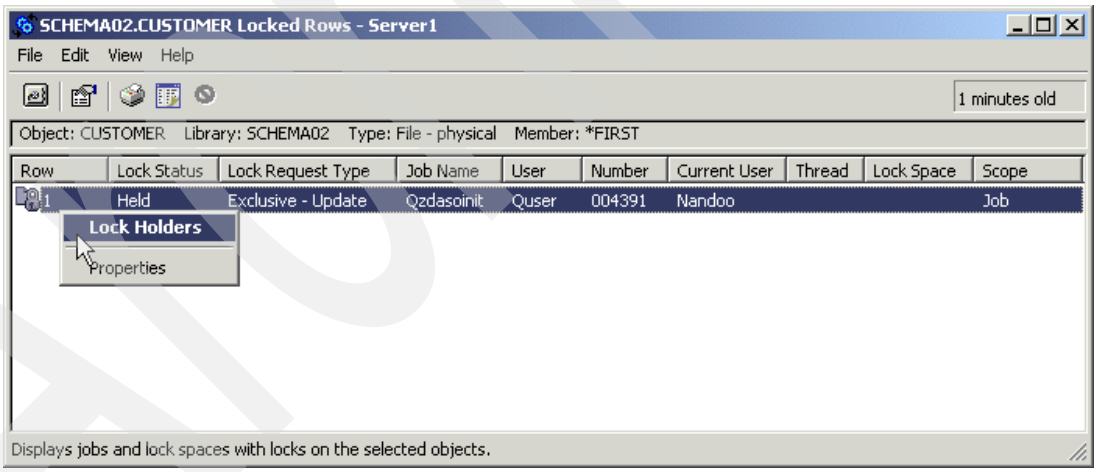


Figure 3-37 Locked Rows window

To view such details as Transactions and Last SQL Statement of the job that is holding the lock, right-click the locked row and select **Lock Holders**. Then you see the Lock Holders window (Figure 3-38).

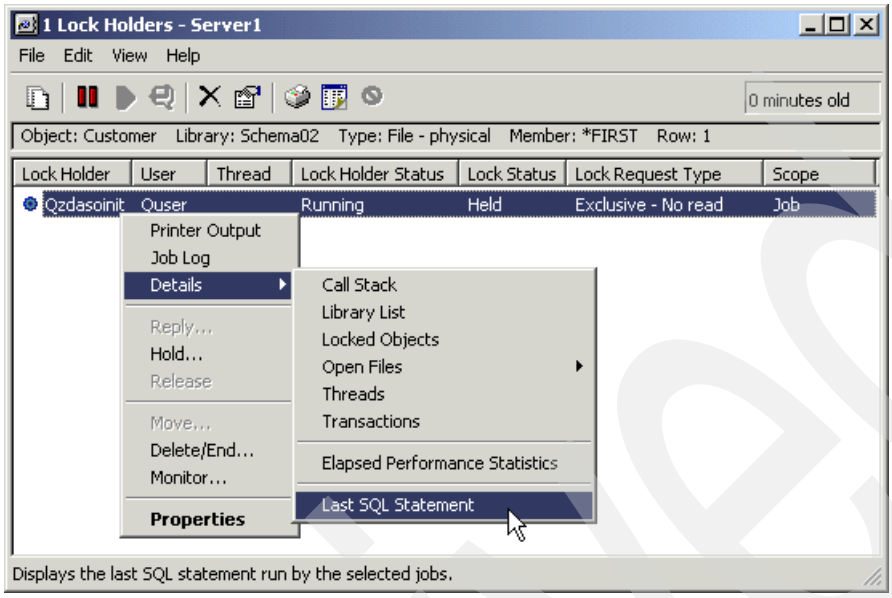


Figure 3-38 Lock Holders window

3.2.3 Views

A view provides an alternative way to look at the data in one or more tables. A view is a logical representation of a table, more than one table, or even other views. Users can use views to access data from one or more tables in an alternate format and sequence than what is represented in the table.

You may find that no single table contains all the information you need. You may also want to give users access to only part of the data in a table. Views provide a way to divide the table so that you deal with only the data you need. A view reduces complexity and, at the same time, restricts access. iSeries Navigator provides a drag and drop mechanism to define and create views.

Creating a view over a single table

The following example illustrates how to create a view that hides sensitive information and provides access to a subset of the remaining data in an alternate format than how it is represented in the base table.

For this example, we use a slightly different EMPLOYEE table that is created as explained in 1.3, “DB2 UDB for iSeries sample schema” on page 6. The only variation is that we added a new column SSN to store the social security number of an employee.

Complete the following steps to create this view:

1. In iSeries Navigator, expand **Databases**, navigate to the database that you want to work with, and expand the schema in which you want to create the view.
2. Select the **Views** container, right-click, and choose **New** →**View**.

3. In the New View window (Figure 3-39), complete these tasks:
   a. Enter a name and description for the view.
   b. For check option, make sure that **None** is selected. A check option on a view limits the values that can be inserted or updated through that view. Refer to the online help for more information about these choices.
   c. Click **OK**.

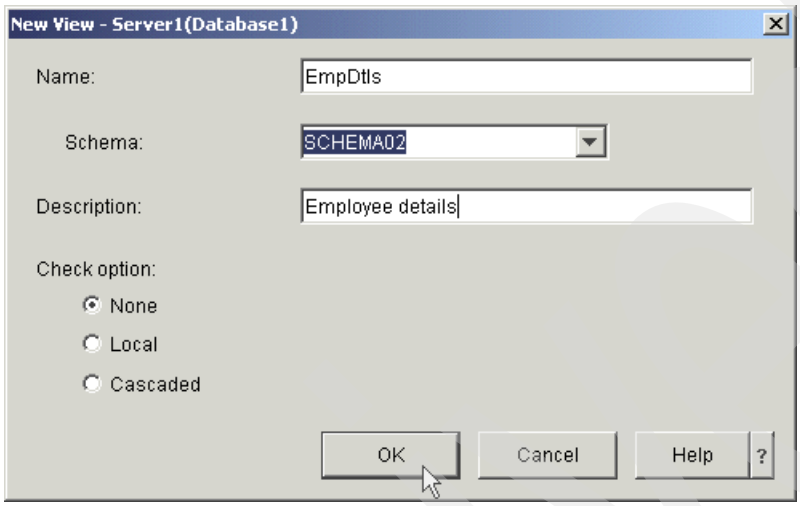


Figure 3-39 New View definition

4. A definition detail window (Figure 3-40) opens where you can select the base table and set other criteria. Complete the following tasks:
   a. In the New View window, click the **Select Tables** button.
   b. In the Browse Tables window that opens, expand the sample schema that you created using the system-supplied stored procedure and select the **EMPLOYEE** table. Click **OK**.
   c. The EMPLOYEE table is now placed in the work area of the New View definition window. Optionally, you can also simply drag and drop the table to the work area.

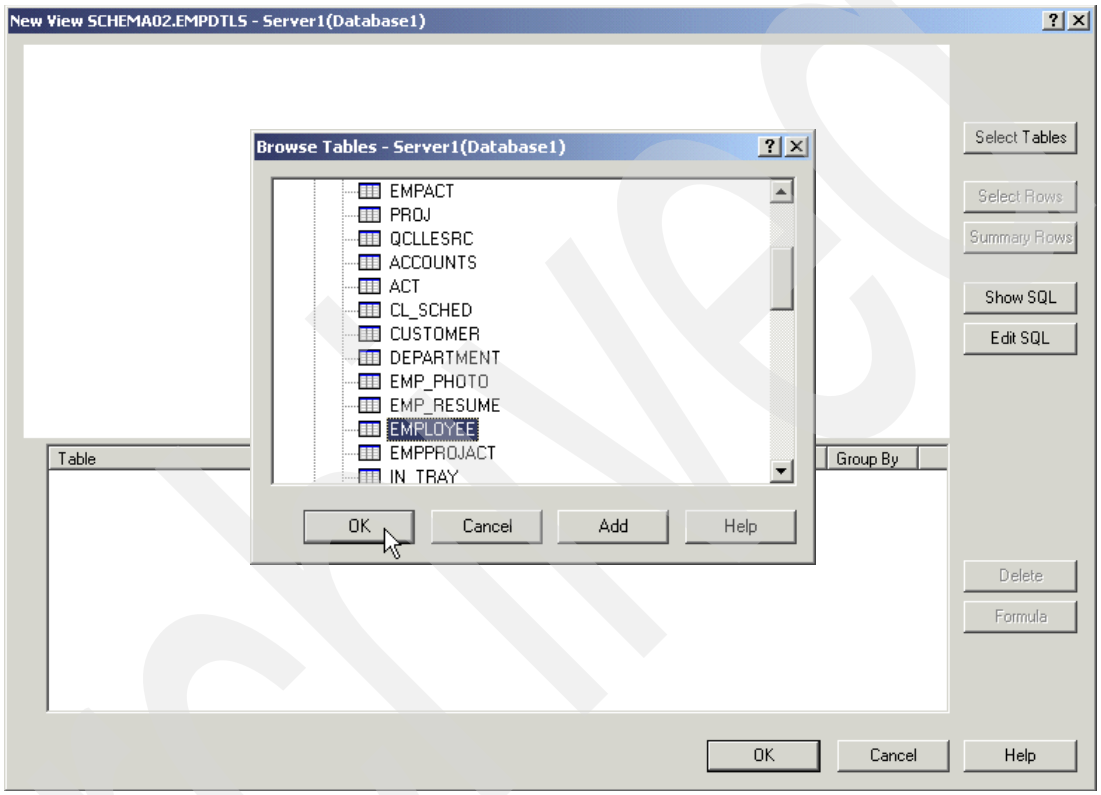


Figure 3-40 New View definition details window

5. Select all the columns, except the SSN and SALARY columns. Then drag and drop them into the Selection grid as shown in Figure 3-41.

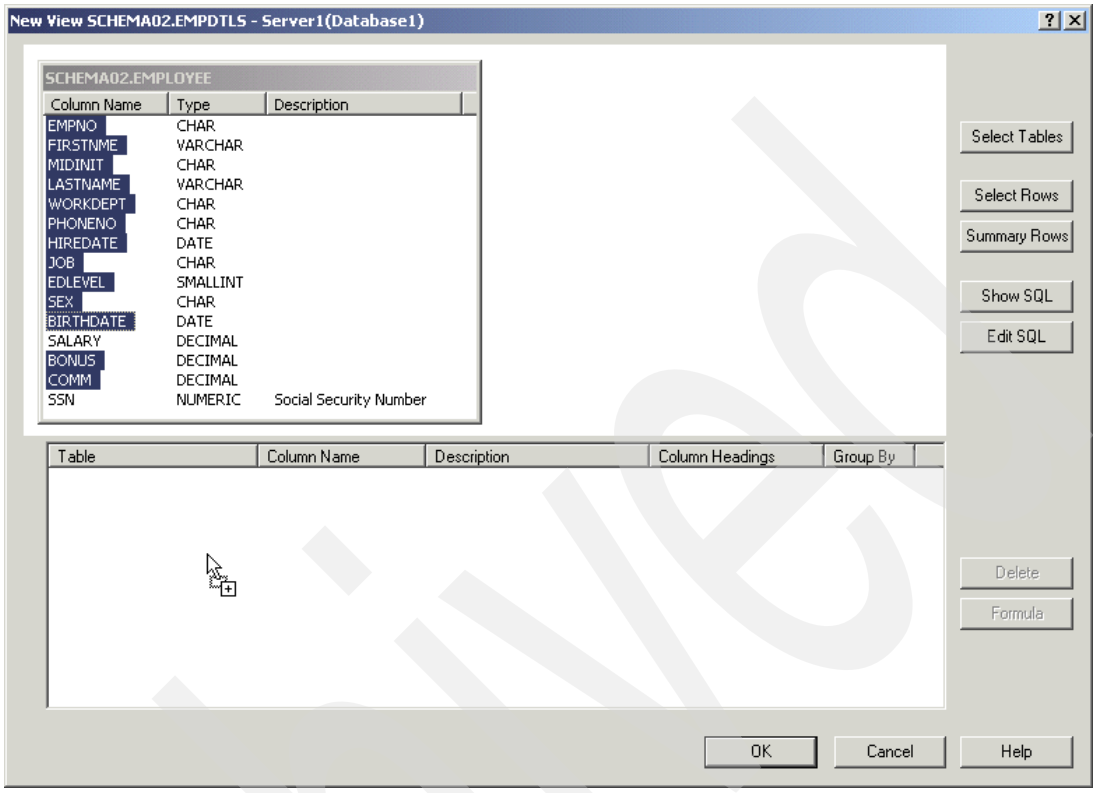


Figure 3-41 Dragging the selected columns to the selection grid

6. Rename the HIREDATE column to `TENURE` in the selection grid and click **Formula** as shown in Figure 3-42.

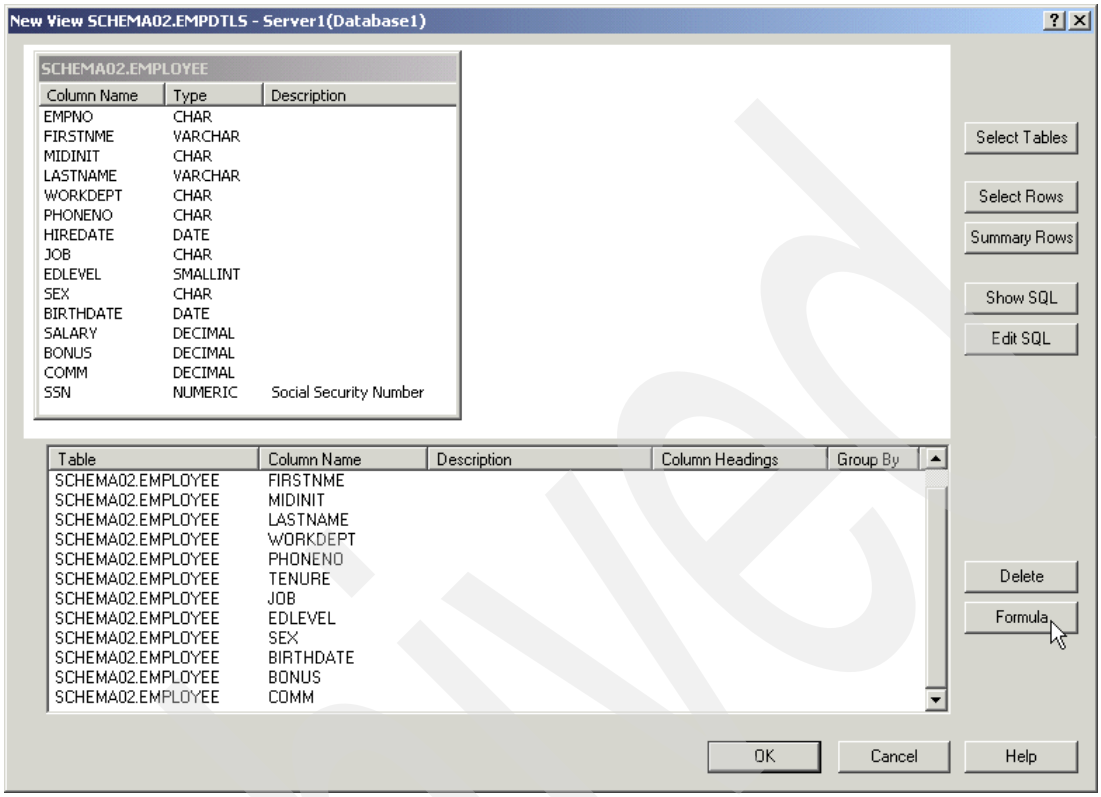


Figure 3-42 Modifying the columns for the view

7. In the Formula for... window (Figure 3-43), complete these tasks:
   a. From the Functions drop-down list, select **Date & Time** and double-click **YEAR**.
   b. Edit the clause and enter the formula using the following syntax:

      `YEAR(CURRENT DATE-SCHEMA02.EMPOYEE.HIREDATE)`

   c. Click **OK**.

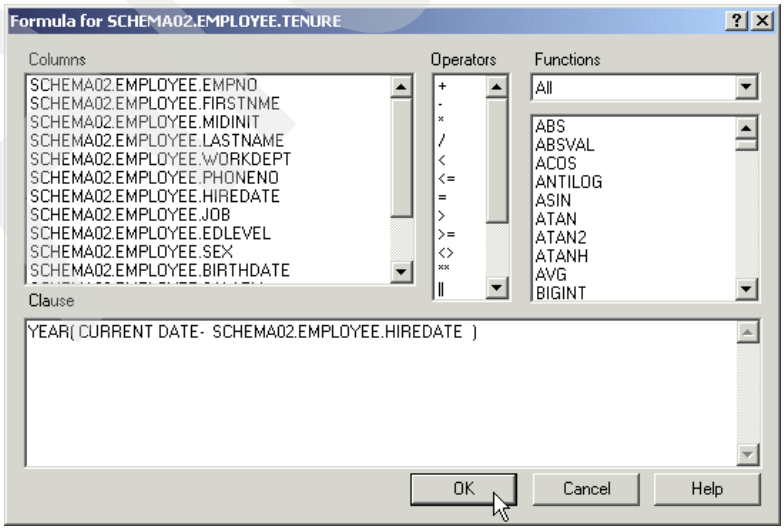


Figure 3-43 Column Formula window

8. Since we modified the formula, in the New View window, click **Show SQL**.
9. In the Show Generated SQL window (Figure 3-44), complete these tasks:
   a. Click the **Check Syntax** button to verify that the statement we entered is correct.

   > **Important:** You can click the Edit SQL button from the New View definition window to modify the actual SQL statement. However, the changes are not carried back to New View definition window. You must run the modified SQL statement from the Run SQL Scripts Center itself to create the view.

   b. In the Check Syntax successful message window (Figure 3-44), click **OK**.
   c. In the Show Generated SQL window, click **OK**.

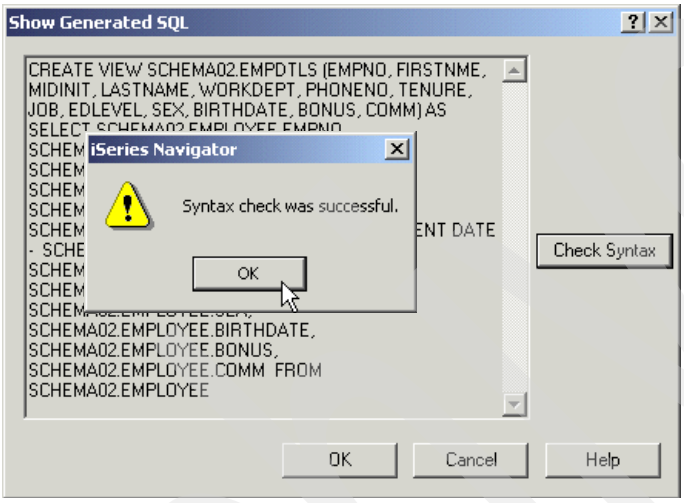
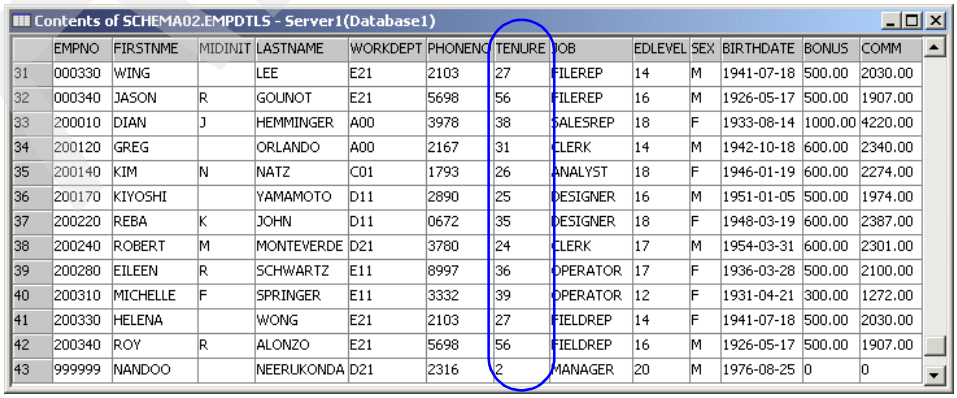


Figure 3-44 Syntax check from Show SQL window

10. Optionally, in the New View definition window, click **Select Rows** or **Summary Rows** to enter a WHERE clause and HAVING clause respectively. We do not need these clauses for this example.
11. In the New View definition window, click **OK** to create the view as defined.
12. To display the contents of this view, select the **Views** container of the schema that we used. Double-click the view that we created. The SSN and SALARY columns are absent in the view. Also, the TENURE column that is derived from HIREDATE shows the number of years elapsed since the hire date.



Contents of SCHEMA02.EMPDTLS - Server1(Database1)

	EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	TENURE	JOB	EDLEVEL	SEX	BIRTHDATE	BONUS	COMM
31	000330	WING		LEE	E21	2103	27	FILEREP	14	M	1941-07-18	500.00	2030.00
32	000340	JASON	R	GOUNOT	E21	5698	56	FILEREP	16	M	1926-05-17	500.00	1907.00
33	200010	DIAN	J	HEMMINGER	A00	3978	38	SALESREP	18	F	1933-08-14	1000.00	4220.00
34	200120	GREG		ORLANDO	A00	2167	31	CLERK	14	M	1942-10-18	600.00	2340.00
35	200140	KIM	N	NATZ	C01	1793	26	ANALYST	18	F	1946-01-19	600.00	2274.00
36	200170	KIYOSHI		YAMAMOTO	D11	2890	25	DESIGNER	16	M	1951-01-05	500.00	1974.00
37	200220	REBA	K	JOHN	D11	0672	35	DESIGNER	18	F	1948-03-19	600.00	2387.00
38	200240	ROBERT	M	MONTEVERDE	D21	3780	24	CLERK	17	M	1954-03-31	600.00	2301.00
39	200280	EILEEN	R	SCHWARTZ	E11	8997	36	OPERATOR	17	F	1936-03-28	500.00	2100.00
40	200310	MICHELLE	F	SPRINGER	E11	3332	39	OPERATOR	12	F	1931-04-21	300.00	1272.00
41	200330	HELENA		WONG	E21	2103	27	FIELDREP	14	F	1941-07-18	500.00	2030.00
42	200340	ROY	R	ALONZO	E21	5698	56	FIELDREP	16	M	1926-05-17	500.00	1907.00
43	999999	NANDOO		NEERUKONDA	D21	2316	2	MANAGER	20	M	1976-08-25	0	0

Figure 3-45 Contents of the sample view

Creating a join view

When a view is based on more than one table or view, the tables or view are said to be *joined*. To illustrate this, we use the same sample database created in 1.3, “DB2 UDB for iSeries sample schema” on page 6. In the sample schema, we create a view that joins the STAFF and ORG tables with a new result column that calculates the total earnings of each staff.

1. Complete steps 1 through 3 on page 50 as explained in “Creating a view over a single table” on page 49. For this example, in the New View window, type the name `STAFFDTLS` and type a description for the view.
2. In the New View definition window, click the **Select Tables** button.
3. The Browse Tables window opens. Complete these tasks:
   a. Expand the sample schema, select **ORG table**, and then click **Add**.
   b. Scroll down, select **STAFF**, and click **OK**.
4. Back in the New View definition window, you should see both tables in the work area. Specify the join condition by dragging the DEPT column from the STAFF table and dropping it precisely on the DEPTNUMB column of the ORG table as shown in Figure 3-46.

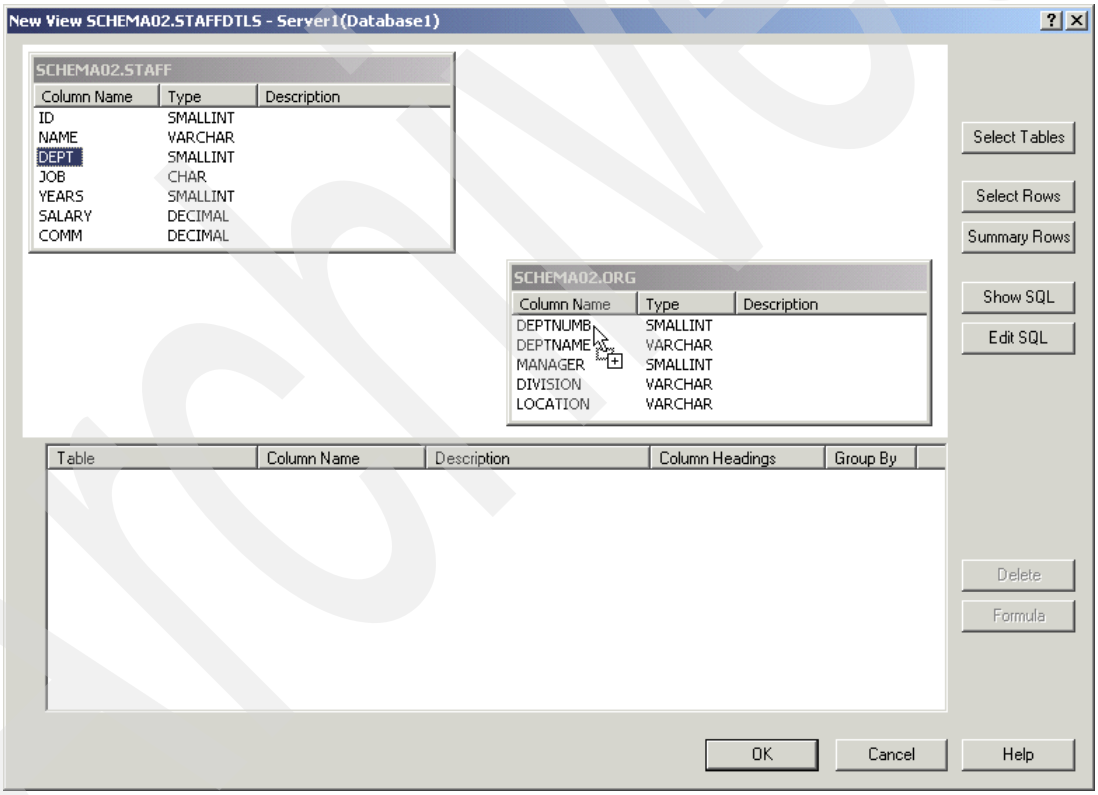


Figure 3-46 Establishing a join on Department number

5. A Join Properties window opens. Make sure that the **Inner Join** radio button is selected (by default) and then click **OK**. This specifies the join condition as:

   `SCHEMA02.STAFF.DEPT = SCHEMA02.ORG.DEPTNUMB`

6. In the New View window (Figure 3-47), complete these tasks:
   a. Drag and drop the columns to the selection grid. Then rename the columns.
   b. Select the **Total Earnings** column.
   c. Click the **Formula** button.

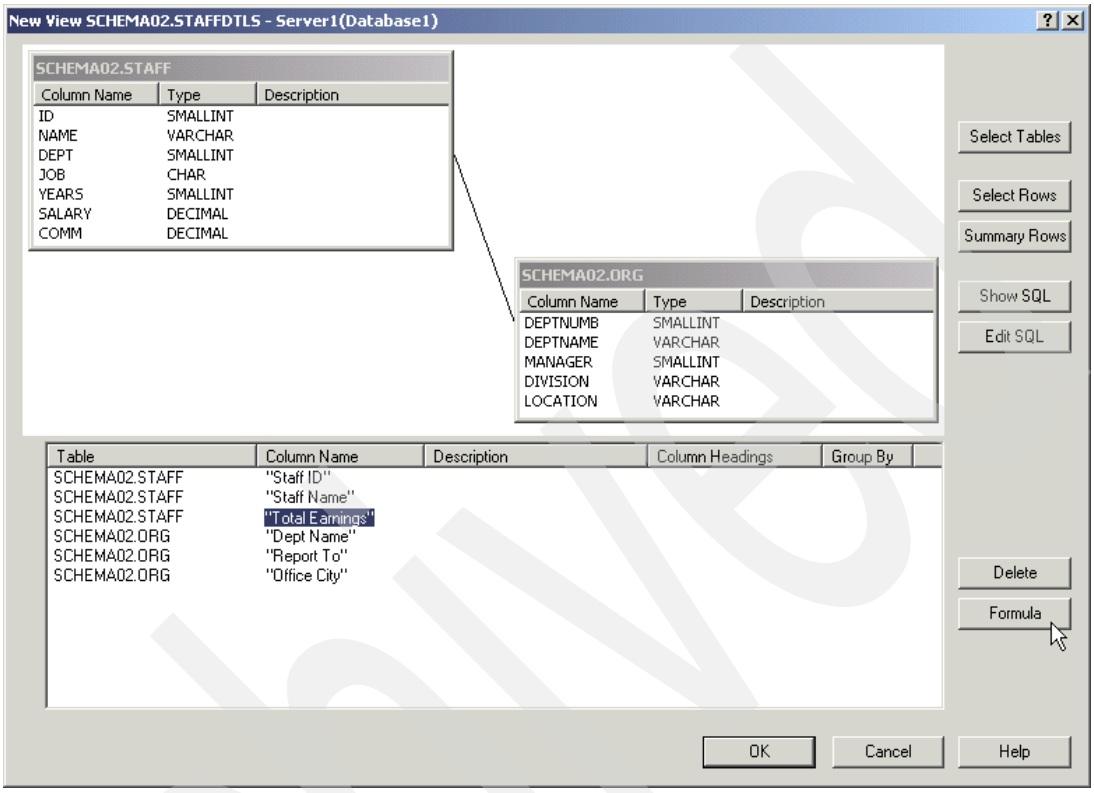


Figure 3-47 Columns selected and renamed

7. A Formula window (Figure 3-48) opens. It shows SCHEMA02.STAFF.SALARY in the clause text box.

> **Note:** For Total Earnings, we intended to use `salary + sales commission`. However, since the SALARY and COMM columns allow null values, we must use the COALESCE function to return 0 when either or both of these column values are missing (null).

In the Formula window, enter the formula and click **OK**.

Optionally, from the Functions drop-down list, you can select and double-click the **COALESCE** function to add to the clause. However, make sure the clause has this syntax:

```
COALESCE (SCHEMA02.STAFF.SALARY, 0) + COALESCE (SCHEMA02.STAFF.COMM, 0)
```

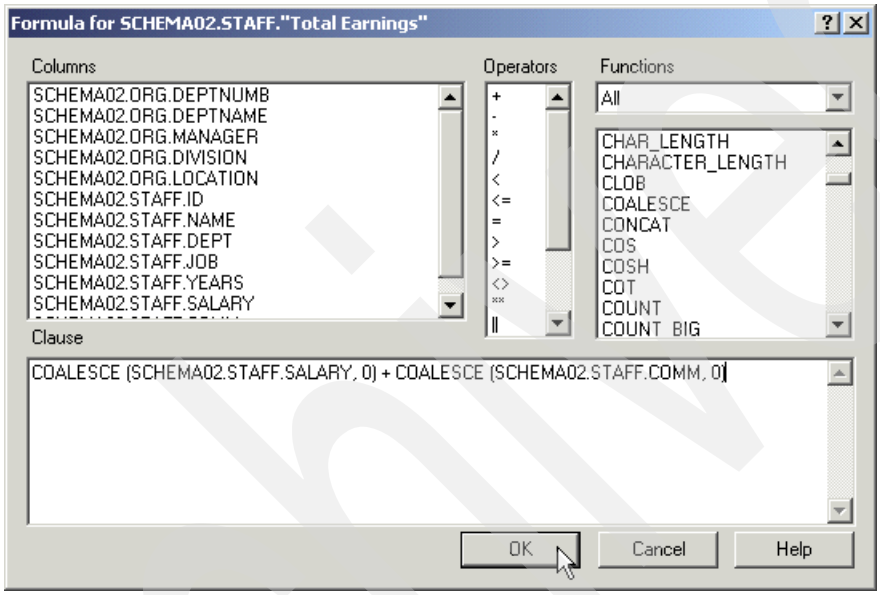


Figure 3-48 Formula for computing Total Earnings

8. Optionally, you can verify that the SQL statement is valid:
   a. In the New View window, click **Show SQL**.
   b. In the Show Generated SQL window (Figure 3-44), complete these tasks:
      i. Click the **Check Syntax** button to verify that the statement we entered is correct.

      > **Important:** You can click the Edit SQL button from the New View definition window to modify the actual SQL statement. However, the changes are not carried back to New View definition window. You must run the modified SQL statement from the Run SQL Scripts Center itself to create the view.

      ii. In the Check Syntax successful message window (Figure 3-44), click **OK**.
      iii. In the Show Generated SQL window, click **OK**.
9. In the New View definition window, click **OK** to create the view as defined.

10. Locate the view that you just created, right-click, and select **View Contents**. The contents should appear as shown in Figure 3-49.

Contents of SCHEMA02.STAFFDTLS - Server1(Database1)

	Staff ID	Staff Name	Total Earnings	Dept Name	Report To	Office City
1	10	Sanders	18357.50	Mid Atlantic	10	Washington
2	20	Pernal	18783.70	Mid Atlantic	10	Washington
3	30	Marenghi	17506.75	South Atlantic	30	Atlanta
4	40	O'Brien	18852.55	South Atlantic	30	Atlanta
5	50	Hanes	20659.80	New England	50	Boston
6	60	Quigley	17458.55	South Atlantic	30	Atlanta
7	70	Rothman	17654.83	New England	50	Boston
8	80	James	13632.80	Mid Atlantic	10	Washington
9	90	Koonitz	19388.45	Great Lakes	100	Chicago
10	100	Plotz	18352.80	Great Lakes	100	Chicago
11	110	Ngan	12714.80	New England	50	Boston
12	120	Naughton	13134.75	South Atlantic	30	Atlanta
13	130	Yamaguchi	10581.50	Great Lakes	100	Chicago
14	140	Fraye	21150.00	Plains	140	Dallas
15	150	Williams	20094.15	Plains	140	Dallas
16	160	Molinare	22959.20	Head Office	160	New York
17	170	Kermisch	12368.60	New England	50	Boston
18	180	Abrahams	12246.25	South Atlantic	30	Atlanta
19	190	Sneider	14379.25	Mid Atlantic	10	Washington

Figure 3-49 Contents of the join view

Note: You cannot alter a view after you create it. You must drop and recreate the view to make any changes.

3.2.4 Aliases

An alias is an *alternate* name for a table or view. Like tables and views, an alias may be created, dropped, and have a comment or label associated with it. No authority is necessary to use an alias. However, access to the tables and views that are referred to by the alias still require the appropriate authorization.

An alias is an object that allows SQL applications to reference a table or a view by another name. In addition, aliases provide an easy way for SQL applications to access data in partitioned tables.

Creating an alias

An alias can be created from the context menu of the table or view that you want the alias to reference. In iSeries Navigator, there are many ways to create an alias. To explain how to create an alias on a partition, we use the Table Partitions window:

1. Select the partition table on which you want to create an alias. Right-click and select **Show Partitions**.
2. In the Table Partitions window (Figure 3-50), select the partition that you want, right-click, and select **New** →**Alias**.

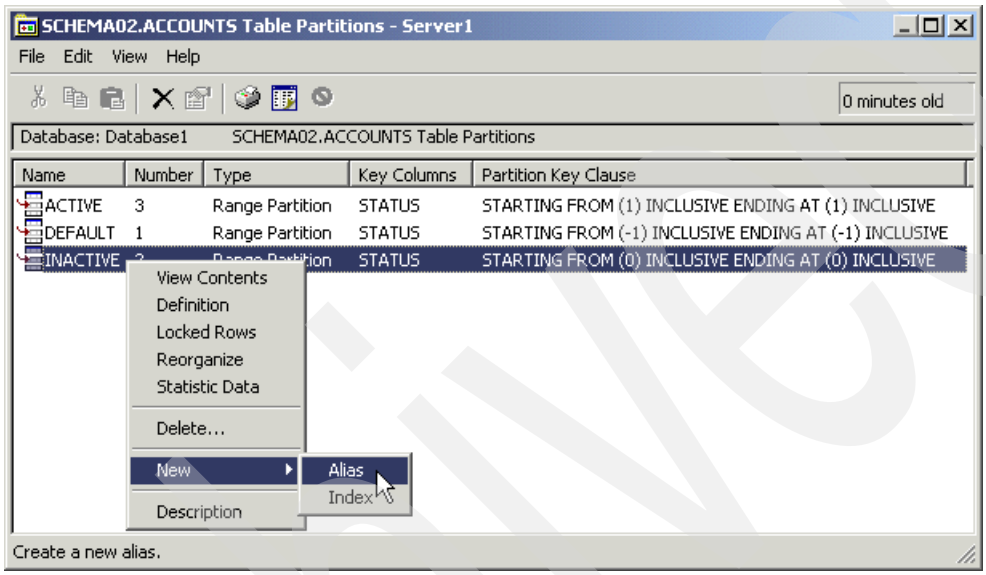


Figure 3-50 Creating an alias from Table Partitions window

3. In the New Alias window (Figure 3-51), complete these tasks:
   a. Type a name for the alias.
   b. Select the schema where you want the Alias to reside.
   c. Optionally, type a description for the alias.
   d. Click **OK**.

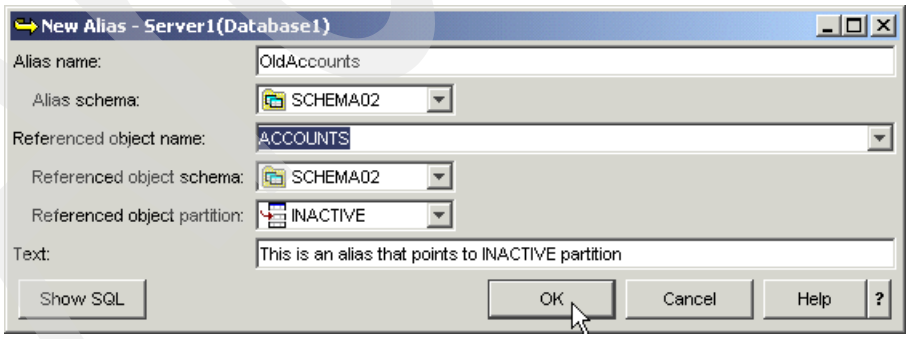


Figure 3-51 New Alias definition window

Note: You can create an alias even though the object, to which the alias refers, does not exist. However, the object must exist when a statement that references the alias is executed.

Deleting an alias

Similar to other database objects, you can drop an alias at anytime by using iSeries Navigator. Dropping an alias has no effect on any constraint, view, or materialized query that was defined using the alias. An alias can be dropped regardless of whether it is referenced in a function, package, procedure, program, or trigger.

In addition, when you choose to delete a table or a view that has one or more aliases that reference it, iSeries Navigator prompts you to select whether you want to delete those aliases as well, as shown in Figure 3-52.

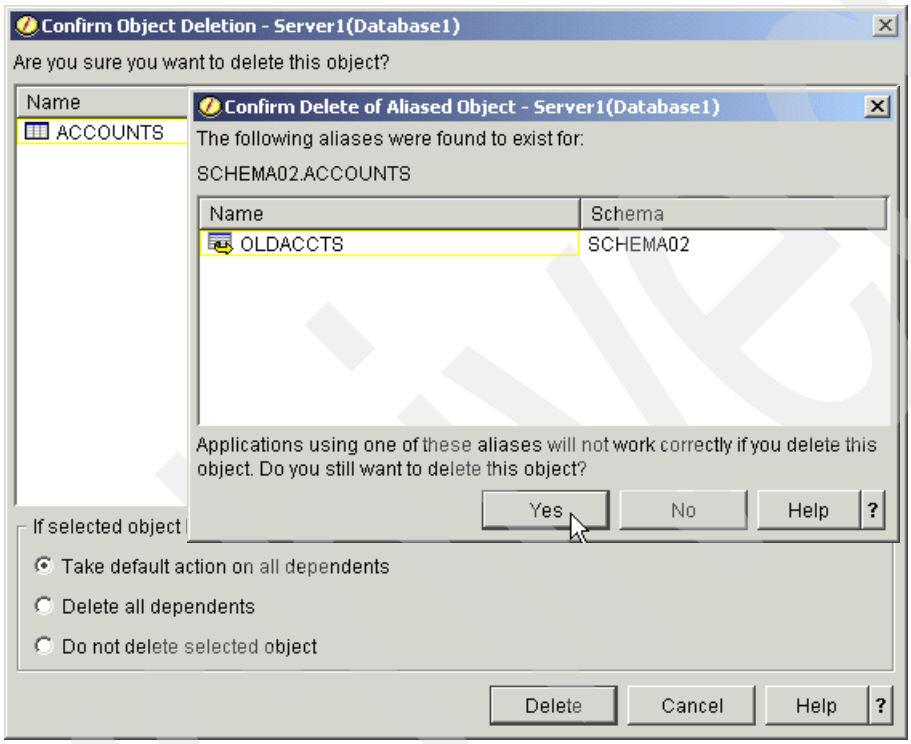


Figure 3-52 Deleting a table that has an alias

3.2.5 Indexes

An index is an ordered set of pointers to rows of a base table. Each index is based on the values of data in one or more table columns. You can use indexes to sort and select data. In addition, indexes help the system retrieve data faster for better query performance. An index is an object that is separate from the data in the table. When an index is created, the database manager builds this structure and maintains it automatically.

There are two kinds of persistent indexes used by the database manager: Binary radix tree indexes and encoded-vector indexes (EVIs). Both can be created using iSeries Navigator.

> **Note:** For more information about indexes and query performance, see **Database →Performance and Optimization →Creating an index strategy** in the iSeries Information Center at:
>
> http://www.ibm.com/eserver/iseries/infocenter

Creating an index for a partitioned table

Indexes can be created as partitioned or non-partitioned. A partitioned index creates an individual index for each partition. A non-partitioned index is a single index that spans all partitions of the table.

> **Restriction:** You can only create a partitioned EVI over a partitioned table. You cannot create a non-partitioned EVI over a partitioned table.

The following steps explain how to create an index for the CUSTOMER table. This is a hash partitioned table with CUSTNUM as the partition key column.

1. Select the **CUSTOMER** table, right-click, and choose **New** →**Index**.
2. In the New Index window (Figure 3-53), complete these tasks:
   a. Type a name for the index.
   b. For Partition, select **For each partition**.
   c. Under Available columns, select the **EMPNO** column and click **Add**.
   d. Optionally, you can select **EMPNO** from the Selected Columns grid. Click **Set Descending** to arrange the index entries in descending order by the column EMPNO. Or click **Set Ascending** to change it back. *Ascending* is the default.
   e. For Index type, select **Not unique**, because we use EMPNO, which is not part of the partition key.

      > **Important:** When creating a unique index, you can partition the index only if the keys of the unique index are the same or are a super-set of the partitioned keys.

   f. Optionally, you can also type a new value or choose one from the Number of distinct values drop-down list. Because we are not creating an EVI, the Number of distinct values in the table parameter is used only as a hint to the optimizer.
   g. In the Text field, type a text description for the index.
   h. Click **OK** to create the index.

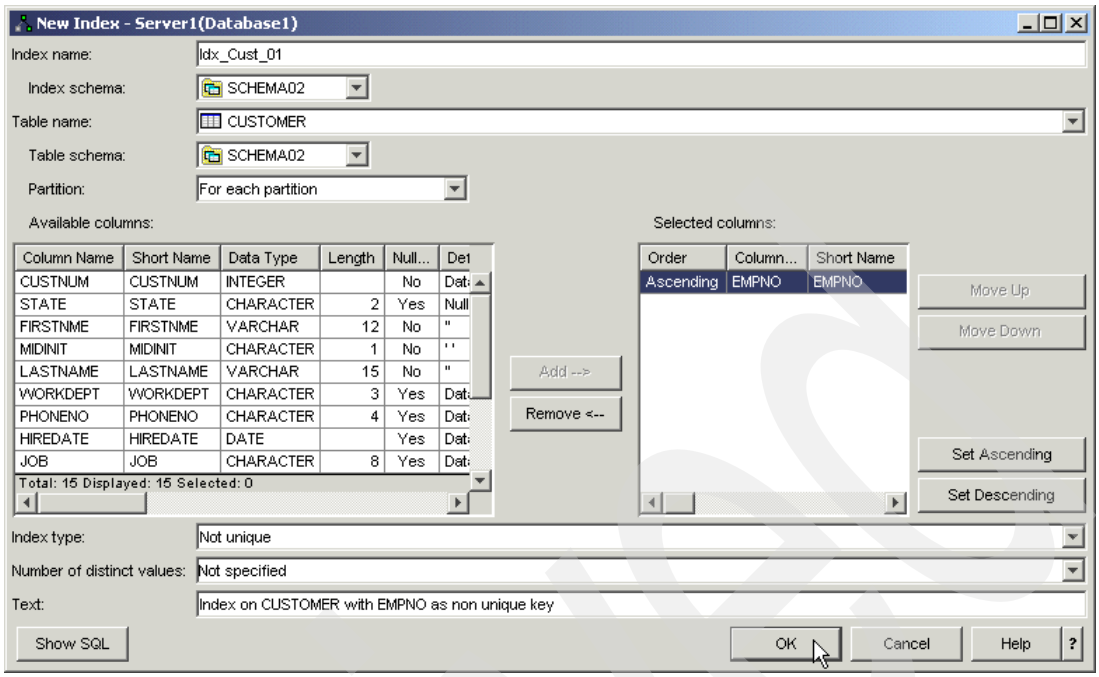


Figure 3-53 New Index definition window

3. When the index is created, optionally, select the index, right-click, and choose **Description** to set other OS/400-specific attributes such as Maintenance and Recovery.

For more information about using indexes with partitioned tables, see the *DB2 Multisystem* topic in the iSeries Information Center at:

http://www.iseries.ibm.com/infocenter

When you reach this Web site, navigate to **Database** →**Reference** →**SQL Reference** →**Tables**. In the right pane, scroll down and click the **DB2 Multisystem** link.

3.2.6 Triggers, procedures, functions, and distinct types

Using iSeries Navigator, you can work with triggers, procedures, user-defined functions, and user-defined distinct types through the corresponding containers provided under a schema. These programs are also available in the context menus of various database objects when applicable. For example, you can define or create a trigger from the context menu of a table.

Triggers

A trigger is a user-written program that is activated by the database manager when a data change is performed to the database. A database change can be an insert, delete or update of a row in a table. The change operation can be an insert, update, or delete high level language statement in an application program, or it can be an SQL INSERT, UPDATE, or DELETE statement. Triggers are useful for such tasks as enforcing business rules, validating input data, and keeping an audit trail.

For a detailed information about how to define external triggers or SQL triggers, or to enable or disable triggers using iSeries Navigator, see Part 3 in *Stored Procedures, Triggers and User Defined Functions on DB2 Universal Dabase for iSeries*, SG24-6503.

Creating a trigger

You can create two types of triggers using iSeries Navigator:

- **External trigger**: On the New External Trigger definition window (Figure 3-54), you can associate an existing procedure or program as a trigger on a specific table when specific events occur. You can refer to the online help for an explanation about each option in this window.

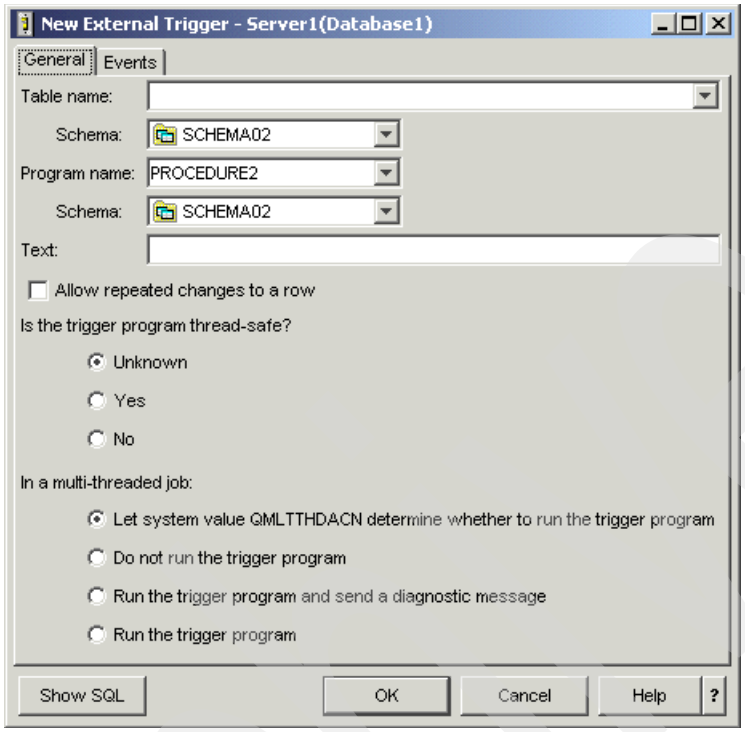


Figure 3-54 New External Trigger definition window

- **SQL trigger**: The New SQL Trigger window (Figure 3-55) enables you to create a new trigger using SQL statements and associate it with a specified table when specific events occur. When an SQL trigger is created, you cannot change the SQL statements. You need to delete and recreate the trigger to change the SQL statements.

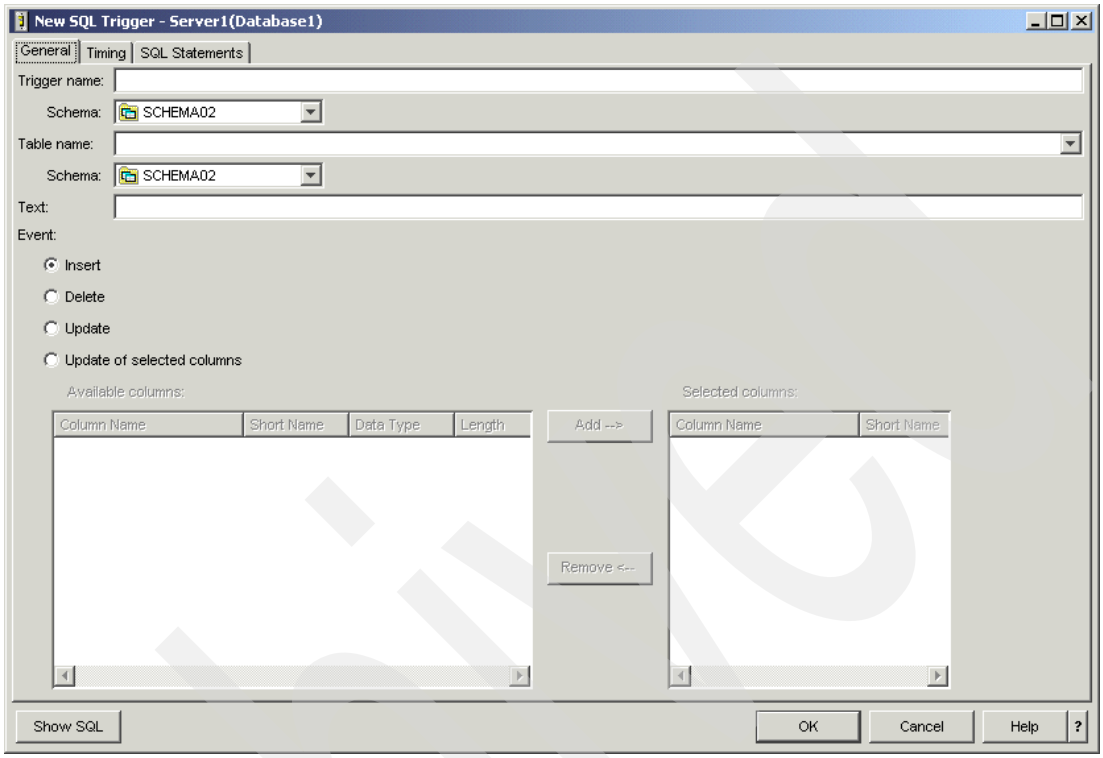


Figure 3-55 New SQL Trigger definition window

Enabling or disabling a trigger

Using iSeries Navigator, you can enable or disable a trigger by clearing the Enabled option in the Trigger definition window (Figure 3-56).

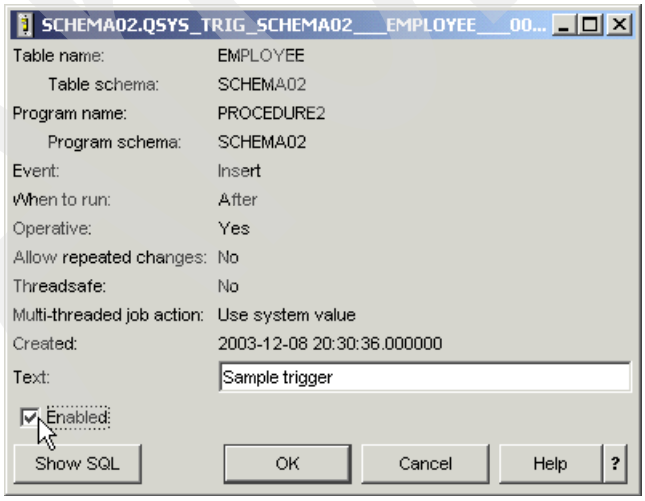


Figure 3-56 Trigger definition window

Tip: You can also enable or disable a trigger by choosing the appropriate option from the context menu of the trigger. Additionally, the Explain SQL option in the context menu of the trigger extracts and displays the access plan etc. of the trigger program. This is the same information that is output by the PRTSQLINF CL command on OS/400.

Procedures

Procedures are compiled programs that you can store on your system. Stored procedures may be called repeatedly, call other programs, and be saved to tape and restored on other systems. Stored procedures can be called locally, on the same system where the application runs, and called remotely on a different system. However, stored procedures are useful in a distributed environment since they may considerably improve the performance of distributed applications by reducing the traffic of information across the communication network.

For more information about creating external or SQL procedures, see Part 2 in *Stored Procedures, Triggers and User Defined Functions on DB2 Universal Dabase for iSeries*, SG24-6503.

Functions

A function is a routine that can be invoked from within other SQL statements and then returns a value or a table. User-defined functions are used to perform customized and often used tasks in applications.

Creating functions

Using iSeries Navigator, you can create three types of functions that can return either a single value or a table. Select the **Functions** container, right-click, and choose:

- *External* to define a program as an external function

  The program for which a function is defined does not need to exist at the time the function is defined.
- *SQL* to define a set of SQL statements as function that can be called from an SQL program
- *Sourced* to create a new function that is based on any built-in SQL function or an existing UDF

To create a new sourced function over an existing function, as shown in Figure 3-57 in the right pane, select a function, right-click, and select **New Based On**. Then specify the details for the new sourced function.

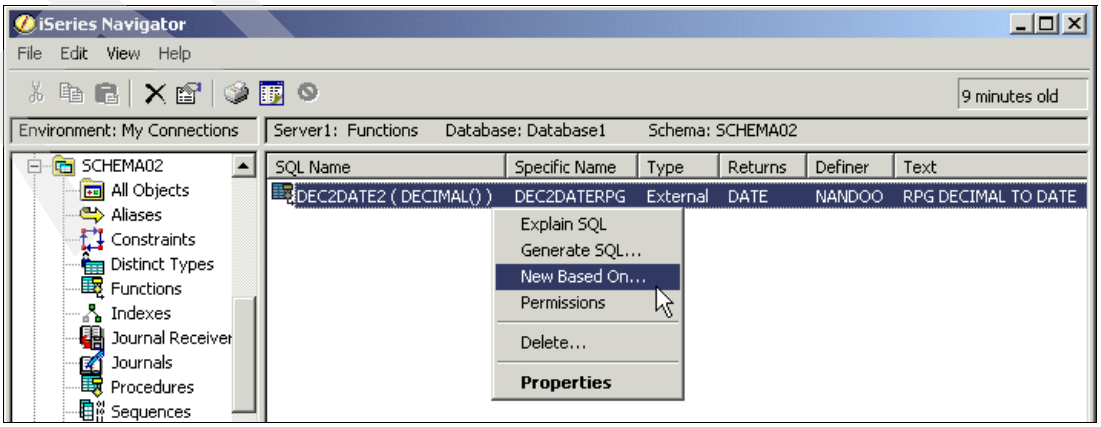


Figure 3-57 Creating a new sourced function over an existing function

Optionally, you may view the functions in the Source function drop-down list in the New Sourced Function definition window. Simply select the Functions container, right-click, and click **New** →**Function** →**Sourced**.

Distinct types

A user-defined distinct type is a data type based on a system supplied data type that the user creates in order to meet specific business needs. This new type behaves independently on the system and can be defined for a variety of behaviors. You cannot create a user-defined type based on another user-defined type. Also, UDTs are created with strong typing. This means that only functions and operators defined on the UDT can be applied to its instances.

Creating a distinct type

Creating a new distinct type based on an existing data type can give you greater control over your data.

In iSeries Navigator, select the Distinct Types container, right-click, and choose **New** →**Distinct Type**. The New Distinct Type window (Figure 3-58) opens. Enter the information as shown in the example to make your UDT available in other SQL statements.

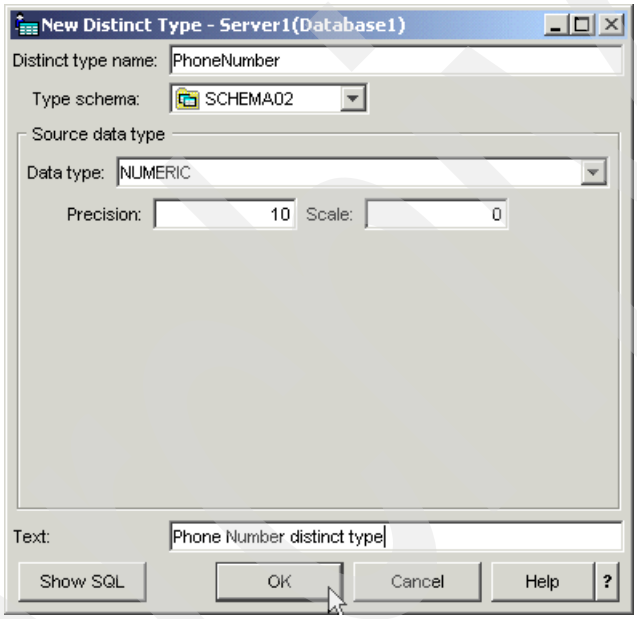


Figure 3-58 New Distinct Type definition window

The New Column definition window (Figure 3-59) shows the available distinct types.

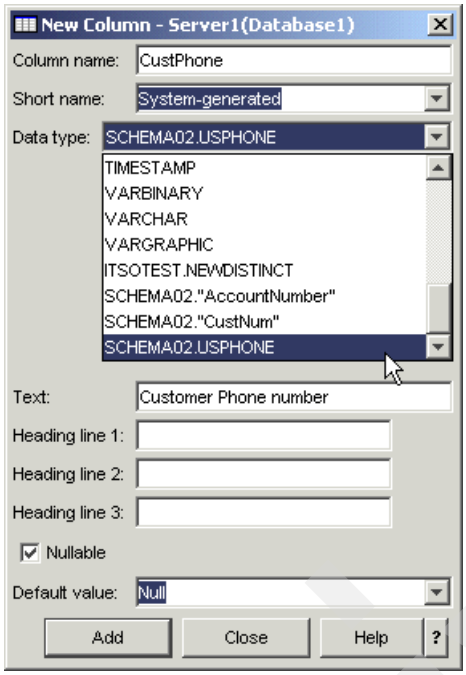


Figure 3-59 Selecting available distinct types

3.2.7 Sequences

Sequences allow you to easily generate sequential values. You can use sequences to replace an IDENTITY column or user-generated integer column. The sequence has similar functionality to these alternatives, but offers much better performance. Sequences are independent objects from a table. Therefore, they are not tied to a column and can be accessed separately. In addition, they are not treated as a part of a transaction's unit of work.

Many database applications require the use of a unique value to identify a particular database row. A classic example of this requirement is an employee database in which each employee is assigned a unique employee number.

Each sequence is a uniquely named database object that can be accessed only by sequence expressions. Sequences are found in a container underneath a named schema in the Database portion of the iSeries Navigator hierarchy. For more information about sequence expressions and how to work with them, refer to *DB2 Universal Database for iSeries SQL Reference* in the iSeries Information Center at:

http://www.iseries.ibm.com/infocenter

When you reach the Information Center, select **Database** →**Reference** →**SQL Reference**.

Differences between identity columns and sequences

While identity columns and sequences are similar in many ways, there are also differences. Examine these differences before you decide which to use.

An identity column has the following characteristics:

- An identity column is defined as part of a table. When a table is created, you cannot change an existing column as an identity column. However, you may change existing identity column characteristics.

- An identity column automatically generates values for a single table.
- When an identity column is defined as GENERATED ALWAYS, the values used are always generated by the database manager. Applications are not allowed to provide their own values during the modification of the contents of the table.
- You can use the IDENTITY_VAL_LOCAL function to see the most recently assigned value for an identity column.

A sequence has the following characteristics:

- A sequence is a system object of type *DTAARA that is not tied to a table.
- A sequence generates sequential values that can be used in any SQL statement.
- Two expressions are used to retrieve the next values in the sequence and to look at the previous value assigned for the sequence:
  - The PREVIOUS VALUE expression returns the most recently generated value for the specified sequence for a previous statement within the current session.
  - The NEXT VALUE expression returns the next value for the specified sequence.

  Using these expressions allows use of the same value across several SQL statements within several tables.

Creating a sequence

To create a new sequence, follow these steps in iSeries Navigator:

1. Select ***your system name*** →**Databases**. Select the database that you want to work with.
2. Expand **Schemas** and select the schema in which you want to create the sequence.
3. Select **Sequences,** right-click, and select **New Sequence**.
4. In the New Sequence window (Figure 3-60), complete the following tasks:
   a. Specify the sequence name for the new sequence. In this example, we type `SEQUENCE01`.
   b. Select the data type for the sequence.

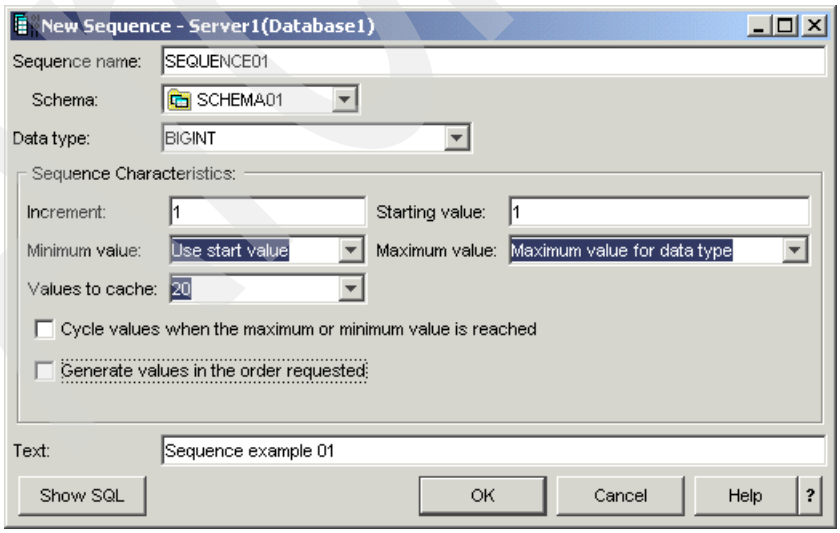


Figure 3-60 Creating a sequence

c. Under Sequence Characteristics, specify the Increment, Starting value, Minimum value, Maximum value, and Values to cache.

- *Increment* specifies the interval between consecutive values of the sequence. The value must not exceed the value of a large integer constant without any non-zero digits to the right of the decimal point. The value must be assignable to the sequence. The default is 1.

  If the value is positive, the sequence of values for the sequence ascends. If the value is negative, the sequence of values descends. If the value is zero, the sequence is a constant.

- The *Starting value* specifies the first value to be generated. The default value is the Minimum value for ascending sequences and the Maximum value for descending sequences. You use this value to start a sequence outside of the range set by the Minimum value and the Maximum value. The value can be any positive or negative value that can be assigned to a column of the data type associated with the sequence, without non-zero digits to the right of the decimal point. If a value is not explicitly specified when the sequence is defined, the default is the Minimum value for an ascending sequence and the Maximum value for a descending sequence.

  This value is not necessarily the value that a sequence cycles to after reaching the maximum or minimum value of the sequence. You can use the Starting value to start a sequence outside the range that is used for cycles. The range used for cycles is defined by the Minimum and Maximum values.

- The *Minimum value* specifies the minimum value at which a descending sequence either cycles or stops generating values, or to which an ascending sequence cycles after reaching the maximum value. This value can be any positive or negative value that can be assigned to a column of the data type associated with the sequence and without non-zero digits to the right of the decimal point.

  The value must be less than or equal to the maximum value. For an ascending sequence, the default value is the Starting value, or 1 if the Starting value is not specified. For a descending sequence, the default value is the minimum value of the data type (and precision, if DECIMAL or NUMERIC) associated with the sequence.

- The *Maximum value* specifies the maximum value at which an ascending sequence either cycles or stops generating values, or to which a descending sequence cycles after reaching the minimum value. This value can be any positive or negative value that can be assigned to a column of the data type associated with the sequence and without non-zero digits to the right of the decimal point.

  The value must be greater than or equal to the minimum value. For an ascending sequence, the default value is the maximum value of the data type (and precision, if DECIMAL or NUMERIC) associated with the sequence. For a descending sequence, the default value is the Starting value, or -1 if the Starting value is not specified.

- In *Values to cache* specify the maximum number of values of the sequence that the database manager preallocates and keeps in memory. During a system failure, all cached sequence values that are yet to be assigned are lost, and therefore, will never be used. The value specified also represents the maximum number of values for the sequence that can be lost during a system failure. A range of sequence numbers can be kept in memory for fast access. When an application accesses a sequence that can allocate the next sequence number from the cache, the sequence number allocation can happen quickly. However, if an application accesses a sequence that cannot allocate the next sequence number from the cache, the sequence number allocation requires an update to the *DTAARA object.

Choosing a high value for CACHE allows faster access to more successive sequence numbers. However, in the event of a failure, all sequence values in the cache are lost. If you do not use the Values to cache option, the values of the sequence are not stored in the sequence cache. In this case, access to the sequence requires an update to the *DTAARA object. The minimum value that can be specified is 2, and the maximum is the largest value that can be represented as an integer. The default is 20.

d. If you want values to cycle after reaching their minimum or maximum, select the **Cycle values when the minimum or maximum value is reached** check box. This options specifies that values continue to be generated for the sequence after the maximum or minimum value is reached.

   If you select this option, after an ascending sequence reaches the maximum value of the sequence, it generates its minimum value. After a descending sequence reaches its minimum value of the sequence, it generates its maximum value. The maximum and minimum values for the sequence determine the range that is used for cycling. When a cycle is in effect, the database manager can generate duplicate values for a sequence.

e. Select the **Generate values in the order requested** check box to specify that the values are generated in order of request. With this option, the performance of the NEXT VALUE sequence expression is worse than if you do not select the option. The default is that the values do not need to be generated in order of request.

f. Optionally, in the Text field, specify a description.

g. Optionally, to view the SQL used to create this object, click the **Show SQL** button. This action launches Run SQL Scripts.

   For more information about Run SQL Scripts, see Chapter 7, “Run SQL Script Center” on page 169. You can then choose to run the SQL from the Run SQL Scripts window. However, if you edit the SQL in the Run SQL Scripts window, the changes are not reflected back to the New Sequence window.

h. Click **OK**.

A sequence is created as a *DTAARA object. Do not change sequence objects by using the Change Data Area (*CHGDTAARA) command or any other similar interface. Doing so may cause unexpected failures or unexpected results when you attempt to use the SQL sequence through SQL.

Tip: You can also create a sequence by right-clicking the schema that you want and selecting **New** →**Sequence**.

Defining constant sequences

You can define a sequence that always returns a constant value. You do this by specifying an Increment value of zero and a Starting value that does not exceed the Maximum value. Or specify the same value for the Starting, Minimum, and Maximum values. For a constant sequence, each time a NEXT VALUE expression is processed, the same value is returned. A constant sequence can be used as a numeric global variable.

Changing a sequence

You can change a sequence in any of the following ways:

- Restarting the sequence
- Changing the increment between future sequence values
- Setting or eliminating the minimum or maximum values to assume the defaults values of the data type
- Changing the number of cached sequence numbers

- Changing the attribute that determines whether the sequence can cycle
- Changing whether sequence numbers must be generated in order of request

To change a sequence in iSeries Navigator, follow these steps:

1. Expand the system that you want to use.
2. Expand **Databases** and the database that you want to work with.
3. Expand **Schemas** and select the schema in which your sequence exists. Right-click your sequence, and select **Definition**.
4. A window opens that is similar to the example in Figure 3-61. You can change all the information in this window, except for the sequence name and the schema.

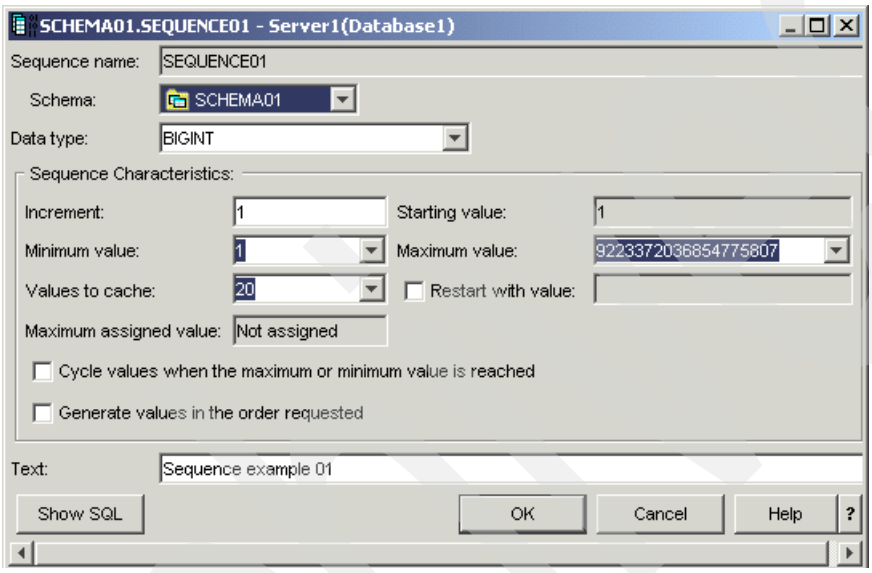


Figure 3-61 Modifying sequences

You can set Data type to any exact numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or NUMERIC) with a scale of zero. Or you can set it to a user-defined distinct type for which the source type is an exact numeric type with a scale of zero. Each of the existing Starting, Increment, Minimum, and Maximum values that are not changed must contain a value that can be assigned to a column of the new data type.

If you select the Restart with value check box, specify any positive or negative value. This value must not exceed the value of a numeric constant that might be assigned to a column of the data type associated with the sequence, without non-zero digits to the right of the decimal point. After you restart a sequence or change the sequence to allow cycling, the sequence numbers may be duplicate values of those that were generated previously by the sequence.

If you select the Restart with value check box, but don't specify value, the sequence is restarted at the value that is specified implicitly or explicitly as the starting value during the creation of the sequence. If you *don't select* the Restart with value check box and specify a new range that is *greater that* the previous one, the sequence continues with the next number of the last range. If you don't select the Restart with value check box and specify a new range that is *lower than* the previous one, the first number that is generated for the changed sequence is the first number of the new range.

Important: When you change a sequence, all cached values are lost. Only future sequence numbers are affected.

3.2.8 Packages

A package is an object that contains control structures that are used to execute SQL statements. Packages are produced during distributed program preparation. The control structures act as the bound or operational form of SQL statements. All control structures in a package are derived from the SQL statements that are embedded in a single source program.

An SQL package contains the control structure that is produced when the SQL statements in an application program are bound to a remote relational database management system (RDBMS). The RDBMS uses the control structure to process SQL statements that are encountered while running the application program.

Learn more about SQL package support in *DB2 Universal Database for iSeries SQL Reference* in the iSeries Information Center at:

http://www.iseries.ibm.com/infocenter

When you reach the Information Center, select **Database** →**Reference**. Then select the **SQL Reference** and **Structured Query Language** topics.

Creating a package

This option allows you to create an SQL package on a relational database from an existing distributed SQL program. This option can be used to create a package when the package was not created at precompile time or if the package is needed at an RDB other than the one specified on the precompile command. A distributed SQL program is a program created by specifying the Relational database (RDB) parameter on a CRTSQL*xxx* (where *xxx* = CBL, CBLI, CI, CPPI, PLI, RPG or RPGI) command.

Follow these steps in iSeries Navigator to create a new package:

1. Expand ***your system name*** →**Databases** →**Schemas and** select the schema in which you want to create the package.
2. Select **SQL Packages**, right-click, and select **New** →**SQL Package** (Figure 3-62).

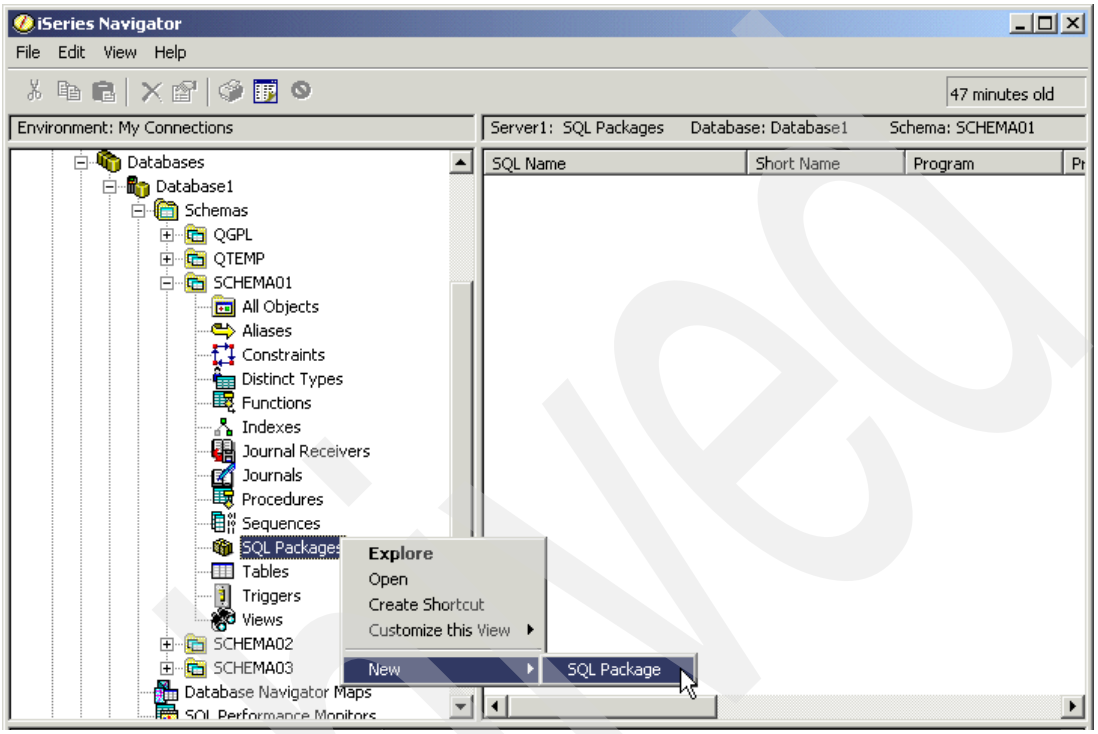


Figure 3-62 Selecting to create an SQL Package

3. The Create SQL Package window (Figure 3-63) opens. Complete the following tasks:
   a. Specify the program and library name for which the SQL package is being created. The program must be a distributed SQL program that was created using one of the CRTSQL*xxx* commands, where *xxx* is CBL, CBLI, CI, CPPI, PLI, RPG or RPGI.
   b. In the Relational Database parameter, specify relational database where the SQL package is being created. Enter `*PGM` to use the relational database name that specified for the Relational database (RDB) parameter of the CRTSQL*xxx* command. You can also specify the *name* of the relational database where the SQL package is to be created. Use the Work with Relational Database Directory Entry (WRKRDBDIRE) command to show the relational database names that are valid for this parameter.
   c. Specify the RDB user name and the RDB user password that is sent to the remote system when you start the conversation. The user name that is associated with the current job is used if you specify `*CURRENT` in the RDB user parameter.
   d. For Default collection, specify the schema name to use for unqualified names of tables, views, indexes, SQL packages, aliases, constraints, external programs, node groups, and triggers. This parameter applies only to static SQL statements in the package.
   e. For Object type, specify the type of program for which an SQL package is created.

f. For Module list, specify `*ALL` or the list of modules in the program for which an SQL package is to be created.

g. For Text ‘description’, specify text that briefly describes the SQL package and its function.

h. Click **OK**.

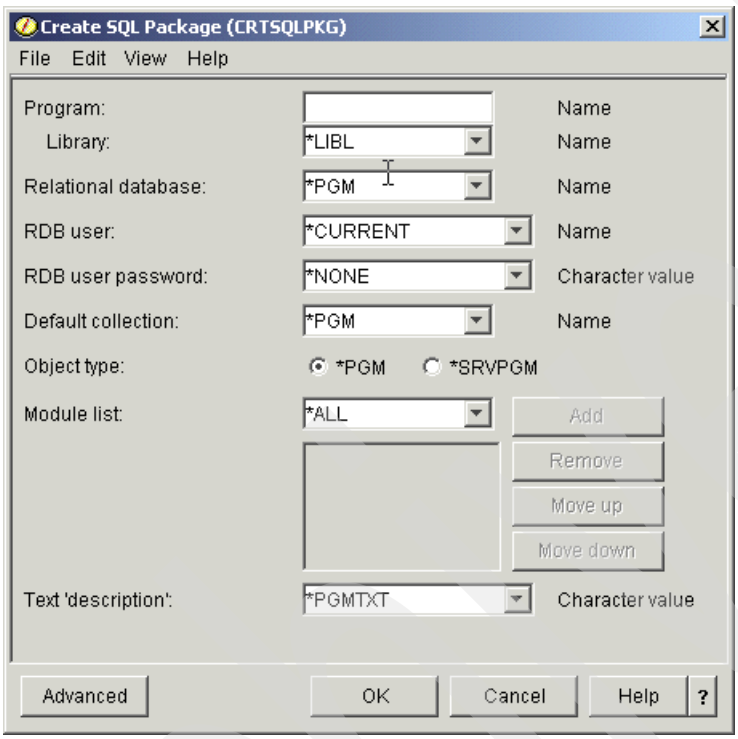


Figure 3-63 Create SQL Package window

The Advanced button allows you to specify the maximum severity level to allow for errors that are detected during SQL package creation. If errors occur at a level that exceeds the level that you specify, the SQL package is not created. You can also specify whether an existing SQL package of the same name in the specified library is replaced by the new SQL package and the printer device file to which the create SQL package error listing is directed. If no errors are detected during the creation of the SQL package, no listing is produced.

3.2.9 Journal management

Journal management provides a means by which you can record the activity of objects on your system. When you use journal management, you create an object called a *journal*. The journal records the activities of the objects that you specify in the form of *journal entries*. The journal writes the journal entries in another object called a *journal receiver*.

Journal entries are records of changes that are made to journaled files and tables. They also record changes of other events that occur on the system, such as changes to a database file, to other journaled objects, or to security-relevant events. The information that is recorded includes the file name and schema to which it belongs, the job ID, user ID, workstation ID, program name, date and time, and type of activity. You can also record information before and after a change takes place. This gives you a more complete picture of the change that is being made.

Journal management provides:

- Decreased recovery time after an abnormal end
- Powerful recovery functions
- Powerful audit functions
- The ability to replicate journal entries on a remote system

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 APIs to write to and read journal entries.

The iSeries database supports two types of journals:

- Local journal

  Use local journal management to recover changes to an object that occurred since the object was last saved, as an audit trail, or to help replicate an object. Setting up journaling locally is a prerequisite for other iSeries functions such as Remote journal management and commitment control.

- Remote journal

  Replicate information from local journals and store them on a remote system. Use remote journal management to establish journals and journal receivers on a remote system that are associated with specific journals and journal receivers on a local system.

  You can use remote journal management with application programs to maintain a data replica. A *data replica* is a copy of the original data that resides on another iSeries server or independent disk pool. The original data resides on a primary system. Applications make changes to the original data during normal operations.

  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.

What's new in V5R3

There are several improvements and additions to journal management. The following items contain a summary of these improvements and additions:

- Changed default for system-managed access path protection (SMAPP) to 60 minutes from 70 minutes
- Enhancements to change journaled objects without ending journaling
- Expanded sequence numbers for your journal receivers
- Recovery enhancements for journaling
- New API for journal entries
- Information enhancements

To learn more about what's new or has changed in Version 5 Release 3 (V5R3), see the *Memo to users* at:

http://www.iseries.ibm.com/infocenter

Creating a journal

The following example explains how to create a local journal in the SCHEMA01 schema and create its associated journal receiver in the SCHEMA02 schema.

> **Note**: To create and set up remote journaling through iSeries Navigator, you must first create the local journal and journal receiver. Then use Properties support for the journal to access actions that set up a remote journal.

1. In the iSeries Navigator window, expand ***your system name*** →**Databases**. Expand the database and schema that you want to work with.
2. Select the **Journals** container, right-click, and select **New** →**Journal**.
3. The New Journal window (Figure 3-64) opens. Complete the following tasks:
   a. Specify the journal name, schema, and description.
   b. Specify the Schema to hold journal receivers. You can select a schema from a list of the current iSeries Navigator session's schema list, except for the schema named to contain the journal. In our example, SCHEMA01 does not appear in the list.

      > **Important:** Although you can place a journal receiver in any schema you want, including SCHEMA01, for OS/400 journaling support, we recommend that you place the schema that is used for the journal receivers in its own user-defined auxiliary storage pool (ASP), separate from the schema that contains the journal itself.

      In our example, we specify schema `SCHEMA02` to emphasize a different schema for the receiver. SCHEMA02 must already exist.
   c. Click **OK**.

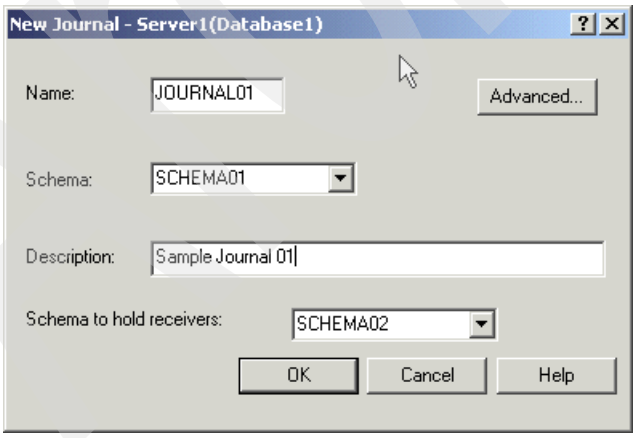


Figure 3-64 Creating a journal

4. 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 window (Figure 3-65). This window shows the default attributes that were used in our example to create the JOURNAL01 journal.

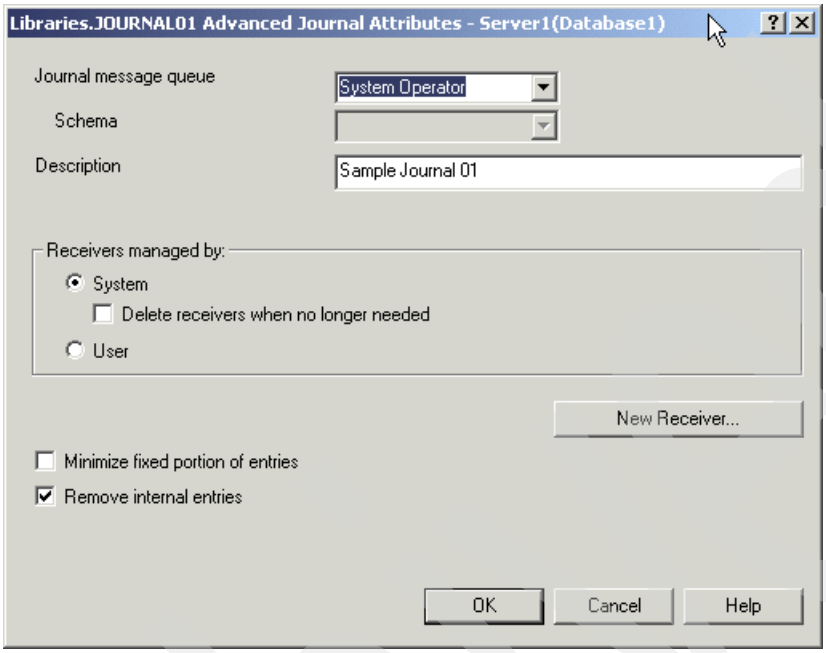


Figure 3-65 Advanced Journal Attributes window

If you click the **New Receiver** button in the Advanced Journal Attributes window, you see the New Journal Receiver window (Figure 3-66). It shows the default new journal receiver attributes.

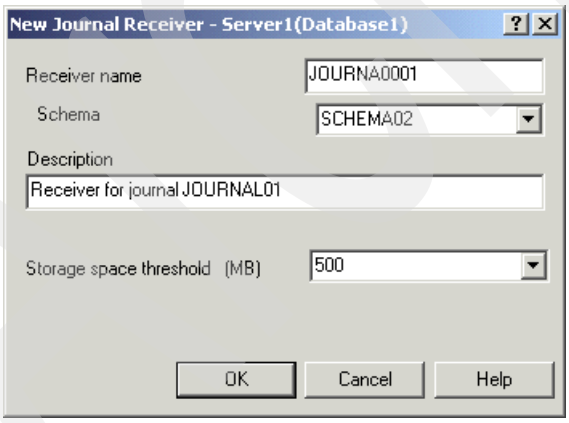


Figure 3-66 Creating a journal (New Journal Receiver)

Journaling a table or group of tables

After you create the journal, follow these steps to journal a table or group of tables:

1. In the right pane, select the journal, right-click, and select **Properties** (see Figure 3-67).

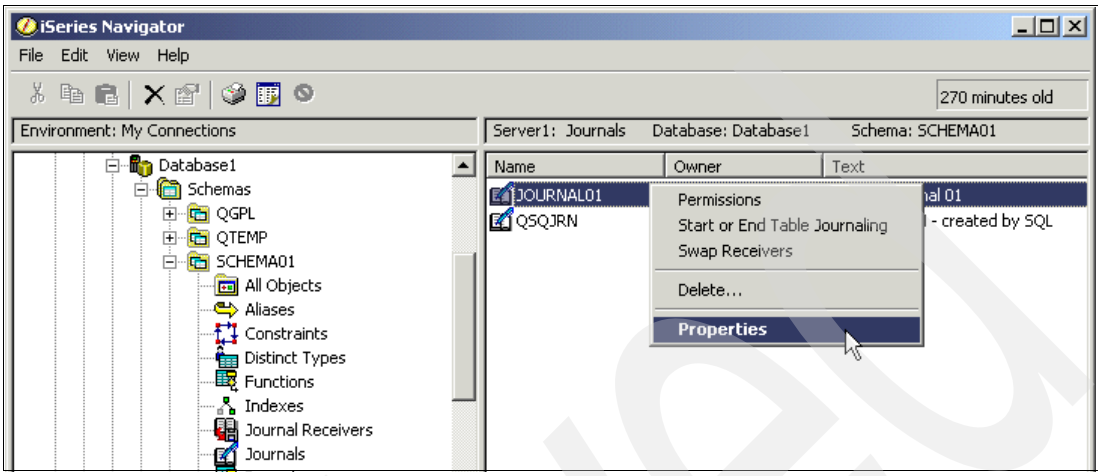


Figure 3-67 Selecting the table or tables to journal

2. The Properties window (Figure 3-68) opens. Click the **Tables** button.

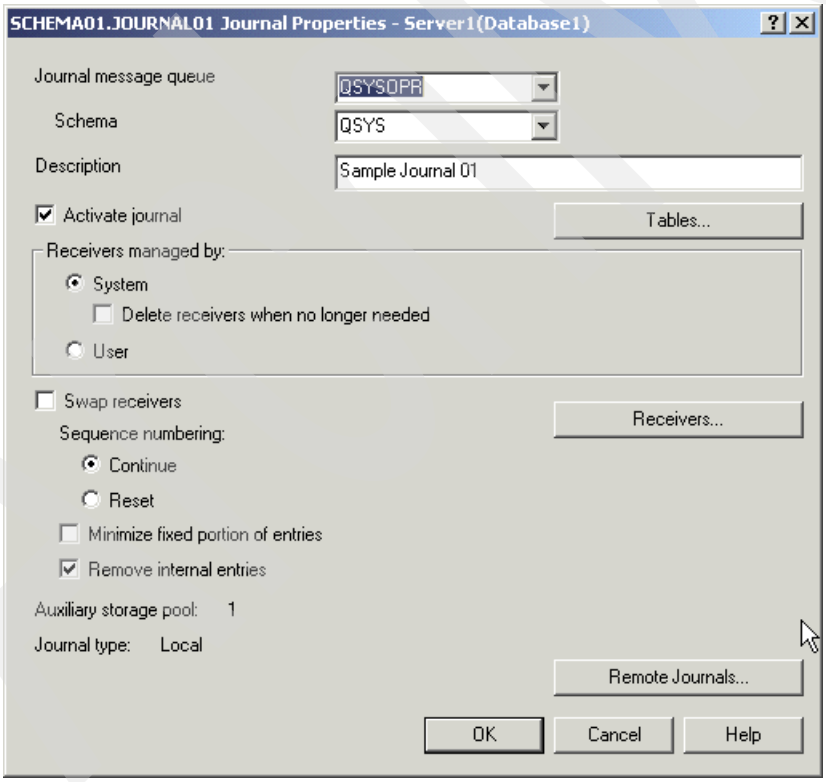


Figure 3-68 Journal Properties window

3. The Start/End Journaling panel (Figure 3-69) opens. Complete these tasks:
   a. Under Tables, expand the **SCHEMA01** schema and select the **EMPLOYEE** table.
   b. Click the **Add** button to move the table to the Tables to journal pane.
   c. Click **OK** to start journaling the EMPLOYEE table.

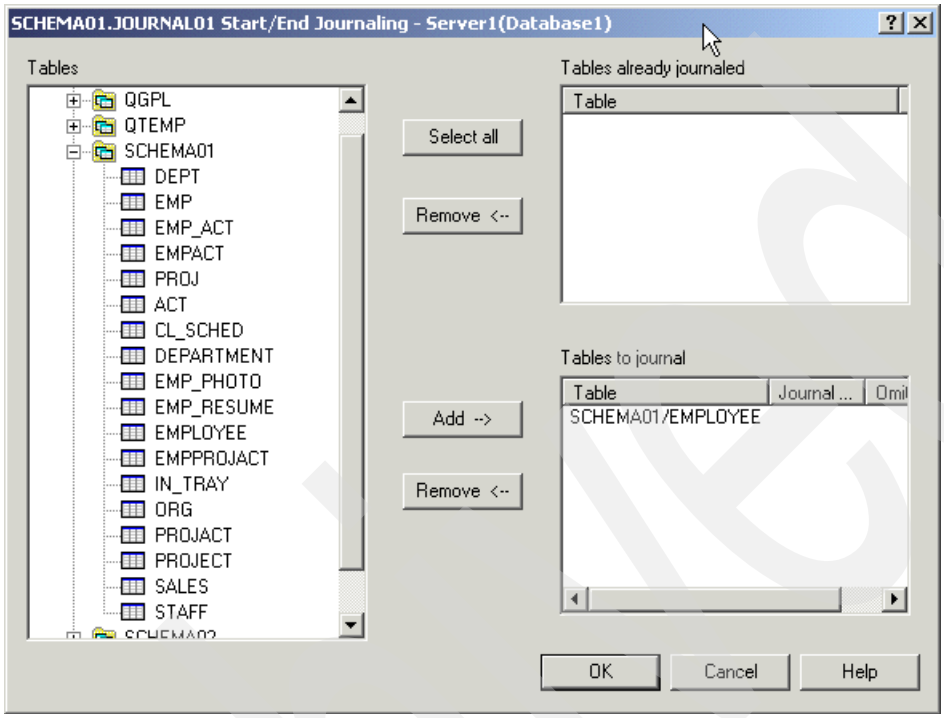


Figure 3-69 Start/End Journaling window

Here is another way to use start and end journaling:

1. Select the table that you want, right-click, and select **Journaling** (Figure 3-70).

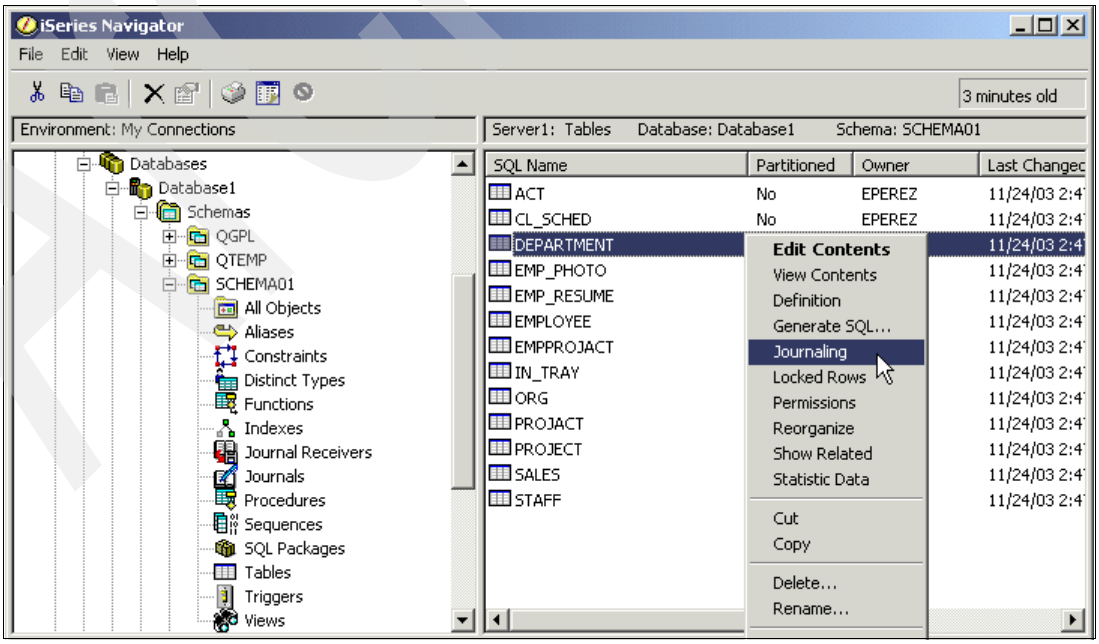


Figure 3-70 Starting a journal for a table

2. The Journaling window (Figure 3-71) opens. It displays information about any journal that is currently or last associated with a table.

   If the status is *Never journaled*, start journaling the table by specifying the name of an existing journal in an existing schema. You can also select the **Journal images before change** and **Include open and close entries** check boxes, depending on the amount of information that you need for recovery or audit.

   Click **Start** or **End**.

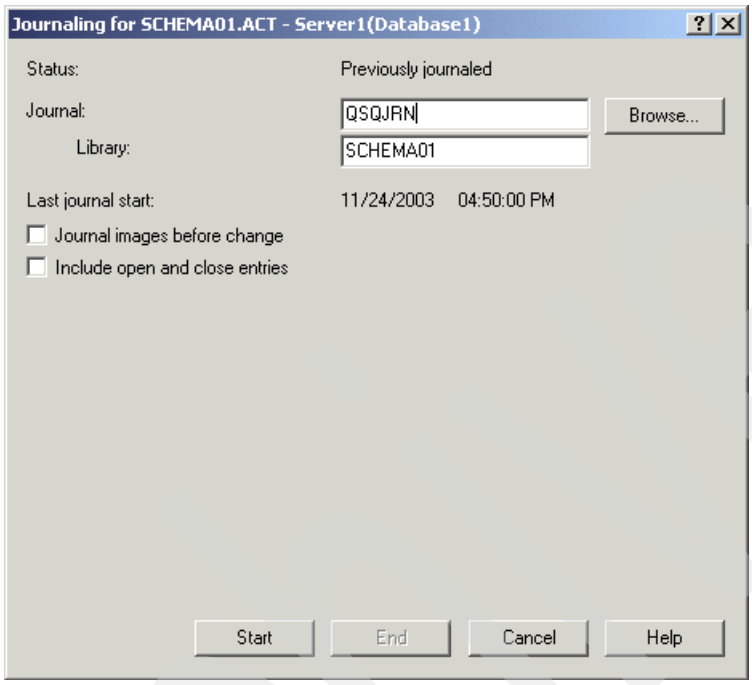


Figure 3-71 Journaling window to start journaling for a table

The following sections discuss 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

The Advanced Journal Attributes window (Figure 3-65 on page 77) contains the following attributes:

- **Journal message queue**: OS/400 issues specific messages for specific changes to the journaling environment. A journaling message is typically issued when a journal receiver reaches its threshold of maximum entries. The message indicates 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 is reached or when the attached journal receiver's sequence number reaches 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 initial program load (IPL), provided that the receiver is not required for commitment control recovery.

Under system managed receivers, you can also specify that OS/400 deletes receivers when they are no longer needed. If you do not select this option, the detached receivers remain on the system until you delete them.

- **Receiver managed by – User**: By clicking User, you take responsibility for changing journal receivers and determining when to delete receivers that 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 places its own entries into a journal receiver. When you select this option, OS/400 deletes these entries from the journal receiver when the system determines that they are no longer necessary.

A good example of these internal entries is those made to support SMAPP (table index). SMAPP journals change to access paths (that is, key columns or fields) independently of whether you use journaling of database tables or files. SMAPP minimizes 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

The New Journal Receiver window (Figure 3-66 on page 77) enables you to specify the following attributes:

- **Journal name and description**: Enter the journal receiver name and journal receiver descriptive text. As shown, the journal name and description are the default values generated by iSeries Navigator. These values are used if you never click the New Receiver button in the Advanced Journal Attributes window.
- **Schema**: Enter the journal receiver schema. The default value shown (SCHEMA02) was specified in the initial New Journal window (Figure 3-64 on page 76).
- **Storage space threshold**: Enter the maximum storage in megabytes that the journal receiver can take. You see the default value of 500 MB.

  The number of journal receiver entries that 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 (Figure 3-65). See the online help for additional details.

Additional journal management

After you create a journal and its initial journal receiver, you can perform additional journal management. Within a schema, right-click either the journal or a journal receiver. Figure 3-72 shows the actions that are possible on an existing journal.

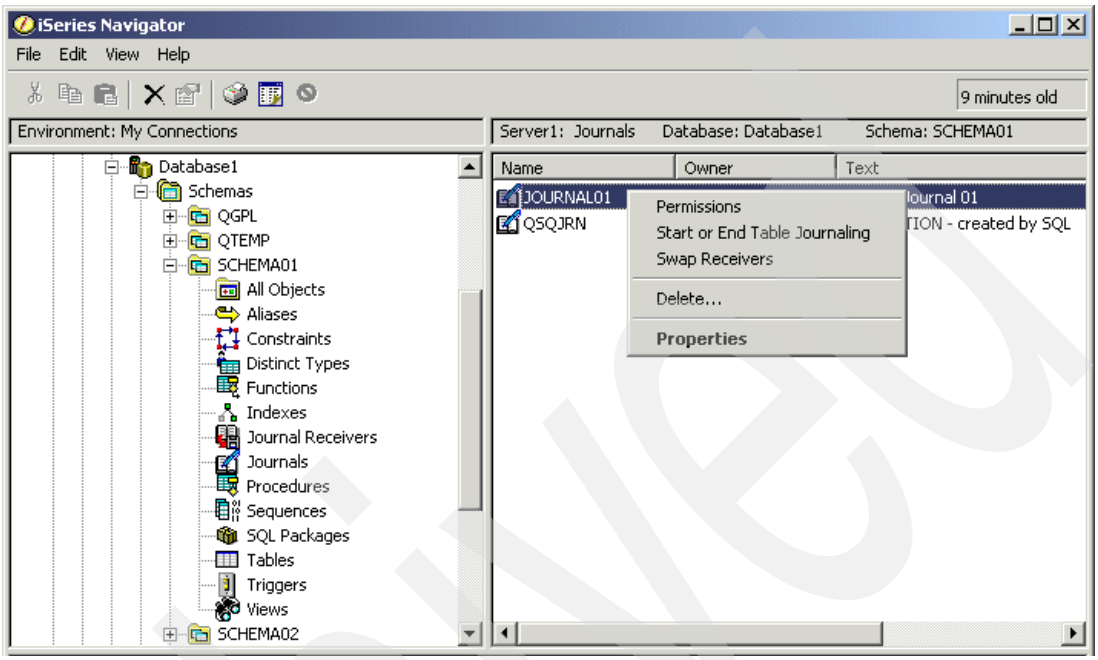


Figure 3-72 Managing a journal

The actions are:

- **Permissions**: This action lets you view and change the authorities to the journal.
- **Start and ends journaling**: This action starts or ends journaling for one or more specific tables. Clicking this action opens the Start/End Journaling window (Figure 3-73).

  The start and end functions correspond to the OS/400 Start Journaling Physical File (STRJRNPF) and End Journal Physical File Change (ENDJRNPF) commands.

  You can start or end journaling, from the item that 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 one to a sequential number suffix to the journal receiver name. You can also manually swap receivers by using an option from the Properties action.
- **Delete**: Clicking this action opens a confirmation window in which you can complete the journal deletion request or cancel it. You can only delete the journal if journaling is ended for *all* objects that are being journaled to the journal.
- **Properties**: This action opens a panel that shows the original create journal attributes. It includes the journal receiver attributes and remote journal attributes, if any. You can create a new journal receiver or remote journal by using the buttons that lead to additional panels.

In the example shown in Figure 3-72, right-click the **JOURNAL01** journal and select **Start and End Table Journaling**. Figure 3-73 shows the Start/End Journaling display for JOURNAL01.

To start journaling for a file or table, you can select the table and click the **Add** button. After you add all that you want journaled to the list box, click **OK**. This starts journaling for these files or tables.

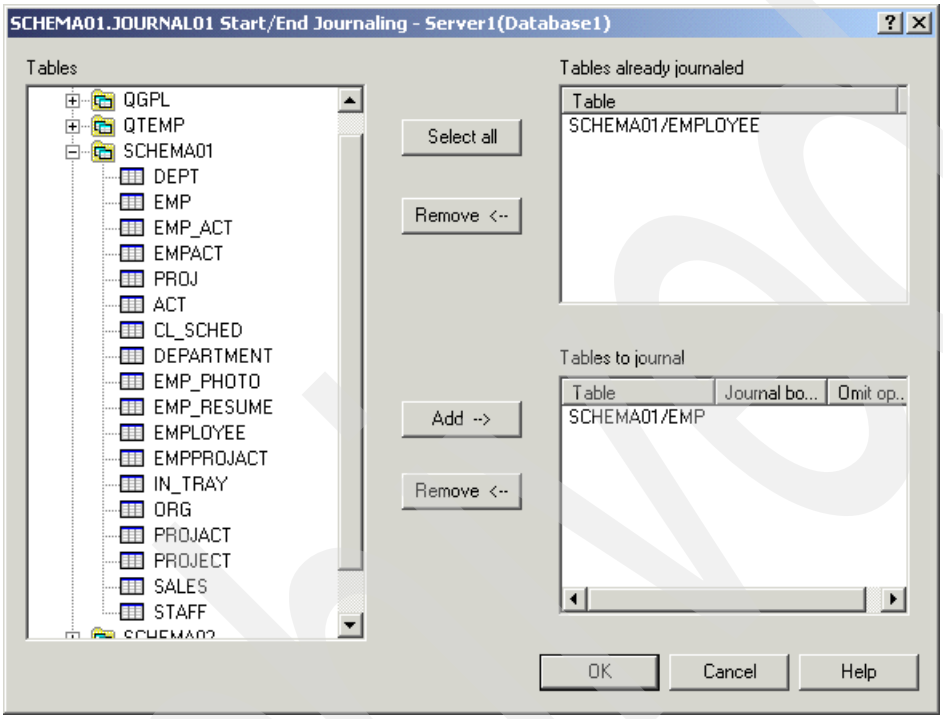


Figure 3-73 Start/End journaling display

In the Tables already journaled box (upper box on the right side), you see the SCHEMA01/EMPLOYEE (system naming convention) table is already being journaled at the time the Properties action was selected. To stop journaling for a file or table, select the file or table in the Tables already journaled box, click **Remove** and then click **OK**.

In the Tables to journal box (lower box on the right side), notice the “Journal bo…” and the “Omit op…” column headings. The Journal bo… heading corresponds to *Journal images before change* option. The Omit op… column heading corresponds to *Include open and close entries* option.

If you click *under* the Journal bo... or Omit op... heading to the right of a table name, an X character appears. If you click again, the X disappears. An X under Journal bo... 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.

Journal properties example

When we right-click the JOURNAL01 journal and select Properties, the Journal Properties window (Figure 3-74) opens. This window shows the original parameters that were used to create the journal. It enables you to make changes and additions.

The *Tables* button gives you access to the Start or End Journaling window. 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 the current status of a remote journal, if any. You can also add a new remote journal.

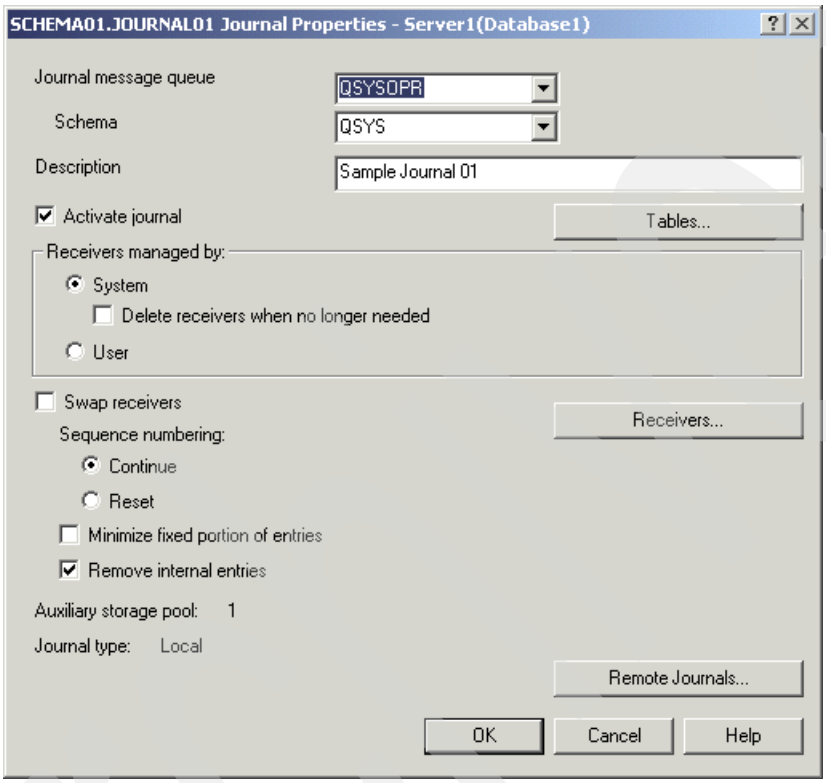


Figure 3-74 Journal Properties window

You can select the Swap receivers check box and optionally specify either Continue or Reset to specify the sequence numbering to be used with the new receiver. Then click OK to detach the current journal receiver and create a new receiver that is attached to the journal. Review online help information (click the ? button and place it over the *Swap receivers* text) to determine whether swap receivers apply to your journaling environment.

In this case, we click the Receivers button to show you the Receivers for Journal window (Figure 3-75). In this example, there is one online receiver (JOURNAL0001). By selecting the JOURNAL0001 journal receiver, the lower portion of the window automatically displays the General properties of this receiver. We already selected the Entries tab information.

We select an online detached journal receiver. Then we click the Delete button to remove the journal receiver and its entries from the system since we no longer need this journaled information.

Note: You cannot delete a journal receiver if the previous journal receivers are not deleted.

If you click the New button, you see an Add Journal Receiver panel. Clicking the OK button makes any new or delete function permanent.

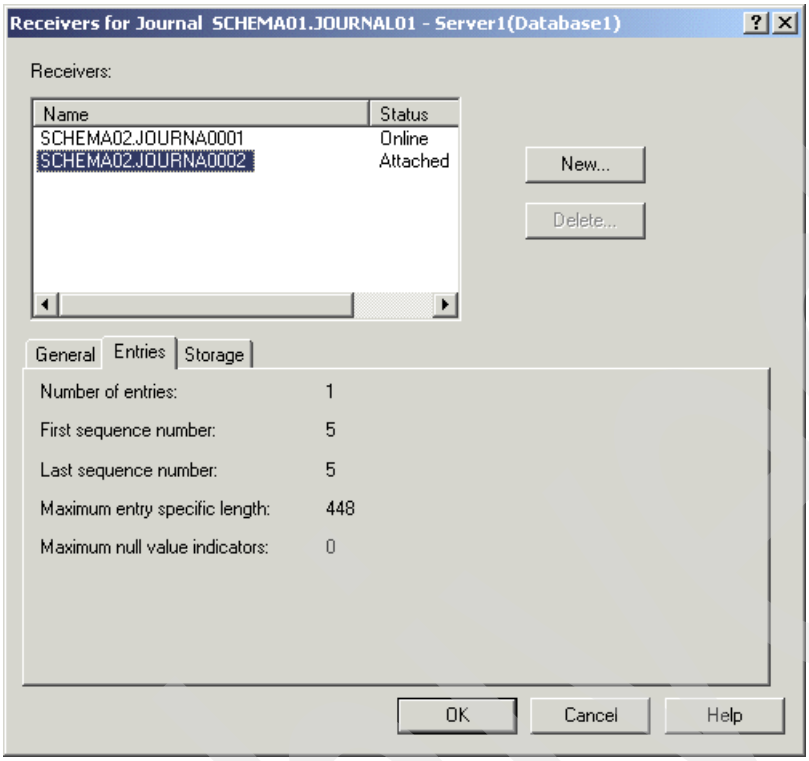


Figure 3-75 Receivers for Journal window

You can 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 Journal Receivers container:

- Double-click the **journal receiver object**.
- Right-click the **journal receiver object** and select **Properties**.

Note: If you want to see the journal entries, you need to use the Display Journal (DSPJRN) command.

3.2.10 Reorganizing tables

Reorganizing a table restores it to its ideal physical organization. The ideal organization for a database table is for its rows to be laid out on pages, ordered by their key values in some frequently used index. You can reorganize a table by compressing out deleted records, by table key, or by a selected index.

Note: In Version 5 Release 1 (V5R1) of iSeries Navigator, you couldn't use the parameter *Reuse deleted rows*. With V5R3, this parameter is now available as a property that you can change at any time. When you create a table, the default value for this parameter is *enable*.

Reorganize table example

In this example, the ORDERS table has an arrival sequence access path. Before the reorganization process, the table has the information shown in Table 3-1.

Table 3-1 Table example (ORDERS)

Relative record number	Customer	Order	OrdDate
1	1050	3000	061503
2	1020	2500	022503
3	deleted		
4	1080	2800	041703

Table 3-2 shows the ORDERS table reorganized using an index with Order column as a key in ascending sequence.

Table 3-2 Table ORDERS reorganized

Relative record number	Customer	Order	OrdDate
1	1020	2500	022503
2	1080	2800	041703
3	1050	3000	061503

Notes:

- If a table with an arrival sequence access path is reorganized using a keyed sequence access path, the arrival sequence access path is changed. The rows in the table are physically placed in the order of the keyed sequence access path that is used. By reorganizing the data into a physical sequence that closely matches the keyed access path you are using, you can improve the performance of processing the data sequentially.
- Reorganizing a table compresses deleted rows, which changes subsequent relative record numbers.
- Because access paths depend on the physical sequence of rows in the table, the sequence of the rows with duplicate key may change after reorganizing a table using a keyed sequence access path.

To reorganize a table using iSeries Navigator, follow these steps:

1. Expand ***your system name*** →**Databases**. Expand the database and schema that contains the table you want to reorganize. Select **Tables**.
2. In the right pane, select the table that you want to reorganize, right-click, and select **Reorganize** as shown in Figure 3-76.

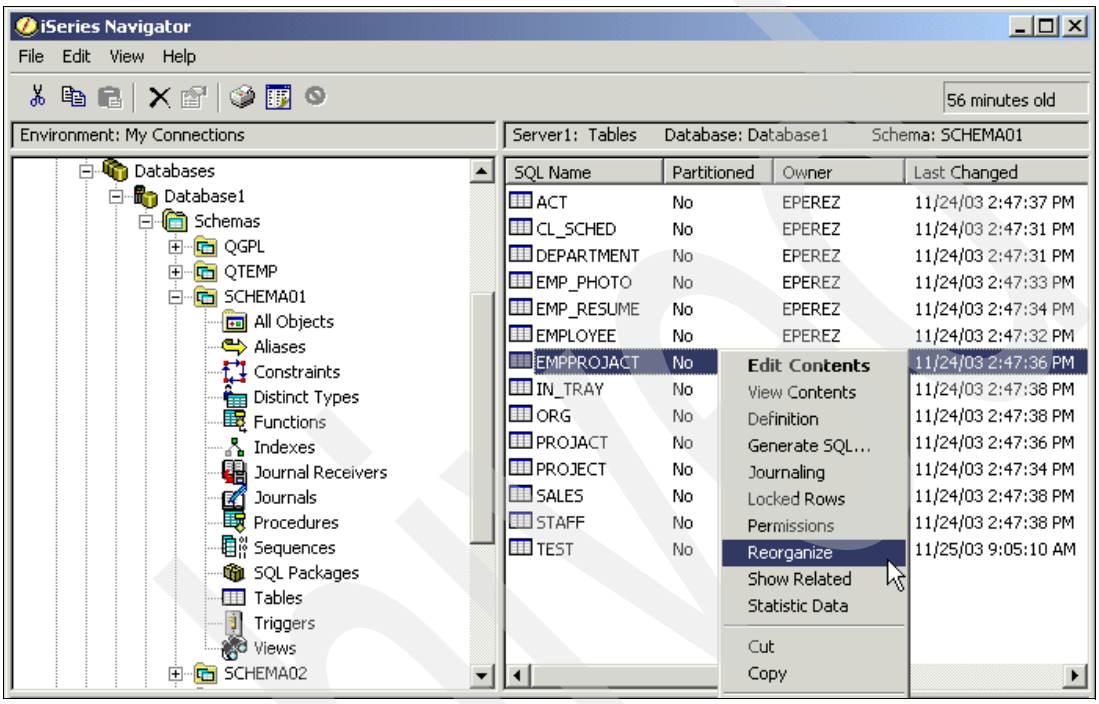


Figure 3-76 Selecting the Reorganize table option

3. In the Reorganize window (Figure 3-77), select the options to specify how to reorganize the rows in the table:
   - **By compressing out deleted rows**: Removes deleted records to make space available for more records. You can also optionally choose to preserve the order in which rows were added to the table:
     - Without Preserving the arrival row sequence: Valid rows at the end of the table are moved to deleted rows until no deleted rows remain.
     - Preserving the arrival row sequence: All valid rows after the first deleted row in the table are moved forward in the table to compress any deleted rows.
   - **By selected index**: Specifies that the rows of the table are rearranged by the key values of an index or keyed logical file that is built over the specified table. You can select only an existing index. Your list of indexes is determined by the schema that you select. If the index that you want to use is not in the list, you can specify the qualified index name.

   With V5R3, you can specify other options in the Reorganize window to control performance and concurrency of the reorganize operation:
   - Specify which partition of a partitioned table (or which member of a multiple member physical file) should be reorganized.
   - Specify whether the reorganize can be suspended and subsequently restarted. If you do not specify that the reorganize can be suspended, the table is allocated exclusively

for the duration of the reorganize operation and can only be suspended by ending the job immediately.

If you specify that the reorganize can be suspended, the file must be journaled since the rows are moved under commitment control to ensure that no rows are lost if the reorganize operation is suspended.

You can also specify whether other users can *read* the table or *change* the table during the reorganize. Locks are acquired for short periods of time on rows that are moved during the reorganize. If concurrent jobs also acquire locks on rows, record lock time-outs may occur. You can change the record lock wait time for the table, or you can use the Override Database File (OVRDBF) command to specify an appropriate record wait time.

– Specify how access paths are handled during and after the table reorganization. If you specify that the reorganize can be suspended, you may also specify to maintain all indexes during the reorganize. No index rebuilds are necessary. Otherwise, you may specify that indexes be rebuilt synchronously or asynchronously.
  - Rebuild at the end: Specifies that the indexes should be rebuilt after the reorganization is complete
  - Rebuild when optimal: Specifies that the database automatically chooses to rebuild indexes or maintain the original indexes based on optimization rules

Click **OK** to start the reorganization process.

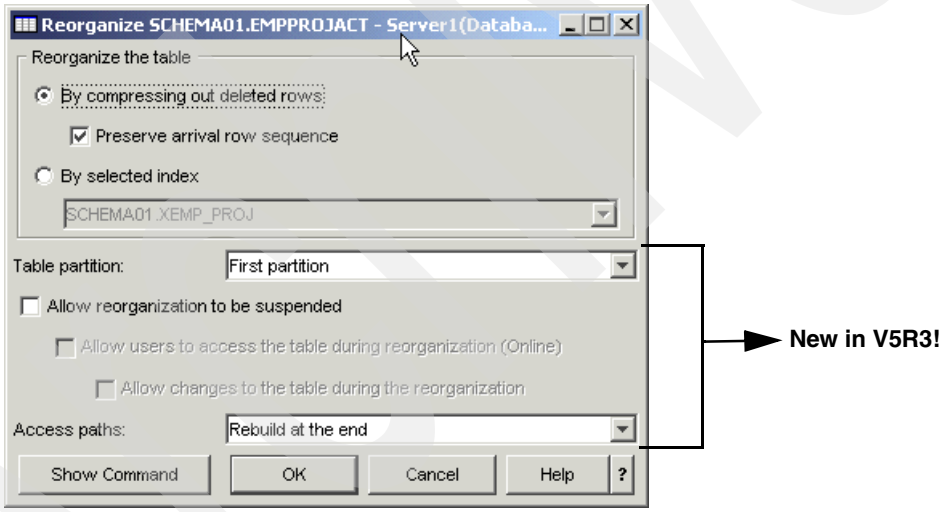


Figure 3-77 Reorganizing a table

4. The Reorganize window (Figure 3-78) opens. In the upper pane in this window, you see the overall status of the process. The messages indicate the individual steps of the reorganization as they occur. The lower pane shows the options that you selected in the Reorganize Table window for the process that is currently running.

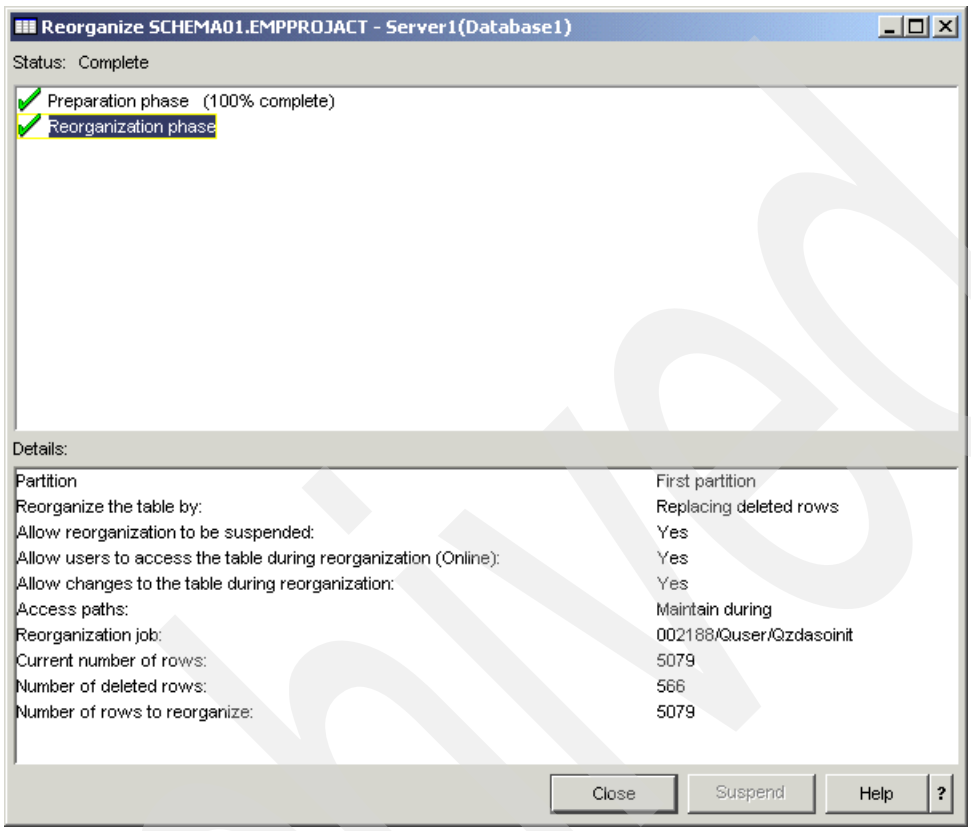


Figure 3-78 Reorganization process

In the upper pane, you can double-click in any of the phases to expand and see more information about the progress of the process. The results are shown in Figure 3-79.

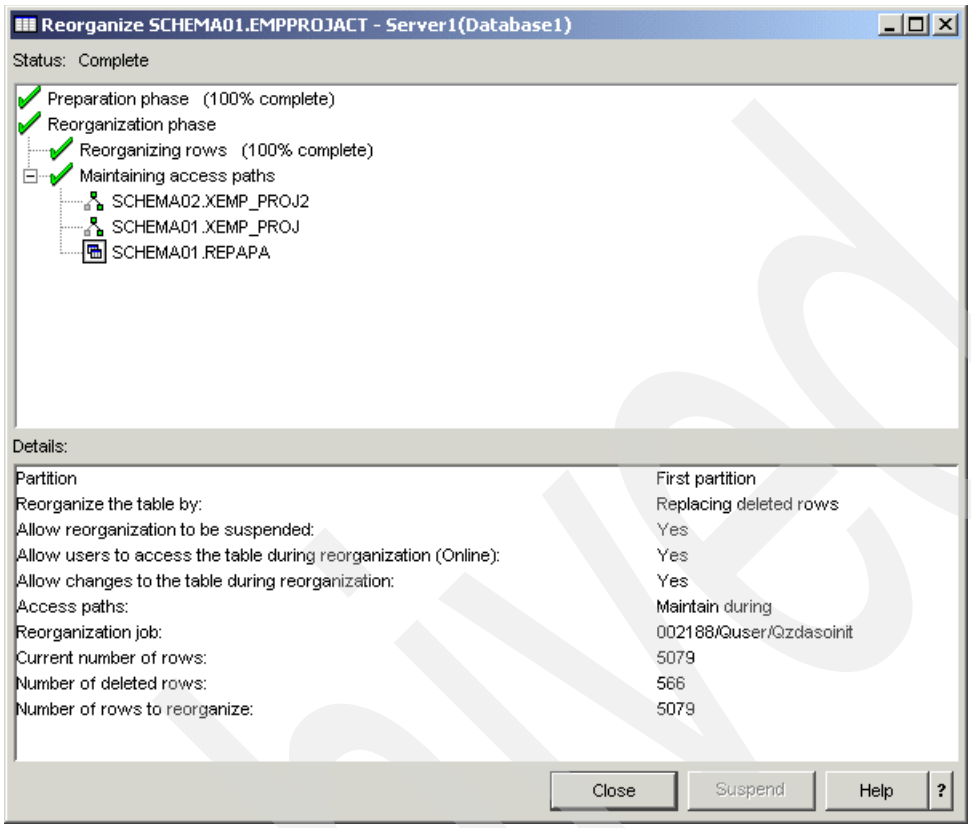


Figure 3-79 Reorganization process detail

Suspending a reorganization process

You can suspend the reorganization while the process is still active. You simply click the Suspend button. The confirmation window, like the example in Figure 3-80, opens.

Note: The reorganization process continues until you click Yes in the confirmation window.

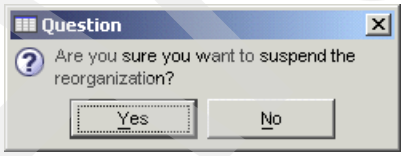


Figure 3-80 Stopping the reorganization process

If the process is still active and you click the Yes button, the process stops and the status for all phases is displayed, as shown in Figure 3-81.

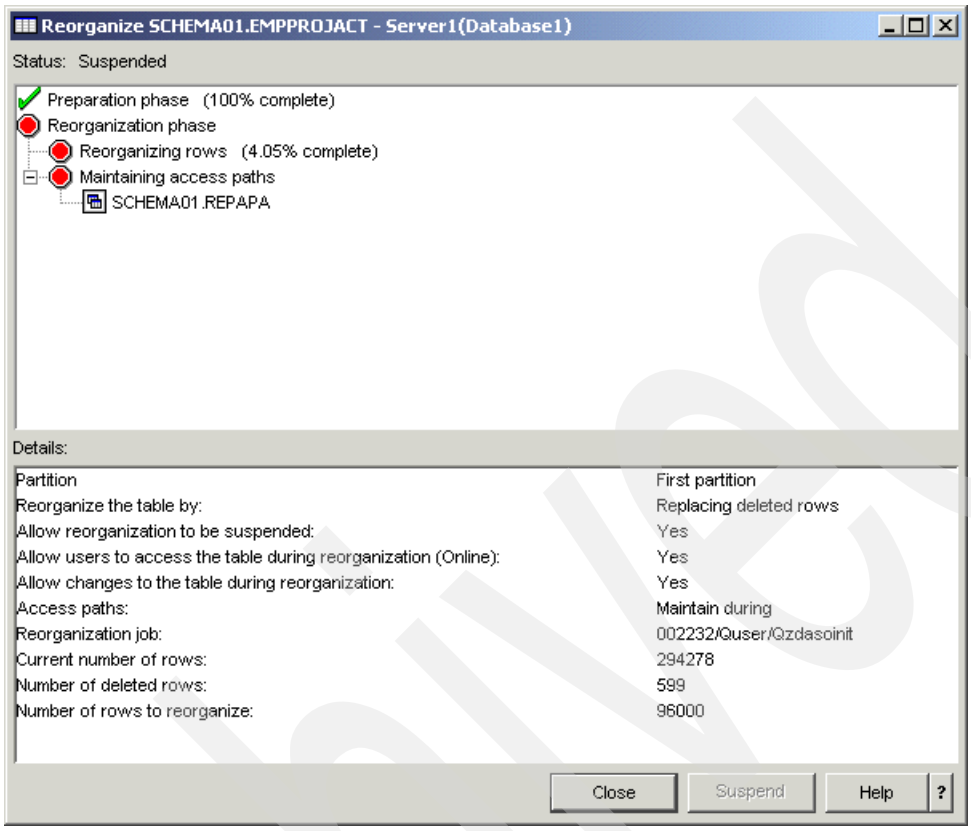


Figure 3-81 Reorganization process suspended

If you suspend the reorganization process for a table, the next time you try to reorganize the table, a window opens that shows information about the last reorganization and the options to resume or to start a new process (Figure 3-82). If you click Yes to resume the process, it begins from the point when it was suspended.

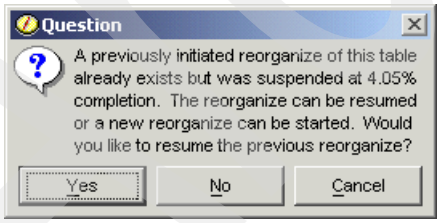


Figure 3-82 Resuming reorganization process

3.2.11 Transactions

A transaction is a group of individual changes to objects on the system that should appear as a single atomic change to the user. A transaction is sometimes called a *logical unit of work* (LUW). Commitment control is used to ensure that either the entire group of individual changes occur on all systems that participate or that none of the changes occur.

There are two different types of transactions that an iSeries server supports: global transactions and database transactions. You can use iSeries Navigator to display information about both transaction types.

Database transactions

A database transaction contains only DB2 UDB for iSeries changes. The transaction is completed by using one of the interfaces provided by the system, such as SQL COMMIT, SQL ROLLBACK, CL COMMIT, CL ROLLBACK, or RPG COMMIT.

> **Important:** Locks that are held on behalf of a database transaction are scoped to the job that performs the transaction. The transactions cannot move from the job that started it.

To work with database transactions using iSeries Navigator, follow these steps:

1. Expand **Databases** and select the database that you want to work with.
2. Expand **Transactions**. Select **Database Transactions** as shown in Figure 3-83.

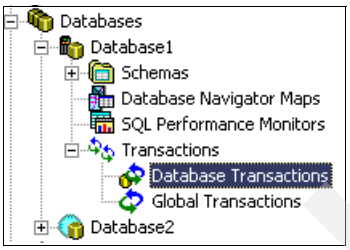


Figure 3-83 Database Transactions container

3. Select the transaction that you want to work with, right-click, and choose one of the following options:
   - **Force Commit** if you want to force the transaction to commit and select **Yes** in the confirmation window
   - **Force Rollback** if you want to force the transaction to rollback and select **Yes** in the confirmation window

     > **Restriction:** You can force a transaction only when the unit of work state is *Prepared* or *Last Agent Pending*.

   - **Cancel Resynchronization** if you want to cancel resynchronization and select **Yes** in the confirmation window

     You can cancel resynchronization when a resynchronization is in process as indicated in the right pane of iSeries Navigator or on the General tab of the transaction Properties window.

The Properties window in Figure 3-84 shows you the unit of work ID, unit of work state, job details, commitment definition, and whether a transaction is being resynchronized.

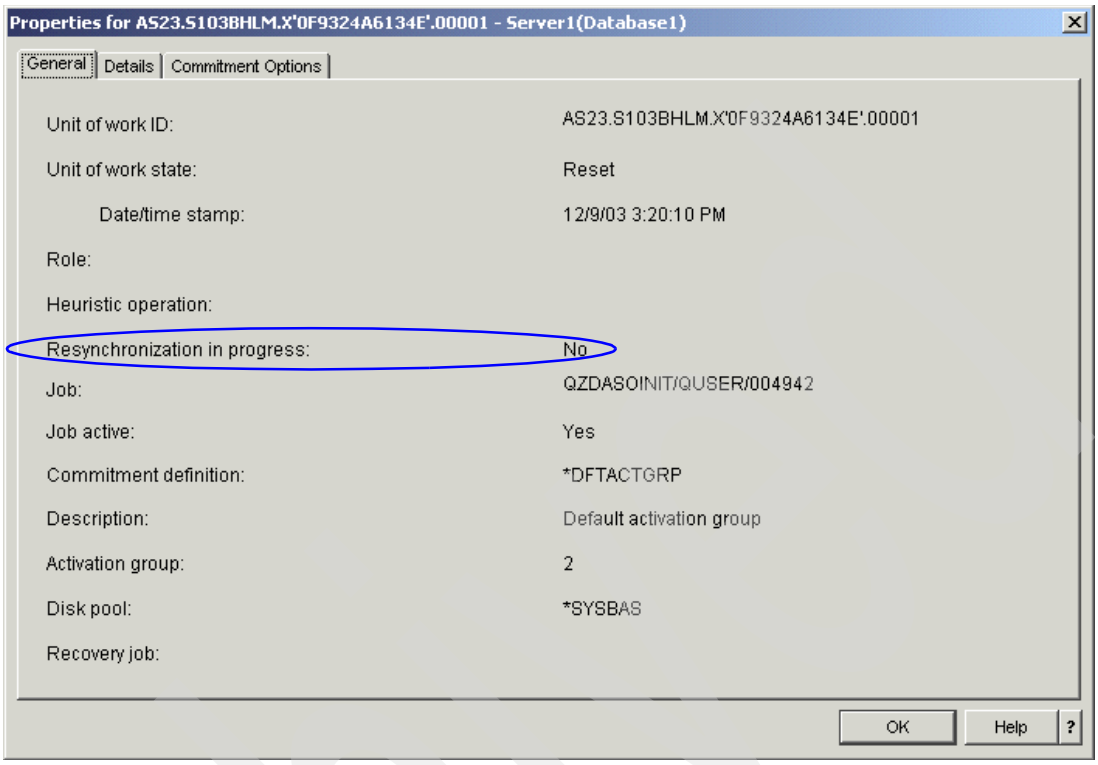


Figure 3-84 Transaction properties

From the context menu of a transaction, you can also choose Jobs to view a list of jobs that are associated with the transaction. From the list of Jobs window, you can work with the locks, as well as see open files and the last SQL statement executed in the job, for example.

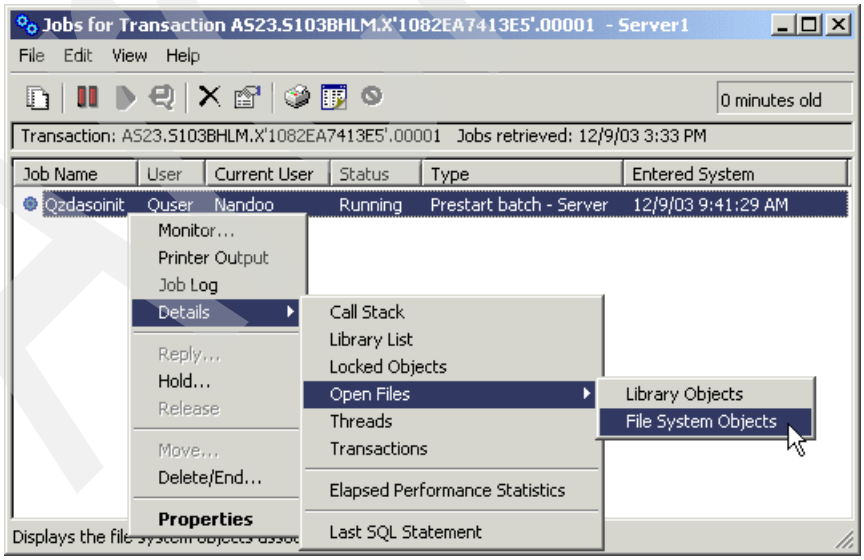


Figure 3-85 List of jobs for a transaction

Optionally, you can choose Resource Status to display resource information about Record, Object, Journal, etc.

Global transactions

A global transaction may contain changes both outside and within DB2 UDB for iSeries. A global transaction is coordinated by an external Transaction Manager using the Open Group XA architecture or another similar architecture. An application commits or rolls back a global transaction by using interfaces provided by the Transaction Manager. The Transaction Manager uses commit protocols defined by the XA architecture, or another architecture, to complete the transaction. DB2 UDB for iSeries acts as an XA Resource Manager when participating in a global transaction.

For more information about global transactions and database transactions, refer to the iSeries Information Center at:

http://www.iseries.ibm.com/infocenter

When you reach this site, select **Databases** →**Commitment Control**. Then select the **Manage commitment control** and **Troubleshoot transactions** topics.

4

Defining and managing constraints

This chapter begins with an introduction to constraints. Then it takes a close look at key constraints, referential constraints, and check constraints. Finally, it explains how to manage constraints using iSeries Navigator.

4.1 Constraints

A *constraint* is a restriction or limitation placed on your tables to ensure that data in your database remains consistent as you add, change, and remove rows. DB2 Universal Database (UDB) for iSeries supports the following types of constraints:

- A *unique constraint* guarantees that the values of a key are unique. Unique and primary keys are the supported unique constraints.
- A *referential constraint* ensures that all non-null values of foreign keys in a dependent table have a corresponding parent key in a parent table.
- A *check constraint* limits the values that are allowed in a column or group of columns.

4.2 Key constraints

A *key constraint* is added to a column in a table to prevent duplicate information. In iSeries Navigator, you can add primary keys and unique keys. When primary keys are joined with a foreign key, they create a relationship between two tables called a *referential constraint*. A table can contain only one primary key, multiple unique keys, and multiple foreign keys.

4.2.1 Unique constraints

A unique constraint acts as a control in a database to ensure that rows are unique. For example, you can specify a customer identification number as a unique constraint in your database. If anyone attempts to create a new customer with the same customer number, an error message is issued.

Unique constraints identify a column or a set of columns in a table where the values must be unique across all rows of that table. The column must be in ascending order and can be null-capable.

Defining a unique key constraint

To define a new unique constraint on an existing EMPLOYEE table using iSeries Navigator, follow these steps:

1. Select the table on which you want to define the key constraint. Then right-click and choose **Definition**.
2. In the Table definition window, click the **Key Constraints** tab.
3. On the Key Constraints page (Figure 4-1), click **Add** to define a new key constraint. Ensure that no other key constraint is already defined on the same column. In this example, we make the SSN column unique.

> **Important:** A table can have multiple unique constraints, but you cannot duplicate such constraints. The same key columns, regardless of order, constitute a duplicate constraint.

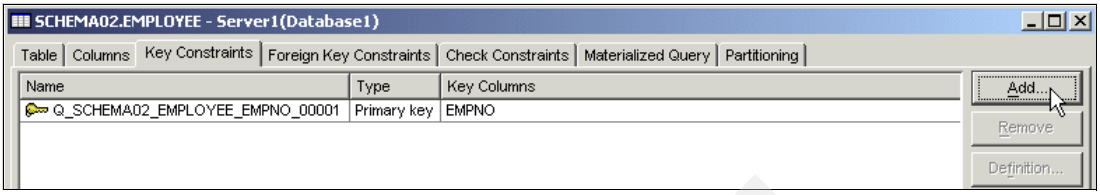


Figure 4-1 Key Constraints tab in the Table definition window

4. In the New Key Constraint definition window (Figure 4-2), complete these tasks:
   a. For Constraint type, make sure that **Unique key** (the default) is selected. Optionally, you can type a name for the constraint or leave it as **System-generated**.
   b. Under Available columns, select the **SSN** column and click **Add**.
   c. Because we are making only one column unique in this exercise, we do not need to set the order for the columns. Click **OK**.

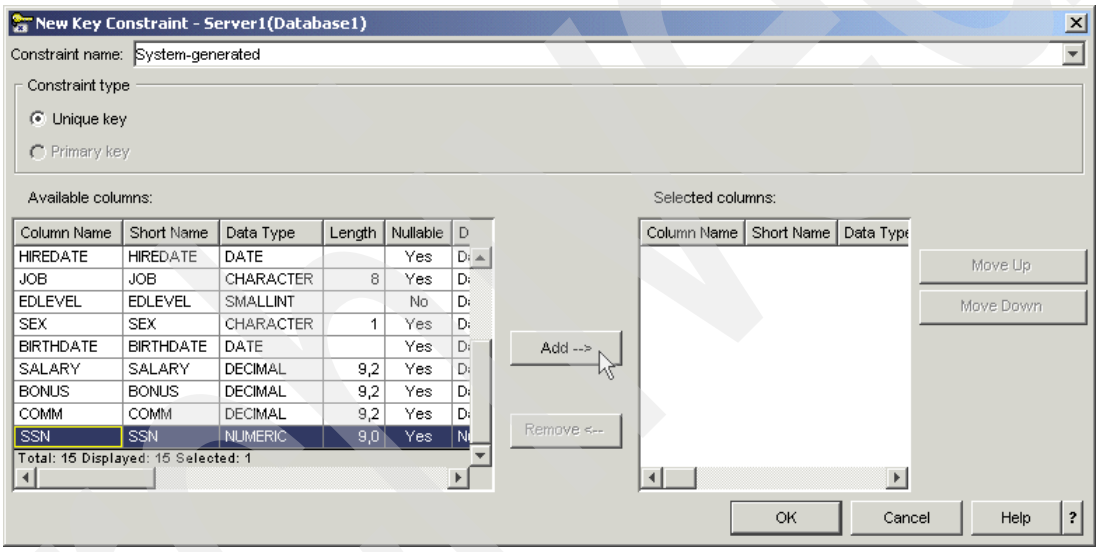


Figure 4-2 New Key Constraint definition window

5. After you return to the Table definition window, click **OK** to create the constraint.

> **Note:** You may modify a constraint only if you defined it during your current table editing session. If you added the constraint and then clicked OK on the Table definition window, then you have read-only access for the constraint. To change the constraint properties, you must drop the constraint and then recreate it with the appropriate changes.

6. If there are any duplicate values for the SSN column in the EMPLOYEE table, you see an error message on an Exception window (Figure 4-3). Click **Job Log**.

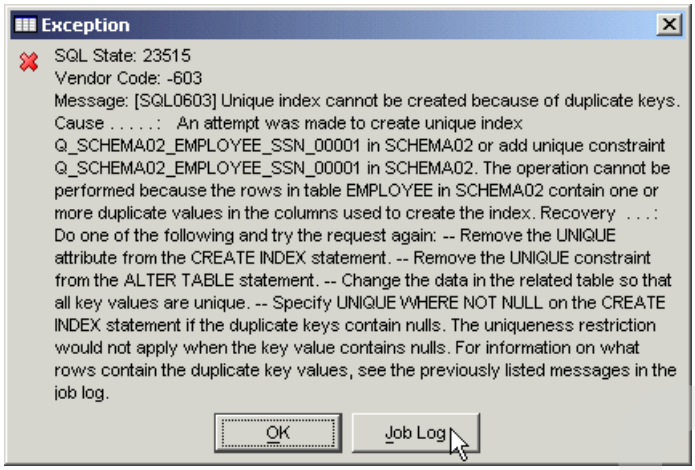


Figure 4-3 Exception - Duplicate key values exist

7. In the Job Log window, double-click each line with a Message ID of **CPF3240**. Make note of the record numbers from the Message help section.
8. Leave the Table definition window open. In iSeries Navigator, select the table, right-click, and choose **Edit Contents**.
9. After the duplicate values are fixed, in the Table definition window, click **OK** to add the unique constraint. This creates a new unique index that is part of the table itself.

Note: A unique key constraint can be added to a partitioned table but it cannot be added to a physical file that can have more than one member.

4.2.2 Primary key constraints

A primary key constraint is a unique key with special attributes that make the key the primary access path for the table. You can define only one primary key constraint for a file.

Primary key constraints identify a column or set of columns in a table where the values must be unique across all rows of that table. The column must be in ascending order and can be null-capable.

Note: When one or more key columns in a primary key are null-capable, a check constraint is implicitly added so that you cannot enter null values in the column or columns. If the column already contains null values, you cannot define a primary key constraint over it.

Defining a primary key constraint

Defining a primary key constraint in iSeries Navigator is similar to defining a unique key constraint. In general, an identity column of a table is a good choice for a primary constraint. Therefore, create a table with an identity column or use an existing table that has an identity column as explained in the following steps. Refer to “Adding an identity column to a table” on page 33 for more information about creating an identity column.

To define a primary key constraint, we use an existing CUSTOMER table defined with SQL statements (Example 4-1). Notice that the CUSTNUM identity column is defined with the CYCLE option. Before you define a primary key constraint, you need to modify the identity column and remove the CYCLE option.

Example 4-1 Defining a primary constraint using the CUSTOMER table defined with SQL statements

```
CREATE TABLE SCHEMA02.CUSTOMER (
   CUSTNUM INTEGER GENERATED ALWAYS AS IDENTITY (
   START WITH 1 INCREMENT BY 1
   MINVALUE 1 MAXVALUE 100
   CYCLE NO ORDER
   CACHE 20 ) ,

   FIRSTNAME CHAR(20) CCSID 37 DEFAULT NULL ,
   LASTNAME CHAR(30) CCSID 37 DEFAULT NULL ,
   ADDRESS1 CHAR(30) CCSID 37 DEFAULT NULL ,
   ADDRESS2 CHAR(30) CCSID 37 DEFAULT NULL ,
   CITY CHAR(30) CCSID 37 DEFAULT NULL ,
   STATE CHAR(2) CCSID 37 DEFAULT NULL ,
   PHONE NUMERIC(10, 0) DEFAULT NULL ) ;
```

Important: When CYCLE is in effect, duplicate values can be generated by the database manager for an identity column. If a unique constraint or unique index exists on the identity column, and a non-unique value is generated for it, an error occurs.

Follow these steps to define the primary key constraint for this table:

1. Select the **CUSTOMER** table, right-click, and choose **Definition**.
2. In the window that opens, click the **Columns** tab.
3. On the Columns page, select **CUSTNUM** and click **Definition** to modify the column.

4. In the Column Definition window (Figure 4-4), clear the **Cycle values when the maximum or minimum value is reached** option and click **OK**.

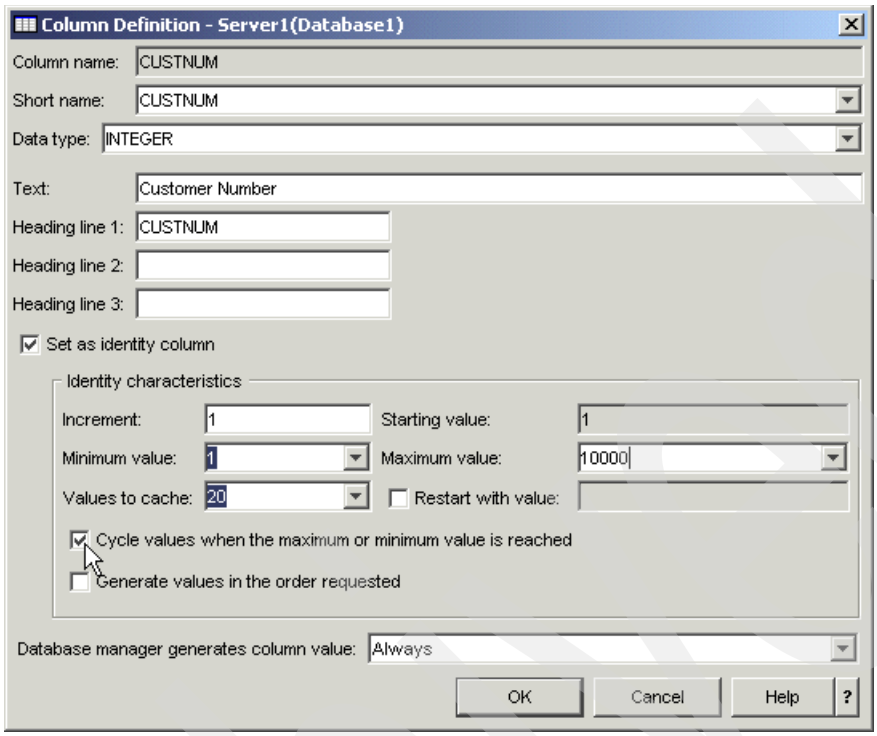


Figure 4-4 Clearing the cycle option for the identity column

5. In the Table definition window, click the **Key Constraints** tab.
6. On the Key Constraints page, click **Add**.

7. The New Key Constraint window (Figure 4-5), complete these tasks:
   a. For Constraint name, type a name.
   b. For Constraint type, select **Primary key**.
   c. Under Available columns, select CUSTNUM and click **Add**.
8. In the Constraint Definition window, click **OK**.
9. In the Table definition window, click **OK**. If there are no errors, the primary key constraint is added to the table.

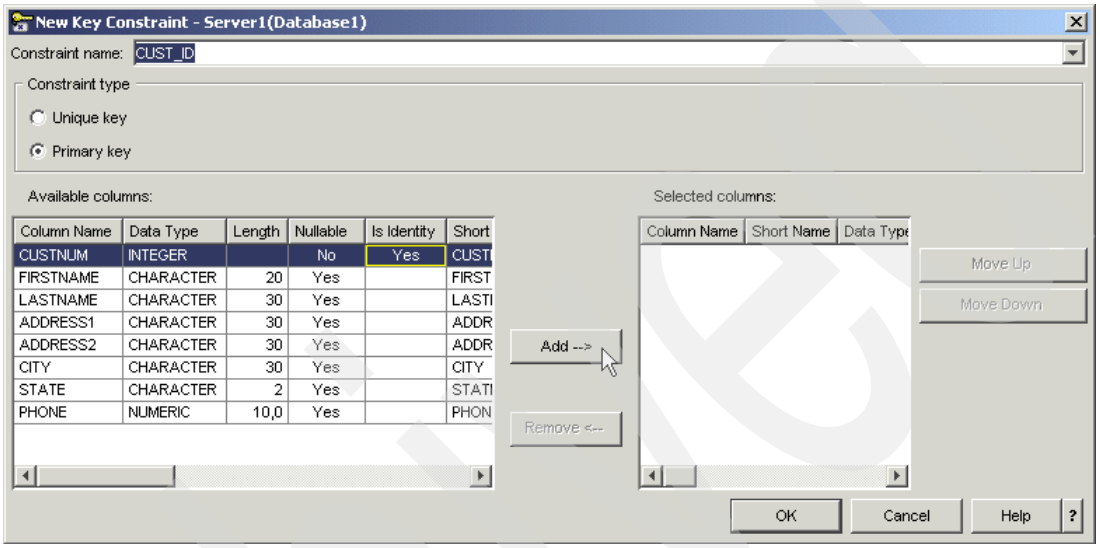


Figure 4-5 Add a column for the primary key

Note: You can add a primary key constraint for a uniquely-keyed physical file (with data description specifications (DDS)) only when the columns of the primary key match those in the access path. If it is a keyed physical file (without UNIQUE), you cannot add a primary key constraint.

4.3 Referential constraints

A referential constraint creates a relationship between a dependent table and a parent table. The system guarantees that, for each row in the dependent table, with a non-null value in all of its foreign key columns, a row exists in the parent table with a matching value in its parent key.

After you set up your business rule as a referential constraint, any program that attempts to insert, delete, or update a row cannot violate that constraint. This relationship is established by creating key values between the two tables. These keys are called the *parent key* in the parent table and the *foreign key* in the dependent table. The parent key can be either a *primary* or a *unique key*.

The state of a database, in which each non-null foreign key value has a matching parent key value, is known as *referential integrity*.

4.3.1 Constraint prerequisites

You can find a full description about the prerequisites and limitations on the database tables and the constraints in the *Database programming* topic (select **Database** →**Programming** → **Database programming**) in the iSeries Information Center at:

http://www.iseries.ibm.com/infocenter

The basic requirement is that your parent key and foreign key must have matching field attributes and definitions. Also, performance is better when the 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, 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 database management system (DBMS) spends verifying referential integrity.

Note: 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 to enforce these relationships at a lower level in the system that eliminates a majority of the overhead associated with application-level enforcement.

Journaling and commitment control requirements

When a referential constraint is defined with a delete or update rule, other than RESTRICT, the system must perform some actions on the corresponding foreign keys each time a delete or an update of the parent key takes place. In the case of a delete, for example, the system 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.

Referential integrity and access paths

DB2 UDB for iSeries uses indexes (or access paths) 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 table, the system first tries to share an existing path. If one cannot be shared, a new access path is created. If the system can share an existing access path, the ownership is transferred to the table itself.

Table constraints are not separate objects and are part of the table description. In fact, when a table is saved, the system also saves all the constraints and their associated access paths that have been defined for that table. For this reason, we recommend that, when a unique keyed access path is required, define a unique constraint instead of an index. These indexes are also considered by the optimizer when implementing a query.

File availability

When adding a referential constraint, the DBMS exclusively locks the table and access paths that are involved. The system must then verify that every foreign key value is valid. When the existing tables contain a large number of rows (hundreds of millions), this process can

possibly run for hours. Consider the impact on table availability before you create a constraint during normal system activity.

Verification of data before defining referential integrity

Before you create referential constraints over existing tables, you may want to check whether any mismatches exist between your candidate parent and foreign keys. You can determine unmatched (or orphan) foreign key values by using the following query:

```
SELECT * FROM MySchema/DependentTable
         WHERE ForeignKey NOT IN
         (SELECT ParentKey FROM MySchema/ParentTable)
```

4.3.2 Creating a referential constraint

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. Using iSeries Navigator, in the New Foreign Key Constraint definition window, you can define a foreign key for a table. The following steps use ORG and STAFF tables to illustrate this:

1. From iSeries Navigator, select the **ORG** table, right-click, and select **Definition**.
2. In the Table definition window, click the **Key Constraints** tab.
3. One the Key Constraints page, click **Add**.
4. In the New Key Constraint window (Figure 4-6), complete these tasks:
   a. Select **Primary key**.
   b. Under Available columns, select **DEPTNUMB** and click **Add-->**.
   c. Click **OK**.

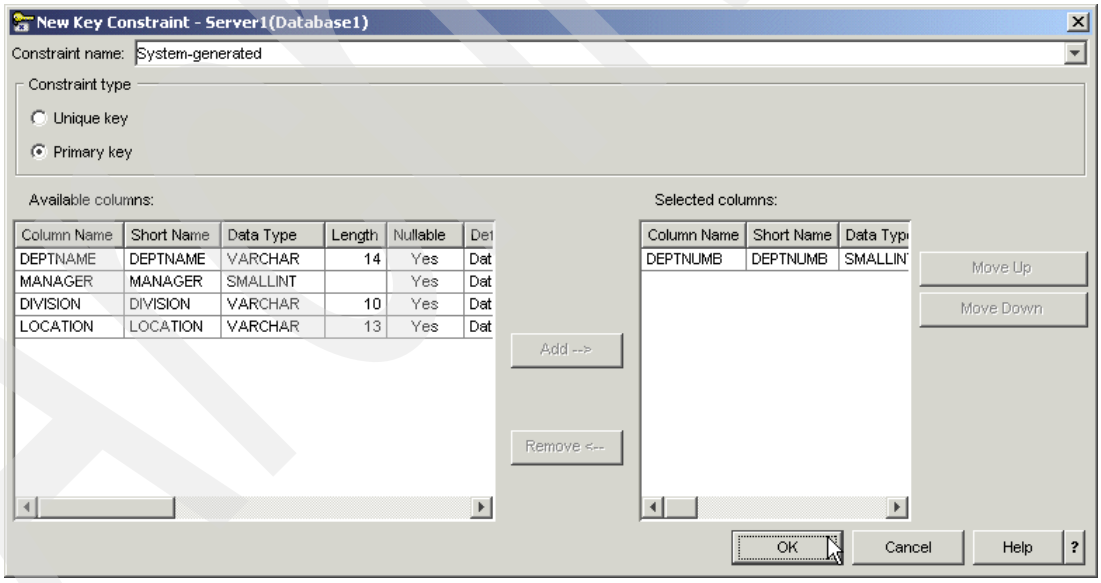


Figure 4-6 New primary key constraint

5. From iSeries Navigator, select the **STAFF** table, right-click, and choose **Definition**.
6. In the Table definition window, select the **Foreign Key Constraints** tab.
7. On the Foreign Key Constraints page, click **Add**.
8. In the New Foreign Key Constraint definition window (Figure 4-7), complete these tasks:
   a. In the Parent table, for Table name, select the **ORG** table. A possible candidate for the parent key is automatically selected and shown under the Key columns section.

b. Under the Available columns section, select the **DEPT** column.

c. Click **Add** to move the column to the Selected columns section and to define the foreign key relationship.

d. From the Action upon delete drop-down list, choose the appropriate action that is to occur on the dependent table when a row of the parent table is deleted. Refer to the online help for more information about each of the available options.

e. Optionally, you can also choose an appropriate action from the Action upon update drop-down list.

f. Click **OK**.

9. In the Table definition window, click **OK** to add the foreign key constraint.

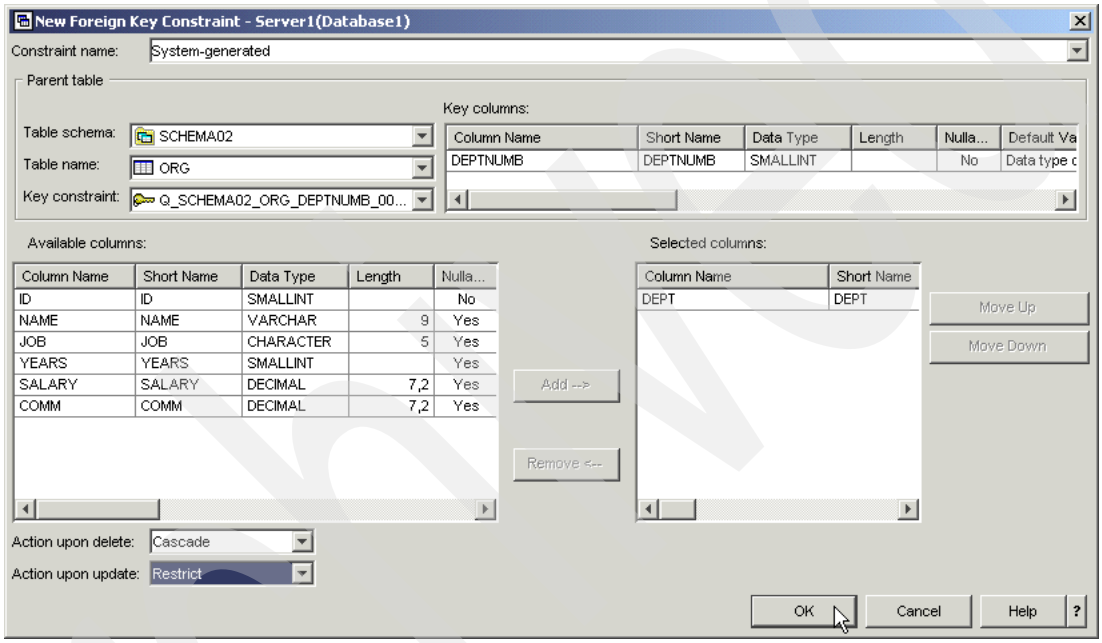


Figure 4-7 New Foreign Key Constraint definition window

Self-referencing constraints

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

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

To illustrate this, we use the sample STAFF table that has a new column MGRID. With the self-referencing constraint, the manager of an employee must also be an employee and be the parent key of their associated employee rows in the same table. After this constraint is established and enforced, you can only insert an employee row if the corresponding manager is already inserted. Therefore, the first row to insert is for the Chief Executive Officer. This row’s foreign key value is NULL. Afterwards, your insertions follow each branch of the hierarchy down to the lowest level.

In this case, you need to define a referential constraint with MGRID as a foreign key and ID as a parent key. STAFF is both a parent and a dependent table as indicated in Figure 4-8.

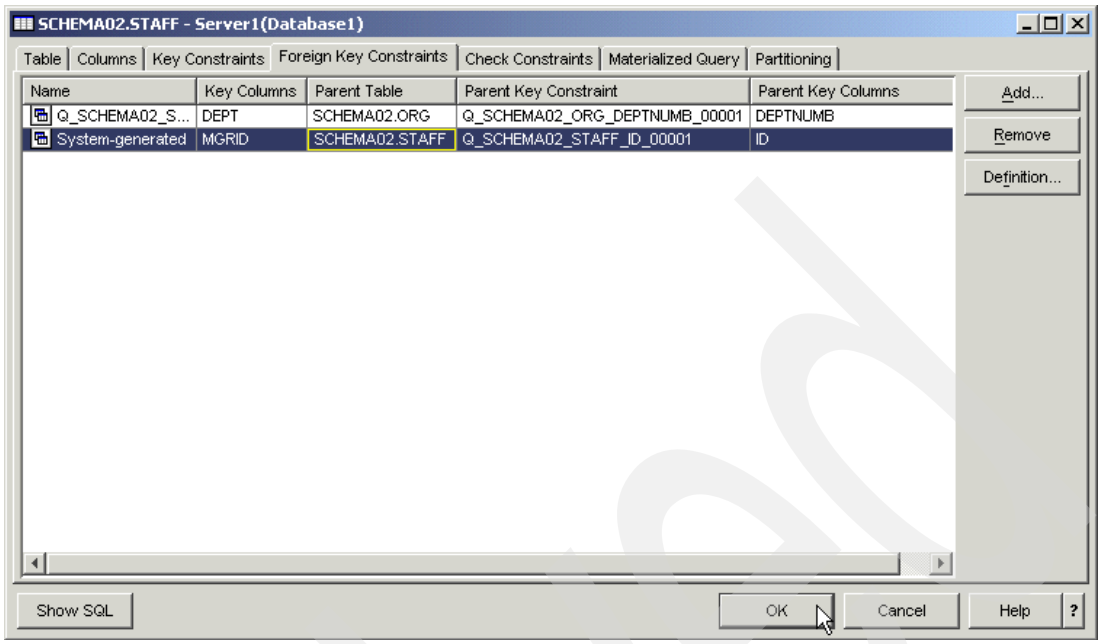


Figure 4-8 Self-referencing constraint

4.4 Check constraints

A check constraint is created on a column to ensure that the value of that column conforms to a specified value or range of values established by your business rules. Whenever you insert or update a column with a check constraint, the check constraint validates the new data against the business rule. It disallows any attempt to violate it.

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

4.4.1 Defining a check constraint

Check constraints simplify business rule checking. Because this checking is done by the database manager, it increases the data integrity and may improve performance rather than when an application does this checking. For example, you may have a business rule, which states that an employee can never have a commission more than the employee's salary. To set this rule as a check constraint on STAFF table, follow these steps:

1. In the Table definition window for STAFF table, select the **Check Constraints** tab.
2. On the Check Constraints page, click **Add** to define a new check constraint.
3. In the New Check Constraint window (Figure 4-9), complete these tasks:
   a. Leave Constraint name as **System-generated** or optionally specify a name for the constraint.

> **Important:** The name specified must be unique across all constraint types in the schema.

b. Under Columns, select the **COMM** column and click the **Add to Check Condition** button.

c. Under the Operators pane, select **<=** and click the **Add to Check Condition** button.

d. Repeat step b and c for the SALARY column.

e. Click **OK**.

4. In the Table definition window, click **OK** to add the check constraint to the STAFF table.

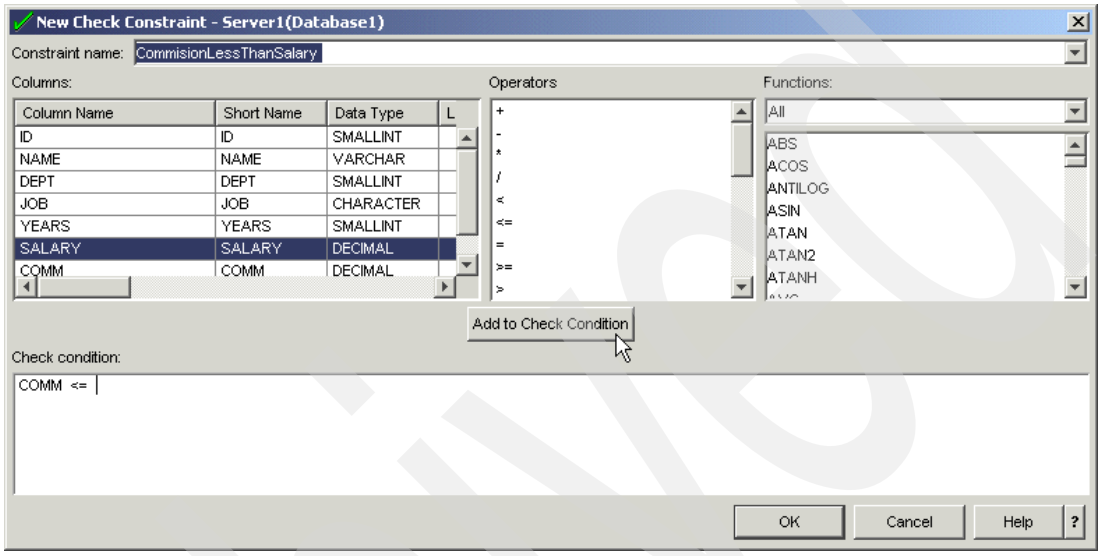


Figure 4-9 New Check Constraint definition window

4.4.2 General considerations

Remember these points when you define CHECK constraints:

- 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.
- DB2 UDB for iSeries does not prevent conflicting constraints from being defined.

> **Important:** The developer must ensure that check constraints are not mutually exclusive. In addition, if the table is a range partitioned table, and if you define a check constraint that does not match this range, then you can *never* insert data into that table.

You must also be aware of the following considerations:

- A table or file has a limit of 300 combined constraints (referential constraints, primary, unique, and check constraints).
- You cannot add constraints to tables in the QTEMP library.

- When you add a check constraint, DB2 UDB for iSeries makes an exclusive lock on the table for the verification of the condition clause.
- Referential constraints must have the parent and dependent table in the same auxiliary storage pool (ASP).

4.5 Managing constraints

Using iSeries Navigator, you can easily manage all types of constraints. You can add, remove, or enable and disable constraints from all the windows where appropriate. The constraints container shown in Figure 4-10 gives you quick access to all types of constraints that exist in a schema. The view from the constraints container indicates whether a constraint is enabled or disabled or is in *check pending state*.

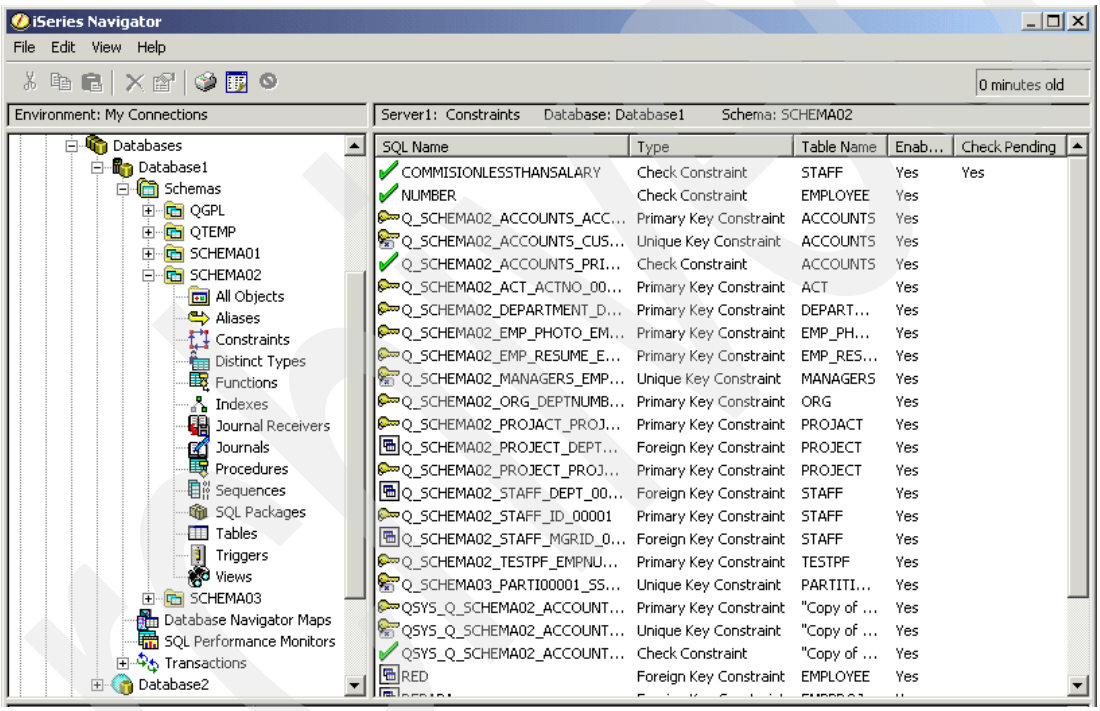


Figure 4-10 Constraints container

4.5.1 Constraint states

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

- **DEFINED state**: The constraint definition exists at the table level, but the constraint is not enforced. Defined constraints are purely by definition and not by function.
- **ESTABLISHED state**: A referential constraint is established when the foreign key attributes match those of the parent key and both tables contain a member. The constraint is now 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.

For more information, refer to *Database programming* in the iSeries Information Center at:

http://www.iseries.ibm.com/infocenter

When you reach this site, select **Database** →**Programming** →**Database programming**.

4.5.2 Managing check pending constraints

Check pending is the condition of a constraint relationship when potential mismatches exist between the parent and foreign keys. When the system determines that referential integrity may have been violated, the constraint relationship is marked as *check pending*. Consider the following examples:

- A restore operation is made where only data in the dependent file is restored. This data is no longer synchronized (a foreign key does not have a parent) with the parent file on the system.
- A system failure allows a parent key value to be deleted when a matching foreign key exists. This can occur only when the dependent and parent files are not journaled.
- A foreign key value does not have a corresponding parent key value. This can happen when you add a referential constraint to existing files that have never before been part of a constraint relationship.

Check pending applies only to constraints in the established state. To remove the check pending status from a constraint relationship, you must disable the relationship, correct the key (foreign, parent, or both) data, and then enable the constraint again. The database then verifies the constraint relationship again.

When a relationship is in check pending status, the parent and dependent files are in a situation that restricts their use. The parent file input/output (I/O) restrictions are different than the dependent file restrictions. Check pending restrictions do not apply to constraints that are in the established and disabled state, which are always in check pending status.

To manage check pending constraints using iSeries Navigator, follow these steps:

1. Select a database, right-click, and choose **Manage Check Pending Constraints** (Figure 4-11).

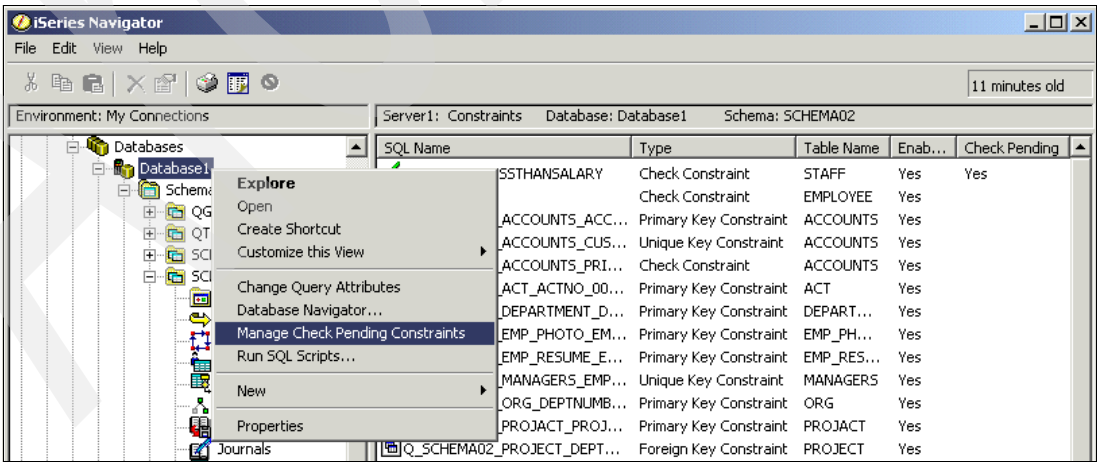


Figure 4-11 Manage Check Pending Constraints option

2. From the Check Pending Constraints window (Figure 4-12), select the constraint, right-click, and then select **Disable** or **Enable**. Or choose **Edit Check Pending Constraint** to make any changes to the data.

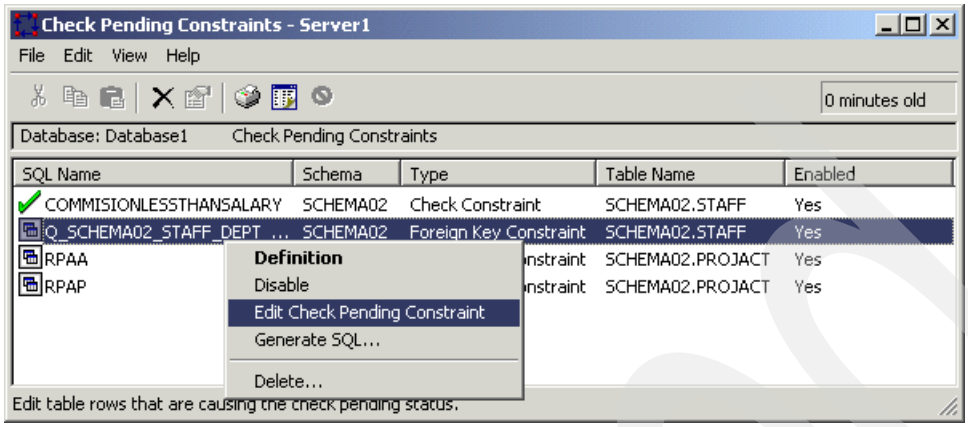


Figure 4-12 The Check Pending Constraint window

3. In the Edit Check Pending Constraint window (Figure 4-13), double-click the cell where you want to make changes. Then click the **Commit** or **Rollback** buttons to make the appropriate changes.

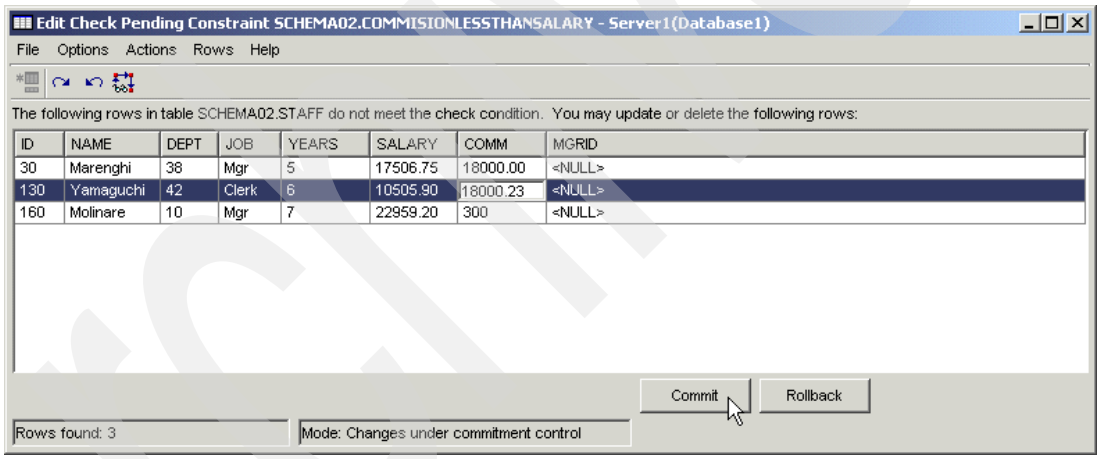


Figure 4-13 Edit Check Pending Constraint window

5

Database Navigator maps

This chapter introduces you to the DB2 Universal Database (UDB) for iSeries Database Navigator maps feature and its capabilities. It covers the following topics:

- 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

5.1 Closeup on Database Navigator

The launch of DB2 Universal Database for iSeries Database Navigator was introduced as part of iSeries Navigator in Version 5, Release 1, Modification 0 (V5R1M0) of iSeries Access. This product allows database administrators (DBAs) to map complex relationships between database objects.

The database component of iSeries Navigator at Version 5, Release 3, Modification 0 (V5R3M0) provides additional graphical interfaces for new functions that include:

- The ability to create and manage tables, aliases, views, indexes, constraints, journals, journal receivers, constraints, sequences, packages, procedures, and system (external) and Structured Query Language (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 DBA 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: A Database Navigator map is not intended to be a data modeling tool as are some existing products in the industry.

To see the benefits of Database Navigator, you must find the relationship between database objects on an iSeries server that is not at iSeries Access V5R1M0. You must use several commands to achieve the same results that you achieve with Database Navigator.

Some of the commands that are needed are:

- `DSPDBR FILE(SCHEMA01/ACT) OUTPUT(*PRINT)`

  This command shows the indexes, views, and constraints that are related to the selected table.
- `DSPFD FILE(SCHEMA01/ACT) TYPE(*CST) OUTPUT(*PRINT)`

  This command shows the details of the constraints that are built over the selected table.
- `DSPFD FILE(SCHEMA01/ACT) TYPE(*ACCPTH) OUTPUT(*PRINT)`

  This command shows the access path that is built over the selected table.

You also need to use the Work with Journal Attributes (WRKJRNA) command to determine which files are journaled to other journals. Although the Display File Description (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 much effort. It is also 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 DBA actually checked the authenticity of the map that was created previously.

The entire process for creating a physical or mental picture of which table is related to which index is difficult to administer. The practical difficulties of keeping this picture up-to-date require time and effort on the part of the DBA. The same is true in regard to the ability to explain the entire picture when training new staff. The process is simplified with the Database Navigator map feature in iSeries Access.

5.2 Database Navigator map interface

Database Navigator enables you to visually depict the relationships of database objects on your iSeries server. The visual depiction that you create for your database is called a *Database Navigator map*. The Database Navigator map is a snapshot of your database and the relationships that exist between all of the objects in the map.

To view the list of Database Navigator maps that are available in the selected database, click the Database Navigator map container. The maps appear in the right side of the iSeries Navigator window as shown in Figure 5-1.

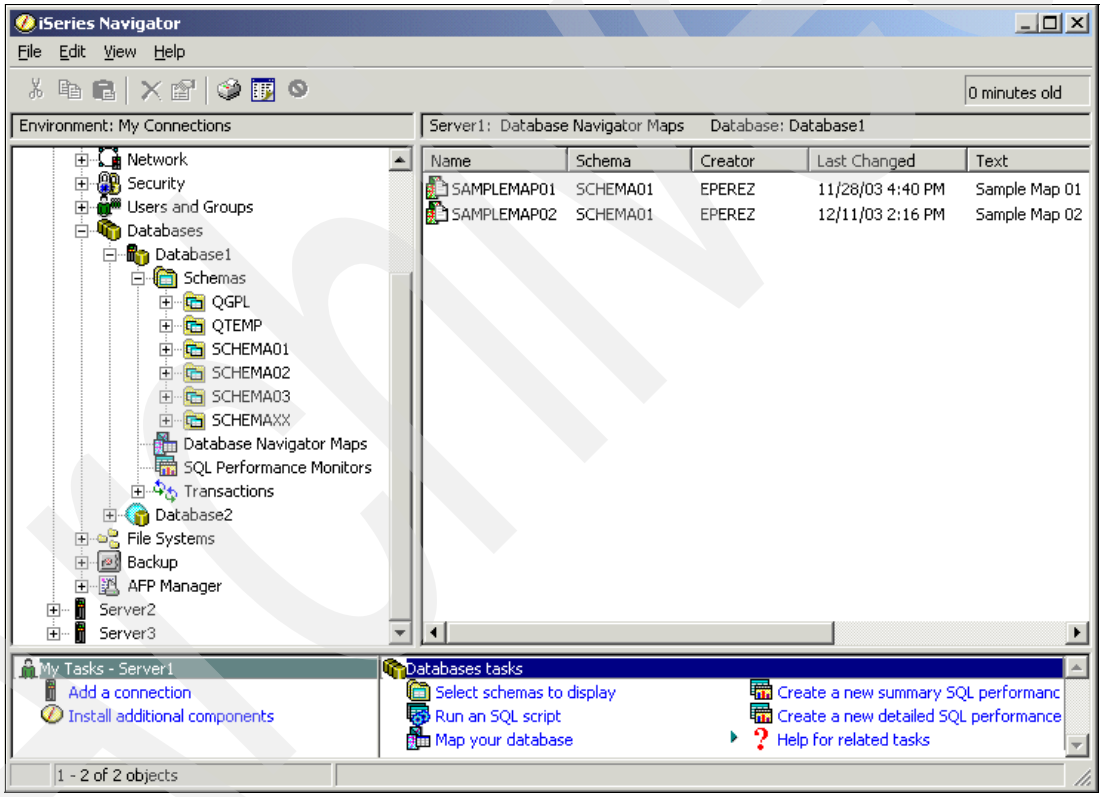


Figure 5-1 Database Navigator maps

Double-click the database map that you want to view. The Database Navigator map window with the selected map appears as shown in Figure 5-2. This map shows 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 a current picture of the database, refresh the map. Click the **View** menu and then select the **Refresh** option.

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 by someone else. Because they are stored on the iSeries server, you must have a connection to the system to open a map.

You can save multiple maps of the same database that reflect the database design at certain points in time. For this, you must use different names. To compare how the database design has changed, simply open and print the appropriate maps and then compare the printouts.

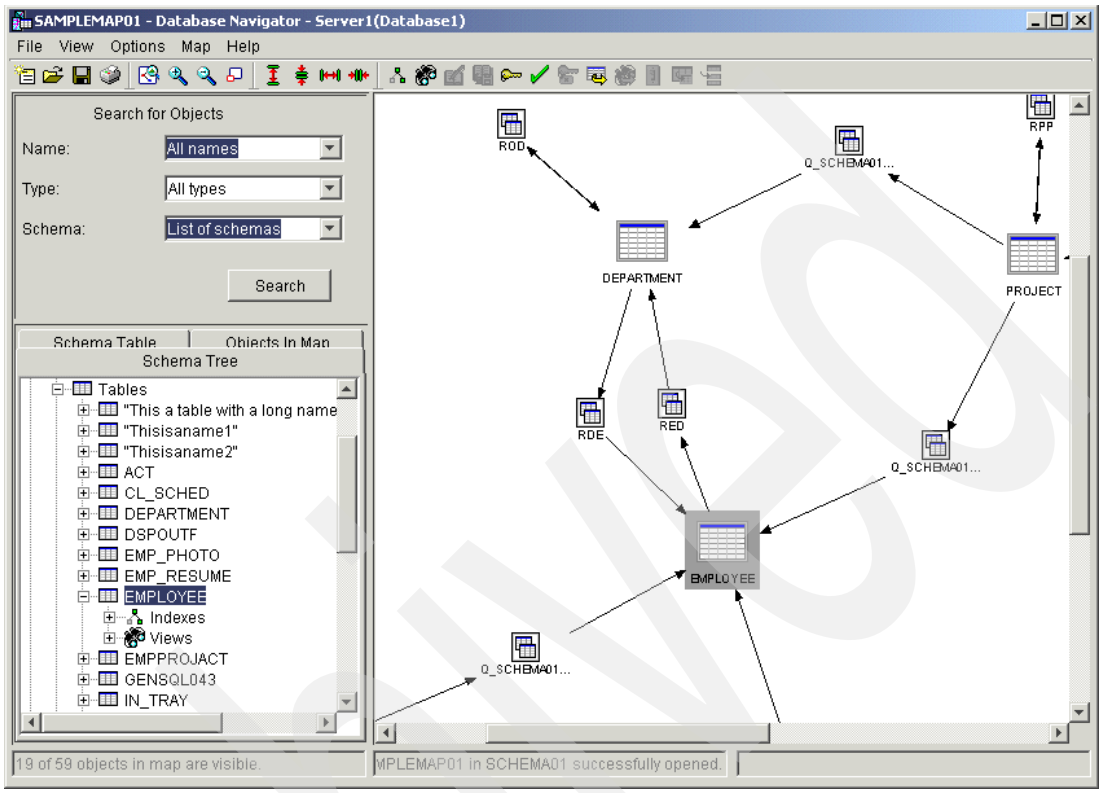


Figure 5-2 Database Navigator map

To create a map, click **Map your database** in the task pad (Figure 5-1). The window shown in Figure 5-3 opens.

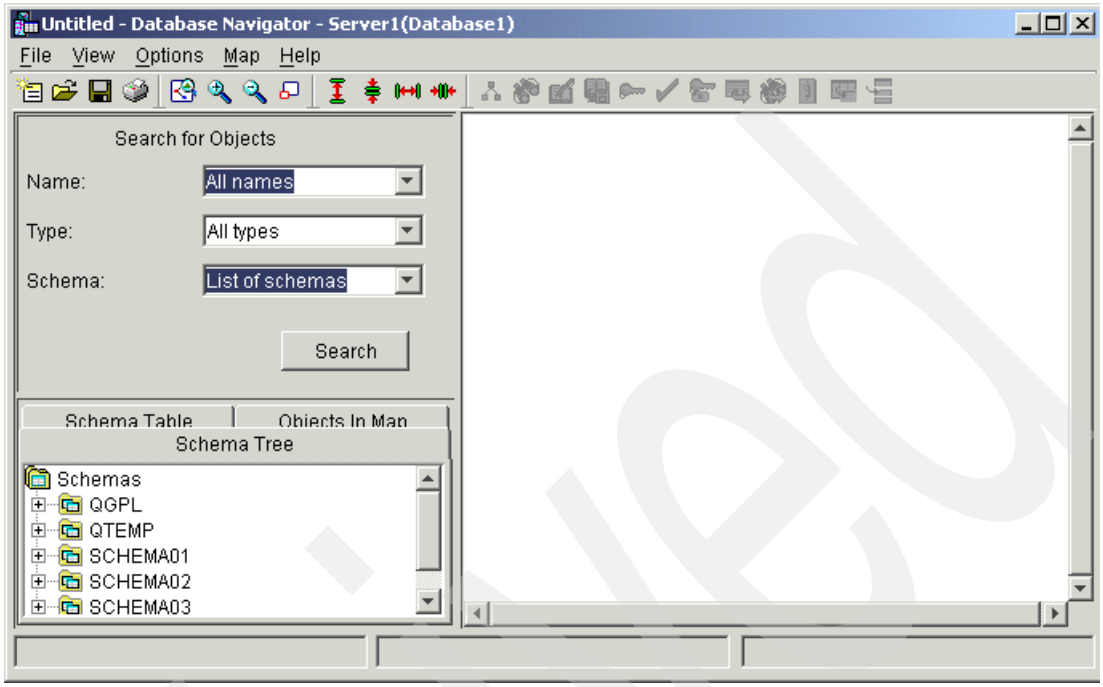


Figure 5-3 Default map display

The primary workspace for Database Navigator maps is a window that is divided into several main areas as shown in Figure 5-4. 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, helps you 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 you can use to specify the name, type, and schema 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 Schema Tree and Schema 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 by 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 into 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:

    - In the search criteria, you can specify single objects or search for generic names using the asterisk (*) (for example, EMPLO*).
    - You can specify all object types or indexes, tables, and views.
    - You can specify one schema from your schema list or all schemas to search on.

– *The lower locator*

This window has three parts:

- The *Schema tree* can either show individual schemas, schemas in your list, or all schemas on the system.
- The *Schema table* shows the tables, indexes, or views of the schemas in the schema tree.
- *Objects In Map* shows all of the objects in the map, regardless of whether they are hidden. Within this display, you can select or deselect objects to place in the map.

Note: Any changes that you make using the search criteria require that you click the Search button to change the Schema Tree or the Schema Table displays.

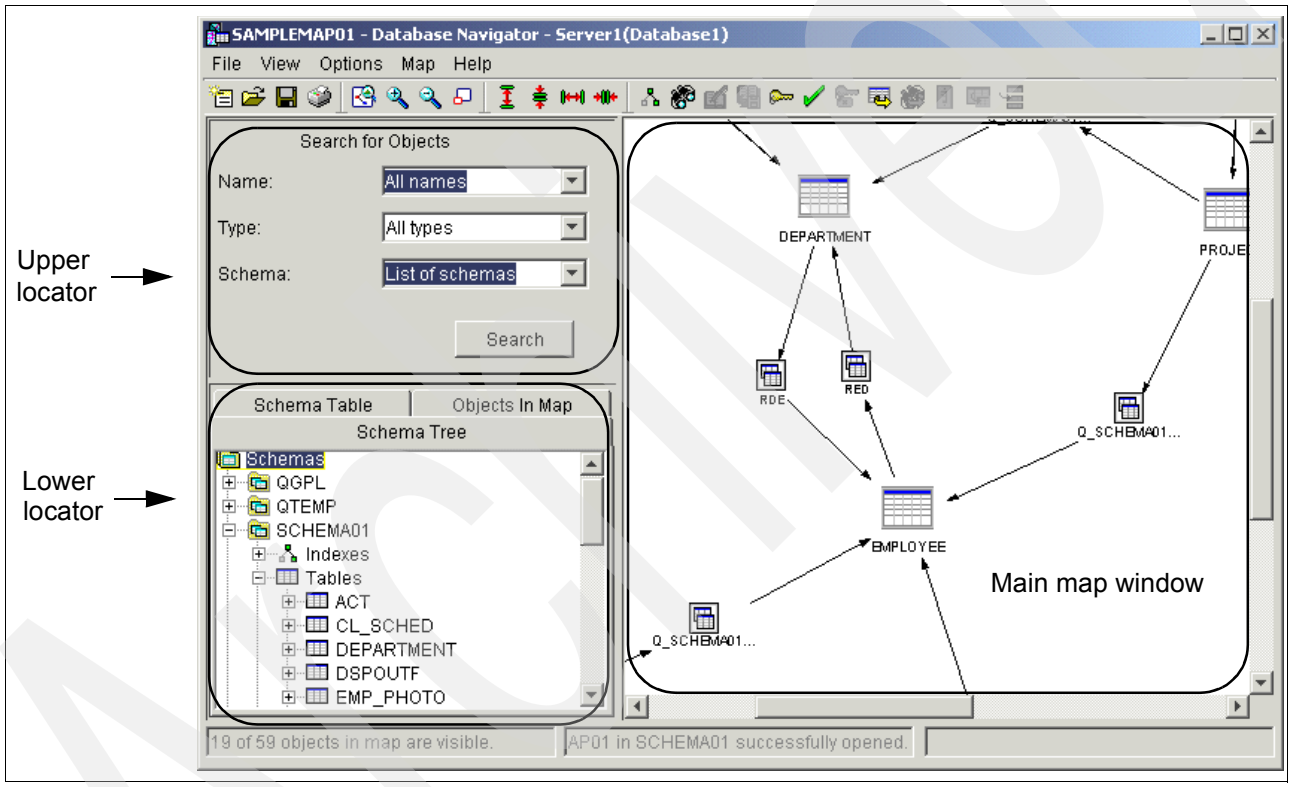


Figure 5-4 Three main windows in the Navigator map display

- Map pane

The map pane is located on the right side of the Database Navigator window. It 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 5-5):

  – *Object status bar* 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* provides a clear description of the actions taken that affect the map and any modifications that are pending.

  – *Modification status bar* indicates whether a modification has been made or is pending.

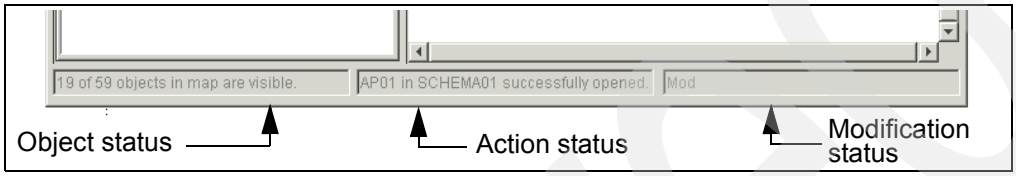


Figure 5-5 The status bar

The Database Navigator map display also supports the following menu options:

- File menu

  You can select from several options, including:

  – *New* allows you to create a new map.

  – *Open* allows you to open a previously saved map.

  – *Close* 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 was previously saved.

  – *Print Preview* allows you to see a print preview of the map.

  – *Print* allows you to print to a previously defined printer.

  – *Exit* closes the map of your database window.

> **Note:** If you make changes 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* updates the database map with any changes that are made.

  – *Object Spacing* allows you to increase or decrease the vertical and horizontal spacings of the objects in the map.

  – *Show Overview Window* open a window (Figure 5-6) that allows you to see an overview of the map that is currently open. This overview allows you to position the main screen to any part of the map. This is particularly useful on large or complicated maps.

  – *Show objects of type* allows you to add objects to the map, such as aliases, journals, etc.

  – *Arrange* allows you to change the map back to the original settings.

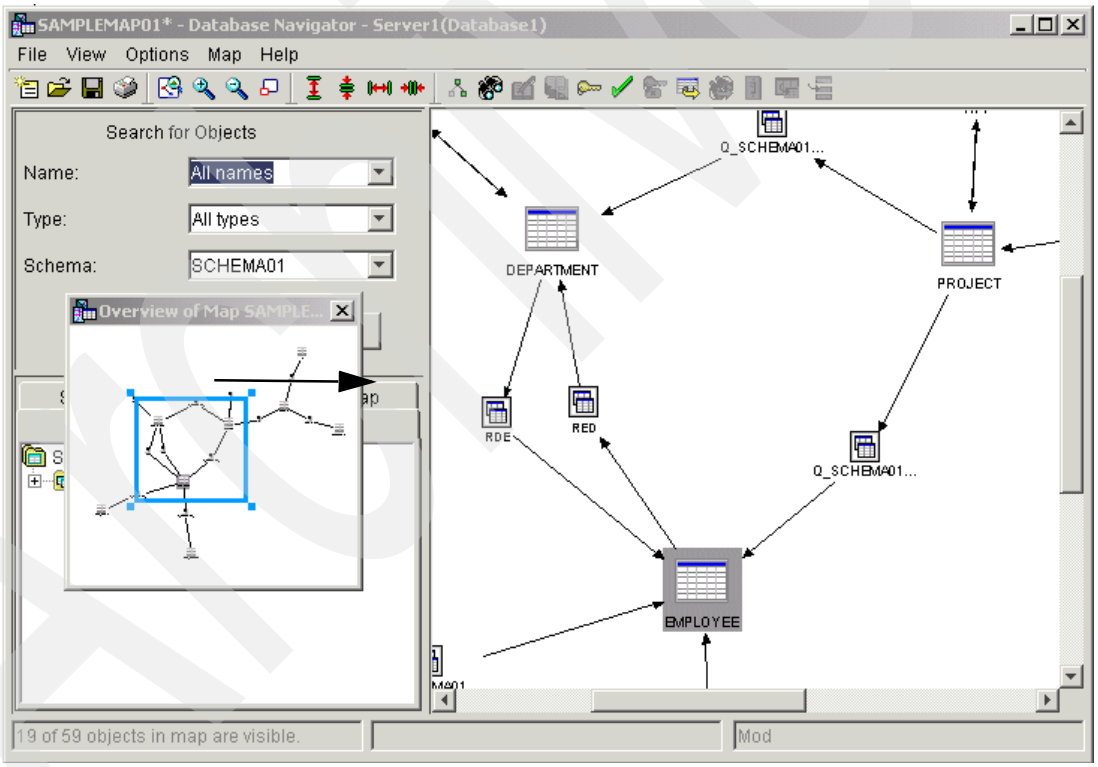


Figure 5-6 Overview window

- Options menu

  The following option are available:

  – *User Preferences* allows you to change the objects that appear on the map as it is created (Figure 5-7). This group box enables you to select the objects that you want to search for and include in a map. By default, all objects in the list, except journals and journal receivers, are selected and included in the map. When you click OK, an extensive search for the objects and related objects is made on your system.

    You can also choose to include flyover help for objects. The default is to include the flyover help.

    Finally, you can see the Generate SQL window when generating SQL for objects. The Generate SQL window enables you to change your options when generating SQL. The default is to bypass the Generate SQL window and to display the generated SQL in the Run SQL Scripts SQL window.

    These preferences are saved for each user and not for each map.

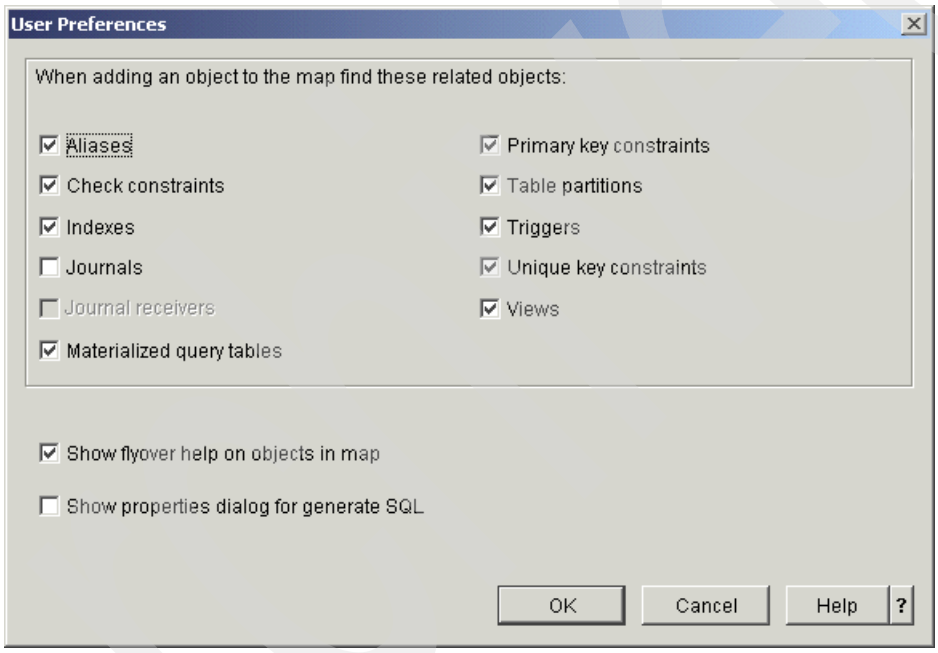


Figure 5-7 User Preferences window

– *Change List of Schemas* allows you to change the schemas that are displayed (see Figure 5-8). If you type a new schema in the Enter schema names box, instead of selecting from a list, the system ensures that the schema exists before it allows the object to be added to your list.

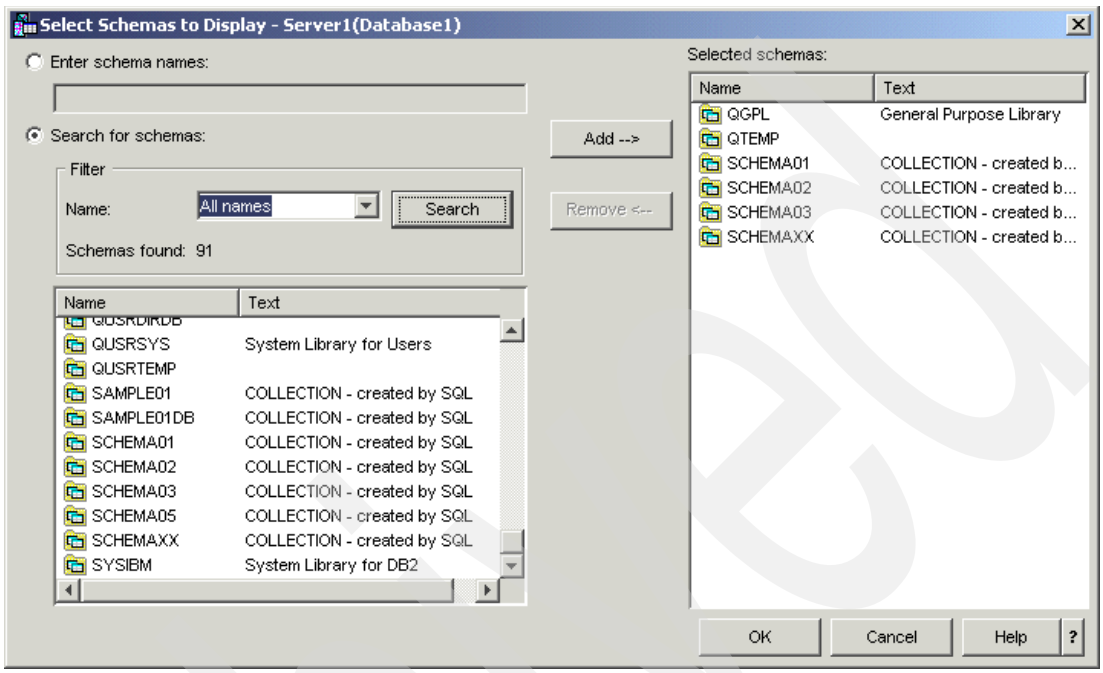


Figure 5-8 Change list of schemas display

- 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. It prompts the Run SQL Scripts window (Figure 5-9) to appear. This window enables you to see the SQL statement that was used to create the object. It also helps you to take individual tables, whole databases, or entire maps of objects to other iSeries servers or SQL platforms.

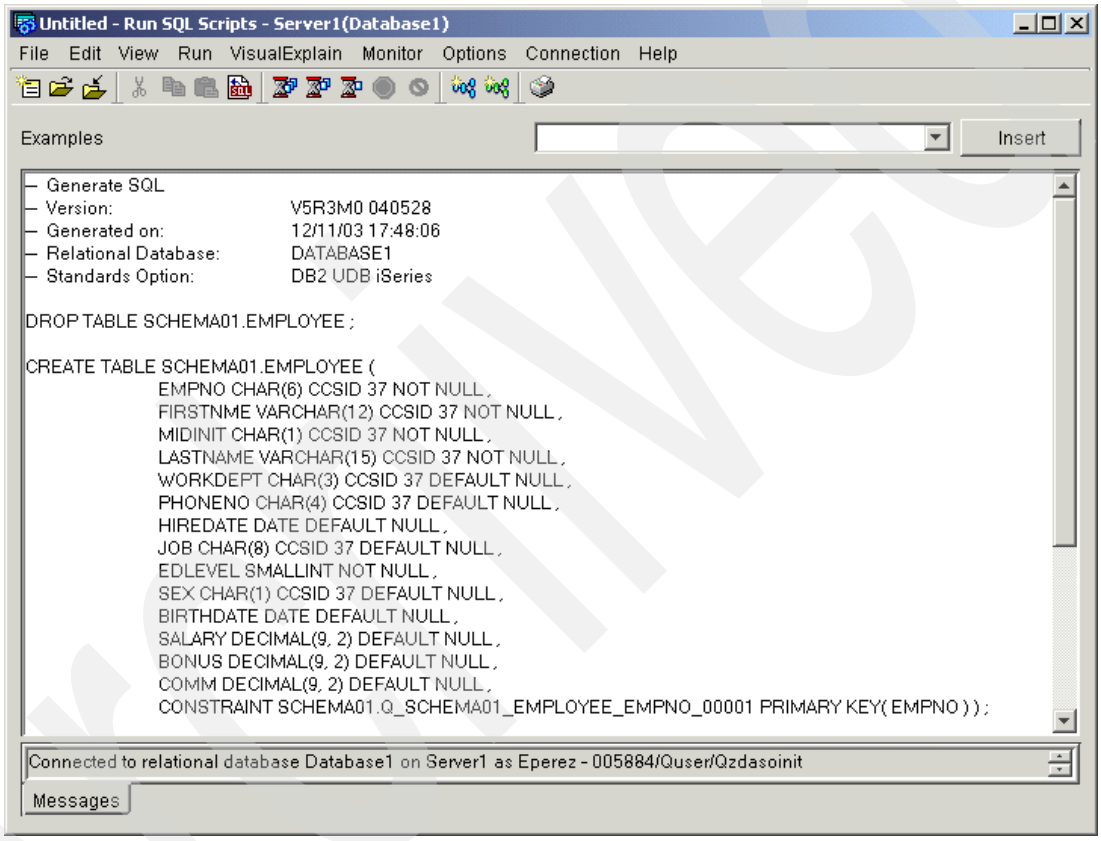


Figure 5-9 Generate SQL window

  - The *Create* option includes the following sub-options to create:
    - Aliases
    - Indexes
    - Journals
    - Materialized query tables
    - Tables
    - Views
- User-defined relationship

  This function enables you to create a user-defined relationship. It helps the DBA 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. It can also help to illustrate relationships that are not physical in the map for debug or education purposes.

A toolbar 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
- Show or hide triggers
- Show or hide materialized query tables
- Show or hide table partitions

Note: If the objects are not available to hide or view, the button is unavailable (grayed out).

5.2.1 Database Navigator map display

The Database Navigator map main display is another interface to manage and change your database using iSeries 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 that are available to you from this display.

Right-click the main window to view the following menus:

- **Create**: This function helps you to create aliases, indexes, journals, tables, views, and user-defined relationships.
- **Zoom**: This function helps you to zoom in or out and fit the map to the window.
- **Generate SQL**: This function helps you to generate the SQL for all objects or only the visible options.
- **Arrange**: This function returns the objects within the map to their position at creation, even if the map was saved previously.
- **Properties**: This function shows you a properties display of the map itself.

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 5-10.

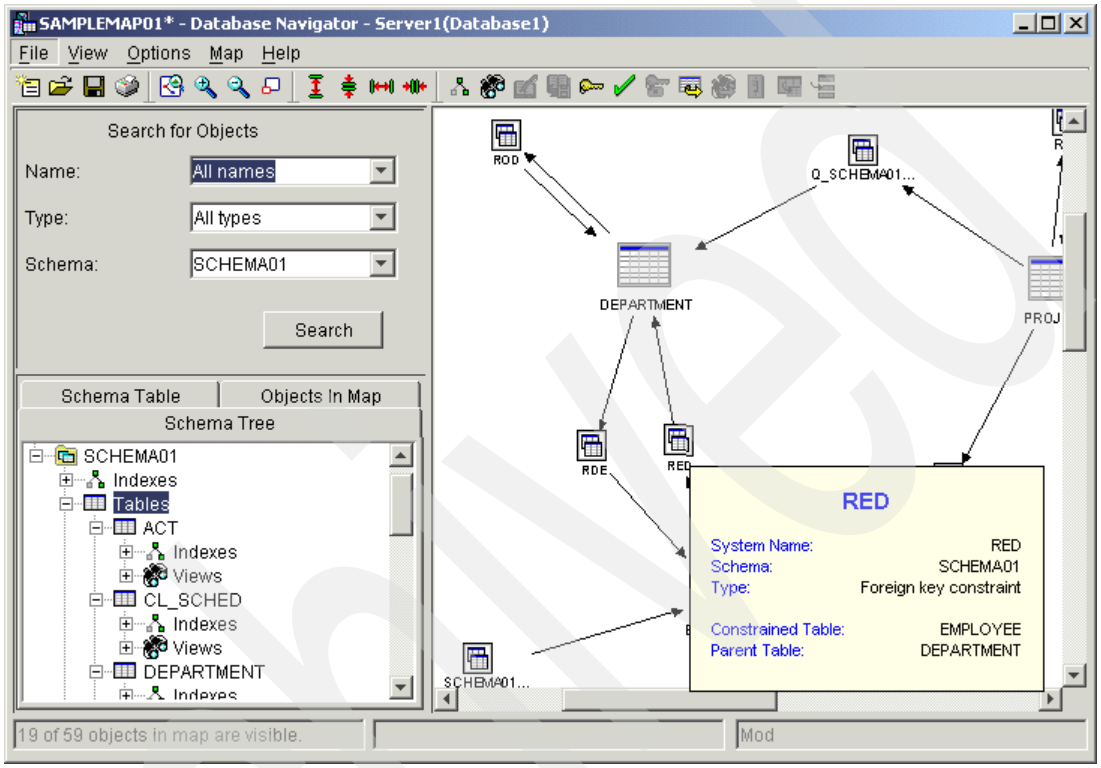


Figure 5-10 Example flyover display

After you refresh the display, a window like the example in Figure 5-11 appears while the refresh runs. After the map is built or refreshed, you can manipulate the objects any way you choose. From within the map display, you can actually move the icons around to suit your requirements.

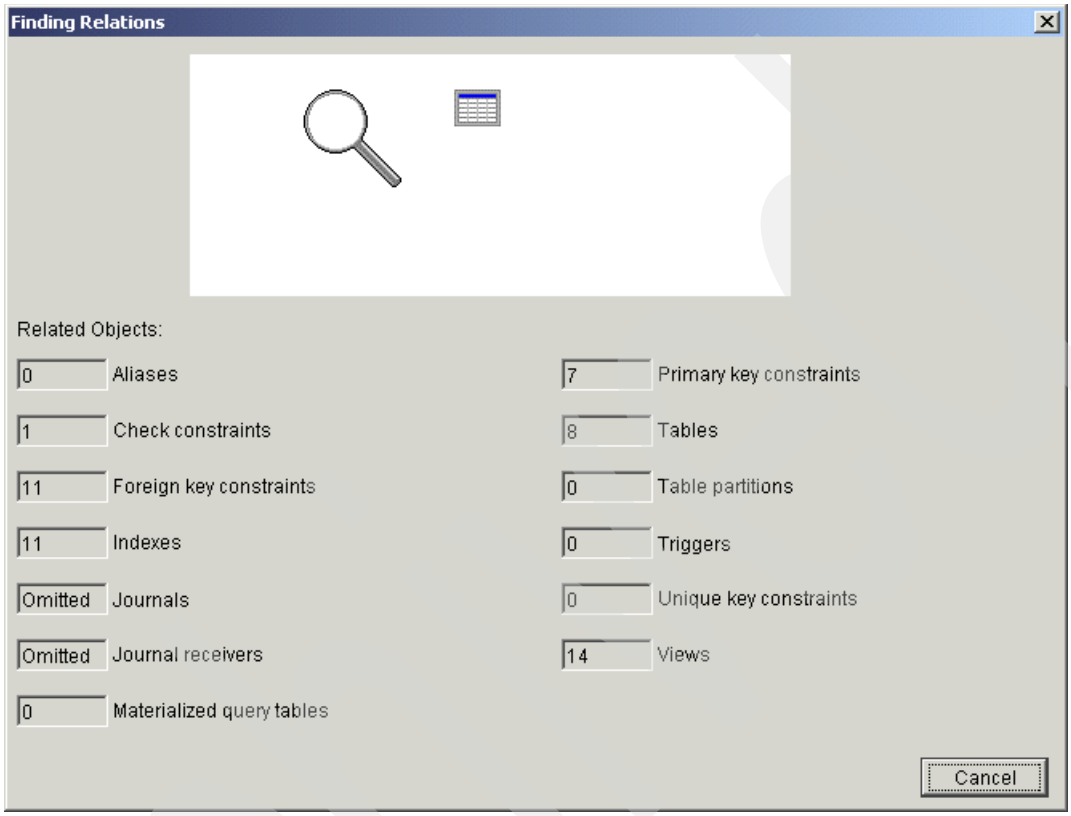


Figure 5-11 Refresh on database in progress

5.3 Available options on each active icon on a map

This section discusses the options that are available from within the map display. To access these options, you simply right-click each of the different objects in the map.

5.3.1 Table options

Figure 5-12 shows the various options that available when you use the active icon for a table within a Database Navigator map. All objects on a map are active. They enable you to manipulate the object without leaving the map. To access this window, you right-click a table within the map.

The options that are available from this window are:

- **Edit Contents**: This allows you to open the file for an update.
- **View Contents**: This shows you the file and its contents (read only).
- **Definition**: This option allows you to change the definition of the table.
- **Generate SQL**: This option creates the SQL script used to generate the object.
- **Journaling**: This specifies whether journaling is turned on.

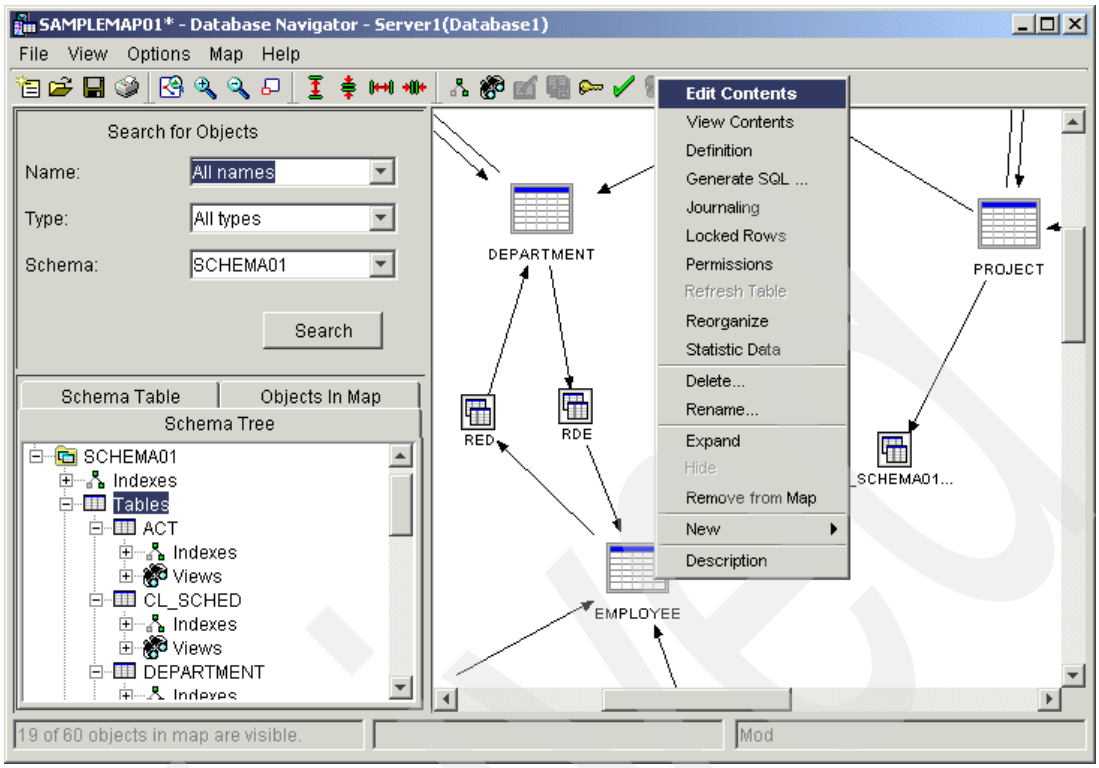


Figure 5-12 Table options in a Database Navigator map

- **Locked Rows**: This shows any rows that are locked on the table.
- **Permissions**: This allows you to set security for a table.
- **Reorganize**: This allows you to reorganize the file by compressing deleted records (by table key or by a selected index).
- **Statistic Data**: This allows you to view statistical data for the columns in the selected table.
- **Delete**: This allows you to delete the object.
- **Rename:** This allows you to rename the object.
- **Expand**: This shows additional details about the table, such as the columns and indexes built over the table.
- **Collapse**: This returns the display to the default setting for the table.
- **Remove from a map**: This option removes the object from the map.
- **New**: With this option, you can create aliases, indexes, and external or SQL triggers.
- **Description**: From within this option, there are six tabs:
  - *General* shows the size of the object, the current number of rows, the number of deleted rows, and whether the table reuses deleted records.
  - *Allocation* 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* shows the current size of the access path, the maximum size, the maximum key length, whether the access path is valid or shared and whether it is journaled, what the maintenance and recovery of the access path is set to, etc.

- *Usage* shows the creation date of the table, the last used date, the last changed date, and other details of the table.
- *Activity* shows the latest activity on the table since the last machine restart.
- *Details* shows the creation date of the table, the maximum row length, and more.

- **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.

5.4 Index options

Right-click an index to access the options shown in Figure 5-13. The options that are available are:

- **Definition**: This option shows you the definition of the index.
- **Generate SQL**: This option creates the SQL script used to generate the object.
- **Permissions**: This allows you to set security for the object.
- **Description**: In this option, there are three views:
  - *Access path*: This shows you general information about access path.
  - *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.

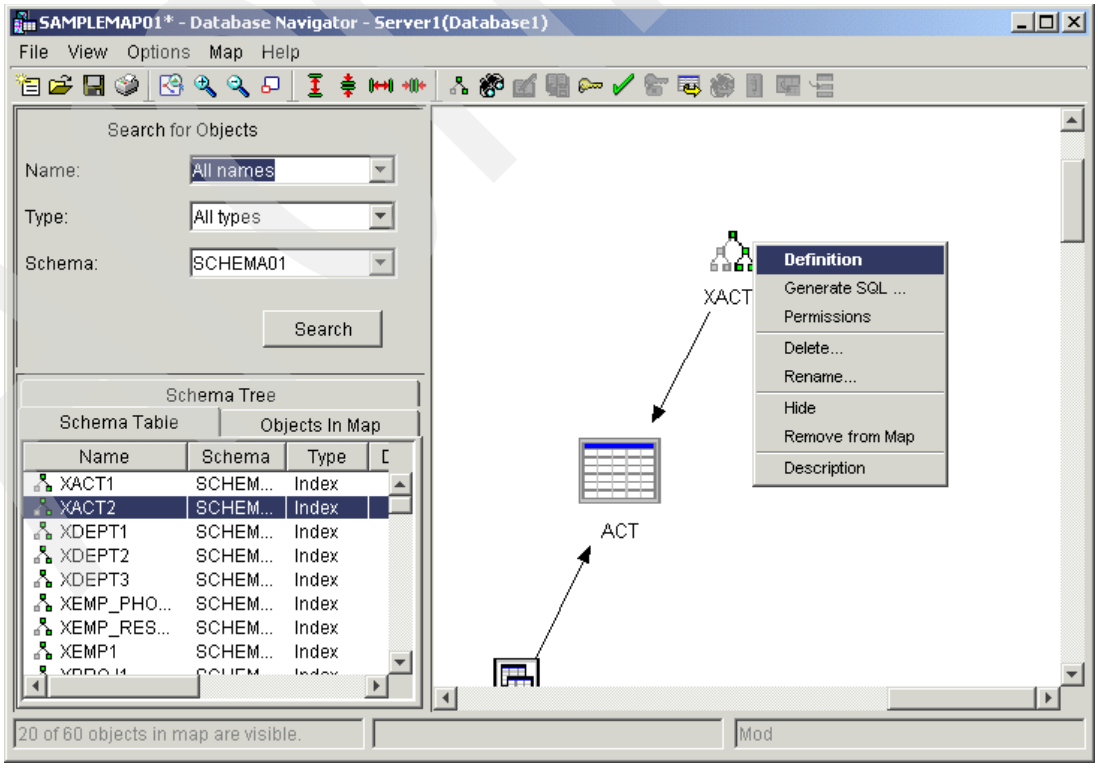


Figure 5-13 Index options in a Database Navigator map

5.4.1 Constraint options

If you right-click any of the key constraints on the map, you see the following options (see Figure 5-14):

- **Definition**: This option allows you to see the definition of the constraint.
- **Generate SQL**: This generate an SQL Script in the Run SQL Scripts window.
- **Remove from Map**: This removes a particular constraint from the map. If the object is not included in the map, the Add to Map function appears highlighted.

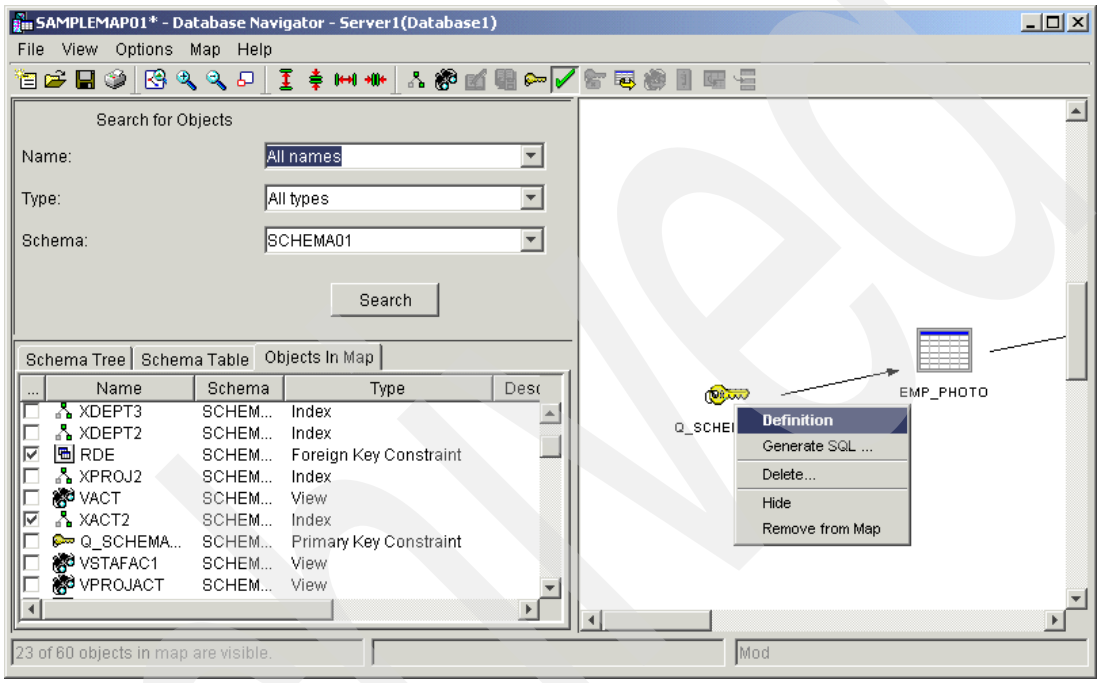


Figure 5-14 Key constraint options in a Database Navigator map

If you right-click any of the check constraints on the map, you see two additional options (see Figure 5-15):

- **Edit Check Pending Constraints**: This option is only enabled when there are check pending constraints.
- **Disable**: This option allows you to disable the check constraint. After you disable the constraint, the option change to enable.

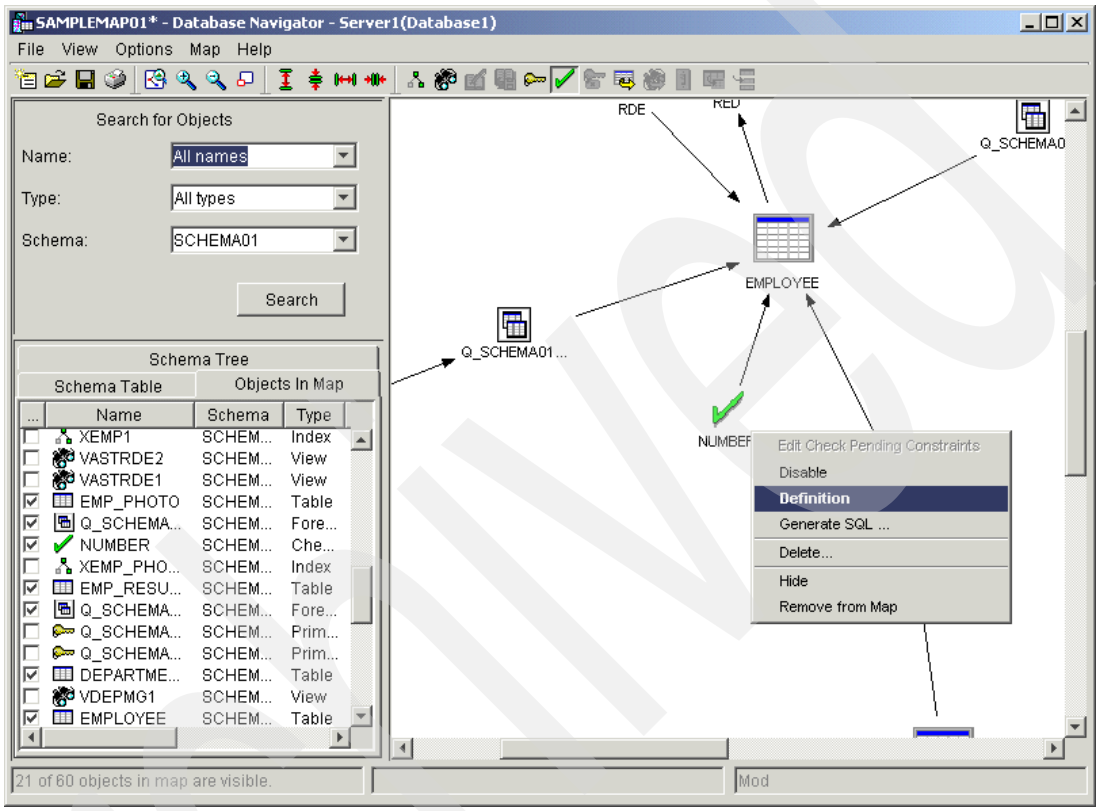


Figure 5-15 Check constraint options in a Database Navigator map

5.4.2 View options

Right-click any view on the map to see the following options:

- **View Contents**: This option shows you the view contents (read only).
- **Definition**: This option shows you the definition 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.
- **Generate SQL**: This option creates an SQL script window that allows you to recreate the table or multiple objects depending on the option selected from the generate SQL window.
- **Permissions**: This option allows you to set security for the view.
- **Delete**: This option allows you to delete the view.
- **Rename**: This option allows you to rename the view.
- **Hide**: This option allows you to hide the view from the map only.
- **Remove from Map**: This option removes a particular view from the map. If the object is not included in the map, the Add to Map function appears highlighted.
- **New**: This option allows you to create an alias for the view.

- **Description**:
  - *Usage* shows you the creation date of the view, the last used date, the last changed date, and other view details. You can also check the option to share open data path.
  - *Details* shows the creation date of the view, the maximum row length, and additional details. On this tab, you can also change such parameters as Description, Maximum wait time, and Maximum wait row time.

5.4.3 Journal options

If you right-click a journal, you see the options shown in Figure 5-16. The various options include:

- **Permissions**: This option allows you to set security for the journal.
- **Start or End Table Journaling**: This option allows you to end or start journaling on any table on the system to the selected journal.
- **Swap receivers**: This option allows you to perform the equivalent of a CHGJRN *GEN from a normal green-screen command.
- **Delete**: This option allows you to delete a particular journal.
- **Hide**: This option allows you to remove the journal from the map only.
- **Remove from map**: This option 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.
- **Properties**: This option shows you the properties of the journal.

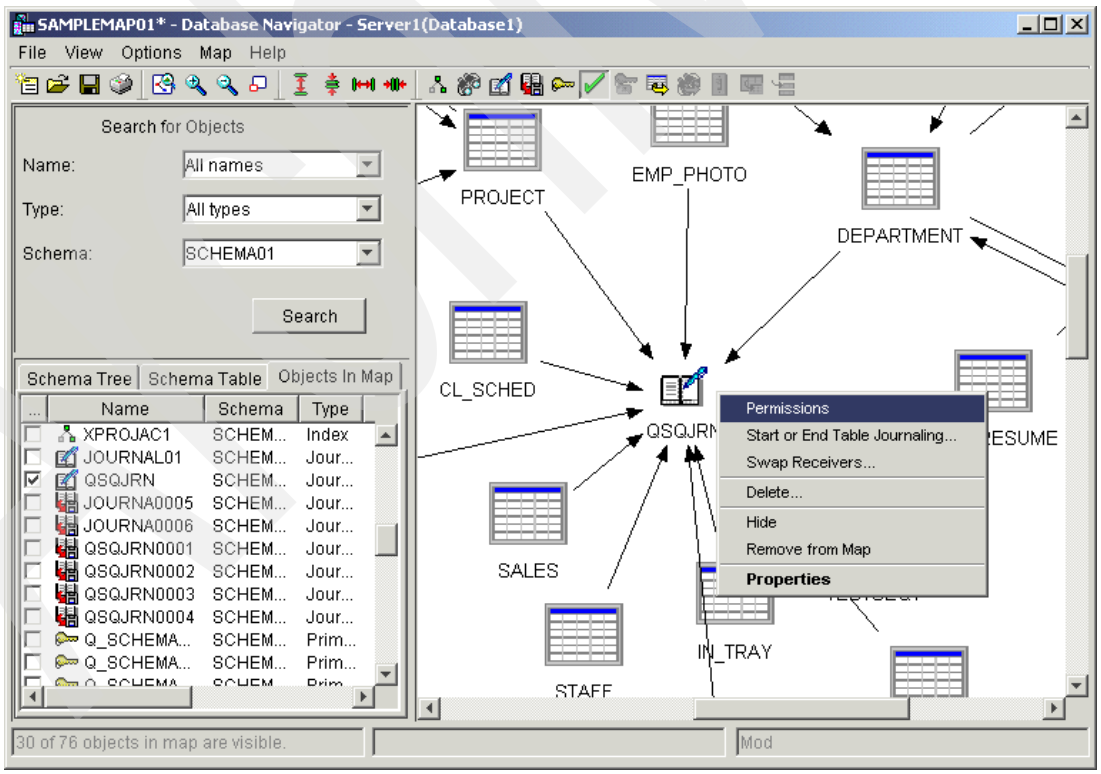


Figure 5-16 Journal options in a Database Navigator map

5.4.4 Journal receiver options

The various journal receiver options include:

- **Permissions**: This option allows you to set security for the journal receiver.
- **Delete**: This option allows you to delete a particular journal receiver.
- **Hide**: This option allows you to remove the journal receiver from the map only.
- **Remove from map**: This option 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.
- **Properties**: This option shows you the properties of the journal receiver.

5.5 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, follow these steps:

1. In the iSeries Navigator window (Figure 5-17), expand **Databases** and the database that you want. Select **Database Navigator Maps**, right-click, and select **New** →**Map**.

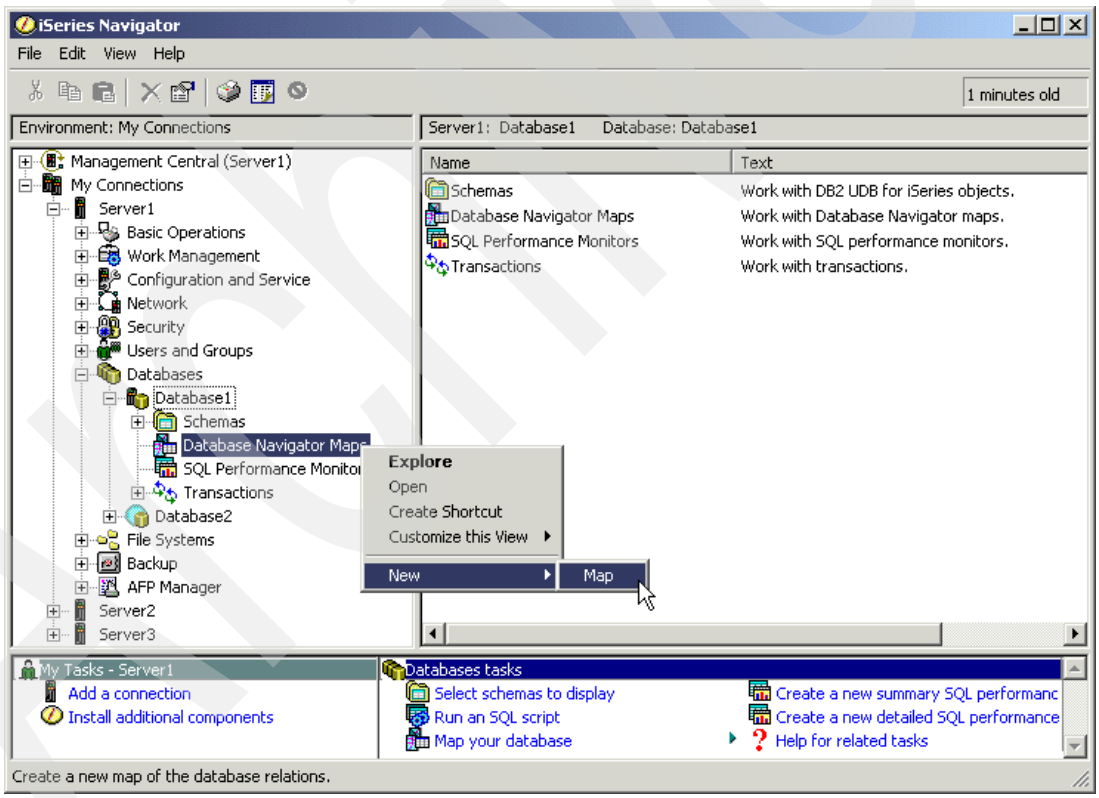


Figure 5-17 Selecting the Database Navigator map option

2. The iSeries Navigator schema list appears in the left side of the Database Navigator window. Double-click the schema that you want to expand the objects.
3. Double-click **Tables** in the locator pane to expand all the tables in a database.

4. Double-click any 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 5-18 for EMPLOYEE table.

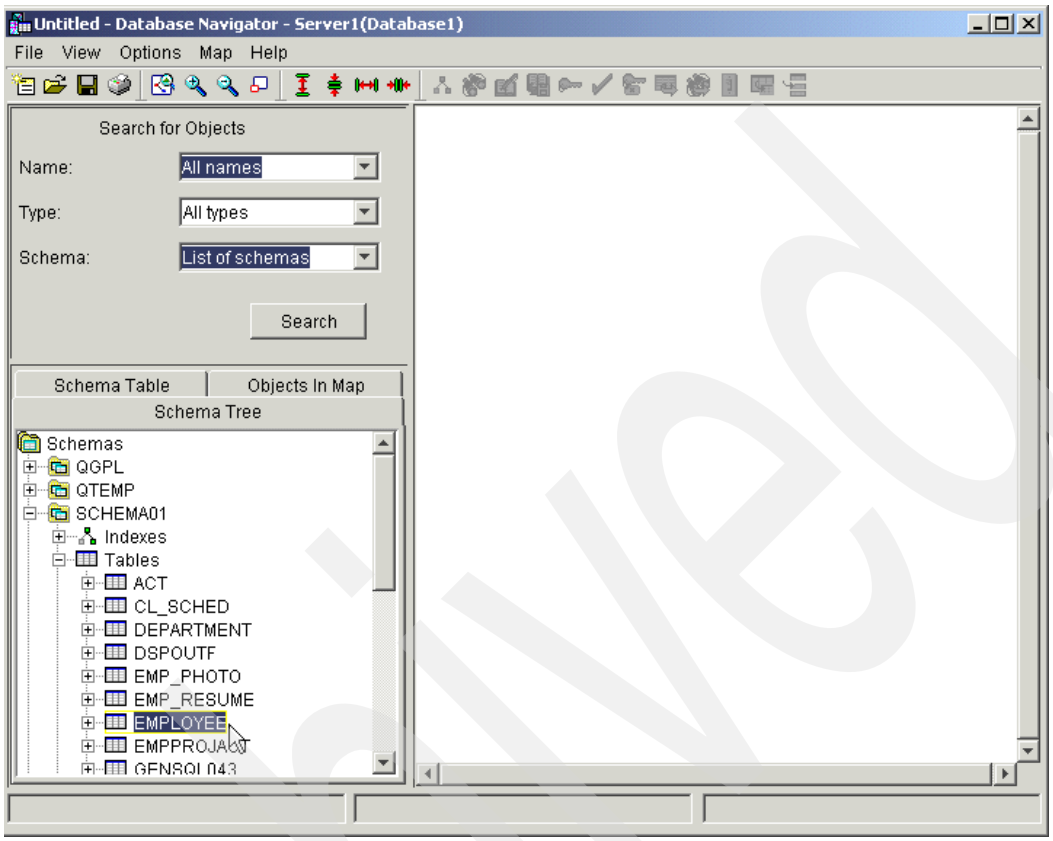


Figure 5-18 Selecting a database to build a map

5. 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 5-19.

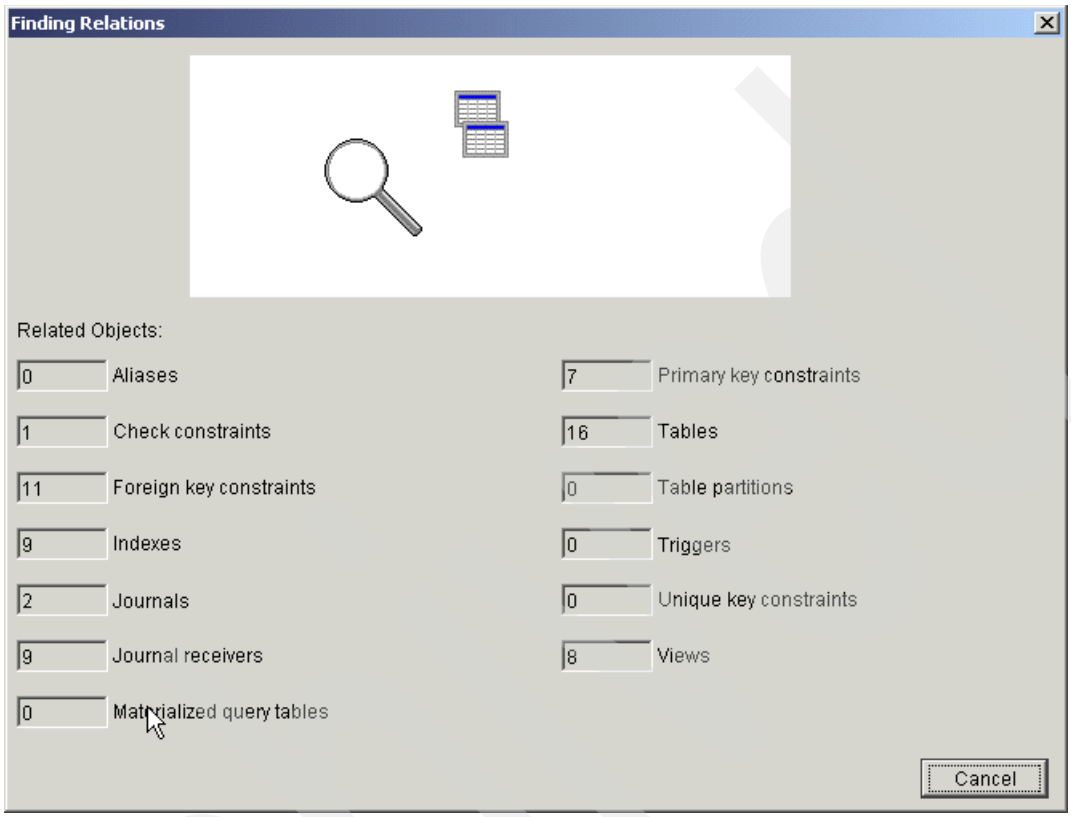


Figure 5-19 Building a Database Navigator map

6. Click the minus (-) sign next to the schema to collapse the tree view.

7. Use the vertical and horizontal scroll bars to navigate the map in the Database Navigator window as shown in Figure 5-20.

> **Note:** You can also click the Map your database task on the task pad at the bottom of the iSeries Navigator window (Figure 5-17 on page 130) to create a map.

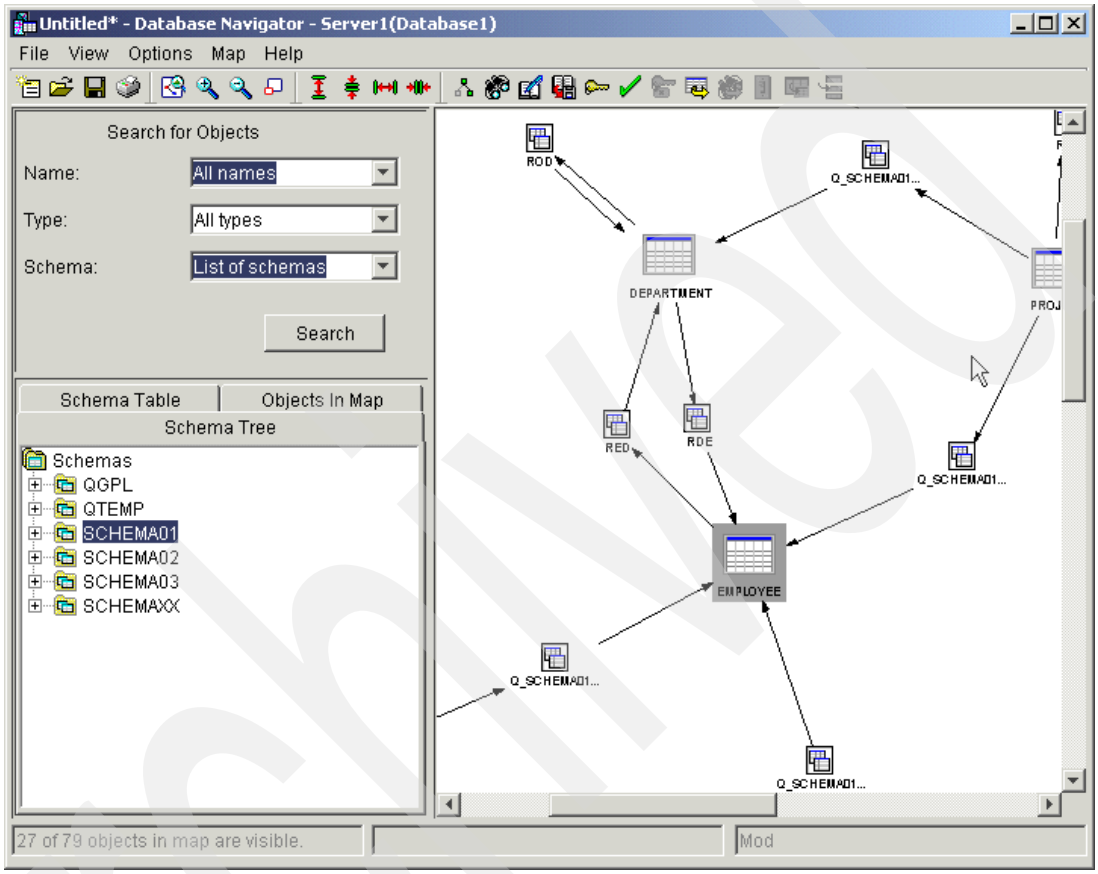


Figure 5-20 Database Navigator map

8. Save this map. Select **File** →**Exit**.
9. If changes are pending, in the Save Changes To window, select **Yes**. You can re-open this map at a later time.

After 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

5.5.1 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** →**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.
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.

5.5.2 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, 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. Select **Options** →**User Preferences**.
3. In the User Preferences window, in the When adding an object to the map find these related objects group box, select the objects that you want to include or clear the objects that you do not want to include. Click **OK**.
4. If you want to refresh the map with the new preferences, in the Information box, click **Yes**.

5.5.3 Changing object placement and arranging objects in a map

When you have a map, it is possible to arrange and move objects in the map.

1. Double-click a table from the list of tables to find it in the map.
2. Drag and drop the table to the left as shown in Figure 5-21.

> **Important:** When you use the arrange option, it removes any customized object position or relationship line that you created. The Arrange option places the map back in a *Default* state.

3. In a free space in the map pane of the Database Navigator window, right-click.

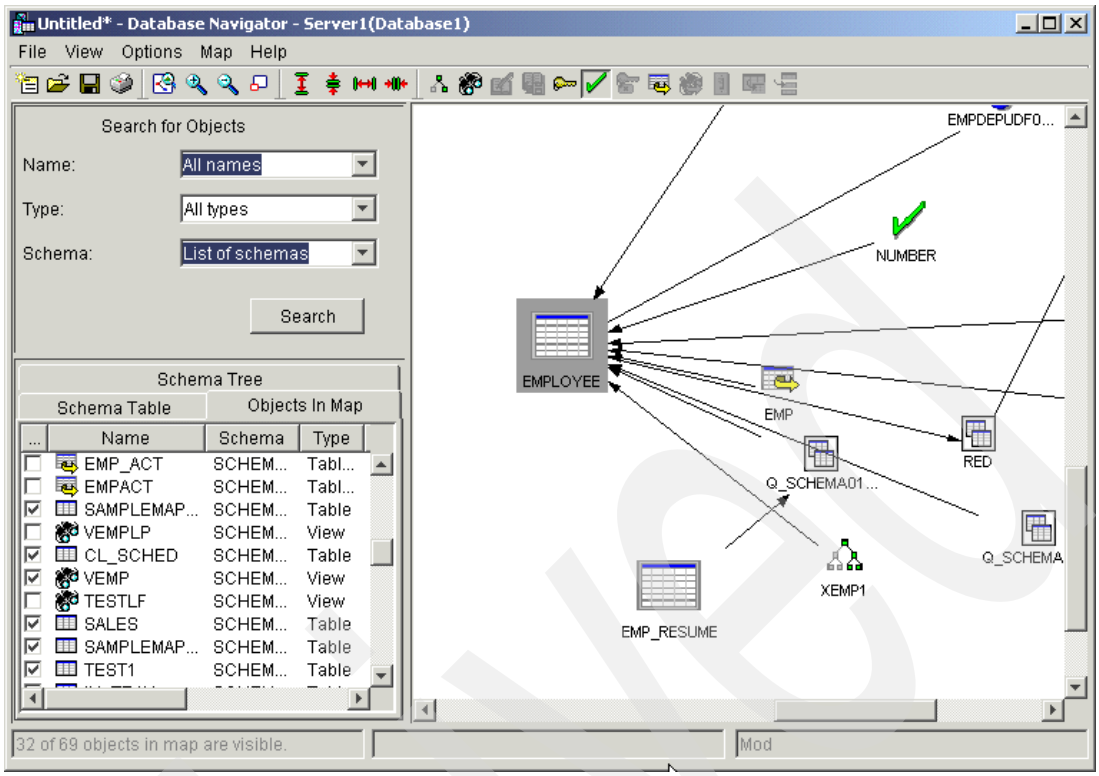


Figure 5-21 Changing object placement

4. The Arrange function appears as shown in Figure 5-22. Select **Arrange** and the arrange type to minimize the line crossing the map.

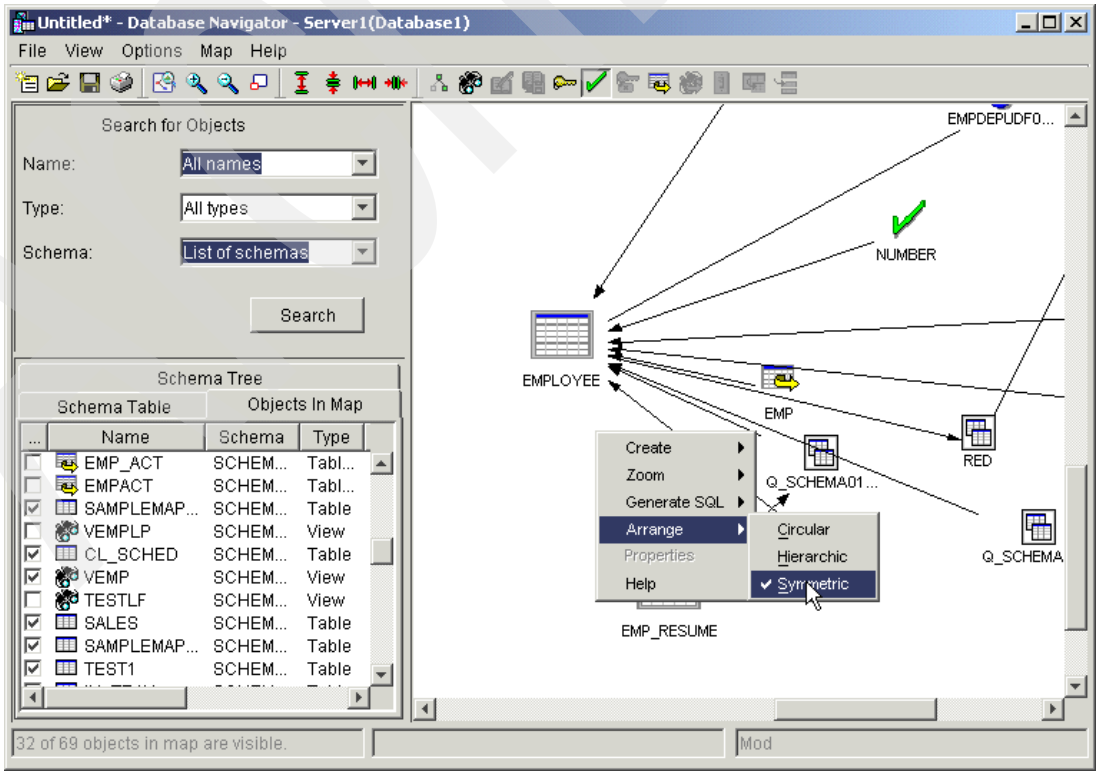


Figure 5-22 Arranging objects in the map

5.5.4 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. This displays your relationship in the map. For example, you may want to create a user-defined relationship to remind programmers about 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. In a free space on the map pane in the Database Navigator window, right-click and select **Create** →**User-Defined Relationship** to create the new object. See Figure 5-23.

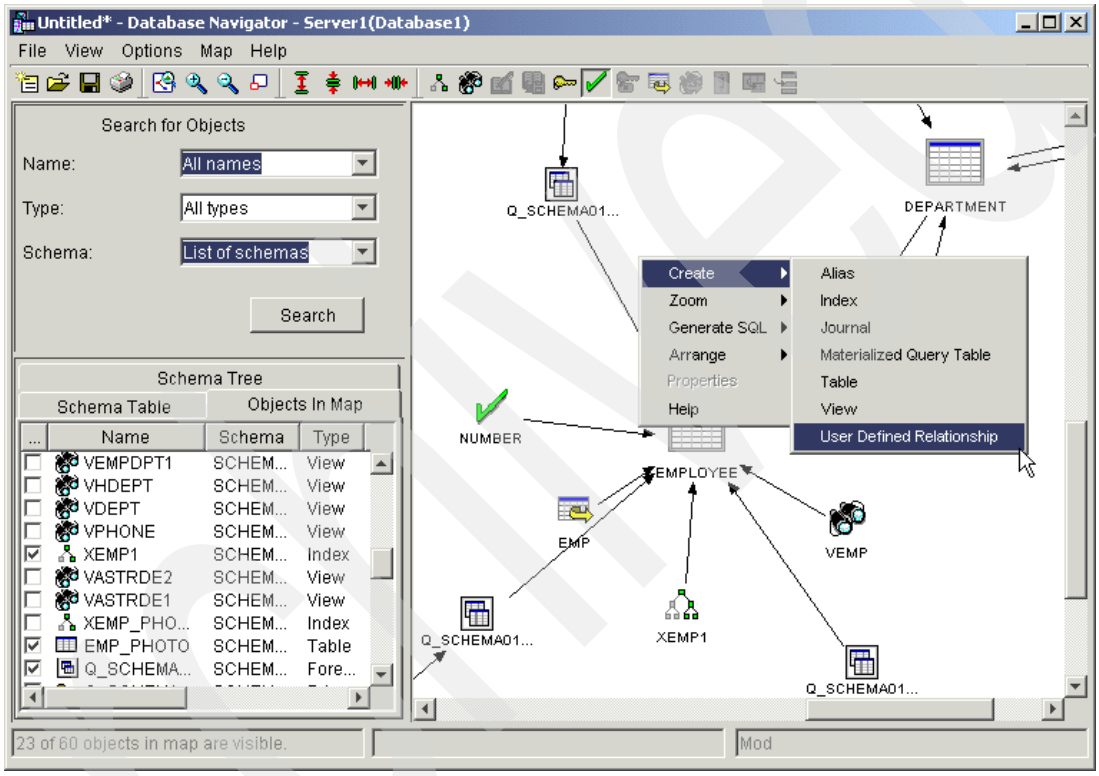


Figure 5-23 Selecting the function to create a user-defined relationship

3. In the Properties of User Defined Relationship window (Figure 5-24), complete these tasks:
   a. Specify a name and a description for the user-defined relationship.

      Unlike some iSeries Navigator functions where the description is optional, it is important to provide a *meaningful* description for your user-defined relationship. This is the only way for you to indicate what the user-defined relationship represents.
   b. Under Select objects in relationship, select the objects that you want to include.
   c. For Shape and Color, choose the preferred shape and color for the object.
   d. Click **OK** to create the user-defined relationship.

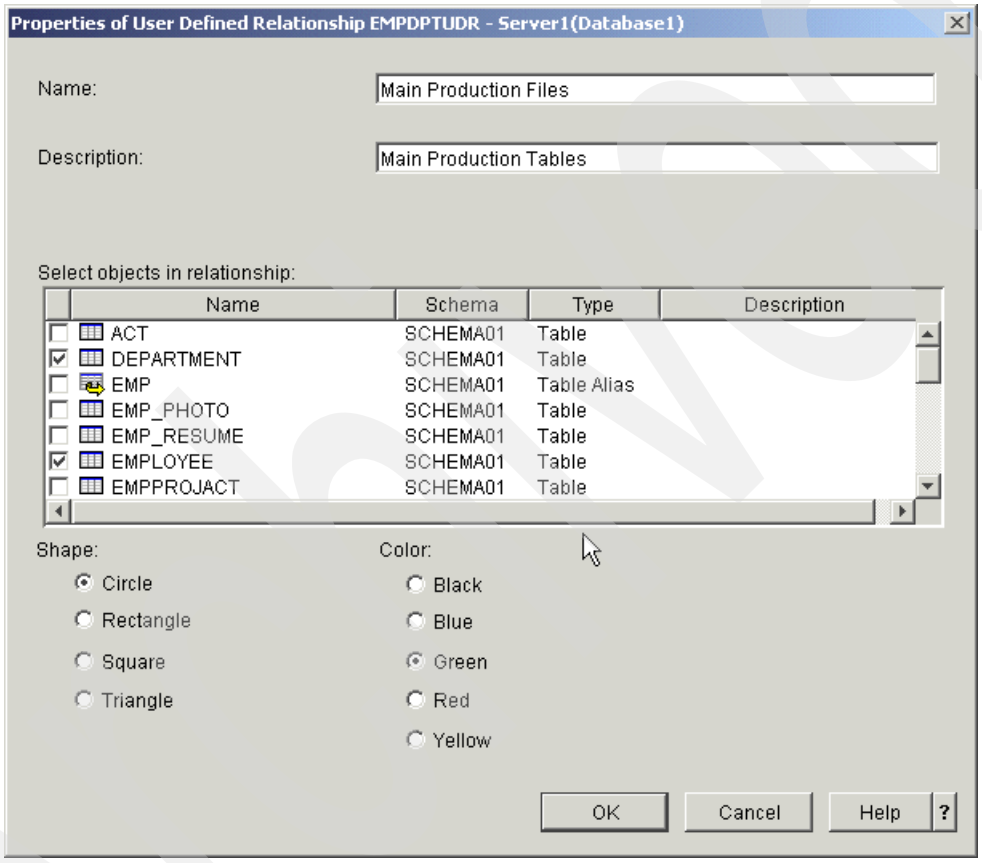


Figure 5-24 Creating a user-defined relationship

You should see a user-defined relationship on the map as shown in Figure 5-25.

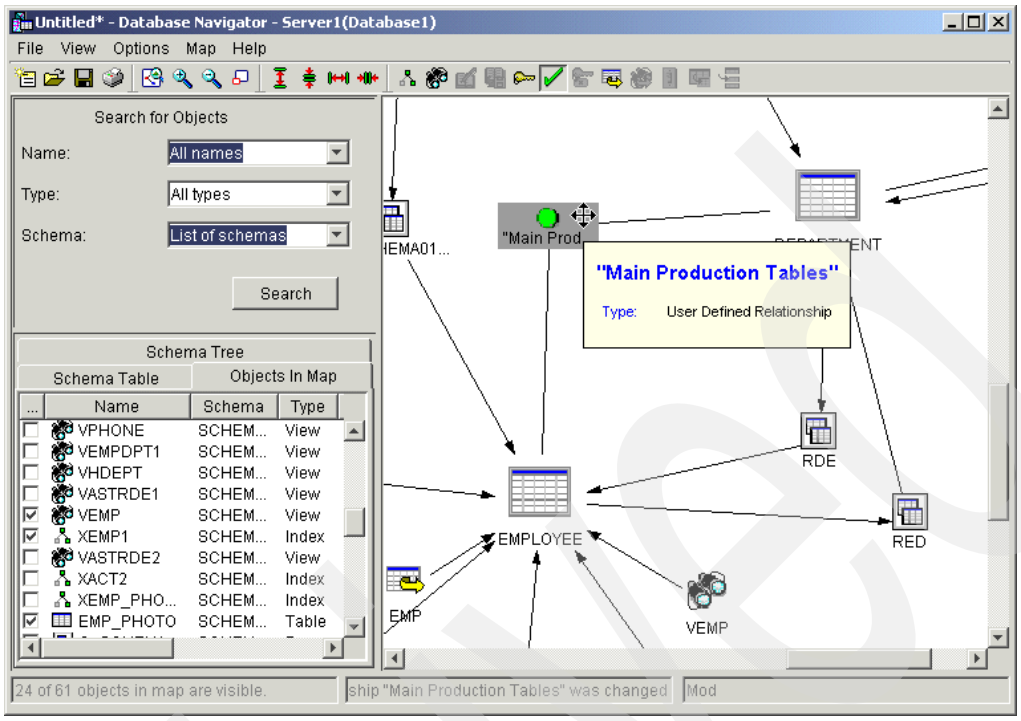


Figure 5-25 Flyover view of a user-defined relationship

5.6 The Database Navigator map icons

Table 5-1 outlines the icons that you may encounter on the Database Navigator map.

Table 5-1 Database Navigator map icons

	The *Library* icon is used in the Database Navigator map 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 to add or remove 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 to add or remove 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 to add or remove 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 to add or remove 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 to add or remove 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 to add or remove 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 to add or remove 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.
	Use this icon to set the zoom on the map so that it fits the current window size.
	Click this icon to increase the level of zoom on the map at the position of the cursor.
	Use this icon to decrease the level of zoom on the map at the position of the cursor.
	This icon invokes the Overview window function. It allows you to position your Database Navigator map panel to any part of a map.
	This icon enables you to decrease the horizontal level of spacing between objects on the map.
	This icon enables you to increase the horizontal level of spacing between objects on the map.
	This icon enables you to decrease the vertical level of spacing between objects on the map.
	This icon enables you to increase the vertical level of spacing between objects on the map.

6

Reverse engineering and Generate SQL

Reverse engineering is a major change that has been included since Version 5, Release 1, Modification 0 (V5R1M0). This function allows you to create Structured Query Language (SQL) for a given schema, table, index, view, etc., and all related objects to them if that option is selected. This enables database administrators (DBAs) to recreate or create duplicates and port to other iSeries servers entire databases or particular parts of a database.

This chapter explains what Generate SQL is, reverse engineering an existing database, and how to generate SQL data definition language (DDL) statements from a database created by data description specifications (DDS).

6.1 A closer look at the Generate SQL function

The Generate SQL function is often referred to as *reverse engineering of database objects*. It provides a graphical user interface (GUI) that allows you to reverse engineer several types of database objects. The results are SQL create statements that are often referred as *DDL statements*.

The Generate SQL function of iSeries Navigator allows you to reconstruct SQL statements used to create existing database objects. With this function, you can reverse engineer database objects. Then you have the option to display the resulting SQL in the Run SQL Scripts window or to save 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 Generate SQL Database Objects support the following objects:

- Schemas
- Tables
- Aliases
- Tables and physical files
- Views and logical files
- Indexes
- Constraints
- Distinct types
- Functions
- Procedures
- Triggers
- Sequences

6.1.1 System requirements and planning

Before you Generate SQL, be sure that you have the following prerequisites:

- Option 12 - Host Servers (5722-SS1)
- TCP/IP Connectivity Utilities (5722-TC1)
- Client Access Express, V5R3M0 (5722-XE1), with the latest service pack applied

6.1.2 Generate SQL features and options

Through the iSeries Navigator map and the schemas display of iSeries Navigator, reverse engineering (Generate SQL) enables you to re-engineer an SQL database or an iSeries database that was not created using SQL.

One use of Generate SQL is to generate the SQL statements of tables, views, indexes, and constraints that were created using the iSeries Navigator interface. For example, when you create a table using iSeries Navigator, there is no method to save 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 into 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.

You invoke the object context from either the Database Navigator map or the Database objects display in iSeries Navigator. To do this, right-click the object and select the **Generate SQL** option.

There is a difference between what appears when you use the two methods. If you select only the Generate the SQL option from the Database objects display, you see the information shown in Figure 6-1.

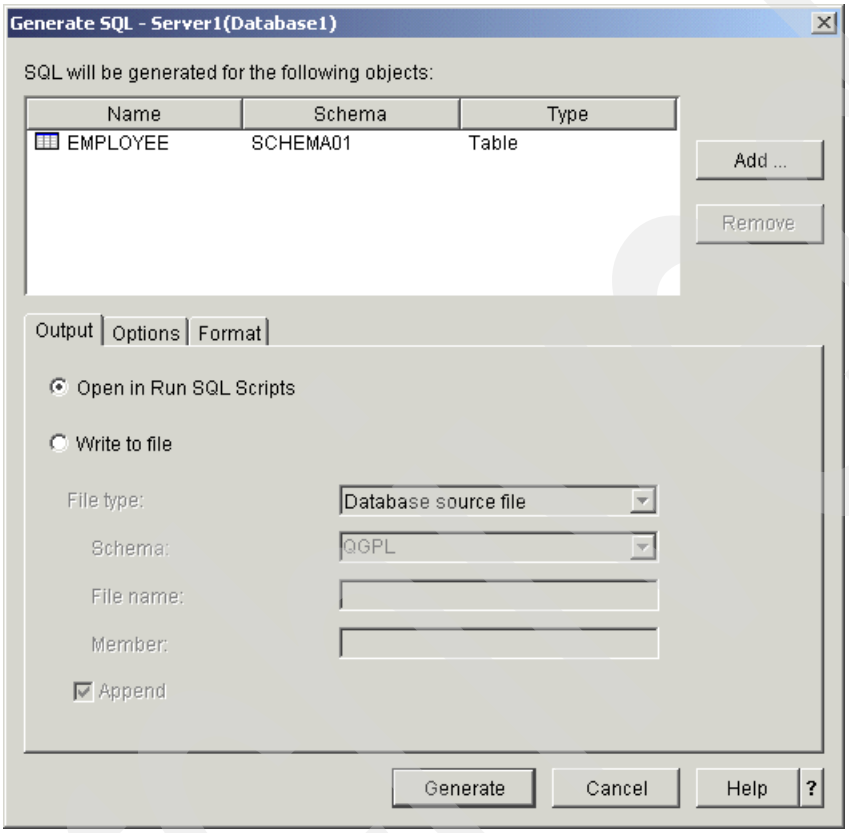


Figure 6-1 Generate SQL display

This method helps you to change the objects that are selected, standard by which they are generated, and the format of the Run SQL script (Figure 6-2).

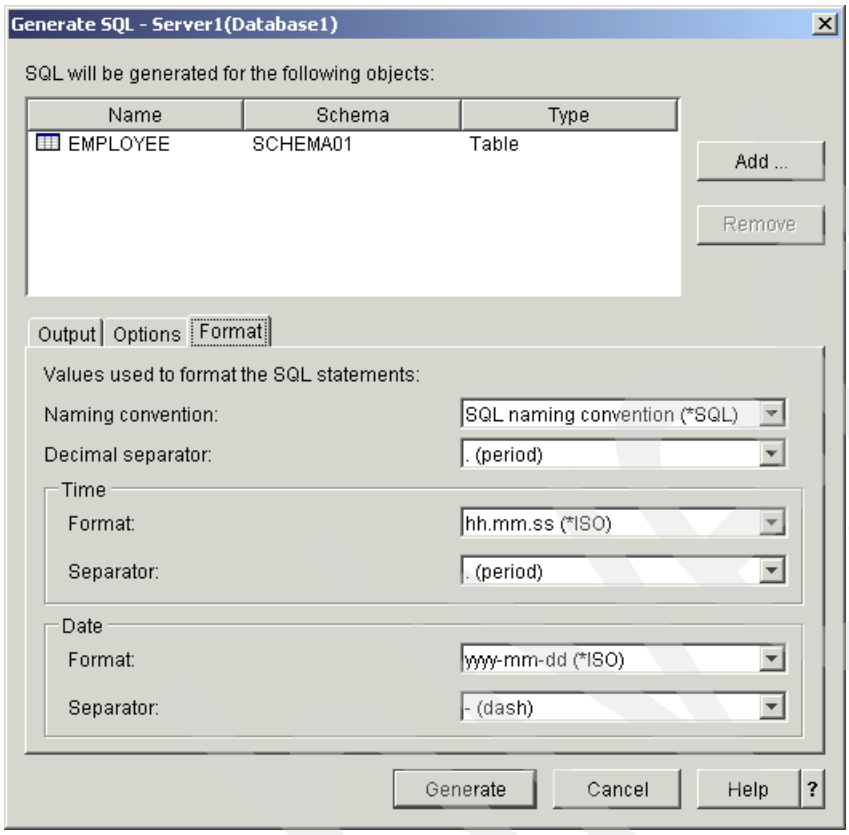


Figure 6-2 Generate SQL format options

This method also enables you to set the options used to create the SQL script (Figure 6-3). The options that you can define include:

- **Standards option**: Under this option, you 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 Generate SQL runs. You may edit this value by using these 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 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. After 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.

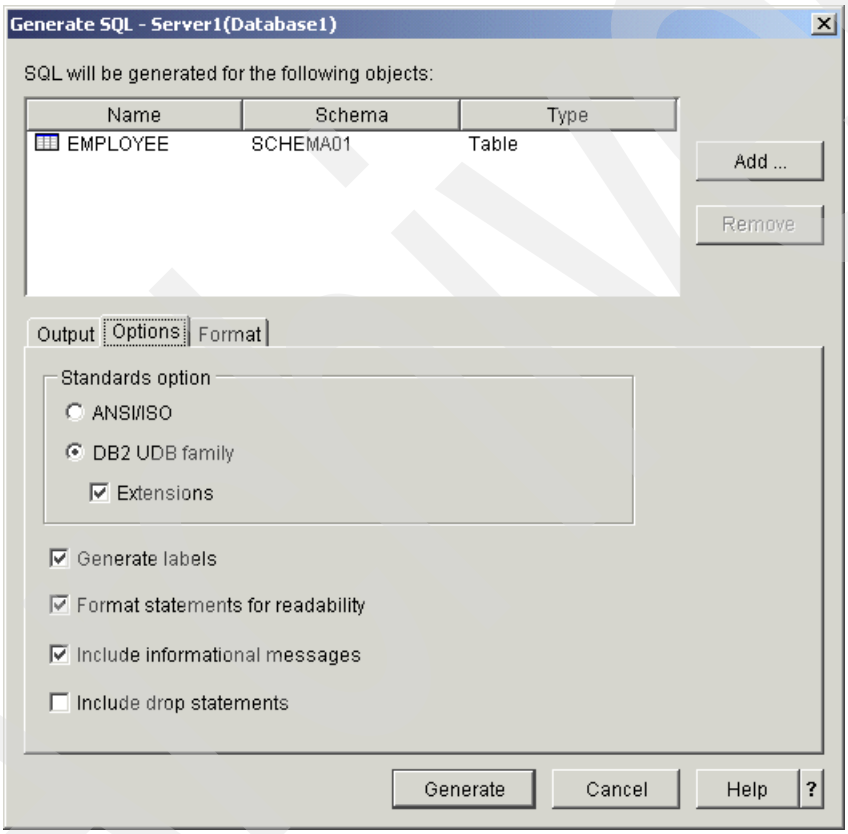


Figure 6-3 Generate SQL options

When you click the Generate button, you prompt the system to generate the SQL and open the Run SQL script window (Figure 6-4).

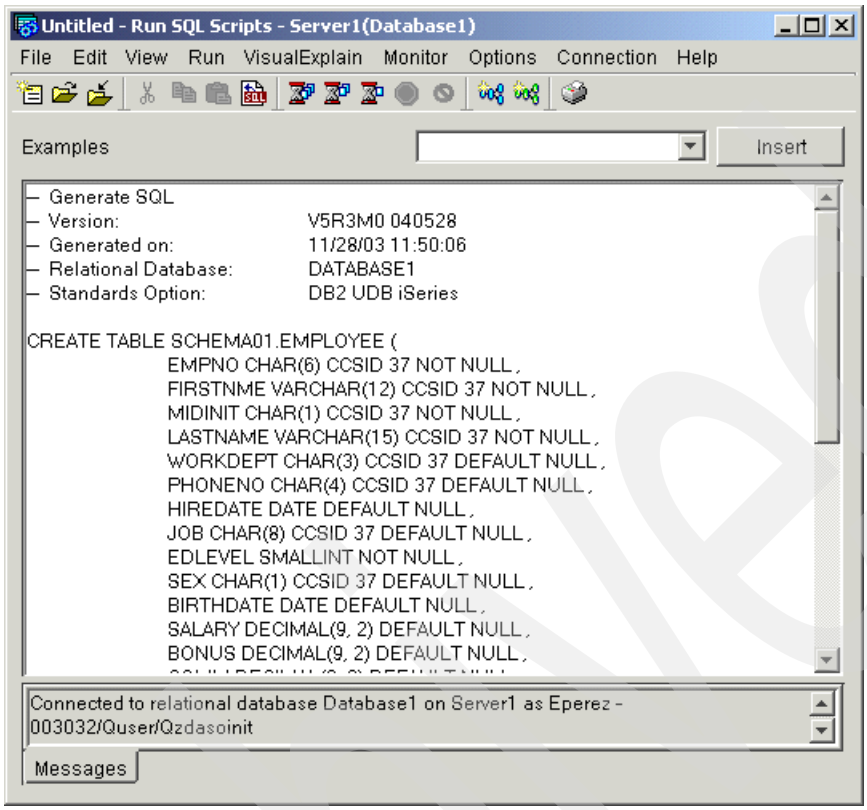


Figure 6-4 Generate SQL Run SQL Scripts window

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 that support SQL. This applies particularly to CASE tools that can use the Run SQL Script as input to recreate the database on other platforms.

6.2 Generating SQL from the schema in iSeries Navigator

With Generate SQL, you can generate the SQL DDL statement for some objects, from your schema in iSeries Navigator. To generate this statement, follow these steps:

1. Start iSeries Navigator.
2. Click the iSeries server that you want to access. Enter your user ID and password.
3. Expand **Databases** and select the database that you want to use.
4. Expand **Schemas**. Then select the schema that you want to use, which in this case is **SCHEMA01**.
5. Click the container that you want to generate SQL. See the database objects that are supported in 6.1, “A closer look at the Generate SQL function” on page 142. You can also click the **All objects** container to select at the same objects of different types.

6. In the right panel, press and hold down the Ctrl key, and locate and select the objects. Then right-click in any of the selected objects and select the **Generate SQL** option as shown in Figure 6-5.

> **Note:** Consider the situation where you click the All objects container and select various objects in the right panel. However, one of the objects is not supported to generate SQL. In this case, when you right-click, the option is not available.

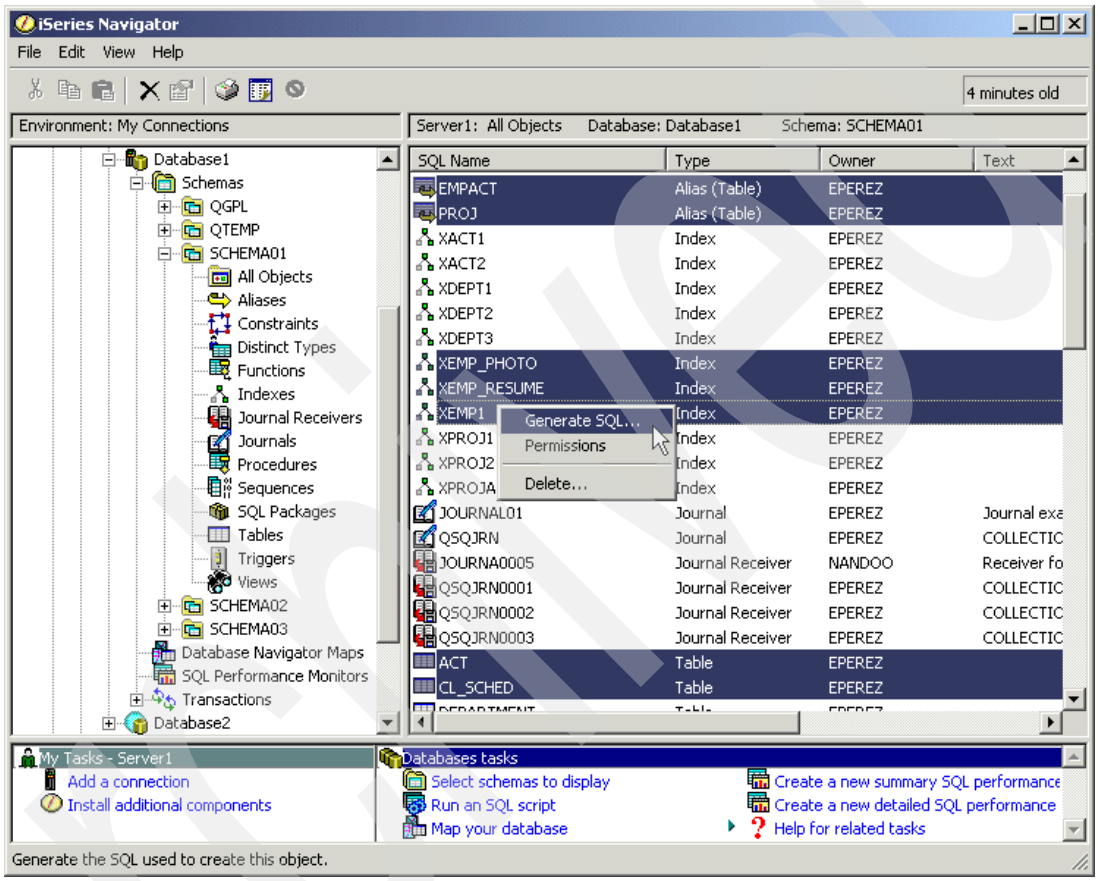


Figure 6-5 Selecting objects from the schema to generate SQL

Important: When you run the Generate SQL function, a new Generate SQL window (Figure 6-6) opens. It provides a list of the objects that were initially selected. It includes the Output, Format, and Options tabs that are used for Generate SQL.

7. In the Generate SQL window (Figure 6-6), click **Generate** to accept the default values.

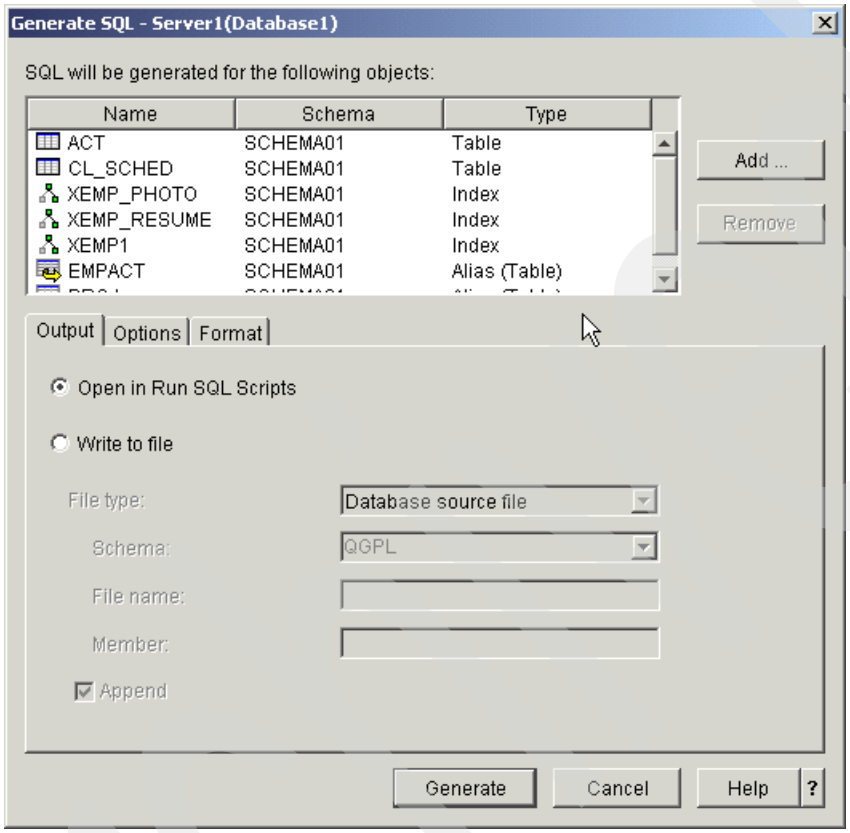


Figure 6-6 Generate SQL window

You can modify the initial list of objects in the Generate SQL window by using the Add and Remove buttons to add new objects or remove objects from the initial list. To generate SQL for all the objects in a schema, simply click the schema and then click the Add button. In the same way, you can add all the objects that belong to any container. If you need select an specific object, you need to expand the tree, select the object, and click the Add button.

Figure 6-7 shows the SQL generating.

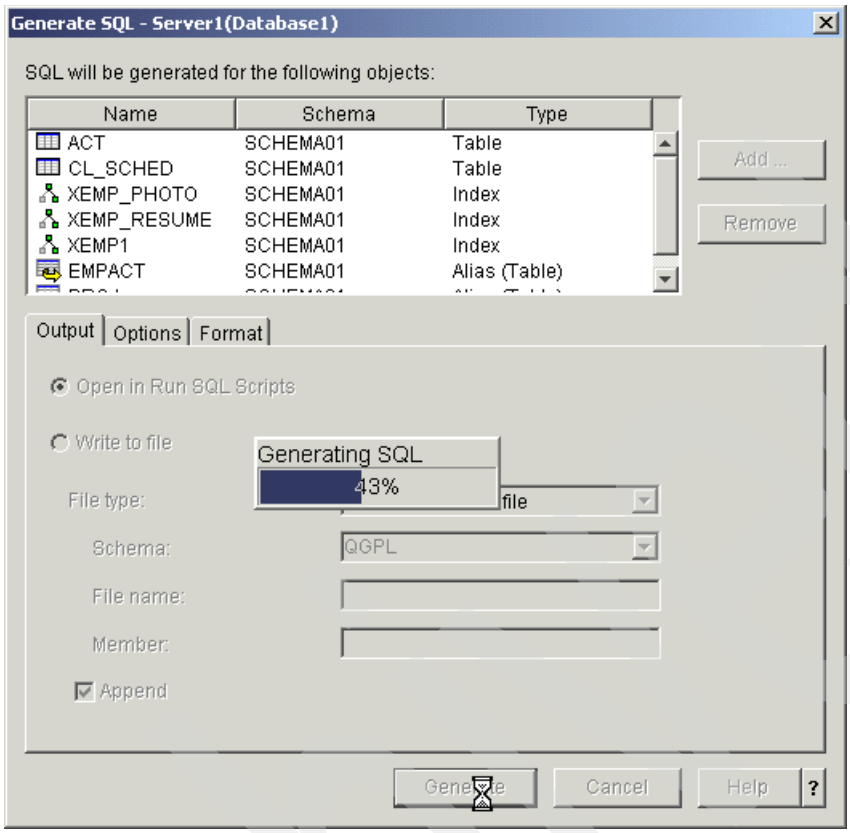


Figure 6-7 Generate SQL display

8. Switch to the new Run SQL Scripts window (Figure 6-8) to see the generated SQL statement.
9. In the Run SQL Scripts window, click **File** →**Save As** to save the SQL script.

> **Important:** You can use the SQL file to replicate your database files on another system (for example, a development system).

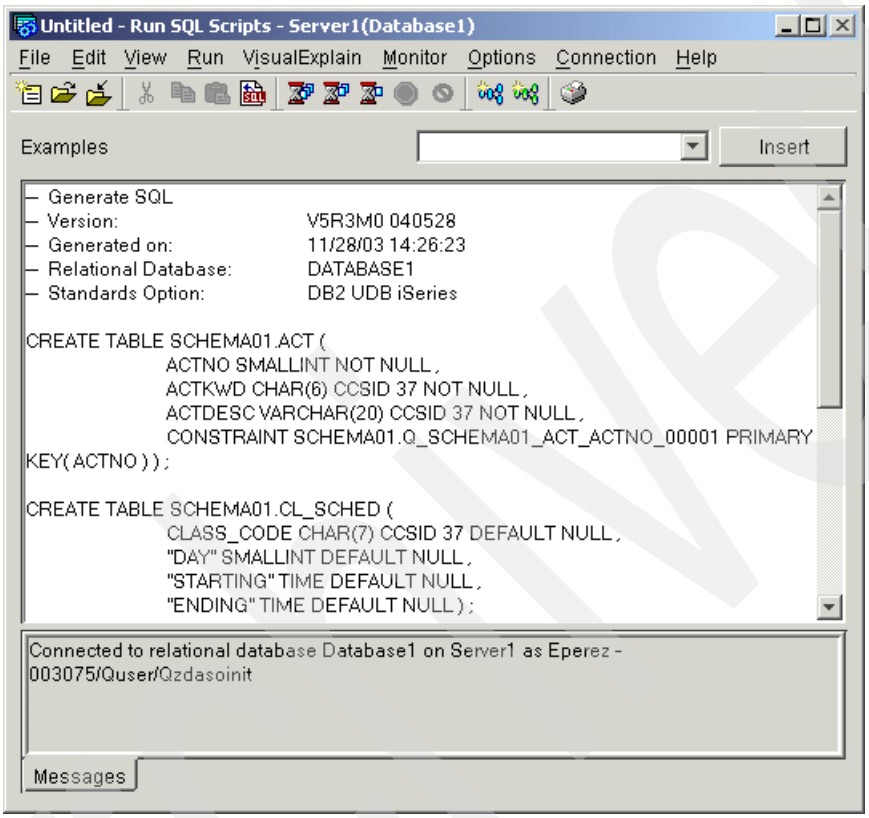


Figure 6-8 SQL generated in the Run SQL Scripts window

6.2.1 Generating SQL to PC and data source files on the iSeries server

You can generate SQL statements to a PC file and to a source member on the iSeries server. The following steps explain how to generate the SQL statements of a group of objects to a PC file:

1. Expand the system that you want to use.
2. Click **Databases** and select the database that you want to work with.
3. Right-click the schema, which is **SCHEMA01** in this example, and select **Generate SQL** as shown in Figure 6-9.

Figure 6-9 Generating the SQL schema in iSeries Navigator

4. In the Generate SQL window (Figure 6-10), complete these tasks:
   a. On the Output tab, select the **Write to file** option.
   b. For File type, select **PC file** to save the generated SQL in this format.
   c. For Location, click **Browse**. Then select your directory from the pull-down list to save your file.

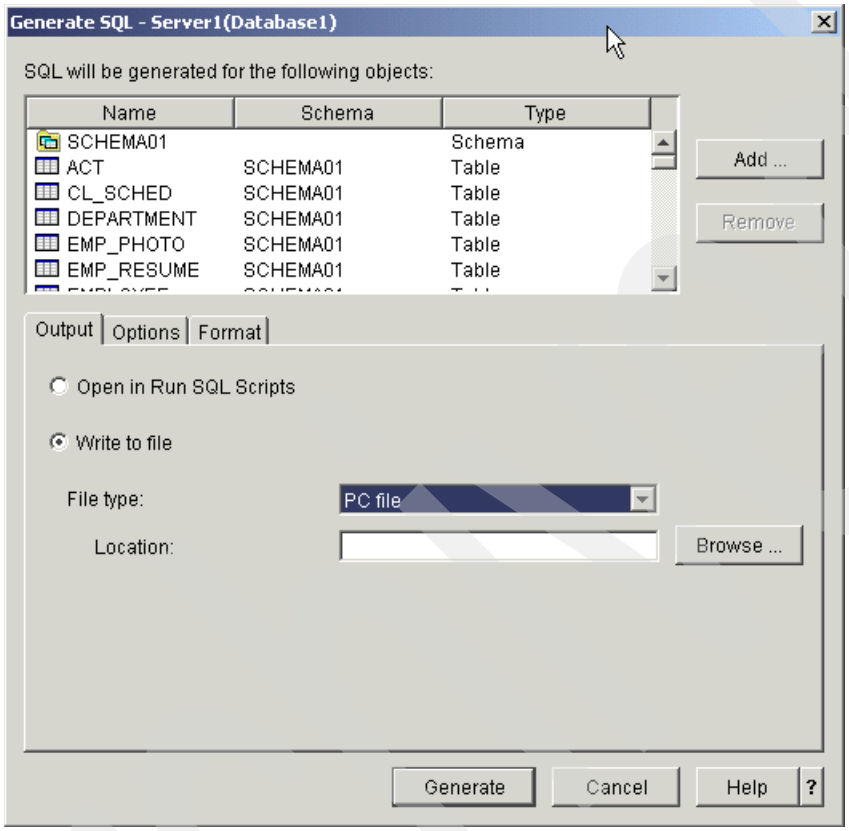


Figure 6-10 Selecting Generate SQL to PC

   d. In the Select window (Figure 6-11), for File name, type the name. In the Files of type field, leave the default SQL files (.sql).
   e. Click the **Select** button to return to the Generate SQL window.

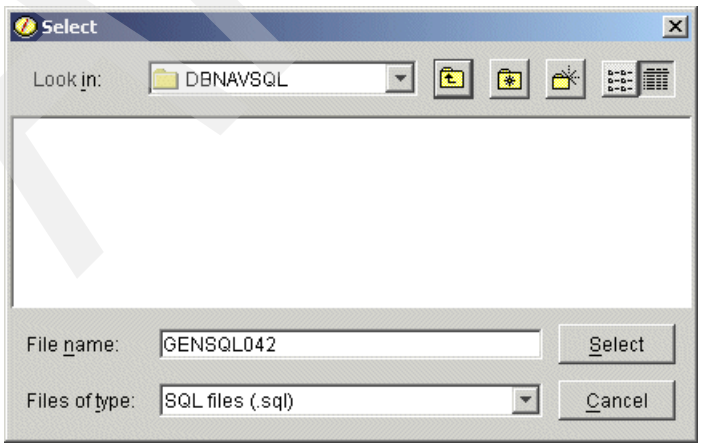


Figure 6-11 Saving the SQL script to a PC file

f. In the Generate SQL window, click the **Generate** button to start generating the SQL DDL statements for all the objects in the schema.

g. A status window opens that shows the progress of the generate SQL process as a percentage (Figure 6-12).

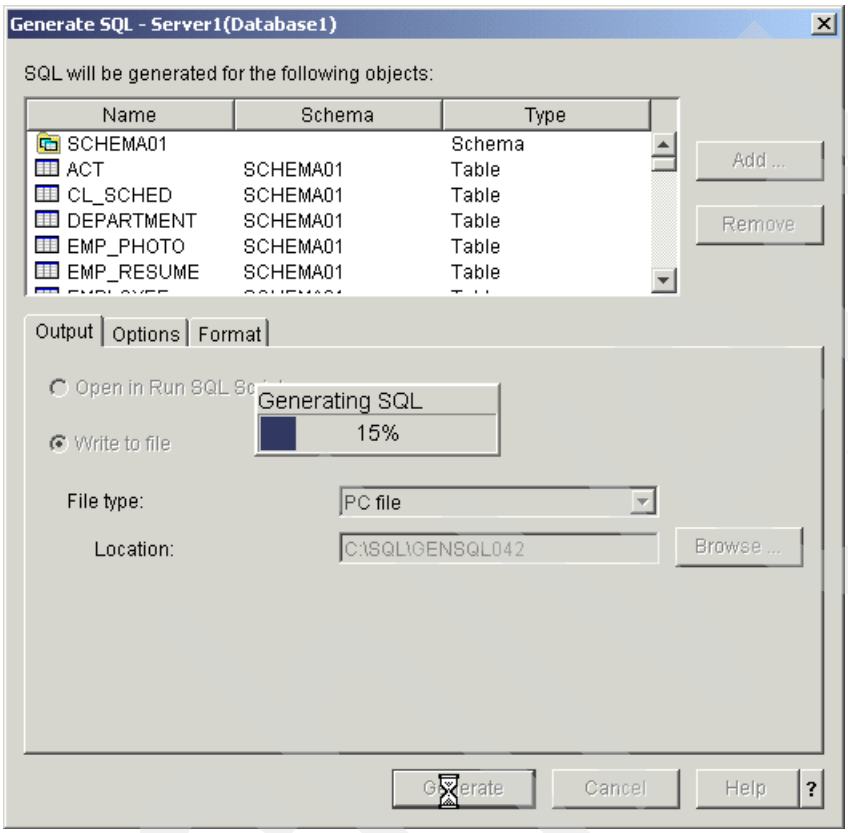


Figure 6-12 Generating SQL window

5. In the iSeries Navigator window (Figure 6-13), in the Database tasks pad, click the **Run SQL Script** icon.

> **Important:** The task pad is a new function that has been added since V5R1. It is located in the lower part of the iSeries 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.

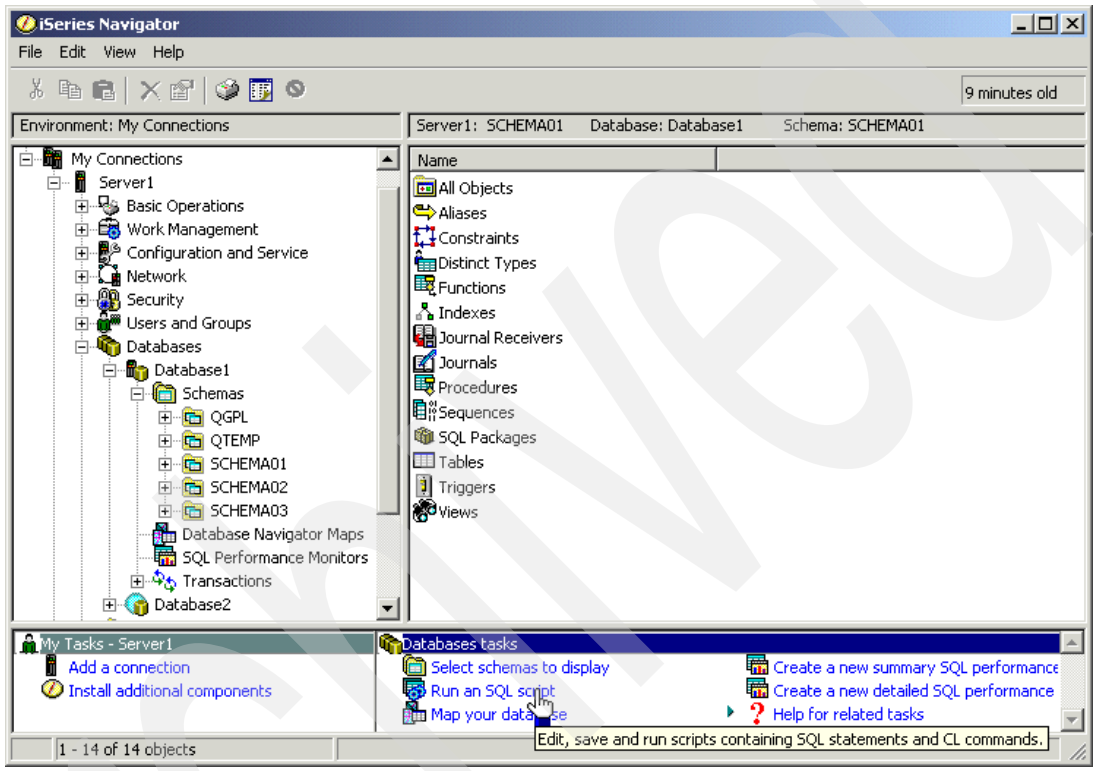


Figure 6-13 Selecting Run an SQL Script from the task pad

6. In the Run SQL Scripts window, click **File** →**Open**.
7. In the Open window (Figure 6-14), complete these tasks:
   a. Click **Look in** and select the directory where you saved your file.
   b. Select the file, **GENSQL042** in our example, and click **Open**.

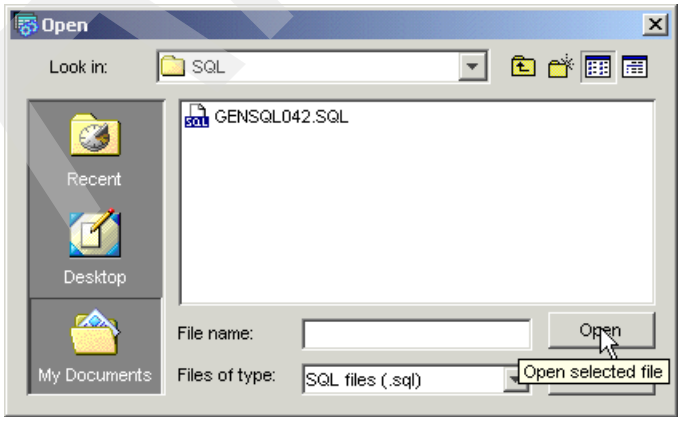


Figure 6-14 Restoring an SQL script file from a PC

8. View the SQL statements generated on the Run SQL Script window (Figure 6-15). Take some time to analyze the order of the statements.

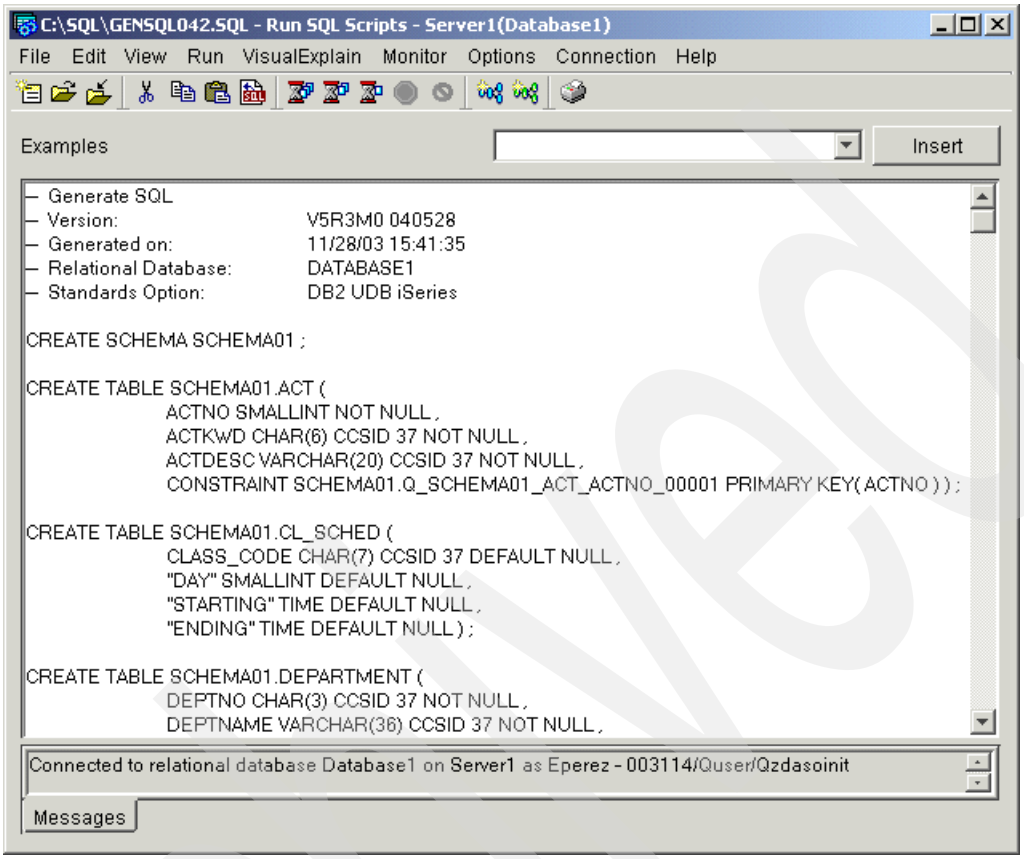


Figure 6-15 SQL Script statement generated

Important: After the statements are generated, you can edit them to create a new copy in another schema and optionally saved them, or you can run them using the SQL Script facility. If you select multiple objects to be SQL Generated, you have the option to run one, some, or all of the statements after any required editing.

To generate the SQL statements of a group of objects to a source physical file on the iSeries server, follow these steps:

1. Click **Databases** and the database that you want to work with.
2. Click **Schemas** and the schema that you want.
3. Select **File** →**Generate SQL** to view the Generate SQL window (see Figure 6-16). This is another way to generate SQL for a group of objects.

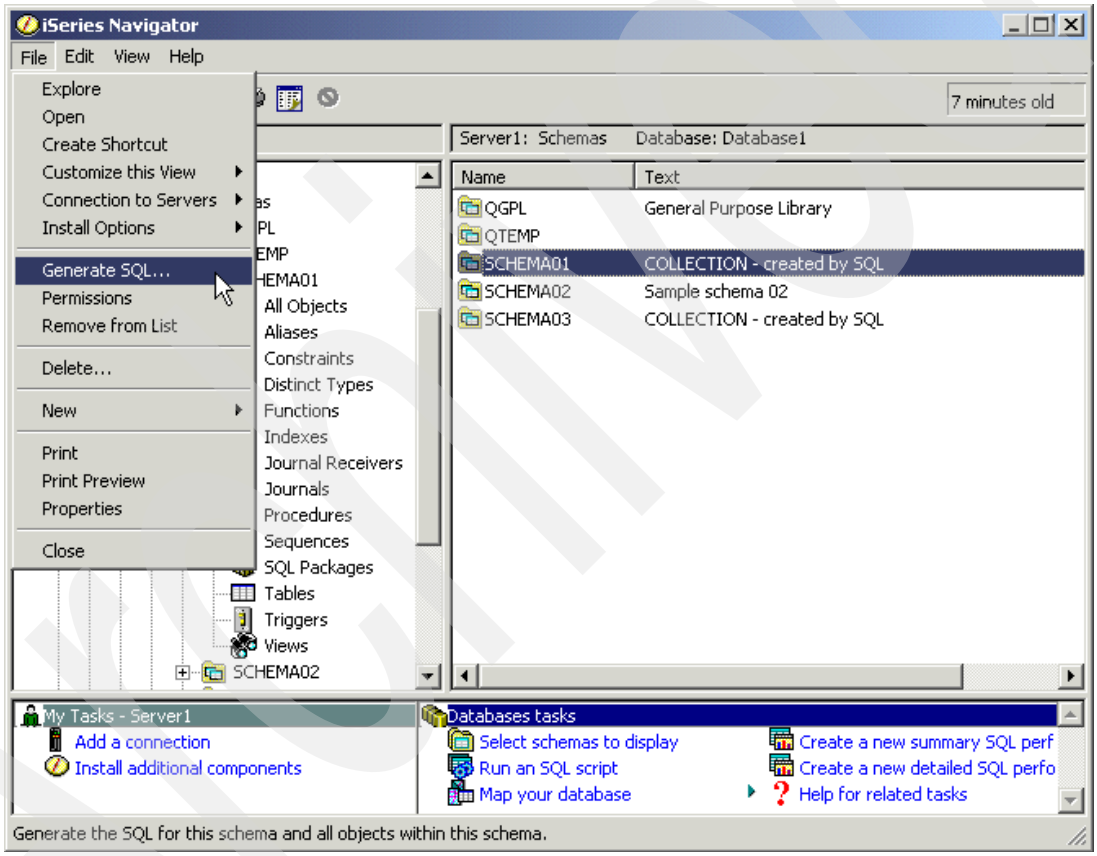


Figure 6-16 Selecting Generate SQL from the File menu

4. In the Generate SQL window (Figure 6-17), complete these tasks:
   a. In the Output tab, select the **Write to file** option.
   b. For File Type, select **Database source file**.
   c. For Schema, select the schema that you want, which is **QGPL** in our case.
   d. Type the file name and the member.
   e. Click the **Generate** button to start the Generate SQL process on the iSeries server.

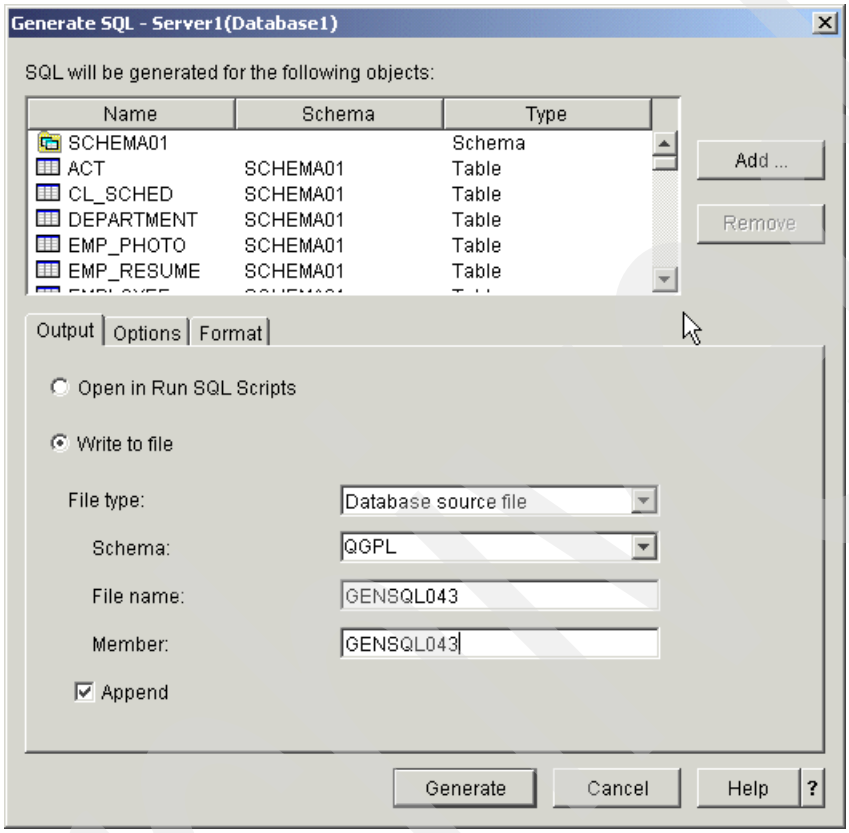


Figure 6-17 Starting the Generate SQL process on the iSeries server

> **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.

5. In the iSeries Navigator window (Figure 6-18), double-click the **GENSQL043** file, in this case, to see the script.

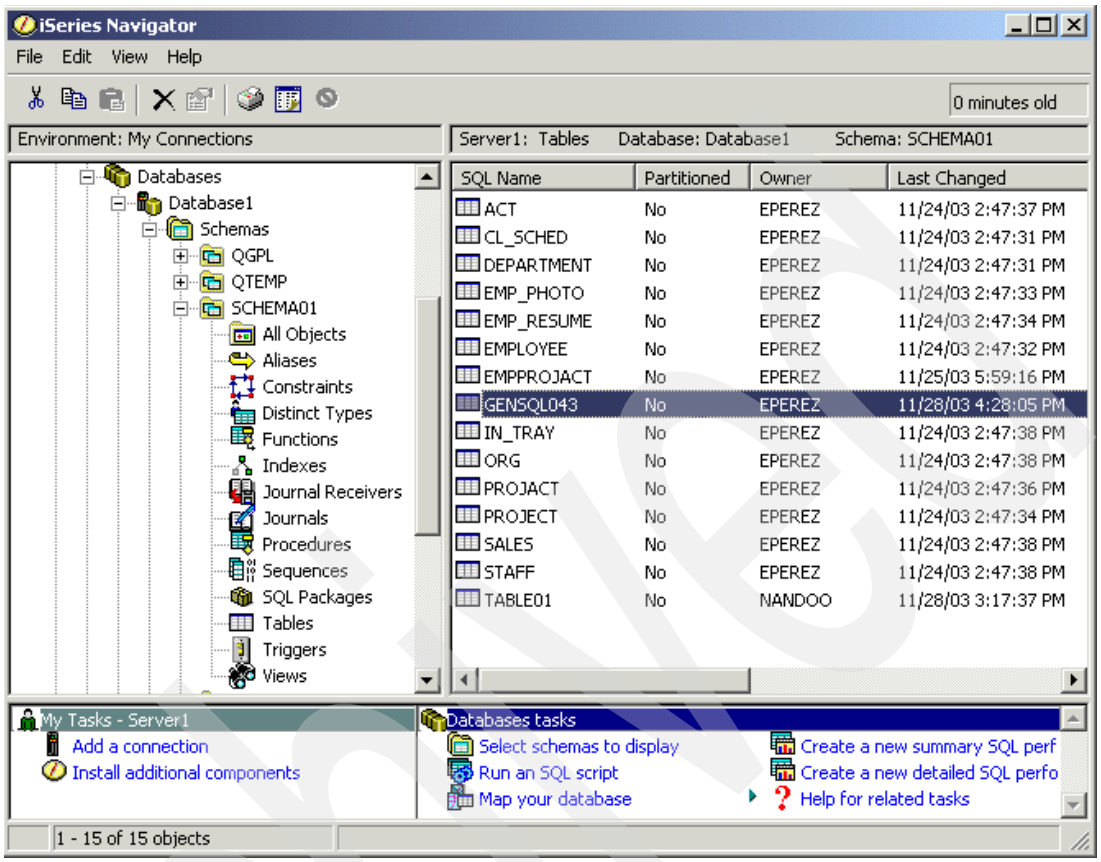
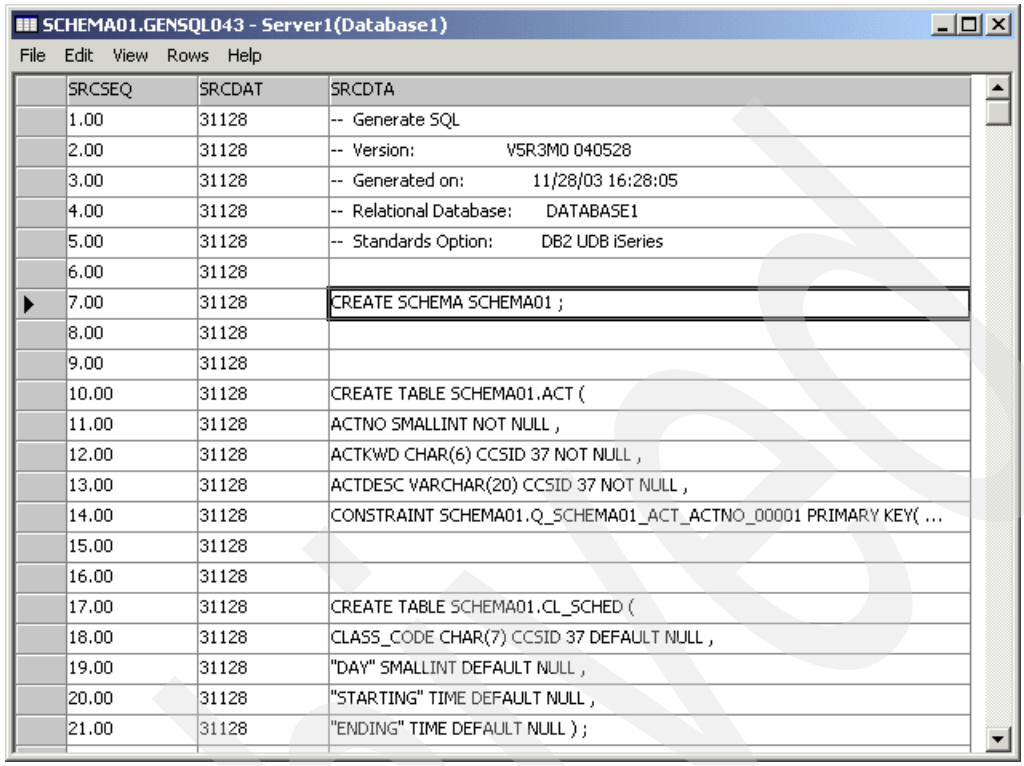


Figure 6-18 Selecting the source physical file to show the Generate SQL Script

6. The script appears as shown in Figure 6-19. Expand the window and use the scroll bar to explore the script file.



SCHEMA01.GENSQL043 - Server1(Database1)

File Edit View Rows Help

SRCSEQ	SRCDAT	SRCDTA
1.00	31128	-- Generate SQL
2.00	31128	-- Version: V5R3M0 040528
3.00	31128	-- Generated on: 11/28/03 16:28:05
4.00	31128	-- Relational Database: DATABASE1
5.00	31128	-- Standards Option: DB2 UDB iSeries
6.00	31128	
7.00	31128	CREATE SCHEMA SCHEMA01 ;
8.00	31128	
9.00	31128	
10.00	31128	CREATE TABLE SCHEMA01.ACT (
11.00	31128	ACTNO SMALLINT NOT NULL ,
12.00	31128	ACTKWD CHAR(6) CCSID 37 NOT NULL ,
13.00	31128	ACTDESC VARCHAR(20) CCSID 37 NOT NULL ,
14.00	31128	CONSTRAINT SCHEMA01.Q_SCHEMA01_ACT_ACTNO_00001 PRIMARY KEY(...
15.00	31128	
16.00	31128	
17.00	31128	CREATE TABLE SCHEMA01.CL_SCHED (
18.00	31128	CLASS_CODE CHAR(7) CCSID 37 DEFAULT NULL ,
19.00	31128	"DAY" SMALLINT DEFAULT NULL ,
20.00	31128	"STARTING" TIME DEFAULT NULL ,
21.00	31128	"ENDING" TIME DEFAULT NULL) ;

Figure 6-19 Exploring the SQL Script file from Operations Navigator

6.2.2 Generating SQL from the Database Navigator map

You can generate the SQL DDL statement from some or all objects in a map generated by the Database Navigator feature (see Chapter 5, “Database Navigator maps” on page 111).

1. Click the Database Navigator icon to display the maps on the right that exist on the iSeries server. See Figure 6-20.
2. Double-click to open the database map that you created.

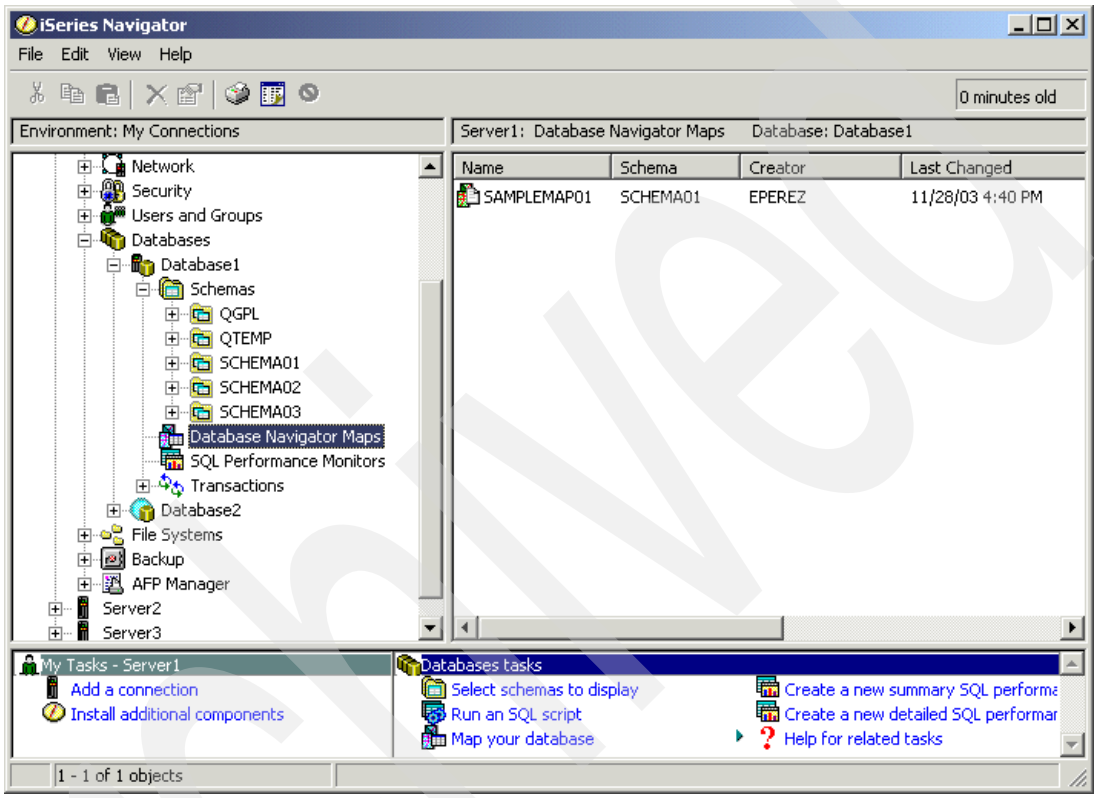


Figure 6-20 Opening the Database Navigator map

3. In Database Navigator (Figure 6-21), click the **View** →**Zoom** →**To Fit Window** to fit all objects on the map in this window.

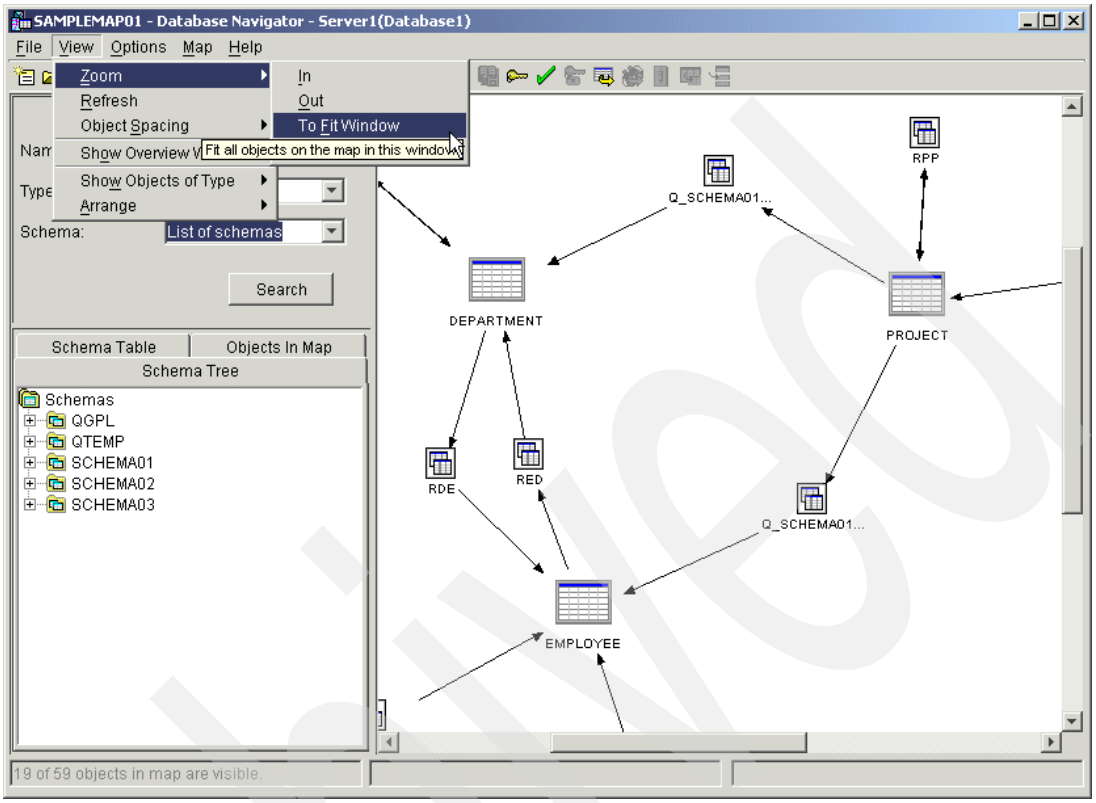


Figure 6-21 Fitting all objects in a map

4. Use the criteria selection in the locator pane and select only the schema that you need (**SCHEMA01** in this example). Click the **Schema** parameter to select your schema as shown in Figure 6-22.

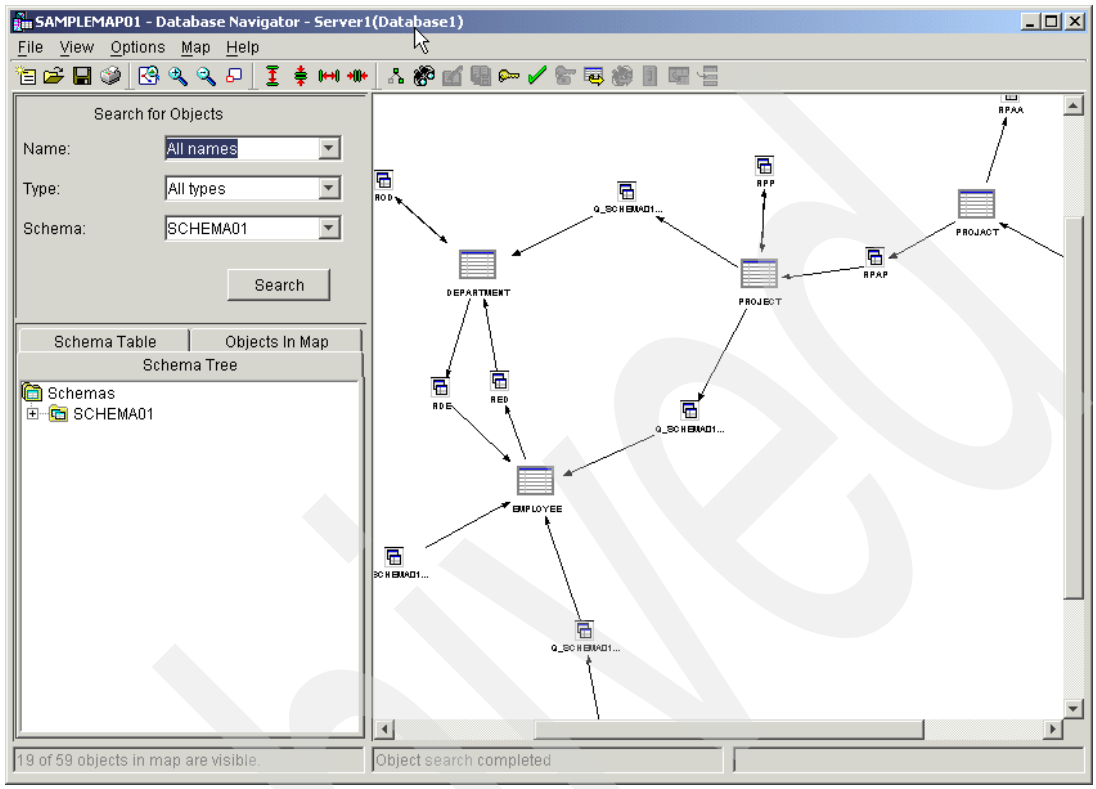


Figure 6-22 Selecting only the sample schema to appear in the Database Navigator map

5. As shown in Figure 6-23, complete these tasks:
   a. Expand the schema to see the found objects, such as tables, indexes, and views.
   b. Expand the Tables database object.
   c. Double-click the table that you want (**EMPLOYEE** table in this example) in the list of tables to find this table in the map.
   d. Right-click the table in the map and select **Generate SQL**.

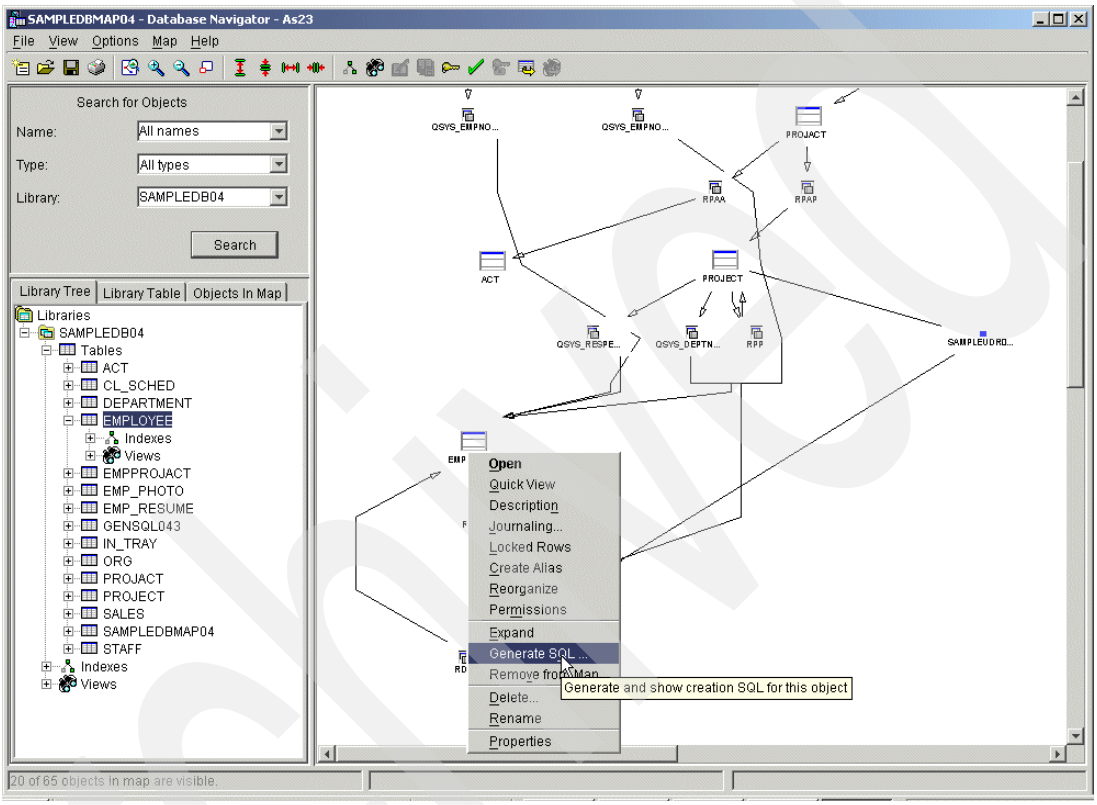


Figure 6-23 Generating SQL for a specific object from the map

6. In the Run SQL Script window (Figure 6-24), explore the Generated SQL statement, using the scroll bar to navigate.
7. Click **File** →**Save As** to save the SQL script.

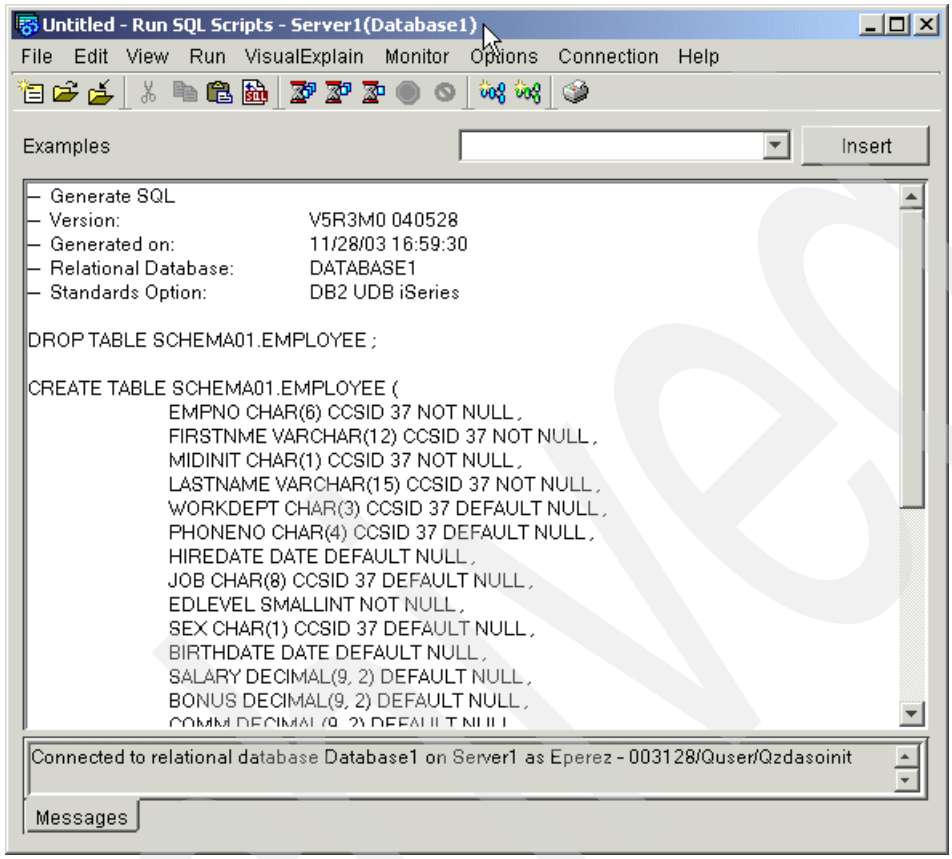


Figure 6-24 Generating SQL from the employee table

Now let's see how to generate the SQL DDL statements for all objects in a schema:

1. Switch to the Database Navigator window.
2. Click the **Map** →**Generate SQL** →**All Objects** to generate the SQL statement for all objects in your schema. See Figure 6-25.

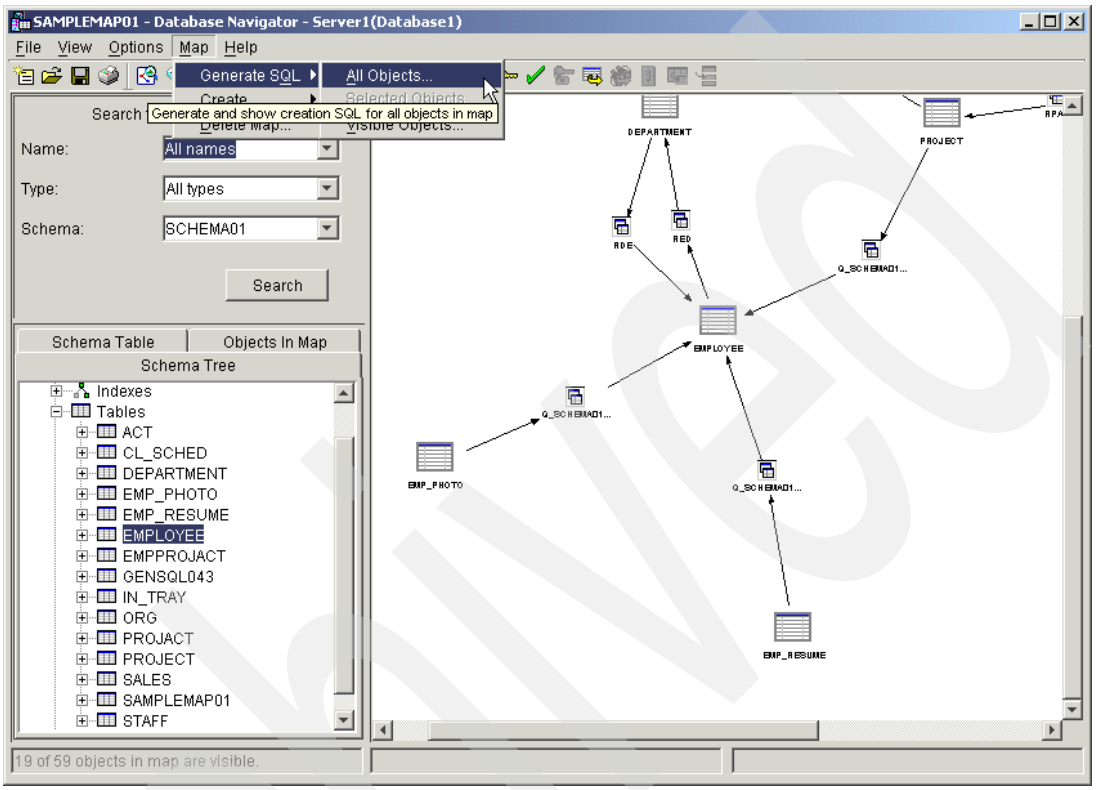


Figure 6-25 Generate SQL for all objects in a schema

3. A status window opens that shows the progress of the Generate SQL as a percentage.
4. Click **File** →**Save As** to save the map.
5. Click **File** →**Exit** to close the Run SQL Script window.

6.2.3 Generating SQL from DDS

The Generate SQL function works with objects that were created using SQL and 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.

To reverse engineer an existing DDS created database, follow these steps:

1. Expand the list of schemas.
2. Change the list of schemas in iSeries Navigator to include the schemas or libraries that have DDS created objects.

3. Right-click the **DDSLIBXX** schema and select **Generate SQL** as shown in Figure 6-26.

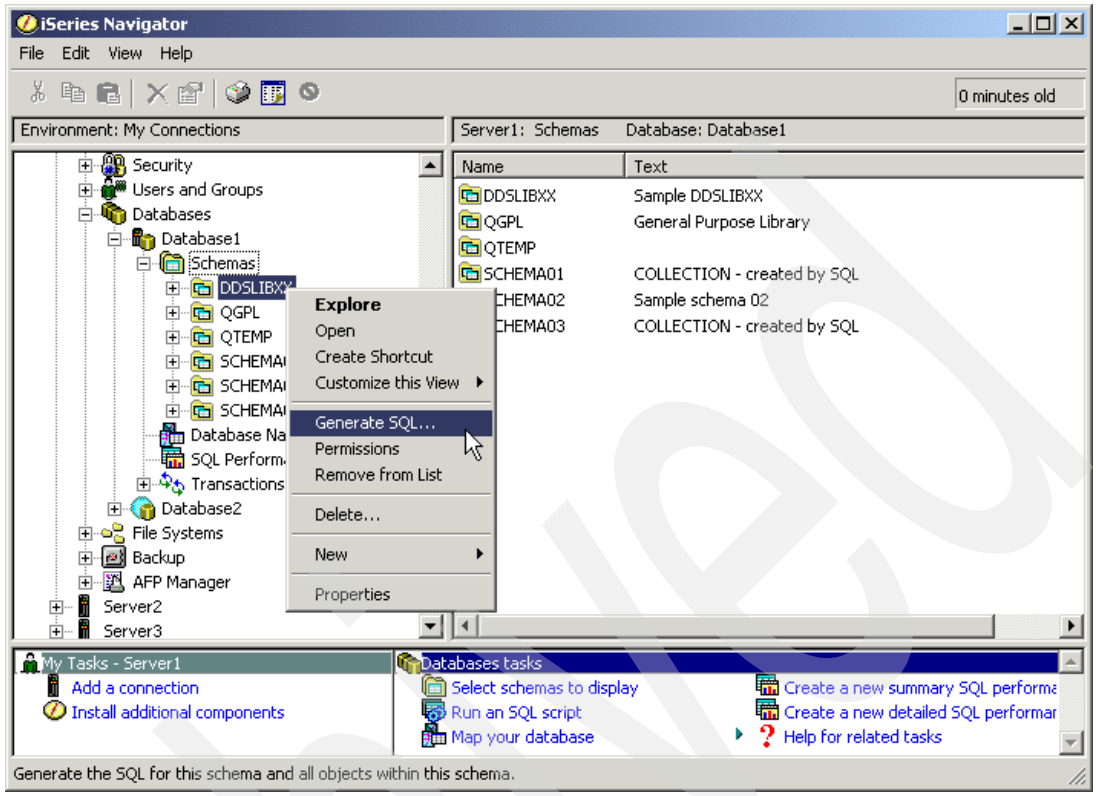


Figure 6-26 Selecting physical files to generate an SQL statement

4. In the Generate SQL window, leave the default options. Click the **Generate** button.

5. The SQL Script Center appears with the generated SQL DDL statements posted in the working area as shown in Figure 6-27.

> **Important:** Some DDS-specific keywords *cannot* be converted to SQL. This appears in the code as messages SQL150x (see Figure 6-27).

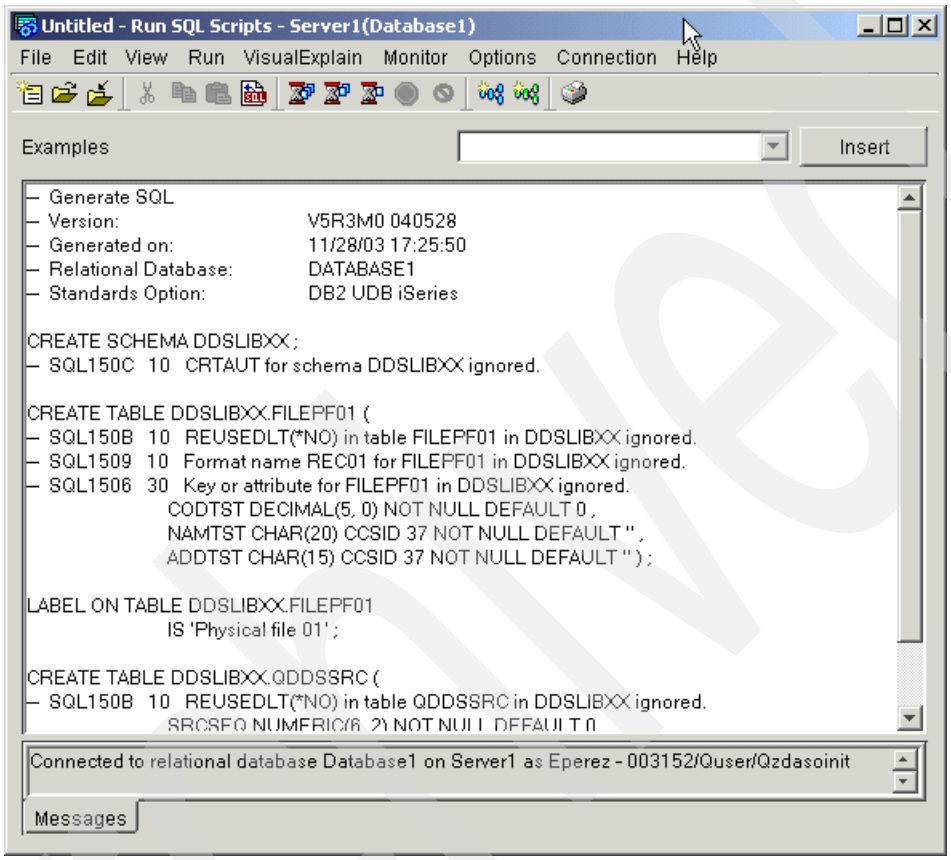


Figure 6-27 Exploring SQL script generated from physical files

7

Run SQL Script Center

Run SQL Script Center is a powerful interface to your iSeries Navigator. It enables you to create, edit, run, and troubleshoot scripts of Structured Query Language (SQL) statements.

This chapter explains how to use Run SQL Script Center, how to run SQL scripts and commands from Run SQL Script Center, and how to use current SQL for a job. It discusses Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC) connections, Change Query Attributes, SQL Assist, SQL Performance Monitors, and System Debuggers.

7.1 Run SQL Scripts

The Run SQL Scripts Center is a powerful interface to your iSeries database. It enables you to use any SQL statements to issue any kind of operations that you are authorized to on the iSeries database objects. The Run SQL Scripts window in iSeries Navigator helps you to create, edit, run, and troubleshoot scripts of SQL statements.

DB2 Query Manager and SQL Development Kit for iSeries (Licensed Product Program 5722-ST1) is not a prerequisite for using Run SQL Scripts. This component of iSeries Navigator uses JDBC to access the server.

Follow these steps to open the Run SQL Scripts window:

1. In the iSeries Navigator window, expand the server you want to use.
2. Expand **Databases**.
3. Right-click the database that you want to work with and select **Run SQL Scripts**. Figure 7-1 shows the Database context menu with the Run SQL Scripts action highlighted. You can also click **Run an SQL script** from the task pad.

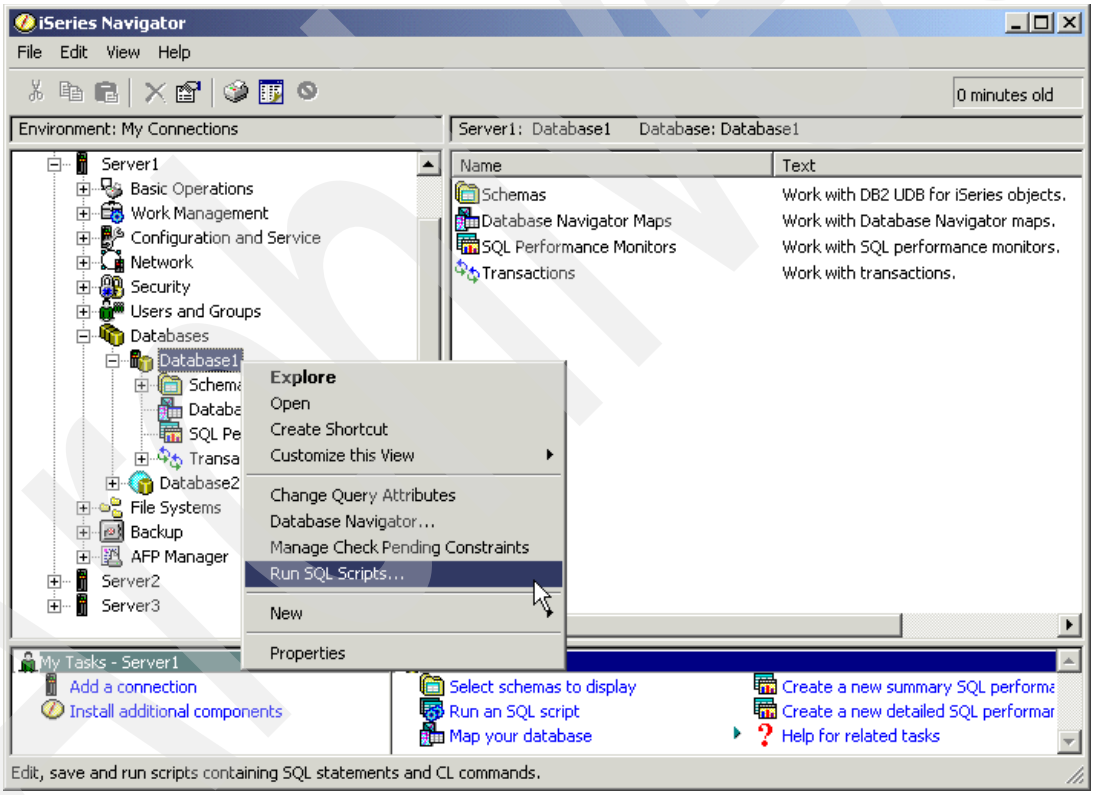


Figure 7-1 Run SQL Script

When the Run SQL Scripts window opens, you are connected to the database from which you launched Run SQL Scripts. You can change your connection to a different database. Figure 7-2 shows an example of the initial Run SQL Scripts window.

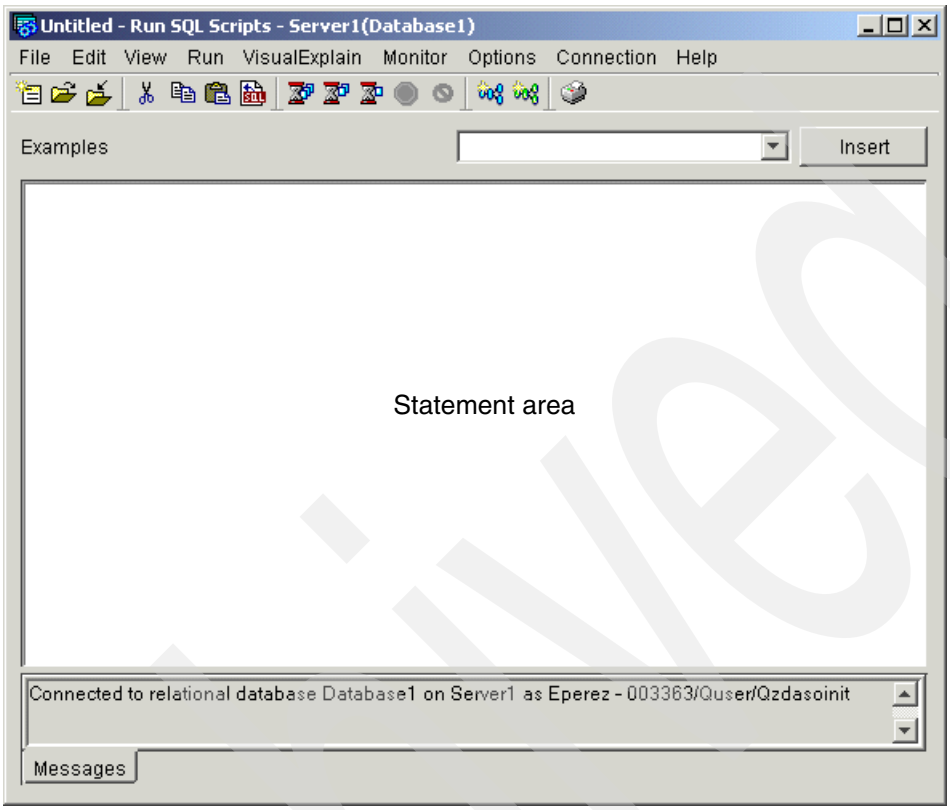


Figure 7-2 Run SQL Scripts: Initial input window

The Run SQL Scripts window enables you to 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. We discuss them later in this chapter.

As shown at the top in Figure 7-2, 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. To display the list of existing SQL statements, click the drop-down list. You can then select an SQL statement from the list shown, click the Insert button, and have it *inserted* into the statement input area.

You can modify the selected SQL statement or enter your own SQL statement. You can run one or more of your entered SQL statements in different ways and stop between statements.

Before we discuss the run actions, refer to Figure 7-3 to see the different panels within the Run SQL Scripts function.

The beginning of the list of provided SQL statements is shown in the drop-down list. This list was produced by clicking the down arrow. In this example, we do not select an SQL statement to be placed into the statement input area. However, if you select one SQL statement from the drop-down list, the statement appears in the “SQL statement example” area or field to the left of the Insert button. Then, when you click the Insert button, the statements are placed into the SQL input area. You can execute the statements placed in this area and then view the results of the SELECT statements by clicking in the tabs at the bottom of the window.

The Run History panel (the gray pane) shows the success and any messages of the SQL statements that are run. When you select the Edit option from the menu bar, you have the option to *clear run history* information.

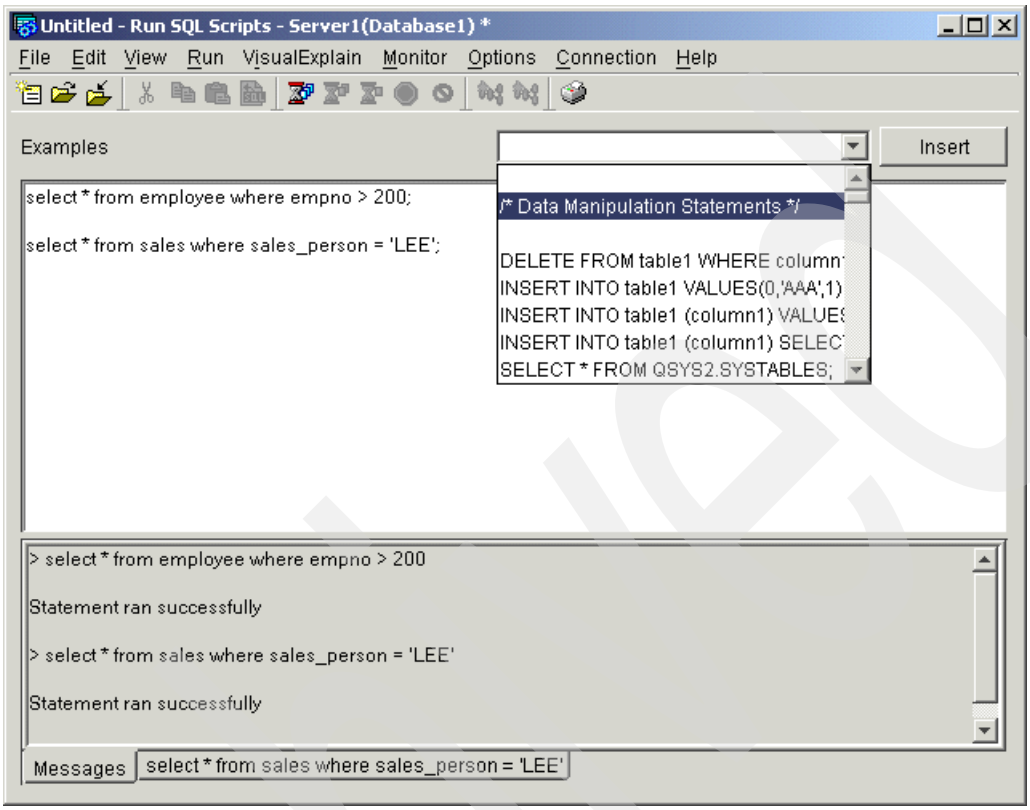


Figure 7-3 Run SQL Scripts window pane example

Figure 7-4 includes some SQL SELECT statements to illustrate more of the power of DB2 Universal Database (UDB) for iSeries that is accessible through iSeries Navigator.

The fourth line in Figure 7-4 (`SELECT * FROM SCHEMA02.ACT;`) shows a select from a table that is in a different schema than the schemas included in the job description's initial library list. This is possible by qualifying the table. In this case, the separator character (.) is valid because the naming convention is set to the SQL format.

By default, SQL statements run on the system to which you are connected. You can use the CONNECT SQL statement to connect to a remote database (DATABASE2 in this example), using OS/400 Distributed Relational Database Architecture (DRDA) over TCP/IP. This is shown in the fifth line (`CONNECT TO DATABASE2;`). Assuming that this CONNECT statement is successful, all SQL statements thereafter are directed to remote database DATABASE2 until an SQL "release all" statement is issued, when the connection returns to access only the local database.

OS/400 supports connections to multiple remote databases during the same session. For example, following the `UPDATE SCHEMA01.TEST TEST SET COLUMN1 = 'A' WHERE TEST.IDCOL >=3;` statement, you can issue a `connect to REMOTEDB` statement. Assuming this is successful, all of the following SQL statements are directed to this remote database. You can then issue a "set connection to DATABASE2" statement that resets the current window back to system DATABASE2. You need to keep track of which system (remote database) you are connected to and on which system you are performing operations.

Note: Distributed Relational Database Architecture (DRDA) is the IBM-defined architecture to access 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 JDBC set of capabilities. On IBM systems, Distributed Data Management (DDM) is a higher level interface to DRDA capabilities.

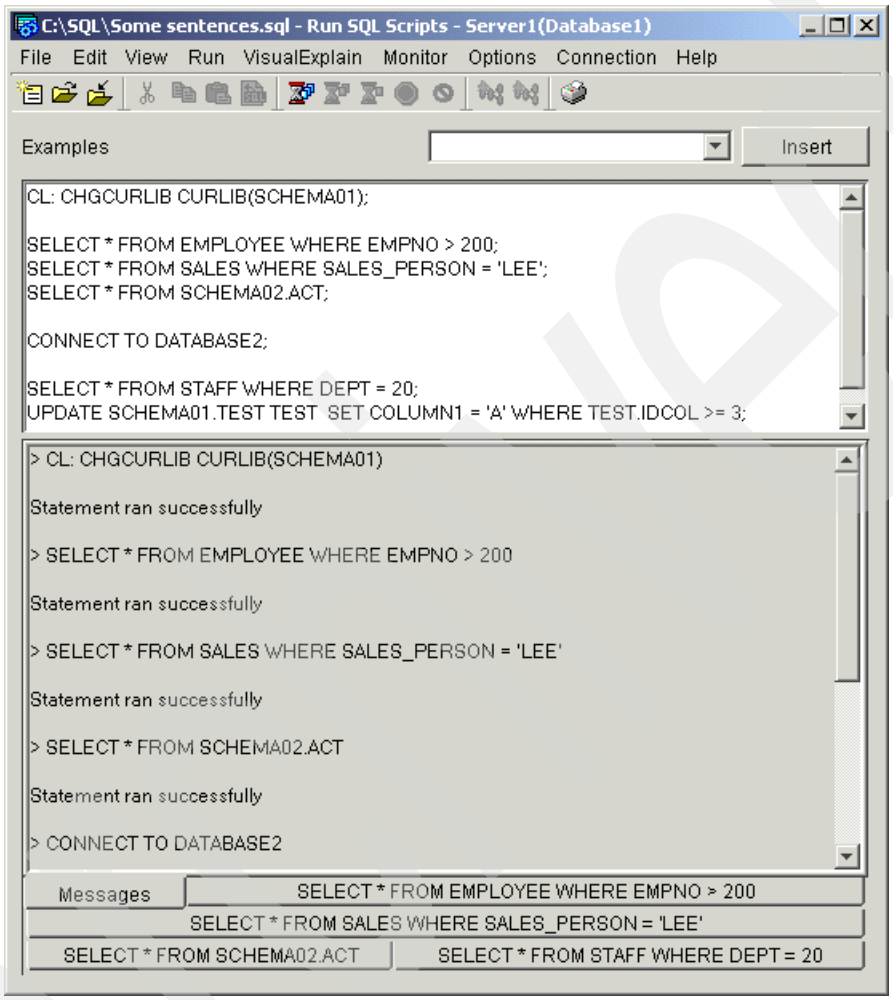


Figure 7-4 Run SQL Scripts: Additional sources

7.1.1 ODBC and JDBC connection

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 (where data is stored). JDBC is an equivalent standard interface for database connectivity from Java applications.

iSeries 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. The production mode job name starts with QZDASOINIT (or QZDASSINIT if Secure Sockets Layer (SSL) is used). In version 4, with ODBC data sources, you can set up an iSeries 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 iSeries Access-provided ODBC driver is done by using the standard ODBC data source administrator, provided with the Microsoft 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 whether to use SSL with the ODBC connection.

Some client applications (including iSeries Navigator) may provide their own unique data source definition. You can create your own data source to limit the schemas that can be used. And, as previously described, you can create 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 iSeries 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 Server1, the data source used by iSeries Navigator is named QSDN_Server1.

  Important: Unless you are an ODBC expert, *do not* change any of the default settings for this data source. If you change them, iSeries Navigator may fail to operate correctly.

- A data source is used if you select 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 you can change by selecting Connections →JDBC Setup as shown in Figure 7-5. 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 schemas and tables to which you are authorized.

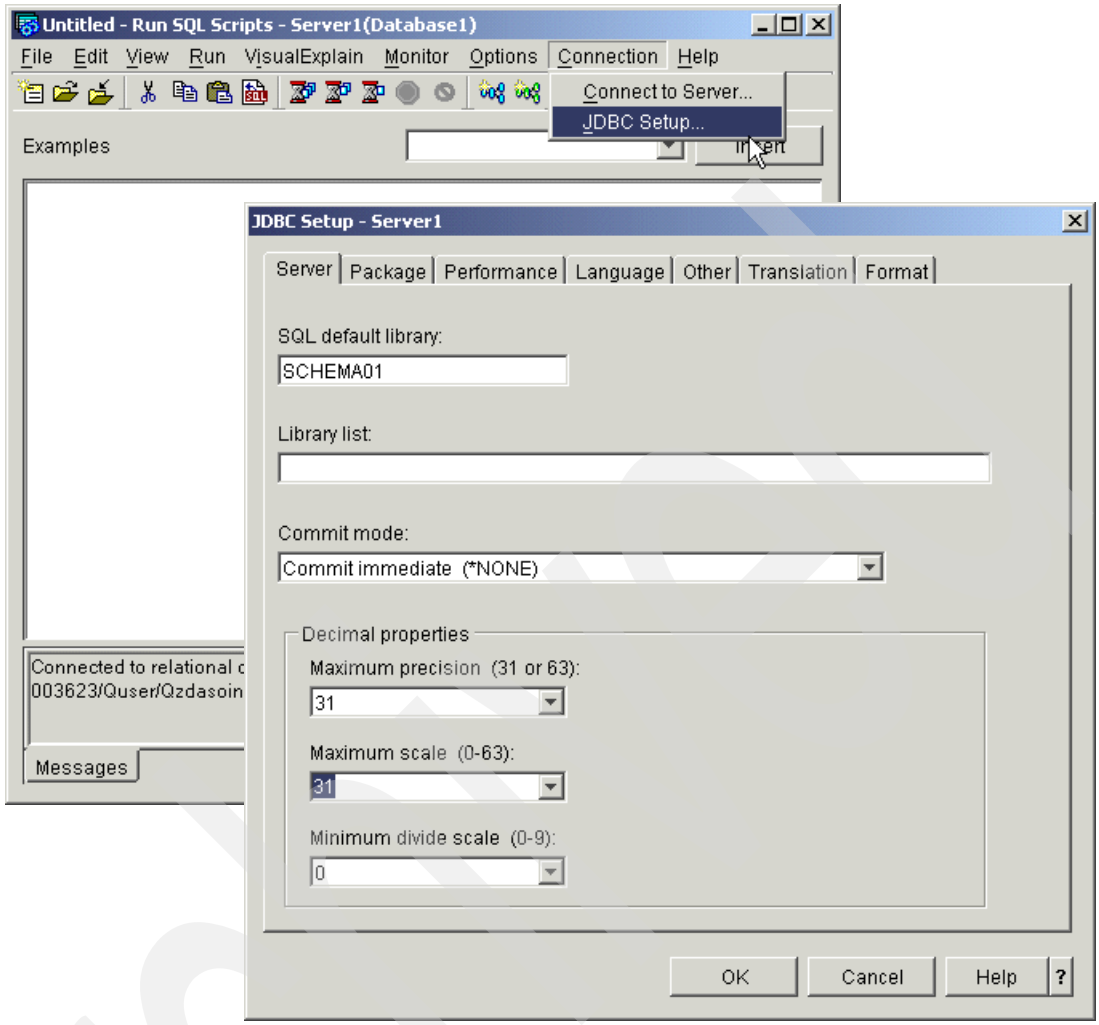


Figure 7-5 JDBC data source panel

Server tab

The SQL default library enables you to specify the default SQL schema. SQL statements can be created without specifying the schema in which an object is located. The objects in the SQL statement are said to be *unqualified*. For example, consider the following SQL statement:

```
Select * from TABLE01
```

TABLE01 is an unqualified object. The schema that is used to locate the table name is called the *implicit qualifier*. If an SQL default library or schema is specified, it is used as the implicit qualifier. The SQL default schema is also the first item in the library list.

If an SQL default schema is not explicitly specified, one of the following cases applies:

- For SQL naming, the implicit qualifier is the run-time authorization. This is a schema with the same name as the user profile specified on the JDBC connection.
- For system naming, the implicit qualifier is the job library list.

A library list enables you to change the set of libraries or schemas available to the user of the JDBC data sources. The default (*LIBL) 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. This list of libraries is used during connections to this data source.

You can separate the library names by commas or spaces. You can either add the libraries to your library list or replace the list entirely. To add to the existing user library list, add `*LIBL` to the list of libraries. All libraries that are listed before *LIBL are added to the front of the user library list. All libraries that are listed after *LIBL are added to the end of the user library list. If no SQL default library is specified and the naming convention is *SYS, the library list is searched for unqualified tables, views, and procedures.

Commit mode controls the level of DB2 UDB for iSeries commitment control. This includes 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 is 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 though they never occurred. If the application abnormally terminates before it issues 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. Also the job that uses 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 on are also part of this commit group. The commit or rollback applies to all of these rows and objects.

The OS/400 default is *NONE, which is not generally supported in the industry. This provides a 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 iSeries 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 iSeries 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.

Important: If you specify a commit value, you must perform a commit to make any changes (update, delete, insert, drop tables, etc.) permanent. If you close the Run SQL script window without specifying a commit, the system performs a rollback for all changes (except for DROP SCHEMA) that you made since you specified the commit value.

You can set the following decimal properties:

- *Maximum precision* indicates the maximum precision of decimal data that will be returned.
- *Maximum scale* indicates the maximum scale used in arithmetic calculations involving decimal data. This value must be smaller than the value of the maximum decimal precision.
- *Minimum divide scale* indicates the minimum scale used in arithmetic calculations that involve decimal data.

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

On the Other tab, you can 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 about 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 iSeries 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 schema qualifier, the system searches for the table within the default library and all libraries or schemas currently in library list. If you are using the SQL naming convention, the ANSI standard specification causes the system to ignore the library list and look *only* in the default library or schema.

For example, consider the case where the default library is SCHEMA01 and the user portion of the session's library list is in the order of SCHEMA02, followed by schema SCHEMA03. Also, consider that the unqualified table name is EMPLOYEE and it is stored in library SCHEMA02. Using the SQL naming convention, the system looks for EMPLOYEE only in schema SCHEMA01 and does not find it, which results in an error condition. Using the system naming convention, the system finds the table and runs the SQL statement successfully. It is successful because first it searches the default schema SCHEMA01 and then it searches in the SCHEMA02 and SCHEMA03 schemas.

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. Your requirements determine the settings.

To modify either data source, refer to the online help or consult the *iSeries Access for Windows* documentation in the iSeries Information Center at:

`http://www.iseries.ibm.com/infocenter`

When you reach this site, select **Programming →iSeries Access**.

7.1.2 Running a CL command under SQL script

In addition to running SQL statements under Run SQL Script, iSeries Navigator enables authorized users 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 7-6.

The command can be any OS/400 or user-defined command. In this example, we used the Display Object Description (DSPOBJD) command with its own set of parameters.

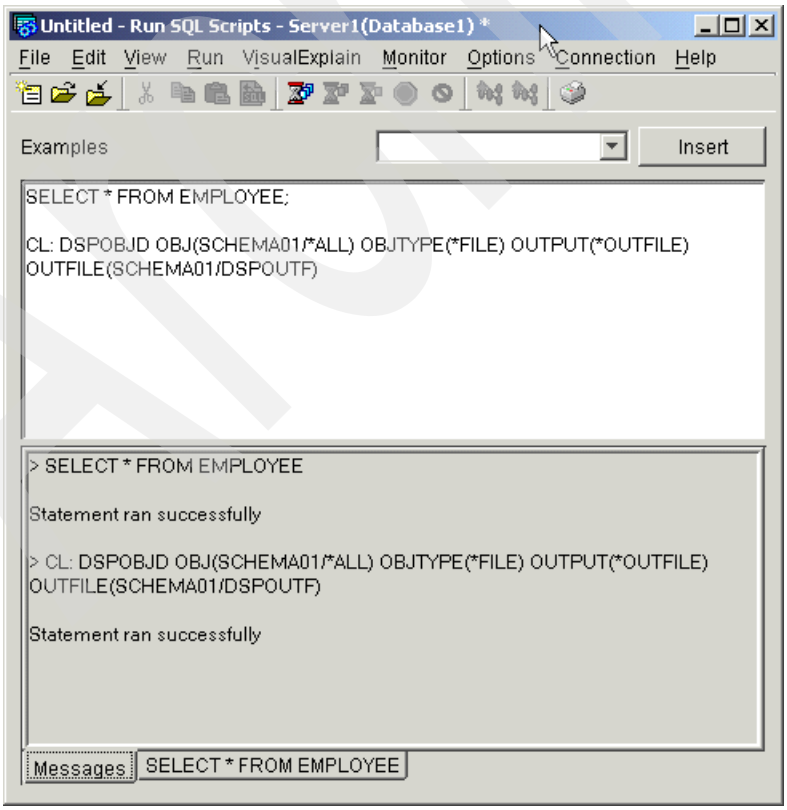


Figure 7-6 Run SQL Scripts: Running a CL command

You may also use much simpler OS/400 commands, such as:

- Adding a new library or schema to the current library list of the iSeries Navigator session using the CL command:

  ```
  CL: ADDLIBLE LIB(SCHEMA01);
  ```

- Sending a message to the system operator using the CL command:

  ```
  CL: SNDMSG MSG('This message is from an iSeries Navigator session from user USER01.')
  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 in which 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:

- The IBM-supplied SQL statement examples include some CL command examples at the end of the SQL statements.
- To make the OS/400 command works from an iSeries Navigator Run SQL Scripts session, ensure that you can find the objects, which are referenced in the command, in the iSeries Navigator session's (job's) library list or 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 iSeries Navigator Run SQL Scripts session's library list. Select **Connection →JDBC Setup** to amend the user part of the library list.

7.1.3 Run SQL Scripts Run options

Figure 7-7 shows the available Run options that are explained in this section.

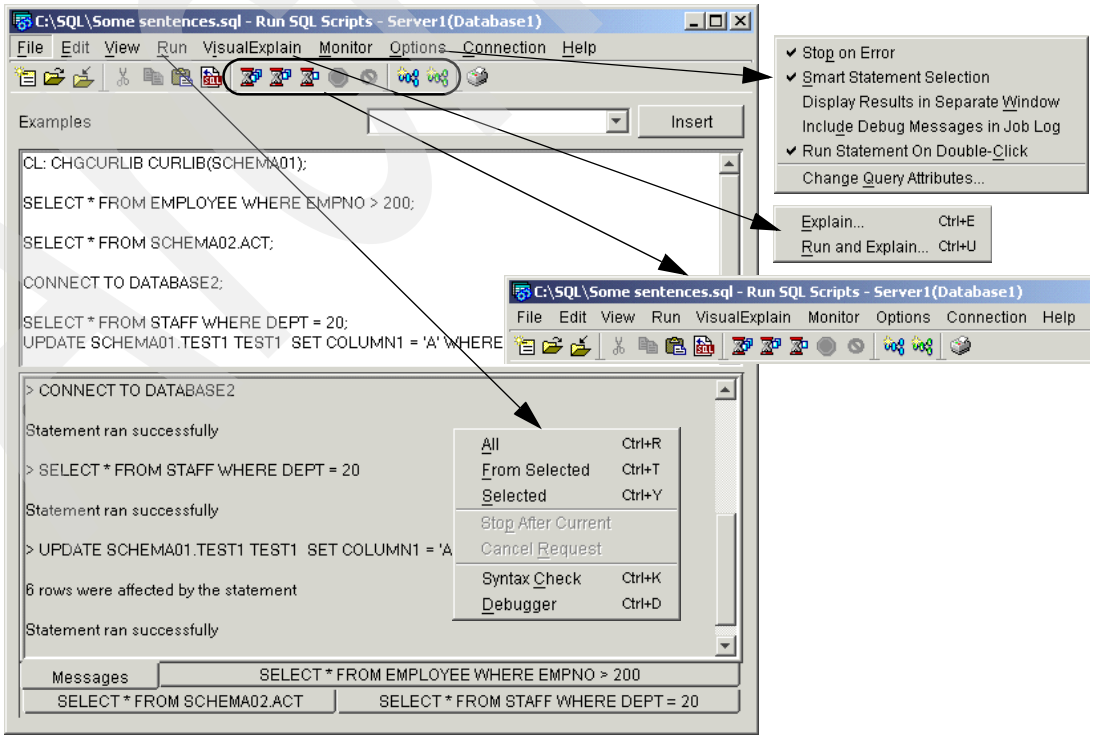


Figure 7-7 Run SQL Scripts: Run options

You can choose from two selection list types to run one or more SQL statements at a single time. From the Run SQL Scripts window, select Run from the menu bar or select one of the *hour glass* Run action icons from the toolbar (circled in Figure 7-7). Each has corresponding functions. Or select the Run option with a key sequence under the Run context menu.

You can pre-specify (defaults are provided) some controls over the Run function through the Options action in the menu bar. We discuss these controls in “Controlling SQL run options” on page 182 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 to run, for example:

```
select * from employee where empno > 200;
```

This is shown in the second line in the statement area in Figure 7-7. You can run only this statement by using *one* of the following actions:

- From the menu, click the **Run** →**Selected**.
- Click the **Run Selected** icon in the toolbar in Figure 7-7.
- Press Ctrl + Y from the workstation keyboard.

Note: If you select a group of statements, you can run them by using one of these actions.

Only the single statement runs. If it is a SELECT statement, the results are presented in the lower pane of the Run SQL Scripts window. If you want the results in a separate window, from the menu, select **Options** →**Display Results in a Separate Window**. The column names are presented as column headings. If you want to select only a subset of columns later, use these headings and displayed column data to help you select the appropriate columns. Figure 7-8 shows some column headings and associated data for the EMPLOYEE table.

SELECT * FROM EMPLOYEE WHERE EMPNO > 200 - Server1(Database1)

EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE	JOB
000210	WILLIAM	T	JONES	D11	0942	1979-04-11	DESIGNER
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29	DESIGNER
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21	CLERK
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05	CLERK
000250	DANIEL	S	SMITH	D21	0961	1969-10-30	CLERK
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11	CLERK
000270	MARIA	L	PEREZ	D21	9001	1980-09-30	CLERK
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24	OPERATOR
000290	JOHN	R	PARKER	E11	4502	1980-05-30	OPERATOR
000300	PHILIP	X	SMITH	E11	2095	1972-06-19	OPERATOR
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12	OPERATOR
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07	FILEREP
000330	WING		LEE	E21	2103	1976-02-23	FILEREP
000340	JASON	R	GOUNOT	E21	5698	1947-05-05	FILEREP
200010	DIAN	J	HEMMINGER	A00	3978	1965-01-01	SALESREP
200120	GREG		ORLANDO	A00	2167	1972-05-05	CLERK
200140	KIM	N	NATZ	C01	1793	1976-12-15	ANALYST
200170	KIYOSHI		YAMAMOTO	D11	2890	1978-09-15	DESIGNER
200220	REBA	K	JOHN	D11	0672	1968-08-29	DESIGNER
200240	ROBERT	M	MONTEVERDE	D21	3780	1979-12-05	CLERK
200280	EILEEN	R	SCHWARTZ	E11	8997	1967-03-24	OPERATOR
200310	MICHELLE	F	SPRINGER	E11	3332	1964-09-12	OPERATOR
200330	HELENA		WONG	E21	2103	1976-02-23	FIELDREP

Figure 7-8 Run SQL Scripts: Sample SQL SELECT output

Running a set of SQL statements

You can run a set of SQL statements that you have in the Run SQL Scripts window. Using the example in Figure 7-7 on page 179, you can run `SELECT * FROM SCHEMA02.ACT;` (third line) through the last statement.

You do this by placing the active screen cursor within the SQL statement text and performing *one* of the following actions:

- From the menu, click the **Run →From Selected**.
- Click the **From Selected** icon (the middle hour glass) in Figure 7-7.
- Press Ctrl + T from the workstation keyboard.

This runs each statement sequentially, beginning with:

```
SELECT * FROM SCHEMA02.ACT;
```

In the example, there are two SELECT statements. For each SELECT statement, a window of data is presented. However, if the SELECT statements are fast enough, you may notice only the last SELECT output.

The two windows are active on the screen. You can view them by selecting the appropriate task from the windows task bar.

If an error occurs and a Stop on error option is selected (as specified under the Options menu, 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. For example, you can run `CL: CHGCURLIB CURLIB(SCHEMA01);` through `UPDATE SCHEMA01.TEST01 ... ;`.

You run all the SQL statements by doing *one* of the following tasks:

- Select **Run →All**.
- Click the **All** icon (left hour glass) in Figure 7-7.
- Press Ctrl + R on the workstation keyboard.

If an error occurs and a Stop on error option is selected (as specified under the Options menu, the program stops, and the statement where the error occurred remains selected.

SQL statement syntax check

Using the Syntax Check option (Run →Syntax Check in Figure 7-7), you can validate a selected SQL statement 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. You can also start this option by selecting an SQL statement and pressing Ctrl + K.

Linking to the Visual Explain component

The two icons under the Options menu in Figure 7-7) provide access to the Visual Explain function, as do the two new menu items under Visual Explain. For more information, refer to Chapter 8, “Visual Explain” on page 213.

Using the *Explain* option (the left icon under the Options menu), or pressing Ctrl + E, allows you to review the Optimizer access plan that is 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 about query attributes, see 7.2, “Change Query Attributes” on page 198. It produces a

visual explanation of the statement but does not access the actual data from the database, and, therefore, avoids the unnecessary I/O load.

Using the *Run and Explain* option (the right icon under the Options menu), or pressing 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 run before the analysis graph is reported.

Controlling SQL run options

By selecting the Options menu on the Run SQL Scripts window (Figure 7-7), you can control what to do if an SQL error occurs. You can also determine the levels of additional information to include 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 7.1.1, “ODBC and JDBC connection” on page 173, for additional information about commitment control. The most complete OS/400 documentation on commitment control is in *Backup and Recovery*, SC41-5304.
- **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 to take to resolve the problem. You may also consider using the iSeries Navigator interface to the SQL Performance Monitor, which is described in 7.4, “SQL Performance Monitors” on page 201, and Visual Explain, described in Chapter 8, “Visual Explain” on page 213.

Figure 7-9 shows an example of an SQL JOIN statement. In this example we use the SQL SELECT with JOIN statement to show the associated job log debug messages issued by the query optimizer.

To see the current iSeries Navigator session's job log, in the Run SQL Scripts window, click **View** →**Job Log**.

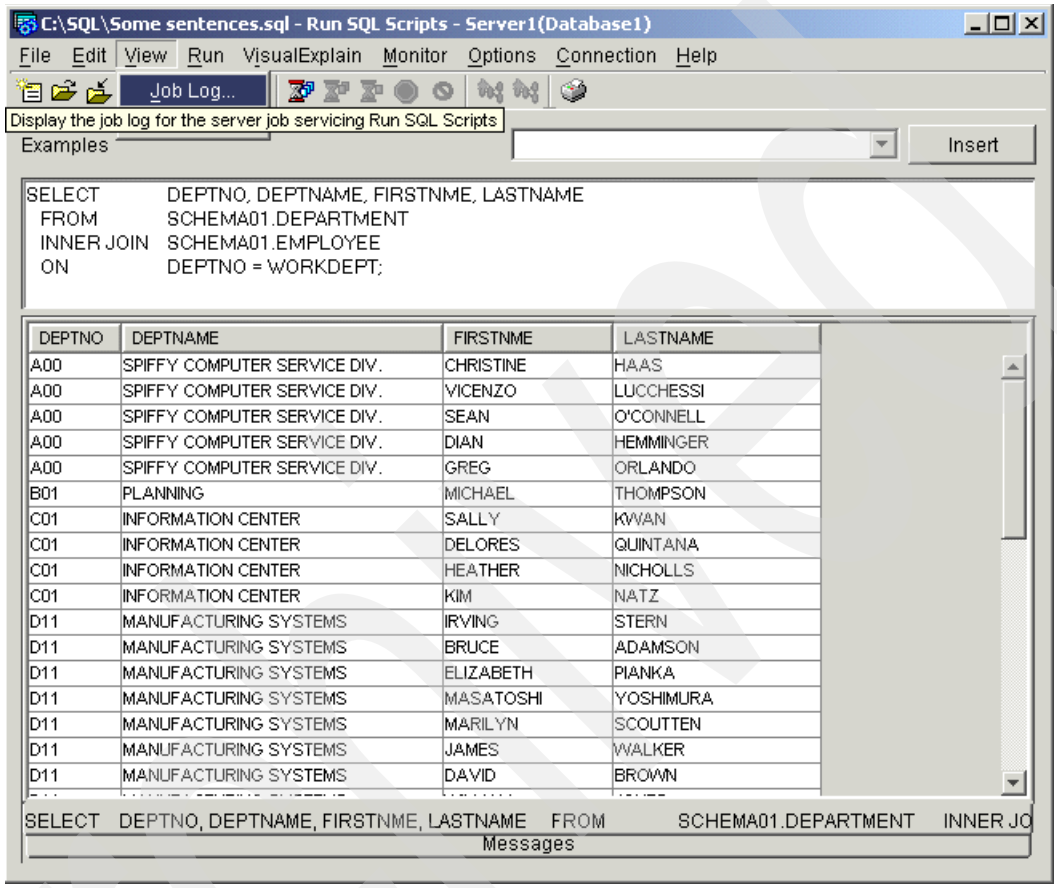


Figure 7-9 Run SQL Scripts: Selecting to view the job log

The Job Log window appears as shown in Figure 7-10. For this job log, we discuss two messages:

- The optimizer's suggestion for an access path (index) to file EMPLOYEE with message ID CPI432F (highlighted in Figure 7-10)
- Error message CPI433A (circled in Figure 7-10)

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. You can set up this table to specify query attributes that the OS/400 query optimizer uses while processing each SQL statement.

If you do not attempt to modify the default OS/400 query processing algorithm through this table, the table is not in the QUSRSYS library, and this message is considered for information only.

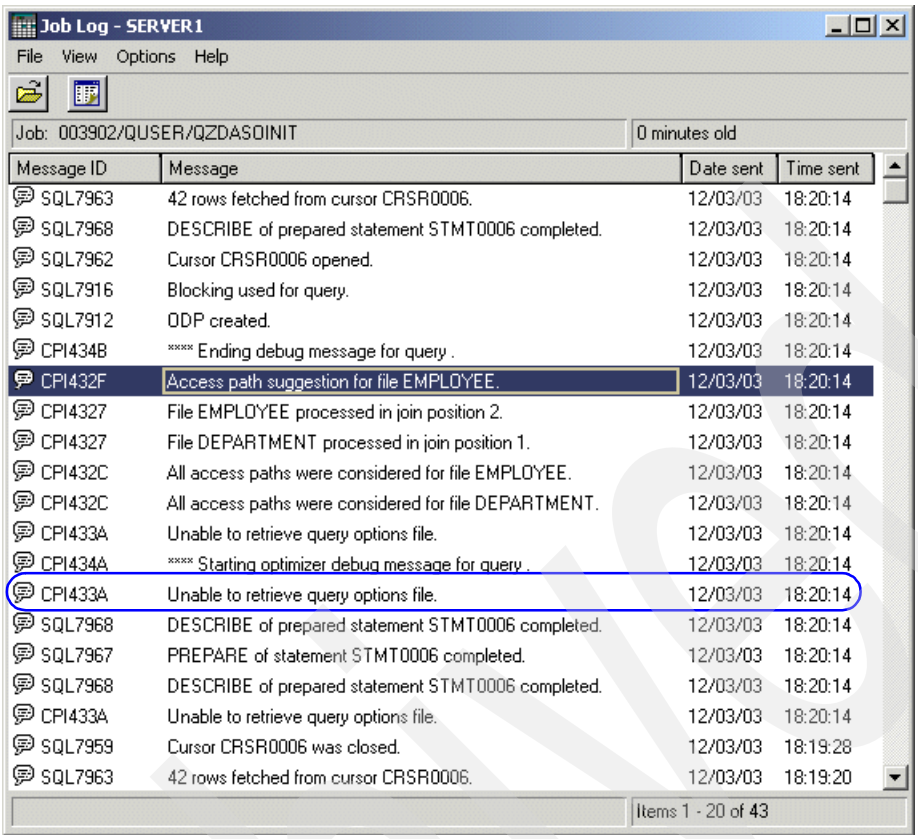


Figure 7-10 Job log for Include Debug Messages

- **Run Statement On Double-Click**: When you turn on this option, you can double-click an SQL statement to run it.
- **Change Query Attributes**: This option enables you to easily modify the query options file QAQQINI for your job, provided that 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 7.2, “Change Query Attributes” on page 198.

7.1.4 SQL Assist

SQL Assist is a graphical user tool that provides a prompted interface to help you to build interactively SQL SELECT, INSERT, UPDATE, or DELETE statements. Details about the structure of each statement is beyond the scope of this redbook. For complete information about these statements and their structure, refer to *DB2 Universal Database for iSeries SQL Reference* in the iSeries Information Center at:

`http://www.iseries.ibm.com/infocenter`

When you reach the Information Center, select **Database** →**Reference** →**SQL Reference**.

All statements created using SQL Assist are sent to the statement input area in Run SQL Scripts window, where you can run or save them. Using SQL Assist, you can edit the statements created in SQL Assist or the statements that were opened, generated, or copied and pasted into the statement input area in Run SQL Scripts.

To do this in the Run SQL Scripts window, select the statement that you want edit and select **Edit →SQL Assist**.

SQL Assist is available from the Run SQL Assist window. To access and use this SQL Assist, follow these steps:

1. In iSeries Navigator, expand the system that you want to use.
2. Expand **Databases** and click one of the databases.
3. In the task pad, click **Run SQL Script**.
4. In the Run SQL Script window (Figure 7-11), click **Edit →SQL Assist**.

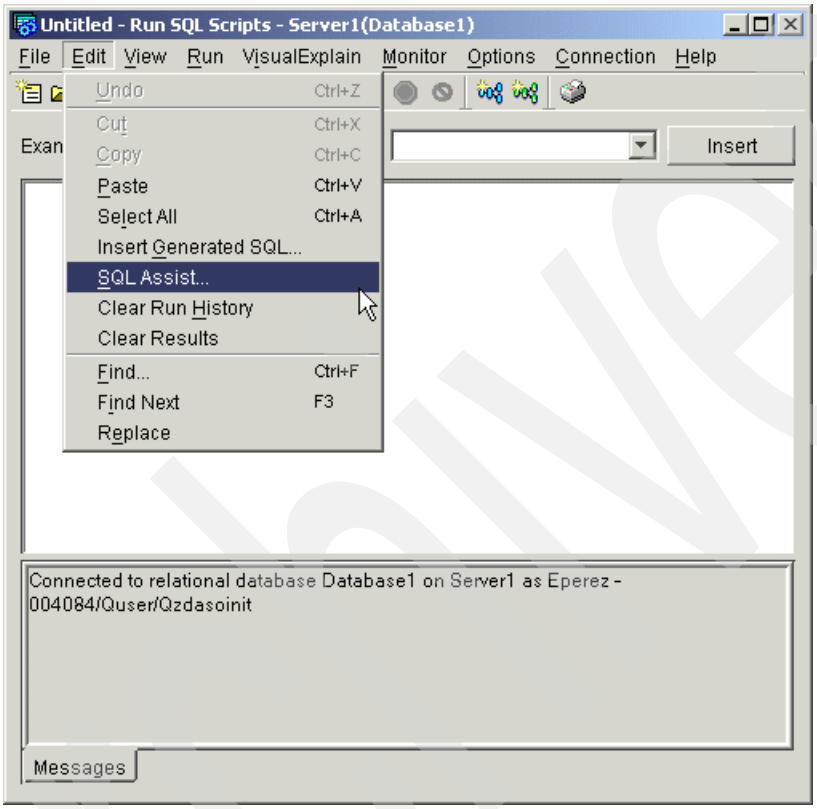


Figure 7-11 Opening SQL Assist

The SQL Assist main window (Figure 7-12) contains three areas:

- Outline view

  This contains a high-level representation of the current SQL statement. You can visually examine an outline of the SQL statement and navigate through the steps of building the SQL statement. Select a node in the Outline view to view details for this element in the Details area.

- Details area

  Use the Details area to add elements to the SQL statement. The Details area changes based on the node that you select in the Outline view. When you select a node in the Outline view and make changes in the Details area, the SQL code is generated in the SQL code view.

- SQL code view

  The SQL code view contains the SQL code that is generated based on the contents of the Outline view and changes that you made in the Details area. The code is syntax-highlighted.

After you open the SQL Assist window, you can change the SQL statement type and create SELECT, INSERT, UPDATE, or DELETE statements.

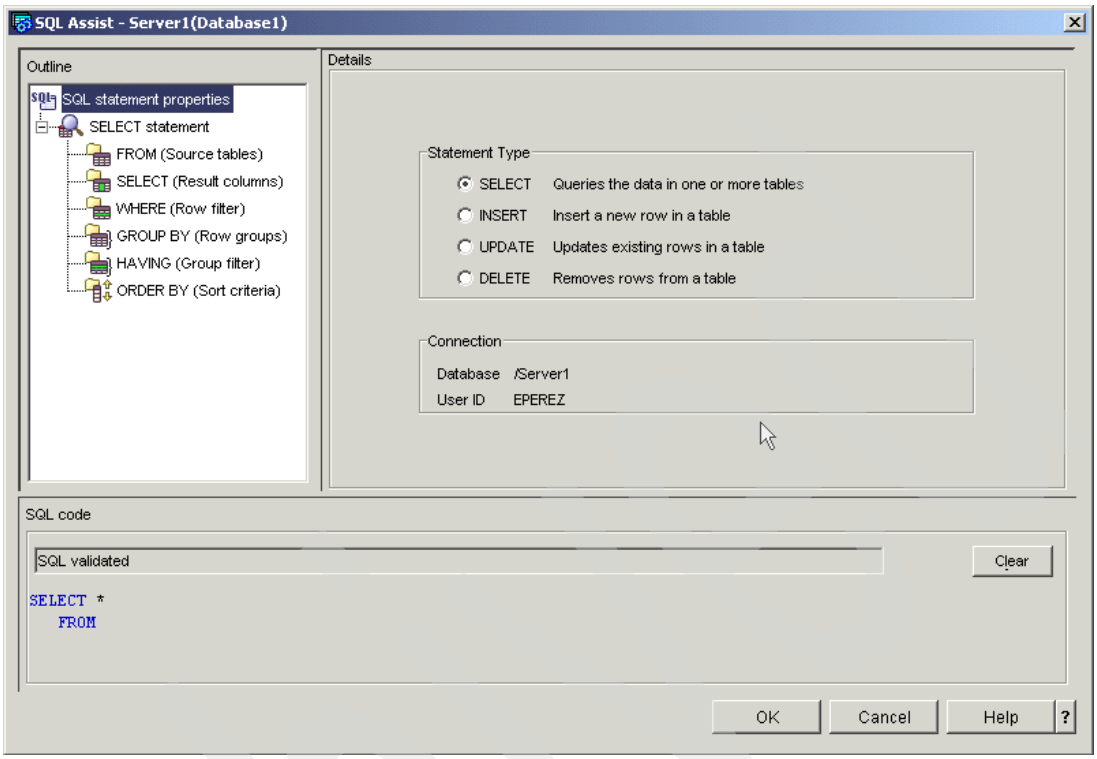


Figure 7-12 SQL Assist window

Changing the SQL statement type

At any time, you can change the type of SQL statement that you are creating using SQL Assist. When you change the type of statement, a message window opens to confirm that you want to discard all changes and start a new statement.

Follow these steps to select the SQL statement type:

1. In the Outline view, click the main **SQL statement properties** node.
2. In the Details area, select the type of SQL statement that you want to create.
3. Click **OK** to close the message window if you are sure that you want to discard all changes. A new template appears in the Outline view.

Creating a SELECT statement

Follow these steps to create a SELECT statement. We use Figure 7-13 to explain some of these steps:

1. In the Outline view, click the main **SQL Statement Properties** node.
2. In the Details area, click **SELECT**.
3. A SELECT statement outline appears in the Outline view. Select tables for the SELECT statement (FROM clause) following these steps:
   a. In the Outline view, click the **FROM (Source tables)** node.
   b. In the Details area, the list of schemas appears that was specified in the Library list parameter in JDBC setup. Expand the schemas and select the table. Press the CTRL key to select more than one table at the same time.
   c. Click the **>** button to add selected tables to the Selected tables list. You can also add a table to the selected table list by double-clicking it in the available tables list. Optionally, under Selected source tables, edit the alias (correlation name) for the table in the Name field.

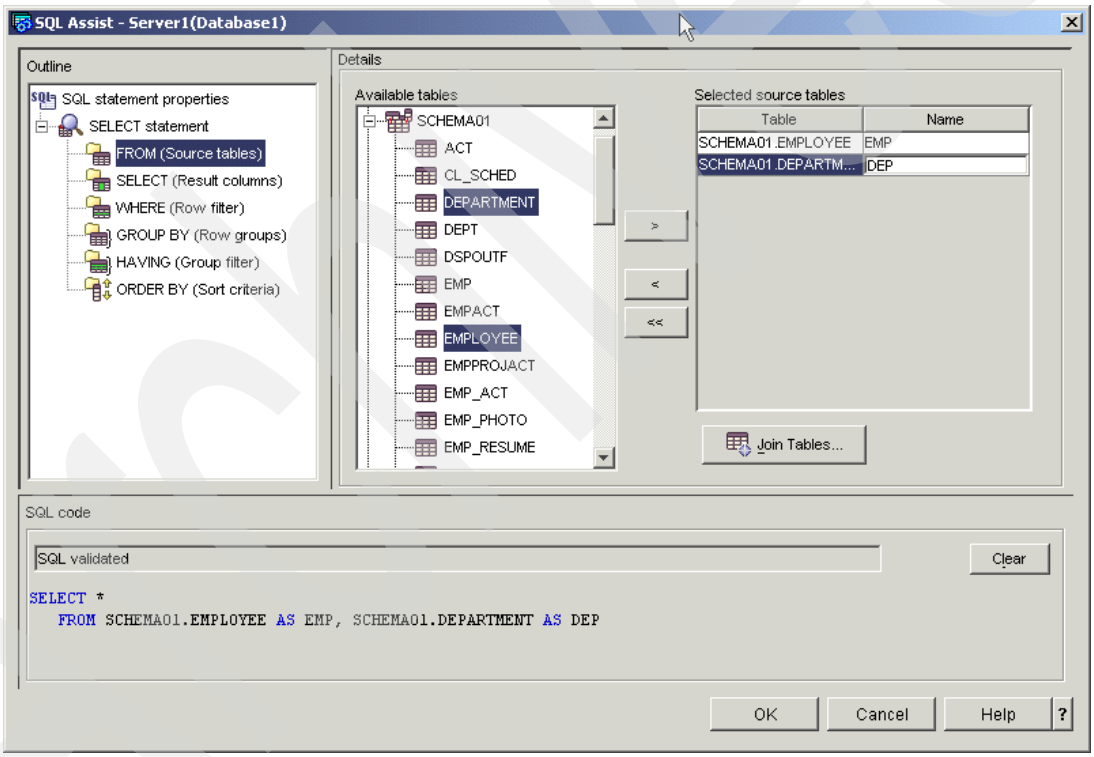


Figure 7-13 Creating an SQL SELECT statement: Selecting the tables

4. If you select more than one table you can join them. Use the Join Tables window (Figure 7-14) to join tables in a SELECT statement. The Join Tables window helps you to perform joins on two or more selected tables. Tables that you selected in the FROM (Source tables) node are listed, and tables that can be joined together are indicated. You can specify join types or select from suggested join types. You can view details for each table including column names, data types, primary keys, and whether columns are nullable.

   To join tables in a SELECT statement, in the Details area for the FROM (Source tables) node, click **Join Tables** button.

5. The Join tables window (Figure 7-14) opens. If no primary/foreign key relationships exist between the tables that you selected in the FROM (Source tables) node, a message appears. It informs you that no join conditions are suggested in the Join Tables window, and some of the features are not available.

   a. In the Joins area (top pane in Figure 7-14), select a table. If any of the other tables in the Joins area are potential candidates for a join with this table, an icon next to these tables changes (see the Department icon in the top pane in Figure 7-14).

   b. You can keep the **Suggest join conditions when joining** check box selected to activate a function that suggests potential join paths when you join tables together.

   c. Hold down the Shift key or the Ctrl key and select another table.

   d. Click the **Join** button.

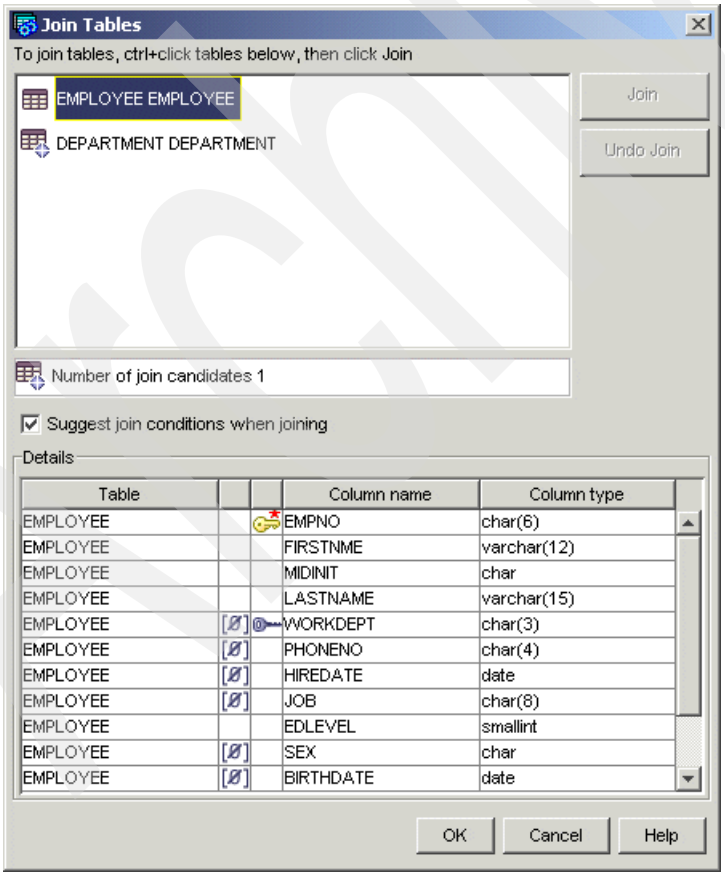


Figure 7-14 Creating an SQL SELECT statement: Joining tables (Part 1 of 2)

   e. If you selected the Suggest join conditions when joining check box and one or more join paths exist between the two tables, a window opens where you can select from all

possible join paths for the two tables (inset in Figure 7-15). In the Suggest Joins window, select a join path and click **OK**.

> **Note:** If you cleared the Suggest join conditions when joining check box and no join paths exist between the two tables, a warning window opens to inform you that no join paths exist. Click **OK** to close the warning window. In the Details area, you can view and edit the join conditions and join type. You must specify valid join conditions before you can save changes in the Join Tables window.

f. In the Joins area (top pane in Figure 7-15), a join appears with an icon next to it.

g. In the Details area, you can view and edit the join conditions and join type. You can change the join type. Either select a join from the Join Type list, or click the **Join Types** button to open a window where the different join types are displayed and described in detail.

h. To edit the join conditions in the Join conditions area, select a column from the First Column list. Then select an operator from the Operator column and the column from the Second Column list.

   Optionally you can click the **Suggest** button to open a window where you can select from a list of valid join paths.

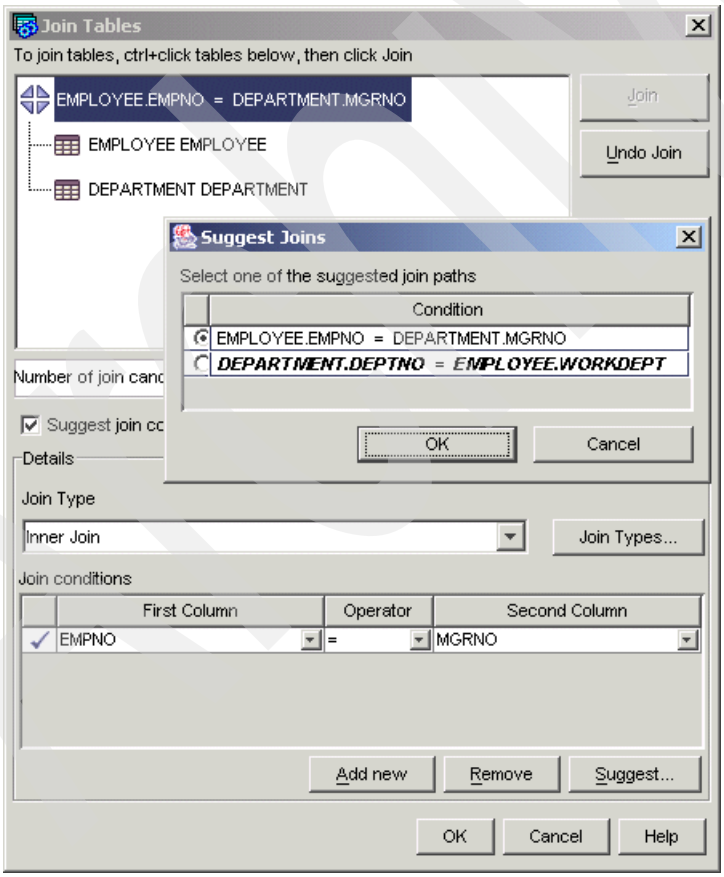


Figure 7-15 Creating an SQL SELECT statement: Joining tables (Part 2 of 2)

i. You can add another table to the join by selecting a join in the Joins area and following steps c on page 188 through h. Additionally you can combine two joins by selecting two

tables that are already joined in the Joins area and following steps d through h. Figure 7-16 shows the two joins combined.

j. Click **OK**. All valid joins are added to the SQL code view.

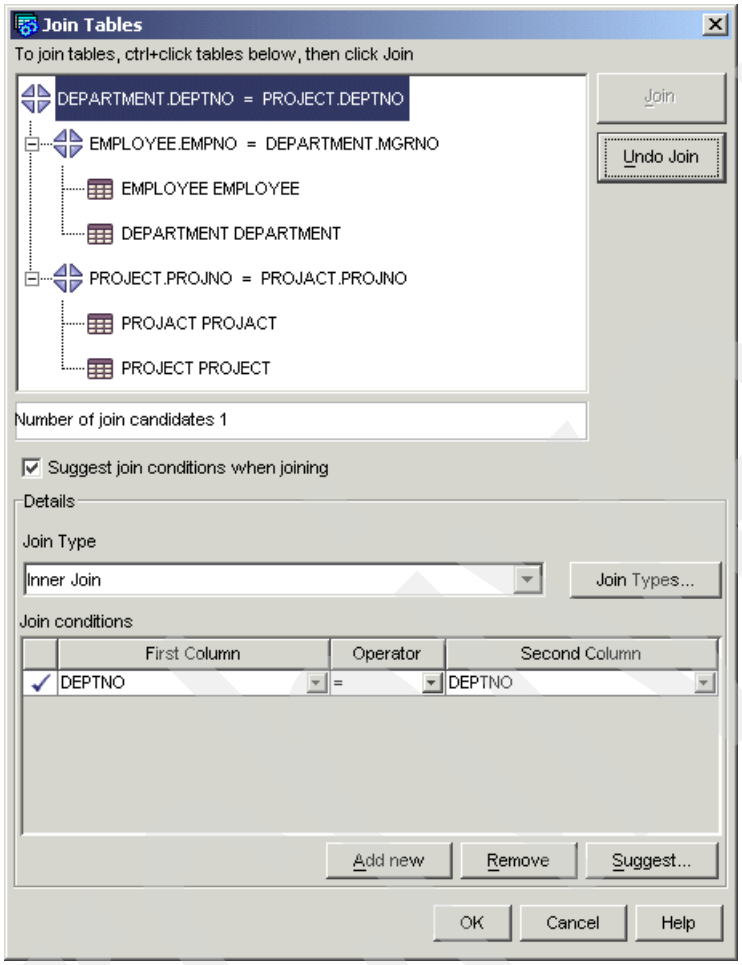


Figure 7-16 Creating an SQL SELECT statement: Joins combined

6. Select the individual result columns, If you do not select columns for the SELECT statement, all columns are returned by default.

   To select result columns for a SELECT statement, follow these steps:

   a. In the Outline view in Figure 7-17, click the **SELECT (Result columns)** node.

   b. A list of tables that you selected in the FROM (Source tables) node is displayed in the Details area. Click a table node in the Available columns list to see the columns in that table.

   c. Select columns from the Available columns list.

   d. Click the **>** button to move the selected columns to the Selected columns list.

      You can click the **>>** button to add all columns in the Available columns list to the Result columns list. Optionally you can click the **<<** button to remove all columns in the Result columns list.

   e. Select to remove duplicate rows (click the **SELECT DISTINCT** check box) if you want to exclude duplicate rows from the result set.

f. To enter a new column name in the Name field next to a selected column, then press Enter to add the new name to the SQL code in the SQL code view.

g. You can click the **...** button next to a selected column to open a window where you can create an expression.

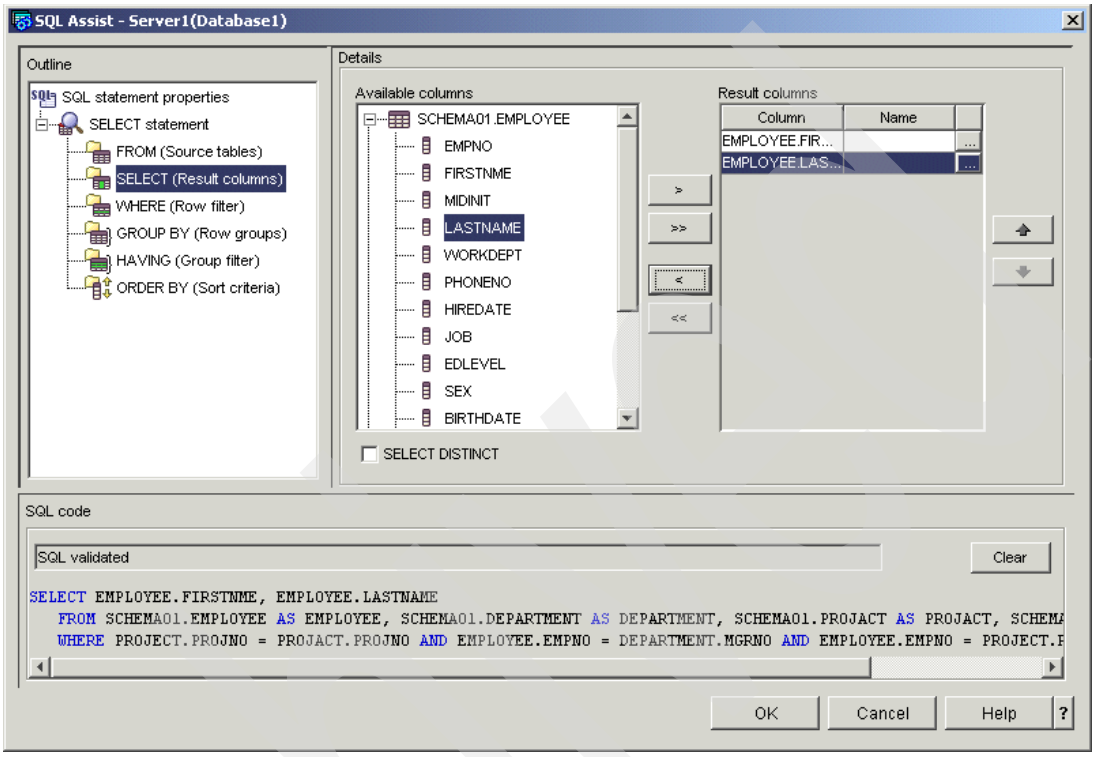


Figure 7-17 Creating an SQL SELECT statement: Selecting columns

7. Specify a search condition. The result of a search condition is derived by application of the specified logical operators (AND, OR, NOT) to the result of each specified predicate. If logical operators are not specified, the result of the search condition is the result of the specified individual predicate. To add search conditions to an SQL statement in the Details area for the WHERE (Row filter) node (Figure 7-18), complete the following steps:

   a. In the Column list, select a column name.

   b. Select an operator from the Operator list. The available options in the Details area change based on which operator you choose. For example, when you select the operator NOT IN, in the Value table, you can:

      - Type a value.
      - Type a host variable name.
      - Click the (**...**) button to open a window if you want to create an expression.
      - Click the third icon on the right, under the Value table, to select from a list of unique value for the selected column.

   c. Click the **>** button to move the completed predicate to the Search Condition area. The predicate is also added to the SQL statement in the SQL code view. The **>** button only adds a predicate to the Search condition area when the Search condition area is empty. If the Search condition area contains existing predicates, clicking the **>** button overwrites any selected predicate in the Search condition area.

   d. Insert additional predicates with AND or OR logical operators by clicking the **AND >** or **OR >** button. The predicate is inserted below the selected item in the Search condition area. The predicate is also added to the SQL statement in the SQL code view.

If you want to modify a predicate in the Search condition area, follow these steps:

a. In the Search condition area, select the predicate. The Predicate area changes to reflect the content of the selected predicate.

b. In the Predicate area, adjust the Column, Operator, or Value settings.

c. Click the **>** button to replace the selected predicate in the Search condition area with the new values in the Predicate area.

d. In addition, you can add parentheses, remove predicates, or remove groups of predicates with the fields and controls below the Search condition area. Changes are added to the SQL statement in the SQL code view.

> **Note:** If a single predicate is selected, the NOT logical operator check box is displayed below the Search condition area. This check box is kept in sync with the NOT logical operator check box above the Predicate area.

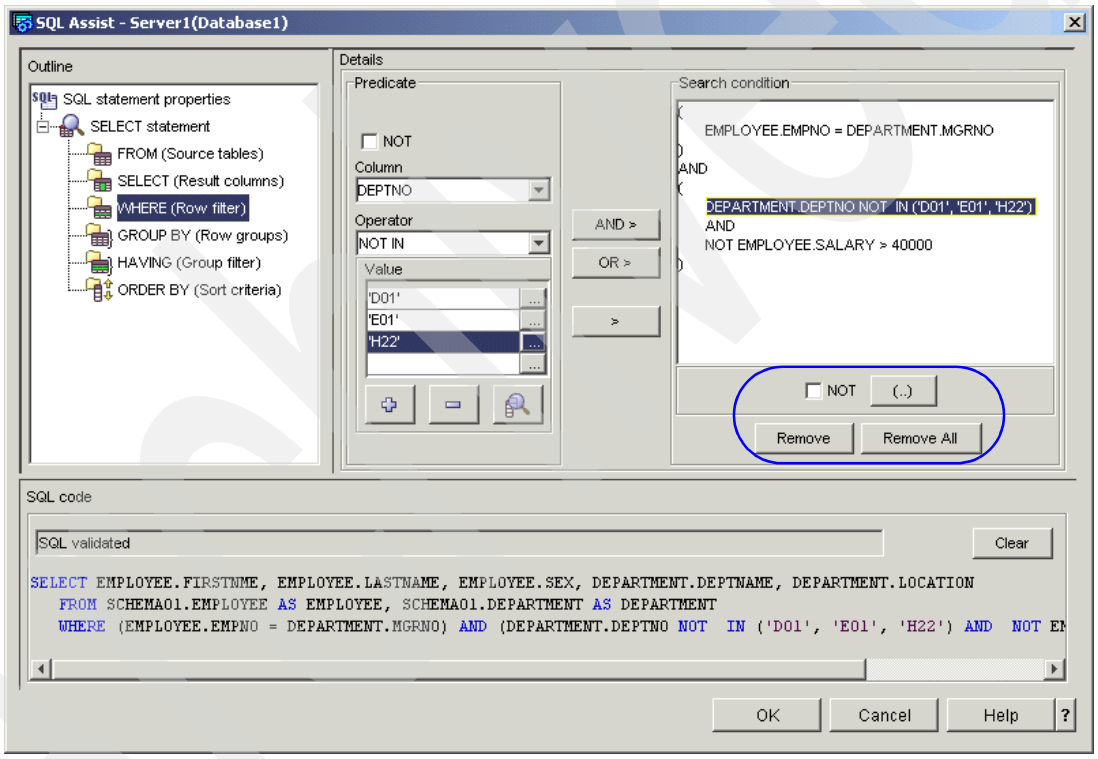


Figure 7-18 Creating an SQL SELECT statement: Adding search conditions

8. Use the GROUP BY clause to specify that rows retrieved by the WHERE clause should be grouped and condensed in such a way that all rows having the same values for the grouping columns will be treated as a single row in the result set.

   SQL Assist automatically adds all columns in the SELECT clause (other than column function columns) to the grouping column list if there are any function columns in the SELECT clause.

   Follow these steps to group rows in the result set:

   a. In the Outline view, click the **GROUP BY** node.

   b. A list of available columns is displayed in the Details area. In the Details area, from the Available columns list, select the grouping columns.

c. Click the **>** button to move the selected columns to the Selected columns list. The grouping columns are added to the SQL statement.

9. You can use the HAVING clause to add search conditions to an SQL statement. You should select grouping columns before you add a search condition in the HAVING clause. Follow the same steps specified to add a WHERE clause.

10. The ORDER BY clause specifies an ordering of the rows of the result table. If a single sort specification (one sort key with associated direction) is identified, the rows are ordered by the values of that sort specification. If more than one sort specification is identified, the rows are ordered by the values of the first identified sort specification, then by the values of the second identified sort specification, and so on.

    Follow these steps to order rows in the result set:

    a. In the Outline view as shown in Figure 7-19, click the **ORDER BY (Sort criteria)** node. A list of available columns is displayed in the Details area.

    b. You can select the **Show all columns** check box to show columns from all tables in the SQL statement in the list of available columns. Or you can select **Show result columns only** to see only the result columns in the list of available columns.

    c. Under the Available columns list, select the columns.

    d. Click the **>** button to move the selected columns to the Selected columns list. The selected columns are added to the SQL statement in the SQL code view.

    e. Click **ASC** or **DESC** in the field next to a column in the Selected columns list to specify whether to order the columns in ascending or descending order.

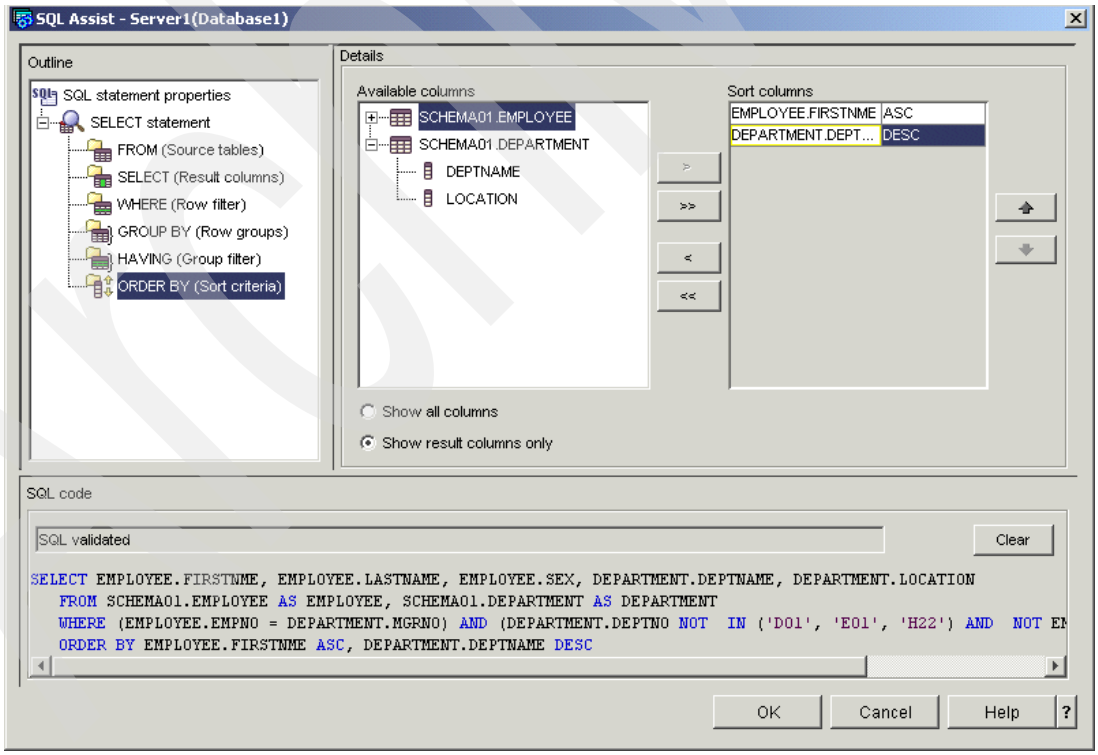


Figure 7-19 Creating an SQL SELECT statement: Grouping columns

11. Click **OK**. The SQL Assist window closes, and the generated SQL code is sent to the statement input area in Run SQL Scripts window.

Creating an INSERT statement

You use an INSERT statement to insert a single row of data into an existing database table. To create an INSERT statement follow these steps:

1. In the Outline view (Figure 7-20), click the main **SQL statement properties** node.
2. In the Details area, select **INSERT**.
3. An INSERT statement outline is displayed in the Outline view. Click the **INSERT INTO (Target table)** node.
4. A list of available schemas is displayed in the Details area. Expand the schema that you want, select a table to insert data into from the Available tables list and click the **>** button to move the selected table to the Selected table list.
5. To specify data to insert into the table selected, in the Outline view, click the **VALUES (Row values)** node.

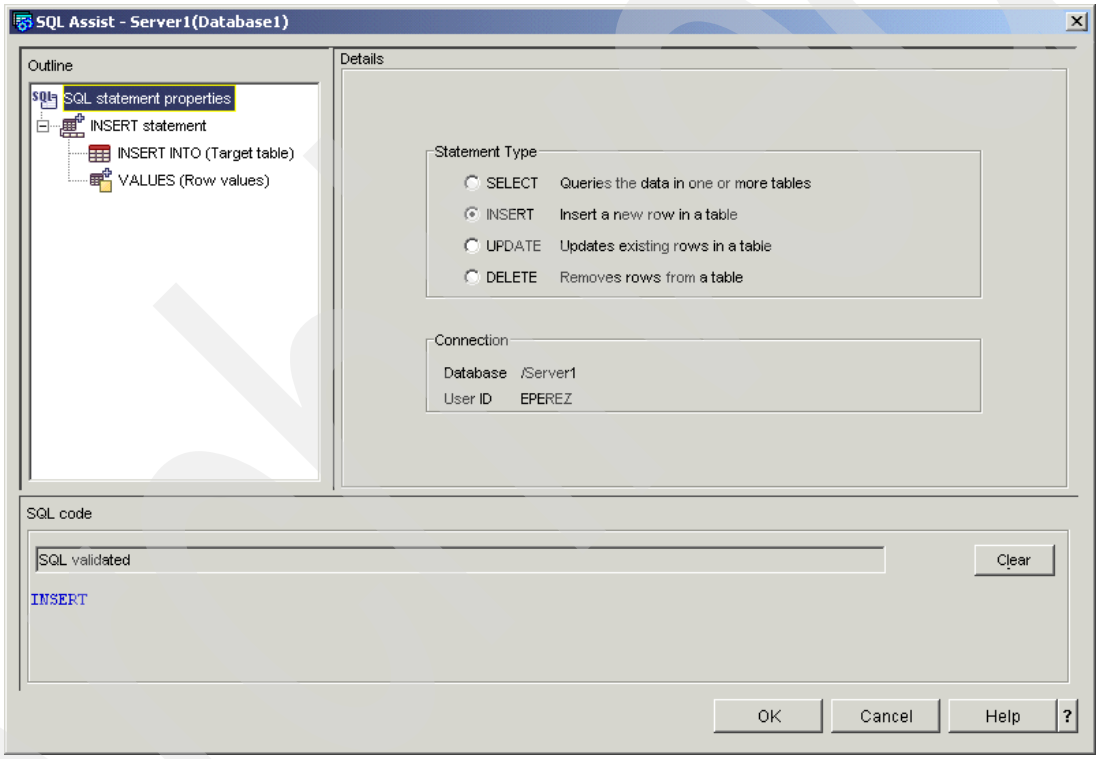


Figure 7-20 SQL Assist: Creating a INSERT statement

6. A list of columns in the table that you selected in the previous step is displayed in the Details area (Figure 7-21). The names and data types of the columns are also listed. In the Value column, specify the values to insert. In the Value column, you can:
   - Type a value.
   - Click **NULL** or **DEFAULT**.
   - Click **Expression** to open a window where you can create an expression.
   - Click **List values** to open a window where you can view a list of existing values for the selected column.
7. Click **OK**. The SQL Assist window closes, and the generated SQL code is sent to input area in Run SQL Scripts window.

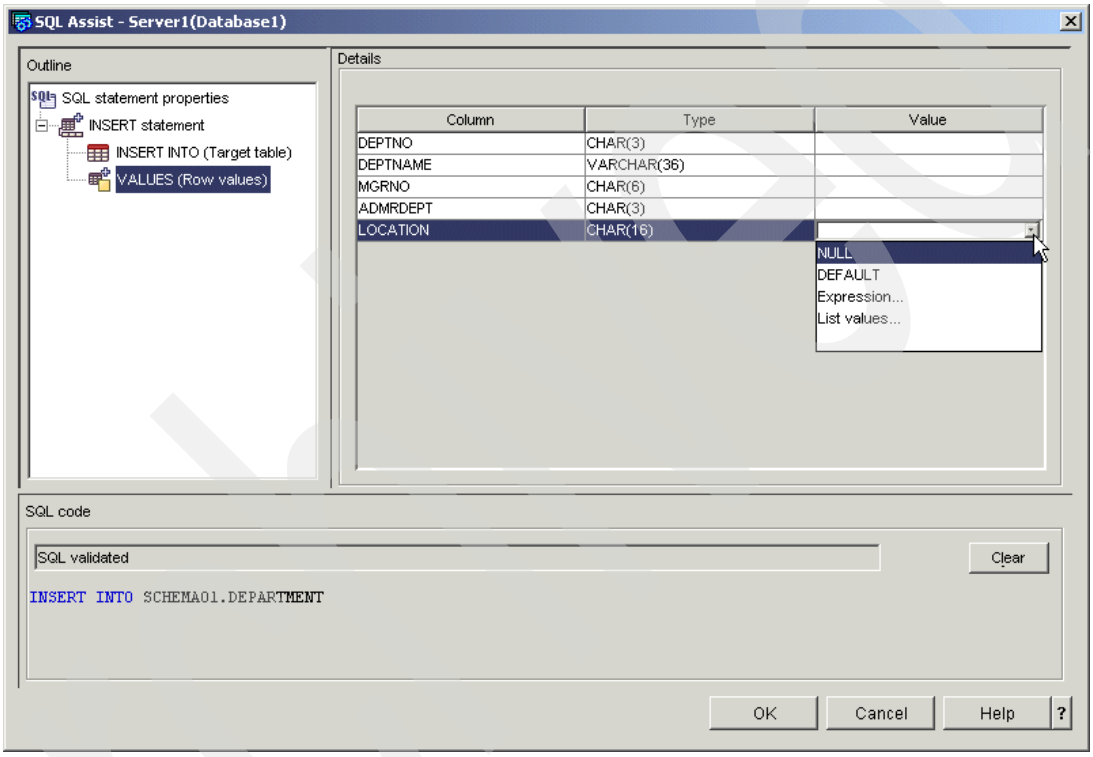


Figure 7-21 Creating an SQL INSERT statement: Adding values to columns

Creating an UPDATE statement

If you want update data in an existing table, you can use an UPDATE statement. To create an UPDATE statement using SQL Assist, follow these steps:

1. In the Outline view (Figure 7-22), click the main **SQL statement properties** node.
2. In the Details area, click **UPDATE**.
3. An UPDATE statement outline is displayed in the Outline view. Click the **UPDATE (Target table)** node.
4. A list of available schemas is displayed in the Details area. Expand the schema that you want, select a table to update from the **Available tables** list, and click the **>** button to move the selected table to the Selected table list.

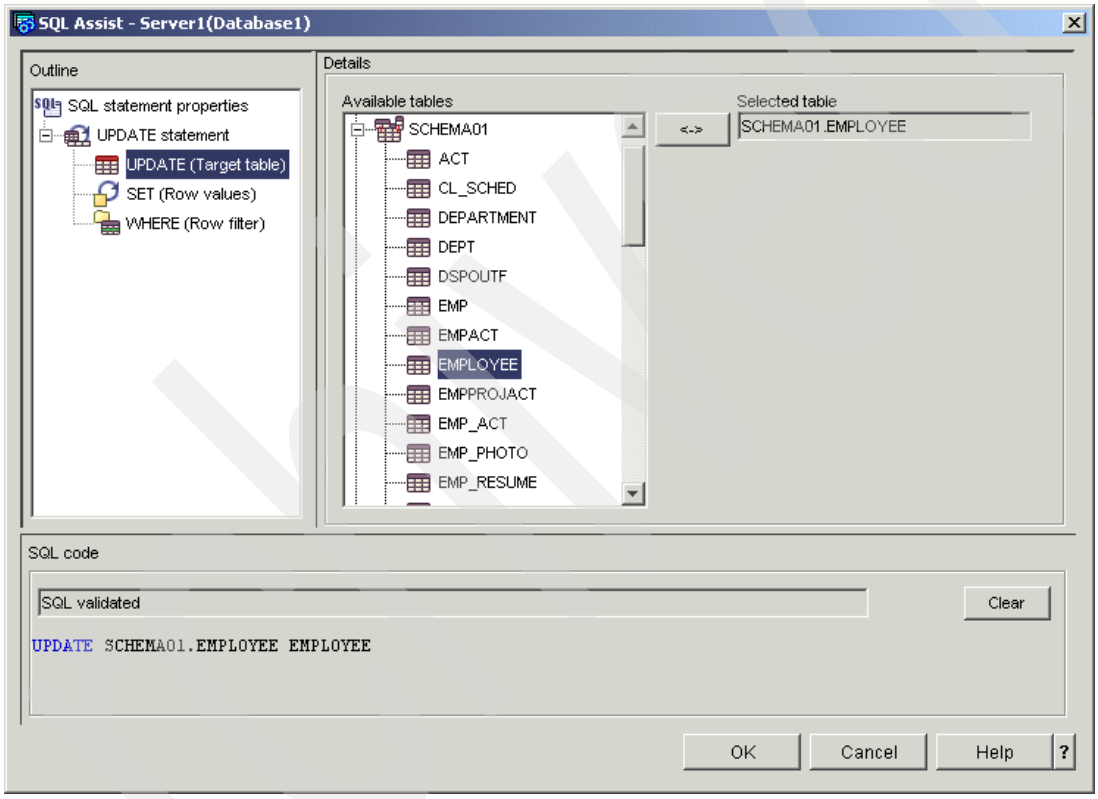


Figure 7-22 Creating an SQL UPDATE statement: Selecting the tables

5. To specify the data to update in a table, click the **SET (Row values)** node.
6. A list of columns in the table that you selected is displayed in the Details area. The names and data types of the columns are listed. In the Value column, specify update values. You can specify values in the same way that you insert values in the INSERT statement as shown in Figure 7-23.
7. Add search conditions to the UPDATE statement. These steps are the same as those for the SELECT statement (refers to step 7 on page 191).
8. Click **OK**. The SQL Assist window closes, and the generated SQL code is sent to input area in Run SQL Scripts window.

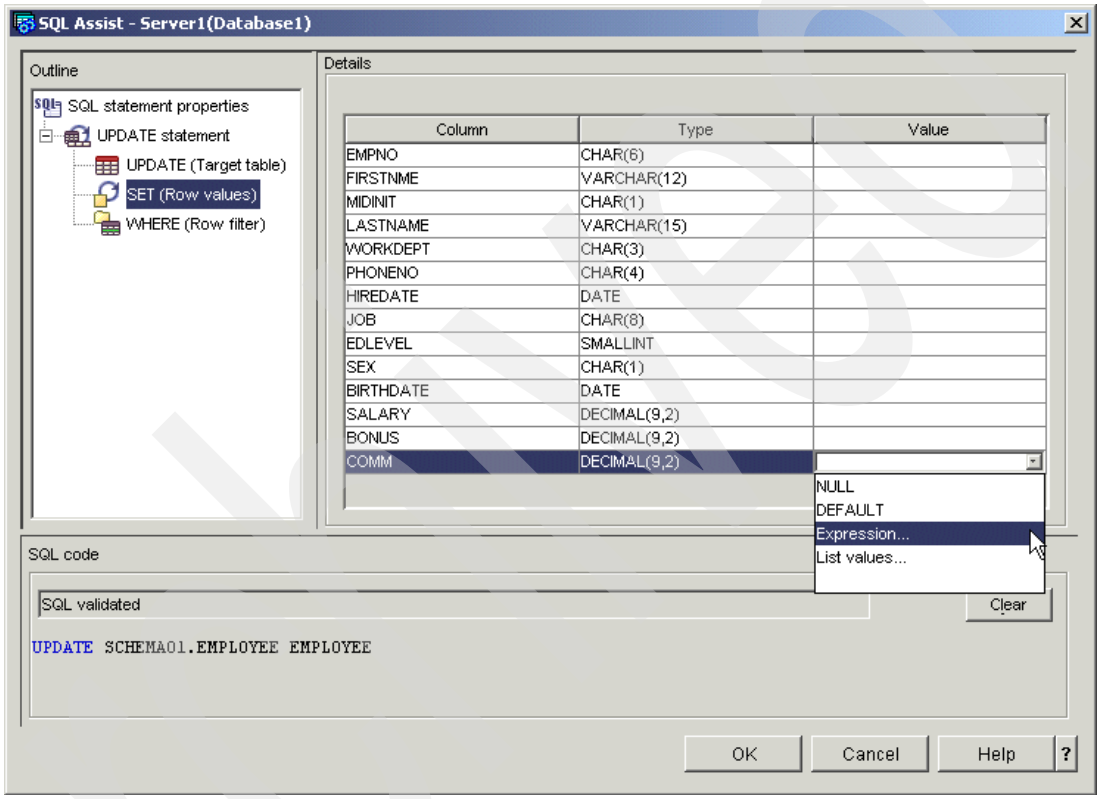


Figure 7-23 Creating an SQL UPDATE statement: Setting values to update

Creating a DELETE statement

Use a DELETE statement to specify criteria for deleting rows of data in an existing table. Follow these steps to create a DELETE statement using SQL Assist:

1. In the Outline view, click the main **SQL Statement Properties** node.
2. In the Details area, click **DELETE**.
3. A DELETE statement outline appears in the Outline view. Click the **DELETE FROM (Target table)** node.
4. A list of available schemas is displayed in the Details area. Expand the schema that you want and select the table to delete data from.
5. Use the **>** button to move the table to the Selected table list.
6. Add search conditions to the UPDATE statement. These steps are the same as those use for the SELECT statement (see step 7 on page 191).

7. Click **OK**. The SQL Assist window closes, and the generated SQL code is sent to input area in the Run SQL Scripts window.

7.2 Change Query Attributes

The Change Query Attributes window enables you to work with performance and status information for the active jobs in the system. The Change Query Attributes item gives you an easy way to change your query options to access the database.

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 about the implications of changing query attributes, refer to *Query performance and query optimization* in the iSeries Information Center on the Web at:

http://www.iseries.ibm.com/infocenter

When you reach this site, select **Database** →**Performance and optimization**.

You can access the Change Query Attributes panel in two ways:

- From iSeries Navigator, right-click in any database and select **Change Query Attributes**.
- From the Run SQL Scripts Center, select **Options** →**Change Query Attributes**.

You see the Change Query Attributes window as shown in Figure 7-24.

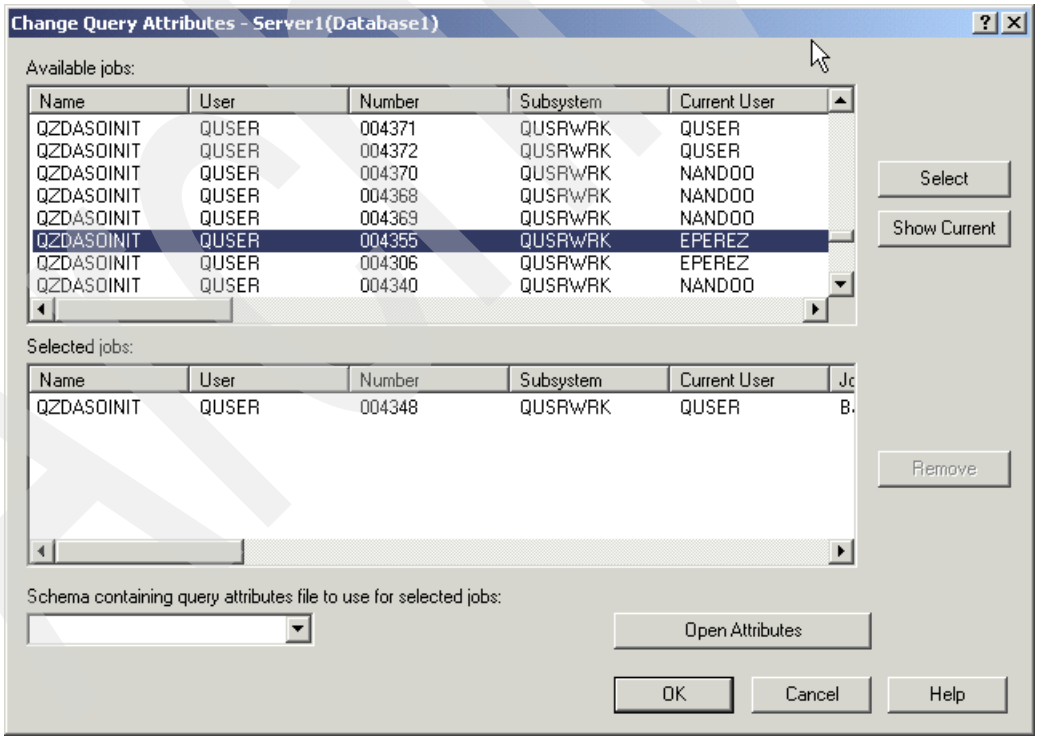


Figure 7-24 Change Query Attributes window

Proceed with the following steps:

1. In the Available jobs section in the upper part of the window, you see a list of all jobs that are currently active in the system. Scroll through the list to locate the job you are

interested in, click its name, and click the **Select** button to move the selection to the Selected jobs section in the bottom part of the window. You can select more than one job and set common query attributes for all of them at the same time. If you want to find the current job, click the **Show Current** button.

2. At this point, you can specify a library or schema in which you want the original QAQQINI file to be copied. You specify this in the Schema containing query attributes file to use for selected topic field.
3. Click the **Open Attribute** button.
4. Edit the copy of QAQQINI that you just made in the library, as shown in Figure 7-25.
   a. Click the cell you want to change and type the new value. As shown in Figure 7-25, we change the setting for MESSAGES_DEBUG from the original value *DEFAULT to *YES, stating that we want debug messages to be recorded for the selected job.
   b. Press Enter to activate your changes.

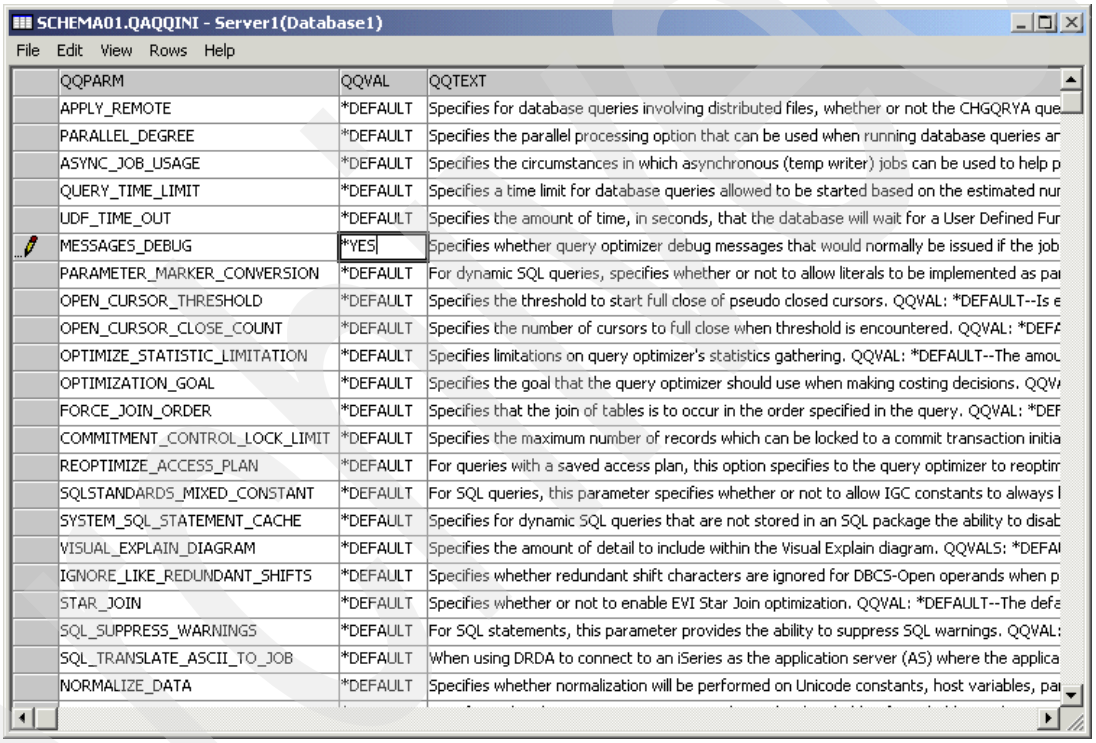


Figure 7-25 Change Query Attributes: Editing QAQQINI

5. You receive a warning message as shown in Figure 7-26. Click **Yes** and close the window.

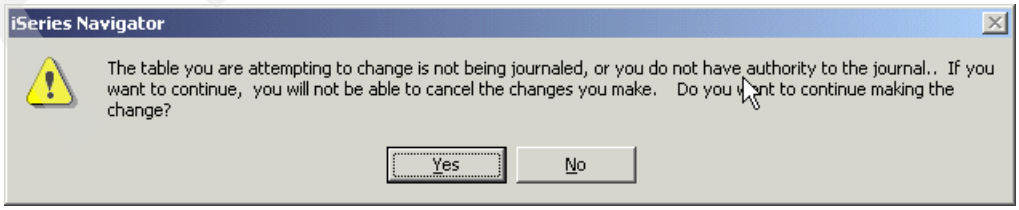


Figure 7-26 Warning message on modification of the QAQQINI file

6. You return to the Change Query Attributes window. Click **OK** to make the change effective.

The options that are currently managed in the QAQQINI file and their values are documented in *Query performance and query optimization* in the iSeries Information Center on the Web at:

http://www.iseries.ibm.com/infocenter

When you reach this site, select **Database** →**Performance and optimization**.

7.3 Current SQL for a job

You can use the Current SQL function to select any job running on the system and display the current SQL statement being run, if any. In addition to displaying the last SQL statement being run, you can edit and rerun it through the automatically linked Run SQL Scripts option and display the actual job log for the selected job or, even end the job. You can also use this function for database usage and performance analysis, with the Visual Explain tool (see Chapter 8, “Visual Explain” on page 213).

To start, in iSeries Navigator, expand the system that you want to use, right-click **Databases** and select **Current SQL for a Job**. Then you see the Current SQL window as shown in Figure 7-27.

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 that you have sufficient authority.

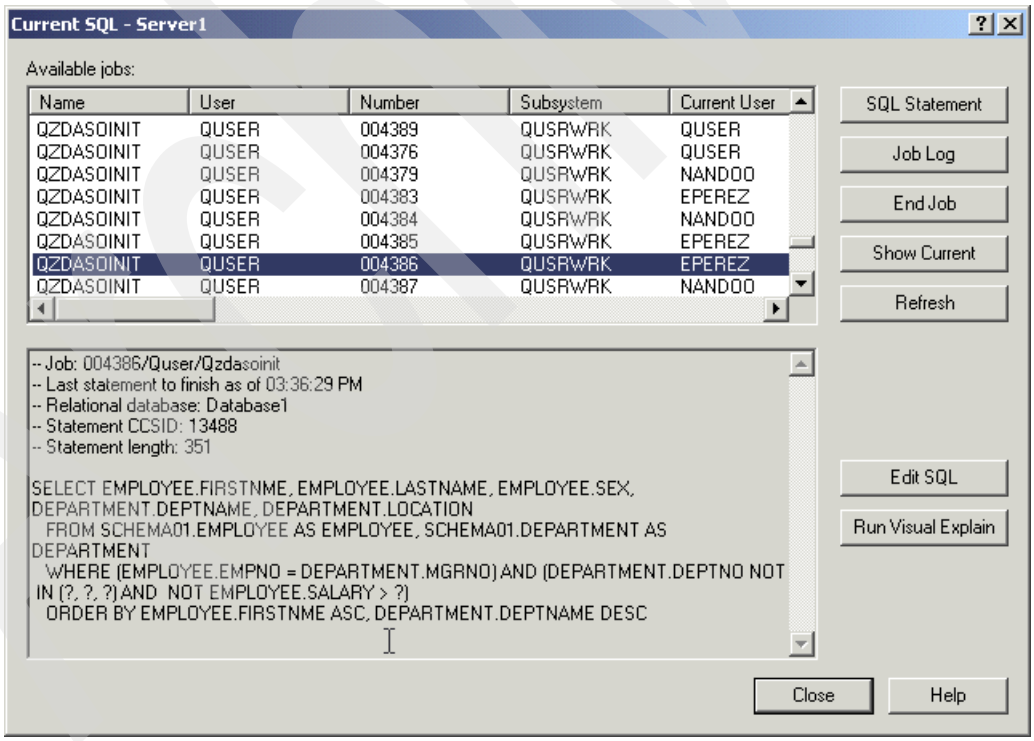


Figure 7-27 Current SQL for a Job

In this example, we selected a JDBC job in the Available jobs section. Then we clicked the SQL Statement button to display the last SQL statement it ran in the bottom part of the window. You can go to the job log by clicking the Job Log button or select a current job by clicking the Show Current button.

After you see the SQL statement in the bottom part of the window, you can click the Edit SQL button to work on this same statement with the Run SQL Scripts Center (Figure 7-28). It is also possible to link into Visual Explain, using the Run Visual Explain button. For information about this tool, see Chapter 8, “Visual Explain” on page 213.

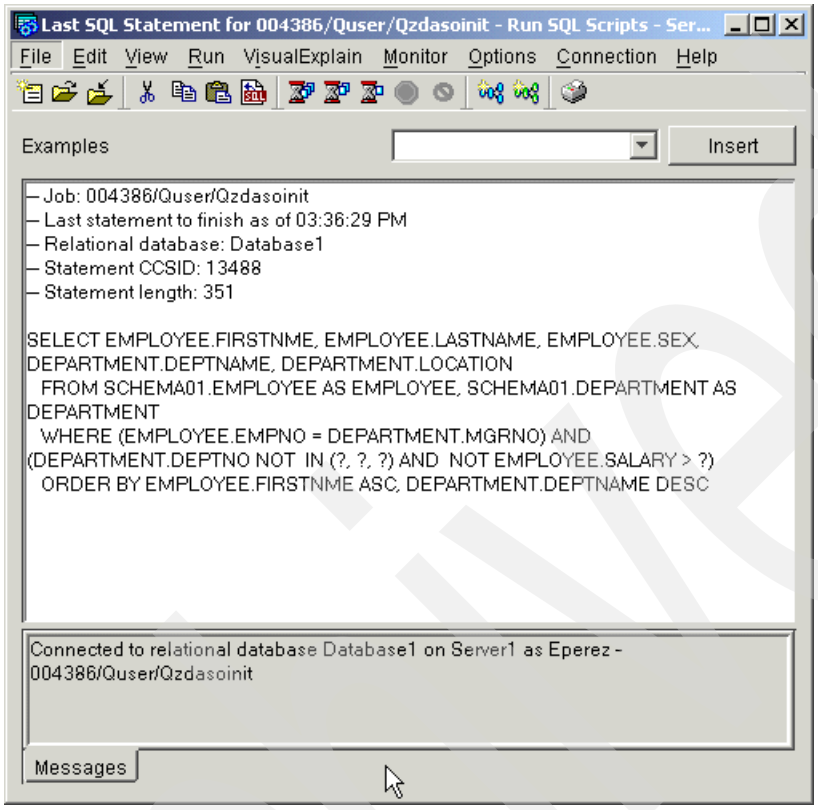


Figure 7-28 Working with a current SQL for a job

As you may have already noticed, all iSeries Navigator database tools are tightly integrated into each other to make it easier for the user to fully exploit their capabilities

7.4 SQL Performance Monitors

System performance is a high priority in any system. The objective is to maximize system resource utilization, while achieving maximum performance throughput.

Monitoring your queries is the first important step to ensure that they are tuned for optimal performance. Consider it as a type of tool for the early determination and identification of potential problem areas.

The SQL Performance Monitor lets you keep track of the resources that SQL statements use. You can monitor specific resources or many resources. The information about resource use can help you determine whether your system and your SQL statements are performing as they should or whether they need fine tuning.

You may receive reports about the poor performance from the database. To find the cause of the poor performance, you start an SQL performance monitor for all jobs running on that database. From the information gathered by this instance of the monitor, you determine which jobs consume most of the time. You then can start monitors for these specific jobs. By using

the information gathered by these monitors, you can fine-tune your queries to optimize performance.

There are two types of monitors that you can choose to monitor your resources:

- Summary SQL performance monitor

  This monitor resides in memory and retains only a summary of data collected. When the monitor is paused or ended, this data is written to a hard disk and can be analyzed. Because the monitor stores its information in memory, the performance impact to your system is minimized. However you lose some of the detail.

- Detailed SQL performance monitor

  This monitor saves detailed data in real time to a hard disk and does not need to be paused or ended to analyze the results. You can also choose to run the Visual Explain based on the data gather by monitor. Since this monitor saves data in real time, it may have a performance impact on your system.

For more information about the SQL Performance Monitor, refer to *Preparing for and Tuning the V5R2 SQL Query Engine on DB2 Universal Database for iSeries*, SG24-6598.

7.5 System Debugger

DB2 UDB simplified the debug of SQL procedures, functions, and triggers in V5R2 with the SQL *SOURCE Debug View. This debug can be simplified even more by using System Debugger.

SQL procedures have always used C code underneath the covers since their arrival. The usage of C code behind the scenes is really not an issue for RPG and COBOL programmers until they need to debug the SQL procedure, function, or trigger. Instead of showing the original SQL procedural statements, the iSeries System Debugger shows the generated C code that is used to implement the original SQL procedure statements.

iSeries System Debugger provides a new graphical user debugging environment on the iSeries server. Use iSeries System Debugger to debug and test programs that run on your iSeries server, including those that run in the OS/400 Portable Application Solutions Environment (PASE) environment. iSeries System Debugger is packaged in a Java Archive (JAR) file with IBM Toolbox for Java.

Since SQL procedures, functions, and triggers are implemented by DB2 UDB as generated Integrated Language Environment® (ILE) C code, they are also supported by the graphical debugger. The graphical debugger supports the following ILE languages:

- ILE C & C++
- ILE RPG
- ILE COBOL
- ILE CL
- Java

Note: Original Program Model (OPM) RPG and COBOL programs can also be used with the graphical debugger if *LSTDBG or *SRCDBG is specified on the compile and the debug session is started with OPMSRC(*YES) specified.

The iSeries System Debugger includes an integrated call stack window, breakpoints groups, variable monitors, and a local variables display. In addition to the new capabilities, the

graphical debugger can be used to debug practically any scenario that a developer can debug using the green-screen interface (via the Start Debug (STRDBG) command).

With the arrival of the *SOURCE debug view in V5R2, programmers can work with the original SQL source debug mode instead of the complex C code generated by DB2 UDB. The *SQL Source debug view can be created in one of two ways. One way is to specify the *SOURCE debug option in the source of the SQL procedure, function o trigger. This method is demonstrated by specifying DBGVIEW = *SOURCE on the SET OPTION clause (see Figure 7-29). An alternative method is to specify a *SOURCE value for the DBGVIEW parameter on the Run SQL Statement (RUNSQLSTM) command.

```
CREATE PROCEDURE SCHEMA01.ADDCOM01 ( IN DEPTNO CHAR(3) ,
                                     IN EVALUA NUMERIC(1, 0) )
LANGUAGE SQL
SET OPTION DBGVIEW = *SOURCE
BEGIN
            CASE
             WHEN EVALUA = 5
              THEN UPDATE EMPLOYEE SET COMM = COMM + 100
                     WHERE WORKDEPT = DEPTNO ;
             WHEN EVALUA = 4
               THEN UPDATE EMPLOYEE SET COMM = COMM + 80
                      WHERE WORKDEPT = DEPTNO ;
             WHEN EVALUA = 3
               THEN UPDATE EMPLOYEE SET COMM = COMM + 50
                      WHERE WORKDEPT = DEPTNO ;
             ELSE UPDATE EMPLOYEE SET COMM = COMM + 20
                      WHERE WORKDEPT = DEPTNO ;
            END CASE ;
END ;
```

*Figure 7-29 SQL procedure with the DBGVIEW=*SOURCE option*

There are several other nuances with the SQL source-level debug that you must know:

- Comments are not saved.
- When stepping through the source, the step may stay on the same SQL statement for several steps, if that SQL statement required multiple lines of generated C code to implement.
- When accessing the value of SQL variables and parameters during the debug session, the EVAL command is still used. Also you must enter the variable and parameter name in capital letters and add a label prefix.

Starting the iSeries System Debugger

To start the iSeries System Debugger, follow these steps:

1. From the Run SQL Scripts menu, select **Run** →**Debugger**.
2. The Start Debug window (Figure 7-30) opens. It dynamically changes appearance, depending on the debug method and type of program object that you want to debug. The window does not display all of the following options at the same time.
3. Select the debug method:
   - **Submit and debug program in batch job**: Submit a batch job that calls the specified program object that you want to debug.
   - **Debug existing job on system**: Submit a debug server to service the existing specified job, which can be active, on hold, or in a job queue that is on hold.
   - **Debug OS/400 PASE**: Select to specify OS/400 PASE debug for the session. A second debug tab appears that allows both the ILE and OS/400 PASE attributes of the target

job to be debugged. This option is useful if the target job has both ILE and OS/400 PASE attributes.

4. Select the program to debug. Depending on the debug method, you can specify the following options:
   - From the list, select the type of program object that you want to debug. You can select to debug programs, service program, or class files. In addition, you can click the Recent button to display a list of program objects that you previously opened with the System Debugger.
   - Specify the program name for programs and service programs or the package name for class files.
   - Specify the library or schema for programs and service programs or the class name for class files.
   - Type any parameters or Java properties that you want to pass to the program object or click the Edit... button to display a window that helps you to edit and save parameters.
   - For Initialization command, type a single OS/400 Control Language command that you want to run before calling the program that you want to debug. For example, you can use this command to set environment variables, library lists, and so on.

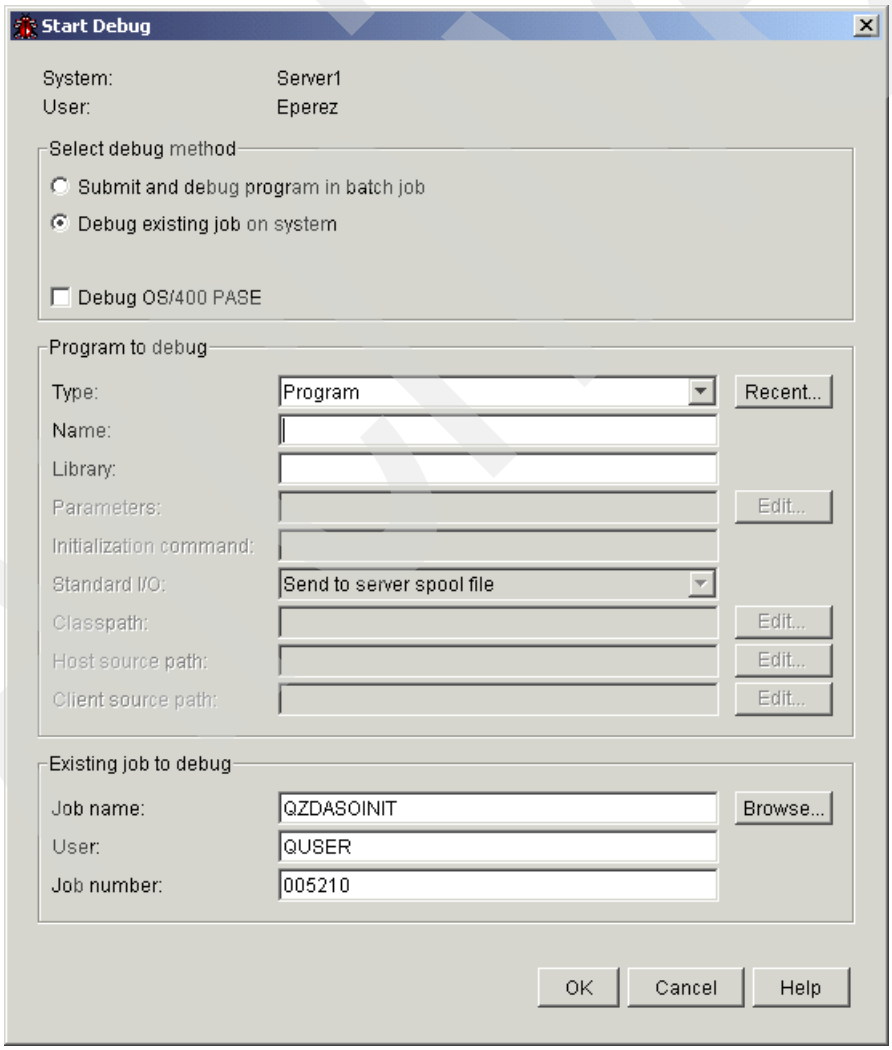


Figure 7-30 Start Debug window

– Select the type of input/output that the program should perform. Possible choices are Send to server spool file or Redirect to client. I/O that is redirected to the client appears in the Standard I/O panel.

– If you selected class files, type the classpath or click the Edit... button to display a window that enables you to edit and save the classpath.

– The Host source path option is also active for class files. Type the host source path or click the Edit... button to display a window that enables you to edit and save the host source path.

– For Client source path, type the client source path or click the Edit... button to display a dialog that enables you to edit and save the client source path.

Note: The Debug Manager searches for Java files in the following order:

- The same directory that contains the corresponding .class file
- The path specified in Host source path
- The path specified in Client source path

5. If you selected the Debug existing job on system debug method, under Existing job to debug (Figure 7-31), specify the name, use, and job number or click the **Browse...** button to search for and automatically enter the required information for an existing job.
6. Click **OK** to start the debug of the program, service program or class file selected.

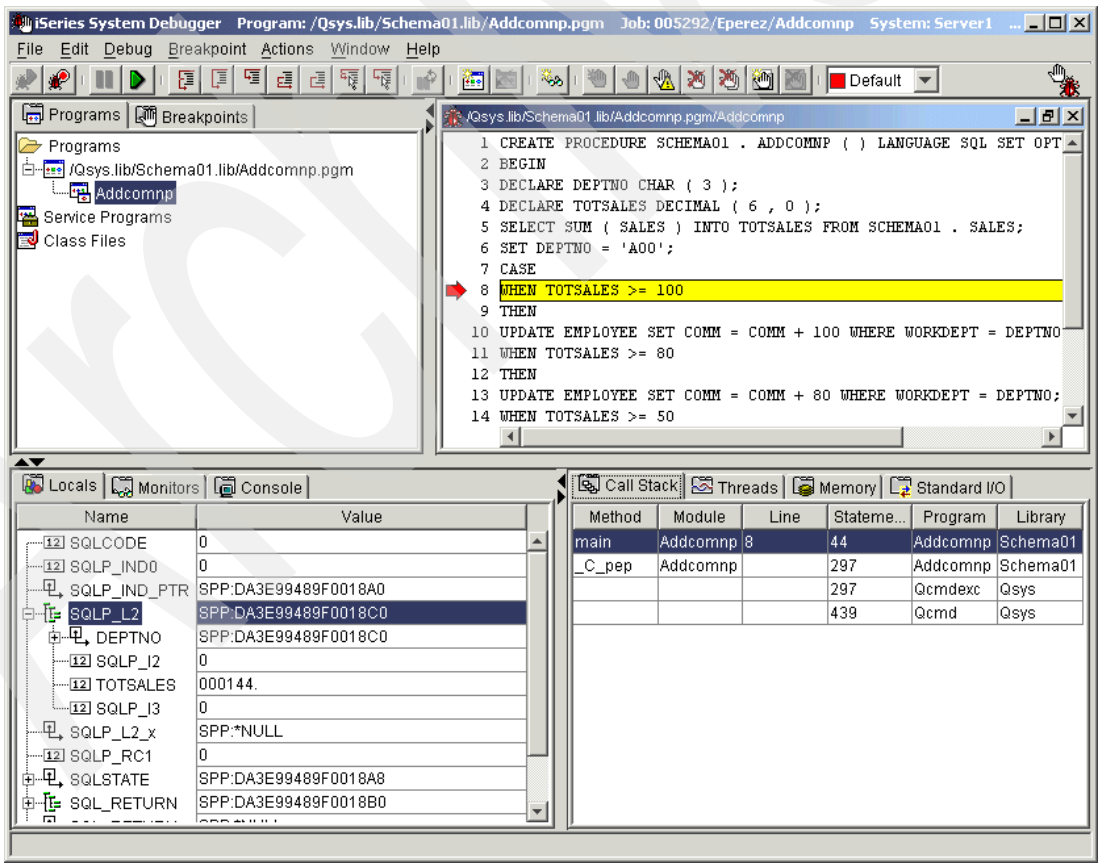


Figure 7-31 iSeries System Debugger

Debug interface components

Figure 7-31 shows the debug interface window with a series of tabbed areas. Each tab provides a different view of the overall debug environment. The debugger interface is a multiple document window, so you can view more than one source file at once.

The System Debugger window provides graphical information about the current status of debugging activities. The window sections are:

- **Programs pane**: The Programs pane is located in the upper left corner of the window. It displays the program (*PGM), service program (*SRVPGM), and class files that are currently loaded on the debugger. *MODULE objects display under the applicable *PGM or *SRVPGM objects. The options for the Programs pane are available from the context menu and the menu bar. Menu options that have an icon appear on the tool bar.
- **Breakpoints pane:** The Breakpoints pane organizes breakpoints into logical groupings with an associated name and color for each group. Any actions taken against a group affects all breakpoints associated with that group. A group can consist of breakpoints from different program objects or modules. The Breakpoints pane supports drag-and-drop operations. You access this pane by clicking the Breakpoints tab next to the Programs tab.
- **Source pane**: The Source window is located in the upper right pane. It displays the source code that is contained in the program that is being debugged. When program processing stops, the current source line position is automatically displayed. The current source line is indicated by a colored bar and moves as the program is debugged.

  You can switch between compiler-supplied source views with the Change Source View option available by selecting Actions →Change Source View from the menu bar or by right-clicking the source windows and selecting Change Source View.

  The source pane also provides the ability to display the contents of variables simply by highlighting any valid variable within the source code. The contents of the variable display in the message section at the bottom of the System Debugger.

  Multiple source panes may be open on the desktop. The Windows menu provides a list of all current source panes.
- **Locals pane**: The Locals pane is located in the lower left corner. It displays the values of local variables when program processing stops. You may edit local variables directly from this pane by double-clicking the variable and entering a new value. The tree pane on the left allows for expanding and collapsing the internals of variables such as arrays and structures. Global variables are viewed in the Monitors pane.

  To optimize debug performance, temporarily suspend updates by right-clicking the pane and selecting Suspend Updates. While in the suspended mode, the pane displays the Suspended indicator, and the pane icon is unavailable. Right-click and select Resume Updates to permit updating the pane information.
- **Monitors pane**: You access the Monitors pane by clicking the Monitors tab next to the Locals tab. This pane displays only those variables that are explicitly selected by the user for review. Any valid variable expression can be monitored. To explicitly select a variable to appear in this window, select a variable from the source window, and then select Actions →New monitor.

  The Monitors pane includes the ability to watch a specified variable. The watch function enables a programmer to request a breakpoint when the content of a specified storage location is changed from its current value. After the watch condition is successfully set, a change to the content of the watched storage location causes program operation to be suspended.

  To optimize debug performance, temporarily suspend updates by right-clicking the pane and selecting Suspend Updates. While in the suspended mode, the pane displays the

indicator Suspended and pane icon is unavailable. Right click and select Resume Updates to permit updating the pane information.

- **Console pane**: You access the Console pane by selecting the Console tab next to the Monitors tab. The Console pane displays host-related message activity such as informational messages and error messages. It also provides a command line entry area that supports a subset of debug commands supported by the ILE system debugger. The results of any command line entry appear in the console log.
- **Call Stack pane:** The Call Stack pane is located in the lower right corner of the iSeries System Debugger. It displays the current call stack of the active thread. Changing the active thread causes the call stack to switch to the new thread. To work with call stack entries, perform any of the actions in the Threads pane, Memory pane, or Standard I/O pane. You can optimize the debug performance in the same way as the Monitors pane.
- **Threads pane:** The Threads pane displays all the threads in the current program and uses an arrow to show the current thread (located in the Programs pane). Use the Threads pane to view information about the threads in your program, set the current thread, and enable or disable thread execution.
- **Memory pane:** The Memory pane displays the memory, in hexadecimal form, that is associated with the program variables. You can use the lists in the Memory pane to select an expression for which you want to see the memory contents or to specify the display format, byte size, and number of bytes displayed per line. When you debug a program written in C or C++, you can change memory contents.
- **Standard I/O pane:** The Standard I/O pane displays the contents of standard output and standard error data streams when they are redirected to the client. I/O can be redirected to the client by selecting the Redirect to client option in the Start Debug window. Output data that is received from the program is immediately displayed in the corresponding text area.

  This pane also provides a standard input entry field for sending input data back to the program. To send data to standard input, type the characters in the Standard Input entry field. Press the Enter key to send the data with a terminating newline. Click the Go button to send the data without a terminating newline.

  Right-click anywhere in the Standard I/O pane to select any of the available options. The options for the Standard I/O pane are also available from the menu bar.

Using breakpoints

You can control the flow of a program by using breakpoints. Breakpoints can be either unconditional or conditional. An *unconditional breakpoint* stops the program at a specific statement. A *conditional breakpoint* stops the program when a specific condition at a specific statement is met. The appropriate module is shown with the source positioned to the line where the condition occurred. The cursor is positioned on the line where the breakpoint occurred if the cursor was in the text area of the display the last time the source was displayed. Otherwise, it is positioned on the command line.

To add a breakpoint to your programs, first select the point in the System Debugger source window where you want to add the breakpoint. Then you can either click the line number to automatically add a breakpoint or click a line of code and perform one of the following actions:

- Select **Breakpoints** →**Add Breakpoint** from the menu bar.
- Right-click the source window and select **Add Breakpoint**.
- Select **Add Breakpoint** from the toolbar.

Arrows are used to show the lines with breakpoints. As shown Figure 7-32, the color (red for Default and Blue for Test) depends on the breakpoint group. To change the current breakpoint group, use the Current Breakpoint Group list on the toolbar to select a breakpoint group. The System Debugger now creates all new breakpoints as a member of this breakpoint group.

To remove a breakpoint or breakpoint group, select the desired object in the Breakpoints pane and use either the menu options (**Breakpoints** →**Remove Breakpoint**, **Remove All Breakpoints**, or **Delete Breakpoint Group**), the context menu, or press the workstation Delete key.

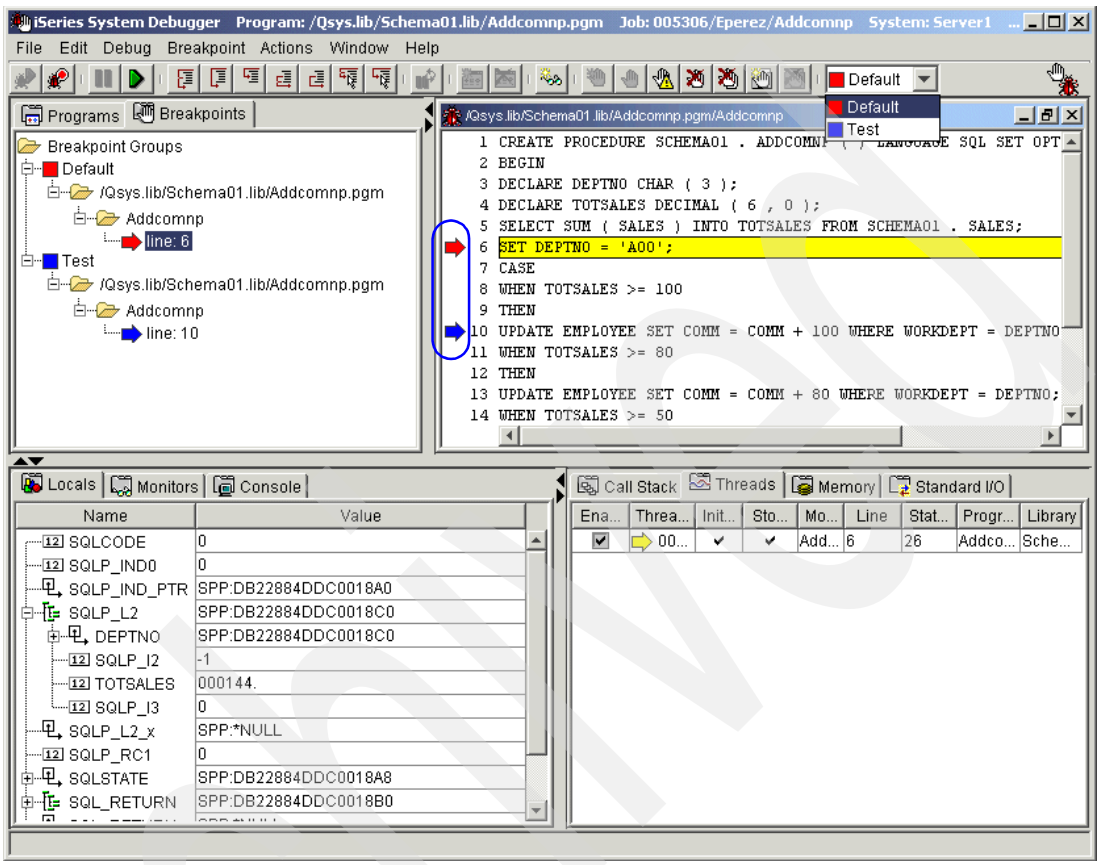


Figure 7-32 Using breakpoints in System Debugger

You can change the properties of a breakpoint. This includes adding conditional parameters, making it thread-specific, and converting it to or from a default breakpoint. To change breakpoint properties, right-click the breakpoint and select **Breakpoint Properties** as shown in Figure 7-33.

The breakpoint properties that you can change are:

- **Breakpoint group**: From the list, select the breakpoint group that you want to contain this breakpoint. By default, all breakpoints belong to the Default breakpoint group.
- **Breakpoint style**: From the list, select the breakpoint style that you want to apply to this breakpoint. By default, all breakpoints have the Default breakpoint style. The *Default* option marks a point that suspends operation during a debugging session.
- **Service entry point**: This option marks a point that suspends operation and starts a debugging session when a job associated with a specific user calls that line of code. The job that calls the line of code must be a program, service program, or optimized OS/400 Java program.
- **Condition**: Set optional conditional breakpoint parameters by typing the parameters. You can use any valid expression as the breakpoint condition.
- **Thread**: From the list, select a thread-specific breakpoint that determines when the debugger will suspend operation.

- Type the name of the user that is associated with the service entry point breakpoint. This option displays only when the breakpoint style is Service entry point.

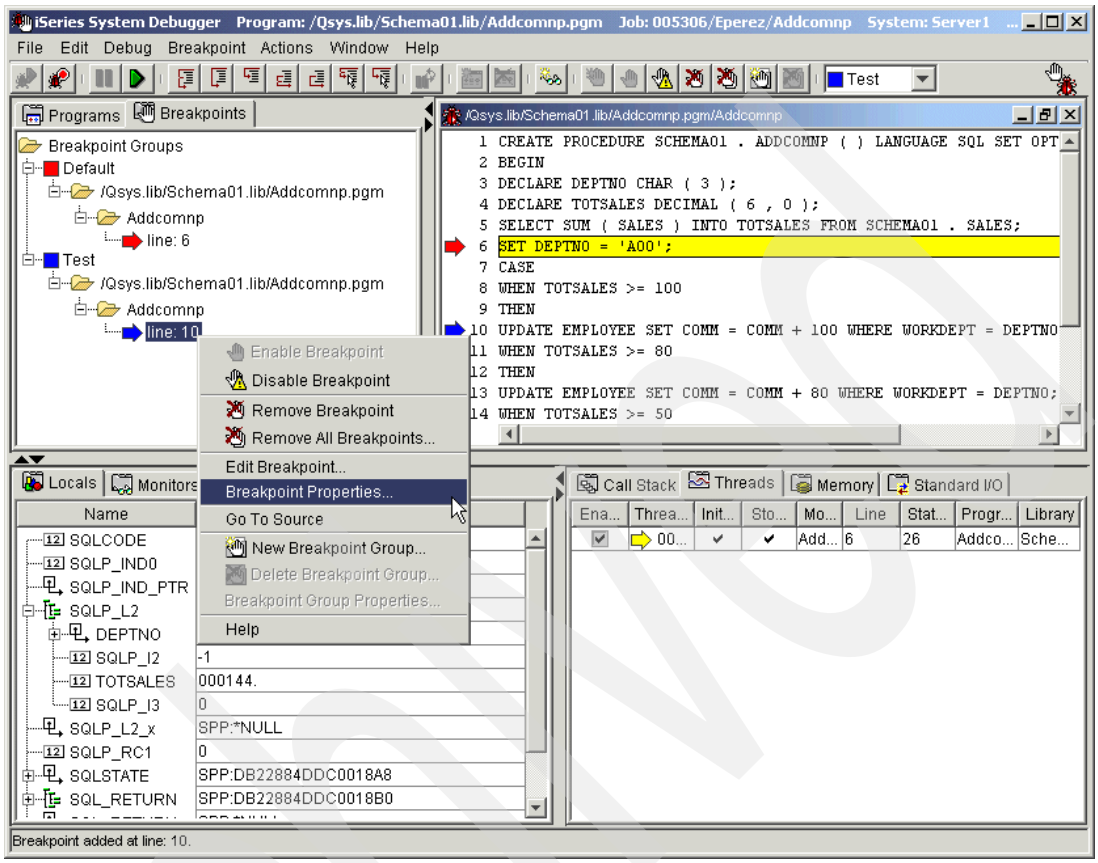


Figure 7-33 Changing breakpoint properties

Additional options to work with breakpoints are available from the context menu, Breakpoint menu option, and toolbar. These options allow you to enable or disable breakpoints, remove one or all, edit or change properties, create a new group, and delete a group. You can find complete information about these options in the online help.

Changing columns displayed

You can change the number, order, or both of columns displayed in the System Debugger for Call Stack, Threads, and Monitors panes. This function is available from the context menu, by selecting **Actions** →**Columns**, or by pressing Shift + C.

To modify the number of columns displayed, select the desired column from the Columns to be displayed section, select **Remove**, and then click **OK**, as shown in Figure 7-34.

The selected column is no longer viewable in the panel. It now appears in the Columns available to display pane.

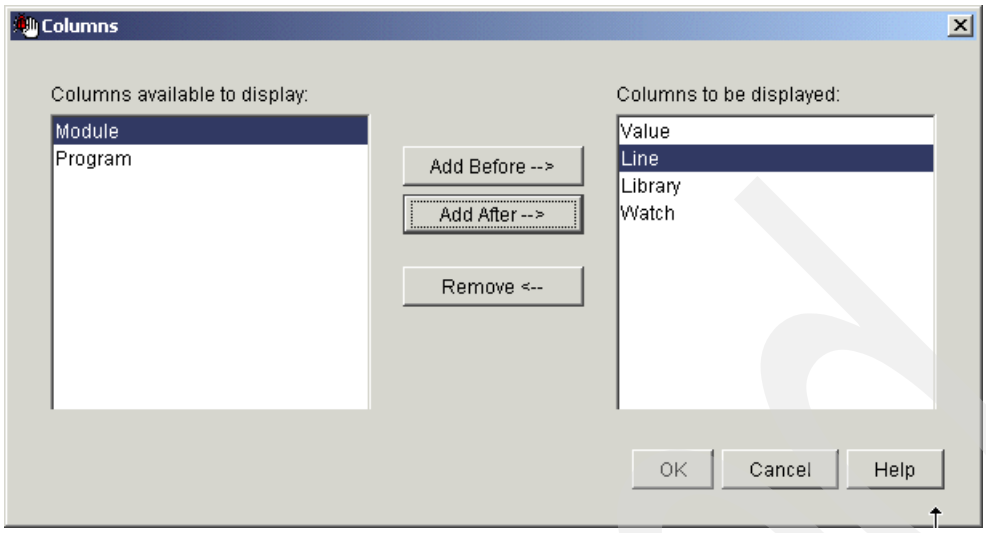


Figure 7-34 Changing the columns to display

Setting program preferences

To set program preferences, select **File** →**Preferences**. Use the tabs on the Preferences window (Figure 7-35) to change the program options. In this window, you can change general preferences, colors, fonts, and key bindings.

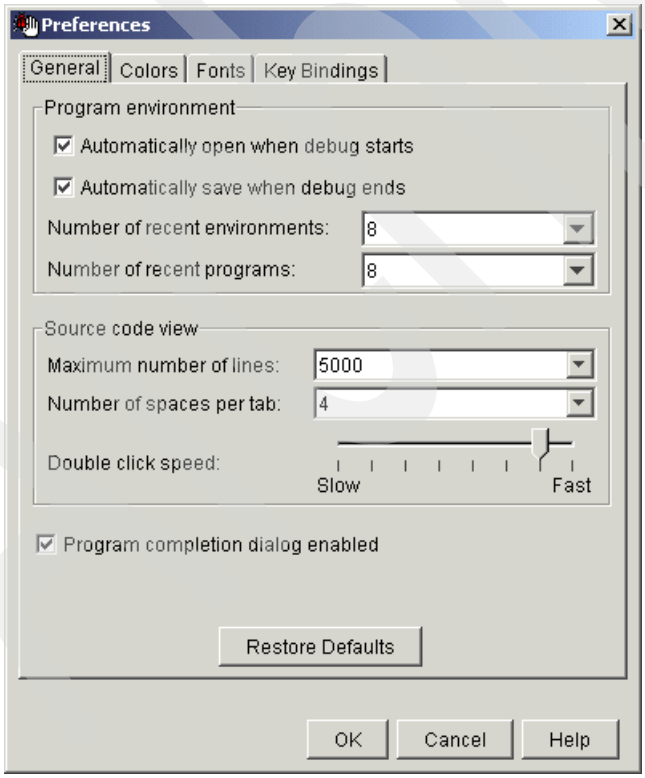


Figure 7-35 Setting program preferences

The tabs are explained in the following list:

- **General tab**
  - Program environment: Automates the opening and saving of debugging environments.
    - Automatically open when debug starts: Select this option to automatically open the previous debugging environment when starting debugging operations. This option is selected by default.
    - Automatically save when debug ends: Select this option to automatically save the current debugging environment when ending current debugging operations. This option is selected by default.
    - Number of recent environments: Select a number from the list to set the number of recently used environments that you want to display under File →Recent Environments.
    - Number of recent programs: Select a number from the list to set the number of recently used programs you want to display under File →Recent Programs.
  - Source code view:
    - Maximum number of lines: Select a number from the list to set the maximum number of lines to display in the source pane.
    - Number of spaces per tab: Select a number from the list to set how many spaces the cursor moves for each press of the Tab key. The default is four spaces per tab.
    - Double click speed: Adjust to select the speed at which two mouse clicks register as a double-click.
  - Select the Program completion dialog enabled check box if you want a message when the program end, as shown in Figure 7-36.

Figure 7-36 Program completion message

  - Click Restore Defaults if you want to restore the original values.
- **Colors tab**
  - Category: Set color preferences to highlight the location of the Program stop position. You can't change this value.
  - Foreground: Select the foreground color that you want to use from the list.
  - Background: Select the background color that you want to use from the list.
- **Fonts tab**
  - Category: Set font preferences for the Source code view. You can't change this value.
  - Font: Select the font that you want to use from the list.
  - Font size: Select the font size that you want to use from the list.
  - Preview: Displays a sample of how your current selections appear.
- **Key Bindings tab**

  Changes the keyboard shortcuts for the different iSeries System Debugger menu selections. Select a menu option or key sequence that you want to change, and then use the Key Sequence window to make your changes.

8

Visual Explain

The launch of DB2 Universal Database (UDB) for iSeries Visual Explain with iSeries Navigator Version 4, Release 5, Modification 0 (V4R5M0) was of great interest to database administrators (DBAs) working in an iSeries server environment. The feature has been described as a quantum leap forward in database tuning for query optimization. Visual Explain provides an easy-to-understand graphical interface that represents the optimizer implementation of the query.

For the first time, you can see, in graphic detail, how the optimizer has implemented the query. You can even see all of the facts and figures that the optimizer used to make its decisions. Best of all, the information is presented in one place, in color, with easy-to-follow displays. There is no more jumping between multiple windows, trying to determine what is happening. Even better, if you currently have iSeries Navigator, you already have Visual Explain.

With all of this in mind, is such a richly featured product complicated to use? As long as you are familiar with database tuning, you will enjoy using Visual Explain and want to learn more.

This chapter answers these questions:

- Where do I find Visual Explain?
- How do I use it?
- What can it be used for?
- Will it tune my Structured Query Language (SQL) queries?
- 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 about the DB2 UDB for iSeries query optimizer and database engine. The recommended way to obtain this skill and build this understanding is to refer to *Preparing for and Tuning the V5R2 SQL Query Engine on DB2 Universal Database for iSeries*, SG24-6598.

8.1 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. 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. 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 no longer reserved only for programmers.

8.2 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 they can involve changing the way that the query statements are structured to create a more efficient access plan.

8.2.1 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 may improve the performance of the query.

The analysis of optimizer debug messages was made easier with the addition of a predictive query governor. By specifying a time limit of zero in the predictive query governor, query optimizer debug messages can be generated in the job log without actually running the query. This means that a query, which may take 16 hours to run, can be analyzed in a few seconds. Some changes can be made to the query or to the database. The effect can be modelled on the query in just a few minutes. The query is then run when the optimum implementation is achieved.

8.2.2 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.

You access the Database Monitor directly from the database component of iSeries Navigator. You can also access it from a 5250 device using the Start Database Monitor (STRDBMON) command or during the collection of performance data with the Start Performance Collection (STRPFRCOL) command.

You can perform an analysis of the statistics that are gathered by using the SQL Performance Monitors in iSeries Navigator. iSeries Navigator provides many predefined reports to assist with the analysis of the performance data collected in this manner.

8.2.3 The PRTSQLINF command

For SQL embedded in a program and package objects, the Print SQL Information (PRTSQLINF) CL command extracts the optimizer access method information from 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.

8.2.4 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.

8.3 A closer look at Visual Explain

In iSeries Access Express, the database component of iSeries Navigator is a graphical way to manage the database. Visual Explain was added to the database component in V4R5M0 and

enhanced in all the Version 5 releases. Visual Explain provides a graphical way to identify and analyze database performance.

8.3.1 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 that represent each unique component. Visual Explain also includes information about the database objects that are considered and chosen by the query optimizer. Visual Explain's detailed representation of the query implementation makes it easier to understand where the greatest cost is being incurred.

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

If query performance is an issue, Visual Explain provides information that can help you to determine whether you need to:

- Rewrite or alter the SQL statement
- Change the query attributes or environment settings
- Create new indexes

Best of all, you do not have to run the query to find this information. Visual Explain has a modeling option that allows you to explain the query without running it. That means that you can try any of the changes suggested and see how they are likely to work, before you decide whether to implement them.

You can also use Visual Explain to:

- View the statistics that were used at the time of optimization
- Determine whether an index was used to access a table

  If an index was not used, Visual Explain can help you determine which columns might benefit from being indexed.
- View the effects of performing various tuning techniques by comparing the before and after versions of the query graph
- Obtain information about each operation in the query graph, including the total estimated cost and number of rows retrieved
- View the debug messages issued by the Query Optimizer during the query execution

Visual Explain is an advanced tool to assist you with the task of enhancing query performance, although it does not actually do this task for you. You still need to understand the process of query optimization and the different access plans that you can implement.

8.3.2 Finding Visual Explain

Visual Explain is a component of iSeries Navigator. You find it in the Database icon. To locate the Database icon, you need to establish a session on your selected iSeries server using the iSeries Navigator icon.

From the SQL Script Center, you can access to Visual Explain directly, either from the menu or from the toolbar. This is explained in 7.1.3, "Run SQL Scripts Run options" on page 179.

Another way to access Visual Explain is through SQL Performance Monitor. SQL Performance Monitor is used to create Database Performance Monitor data and to analyze the monitor data with predefined reports.

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

8.4 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.

8.4.1 The SQL Script Center

To access the SQL Script Center, expand **Databases**. Then select any database, right-click, and select **Run SQL Scripts**. The Run SQL Script window (Figure 8-1) opens with the toolbar.

Reading from left to right, there are icons to create, open, and save SQL scripts, followed by icons to cut, copy, paste, and insert generated SQL (from V5R1) statements within scripts.

The hour glass icons indicate to run the statements in the Run SQL Scripts window. From left to right, they run all of the statements, run all of the statements from the cursor to the end (From Selected), or run the single statement identified by the cursor position (Selected).

To the right of the hour glasses 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 8-1 Toolbar from Run SQL Scripts

There are two Visual Explain icons in the colors blue and green. The left Visual Explain icon (blue) helps to explain the SQL statement. The right Visual Explain icon (green) enables you to run and explain the SQL statement. Both of these options are also available from the Visual Explain menu (Figure 8-2). You may choose either option to start Visual Explain.

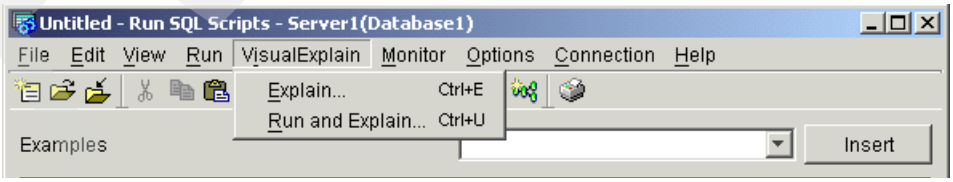


Figure 8-2 SQL Script Center Visual Explain options

You can use SQL Performance Monitors to record SQL statements that are explainable by Visual Explain. We recommend that you obtain access via the SQL Performance Monitors icon, because it provides the full list of monitors.

An SQL script is defined as one or more statements from the Run SQL Script working area under 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.

8.4.2 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.

To optimize an SQL statement, the optimizer validates the statement, gathers statistics about the SQL statement, and creates an access plan. When you choose the Visual Explain Only option, the optimizer processes the query statement internally with the query time limit set to zero. Therefore, it proceeds through the full validation, optimization, and creation of an access plan and then reports the results in a graphical display.

Note: When you choose Visual Explain Only, Visual Explain may not be able to explain such complex queries as hash join, temp join results, etc. In this case, you must choose *Run and Explain* for the SQL statements to see the graphical representation.

8.4.3 Run and Explain

The Run and Explain option (Ctrl + U or the green toolbar icon) also submits the query request to the optimizer. It provides a visual explanation of the SQL statement and the access plan used when executing the statement. It provides a detailed analysis of the results through a series of attributes and values associated with each of the icons.

However, it does not set the query time limit to zero and, therefore, continues with the execution of the query. This leads to the display of a results window in addition to the Visual Explain graphics.

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.

8.5 Navigating Visual Explain

The Visual Explain graphics window (Figure 8-3) is presented in two parts. The left side of the display is called the *Query Implementation Graph*. This is the graphical representation of the implementation of the SQL statement and the methods used to access the database. The arrows indicate the order of the steps. Each node of the graph has an icon that represents an operation or values returned from an operation.

The right side of the display has the *Query Attributes and Values*. The display corresponds to the object that has been selected on the graph. 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.

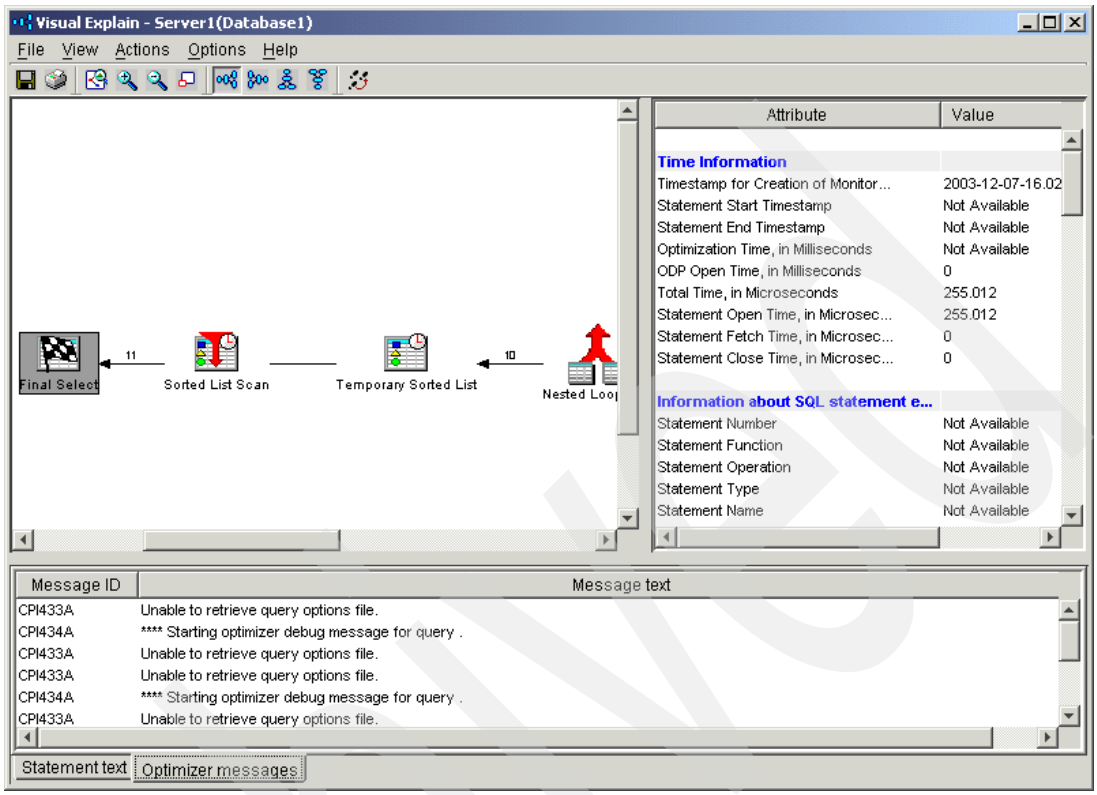


Figure 8-3 Visual Explain Query Implementation Graph and Query Attributes and Values

The default settings cause the display to be presented with the final result icon (a checkered flag) on the left of the display. Each icon on the display has a description and the estimated number of rows to be used as input for each stage of the implementation.

Clicking any of the icons causes the Query Attributes and Values display to change and presents the details that are known to the query for that part of the implementation. You may find it helpful to adjust the display to see more of the attributes and values. Query attributes and values are discussed further in 8.5.5, “Visual Explain query attributes and values” on page 228.

When you right-click any of the icons on the display, an action menu is displayed. The action menu has options to assist with query information. It can provide a shortcut to table information to be shown in a separate window. You can find more details in 8.5.2, “Action menu items” on page 223.

You may find the following action menu items selectively on different icons:

- **Table Description**: Displays table information returned by the Display File Description (DSPFD) command
- **Index Description**: Displays index information returned by the DSPFD command
- **Create Index**: Creates a permanent index on the iSeries server
- **Table Properties**: Displays object properties
- **Index Properties**: Displays object properties

- **Environment Settings**: Displays environment settings used during the processing of this query
- **Additional fly-over panels**: Exist for many of the icons

  By moving the mouse pointer over the icon, a window appears with summary information about the specific operation. See Figure 8-4.

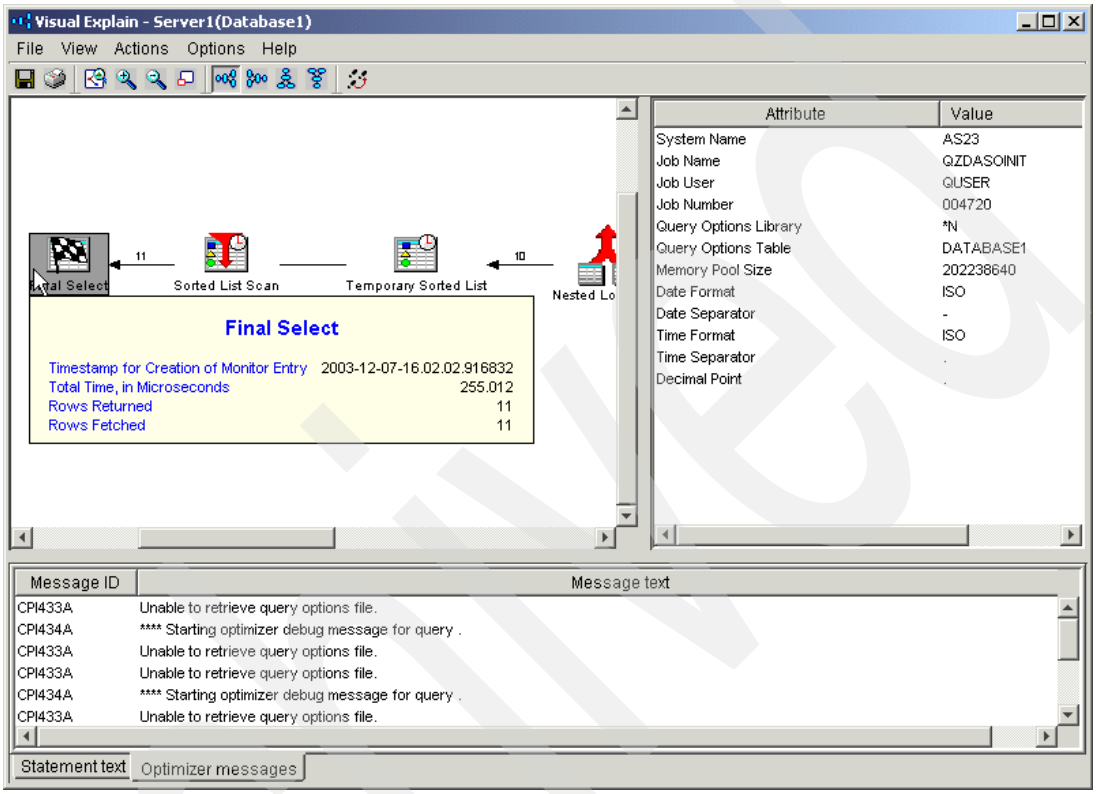


Figure 8-4 Table scan flyover panel

The Visual Explain toolbar (Figure 8-5) helps you to navigate the displays. The first four icons (from left to right after the printer icon) help you to control the sizing of the display. The left-most icon scales the graphics to fit the main window. For many query implementations, this leaves the graphical display too small to be of value. The next two icons allow you to zoom in and out of the graphic image.

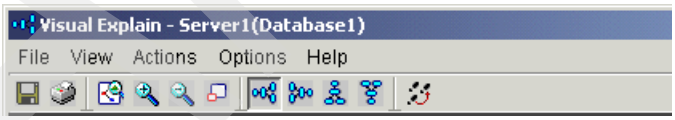


Figure 8-5 Visual Explain toolbar

The fourth icon (Overview) creates an additional window Figure 8-6 that shows the Visual Explain graphic on a reduced scale. This window has a highlighted area, which represents the part of the image that is currently displayed in the main window.

In the Overview window (Figure 8-6), you can move the cursor into this highlighted area that is shown in the main window. The mouse pointer changes so you can drag the highlighted area to change the section of the overall diagram that is shown in the main window.

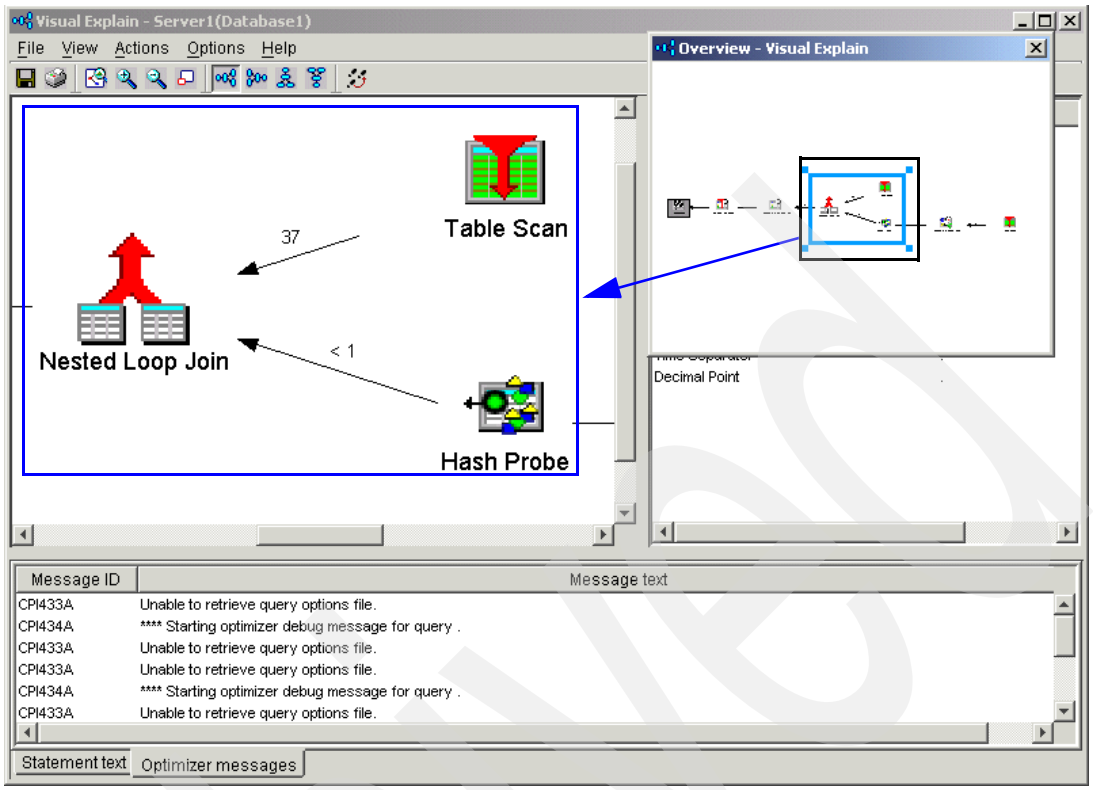


Figure 8-6 Visual Explain Overview window

The default schematic shows the query with the result on the left, working across the display from right to left, to allow you to start at the result and work backward. The 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*.

You can use the last icon, Statistics and index advisor (new in V5R2), to identify missing or stale statistics and to specify how the statistics will be collected (Figure 8-7).

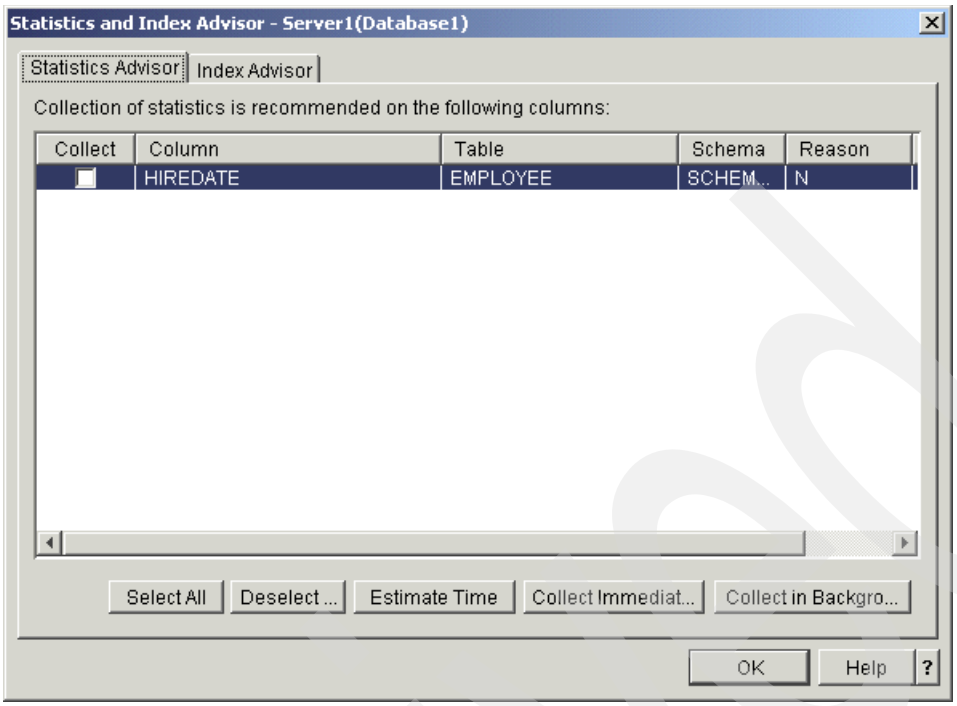


Figure 8-7 Statistics and Index Advisor: Statistics Advisor tab

Additionally, the query optimizer analyzes the row selection in the query and determines, based on default values, if creation of a permanent index would improve performance. If the optimizer determines that a permanent index would be beneficial, it returns the key columns necessary to create the suggested index. On the Index Advisor tab, you can see the indexes that are recommended for creation as shown in Figure 8-8. You can select the index that you want and click the Create button to create the index selected.

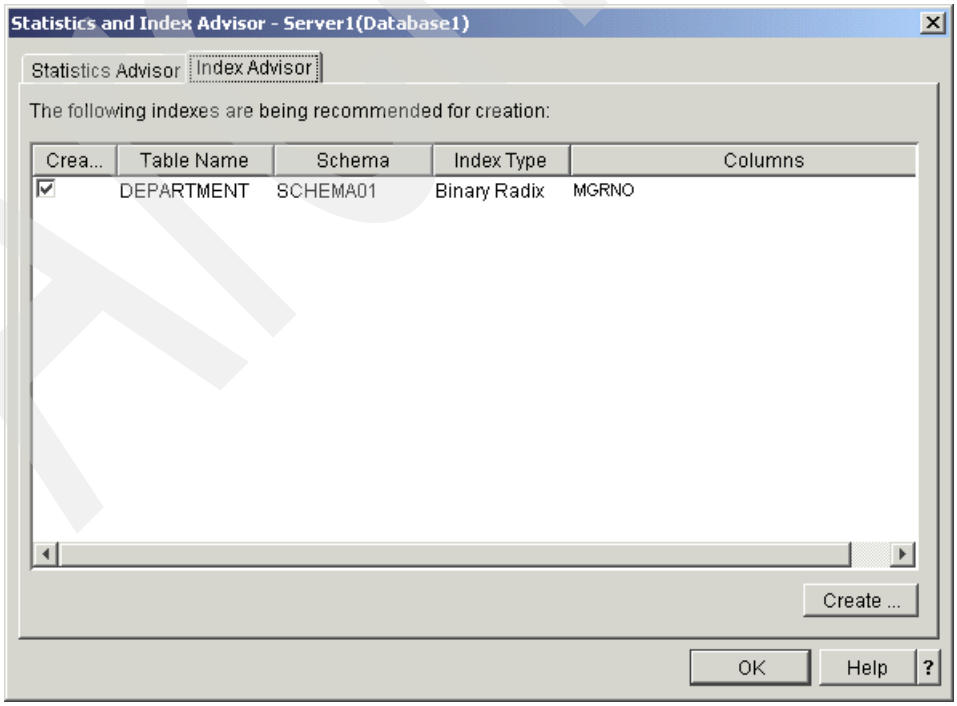


Figure 8-8 Statistics and Index Advisor: Index Advisor tab

8.5.1 Menu options

The menu options above the toolbar icons are File, View, Actions, Options, and Help. Starting with V5R1, the ability to either print or save the Visual Explain output as an SQL Performance Monitor file was added. 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 or hide the estimated number of rows, processing time, or the degree of parallelism that the query is processing at each stage of the implementation.
- Icon labels allow you to show or hide the description of the icons and the object name associated with the icon.
- You can highlight expensive icons by number of returned rows and processing time.
- You can also highlight advised indexes and icons in your graph where the optimizer recommends that you create a permanent index. The icon is highlighted in the graph and the relevant text fields are highlighted in the Icon attributes and values table as shown in Figure 8-9.

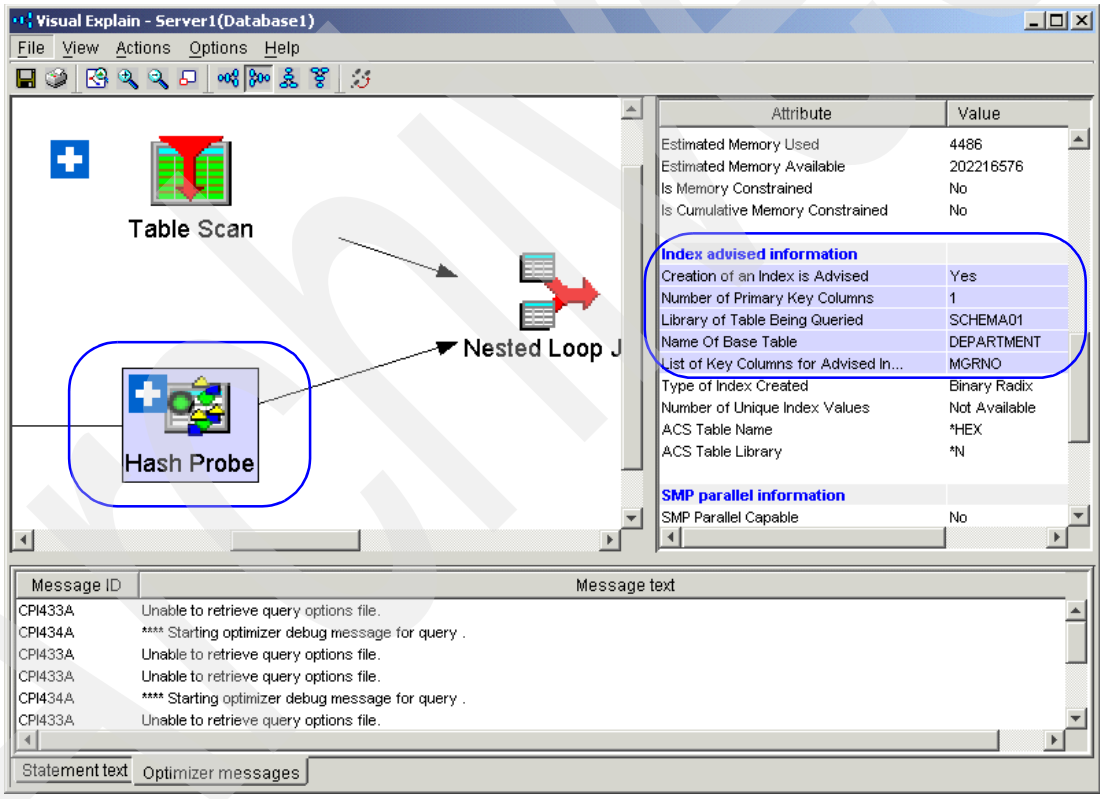


Figure 8-9 Highlighting Index advised information

8.5.2 Action menu items

The Actions menu item replicates the features that are available on the display. When you right-click a query implementation icon, a menu appears that offers further options. These options may include one of more of the items in the following sections.

Table Definition

The Table Definition (Table properties in V5R1) menu item opens the same window that is used to create the table. It enables you to change the definition of the table including columns, constraints and partitions.

Table Description

The Table Description menu item takes you into the graphical equivalent of the DSPFD command. From here, you can learn more information about the file. The description has several tabs to select to find further information. A limited number of changes can be made from the different tab windows.

Index Description

The Index Description attributes can be accessed to obtain further information about the index. Several changes are allowed to an index from these windows, including access path maintenance settings.

Index Definition

The Index Definition window shows the columns that exist in the table. A sequential number is placed next to the columns that form the index, with an indication of whether the index is ascending or descending. The display also shows the type of index.

Create Index

From the temporary index icon, the Create Index menu item takes you to a window where the attributes of the temporary index are completed as shown in Figure 8-10. Simply click **OK** to create a permanent index.

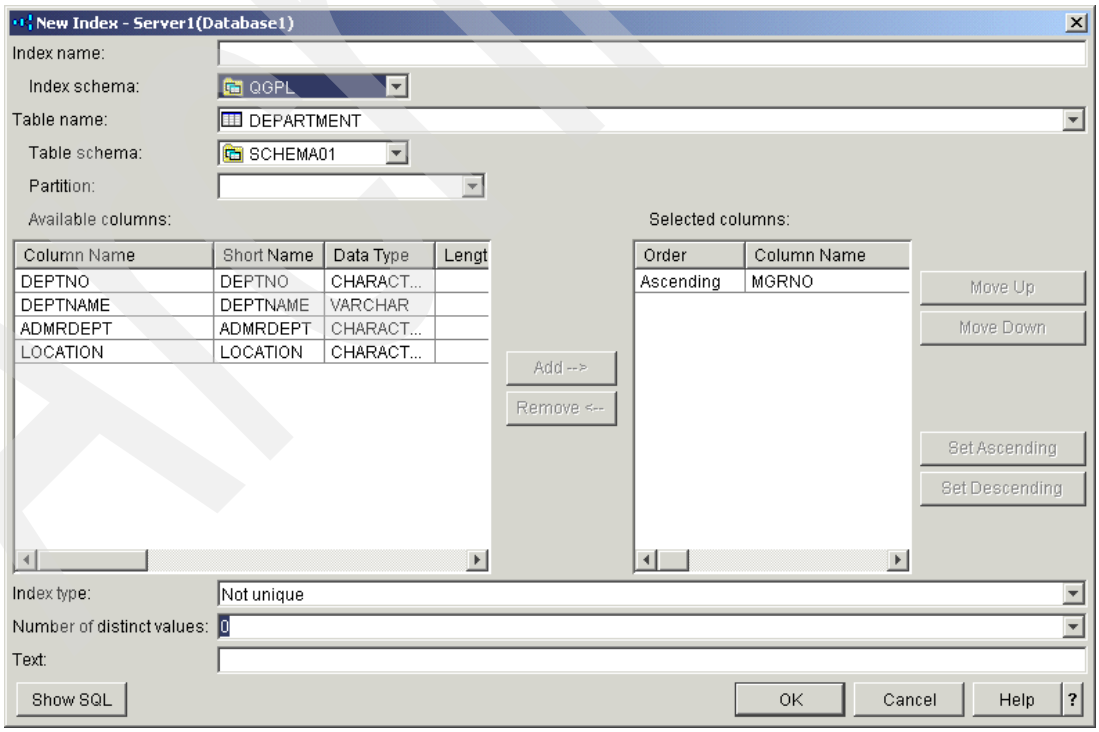


Figure 8-10 Creating an index from Visual Explain

You need to enter an index and schema name. The type of index is assumed to be binary radix with non-unique keys.

Note: The Create Index menu item is available from any icon where an index is advised (for example, table scan, key positioning, key selection) in addition to the temp index icon. This is one of the user friendly features of Visual Explain. It gives you the ability to easily create an index that the optimizer has suggested.

Statistic Data

Use the Statistics Data window (Figure 8-11) to view statistical data for the columns in the selected table. The statistical information can be used by the query optimizer to determine the best access plan for a query. The data presented here can be used to decide whether to collect more statistics, delete existing statistics for a column, or to view more detailed column statistics data.

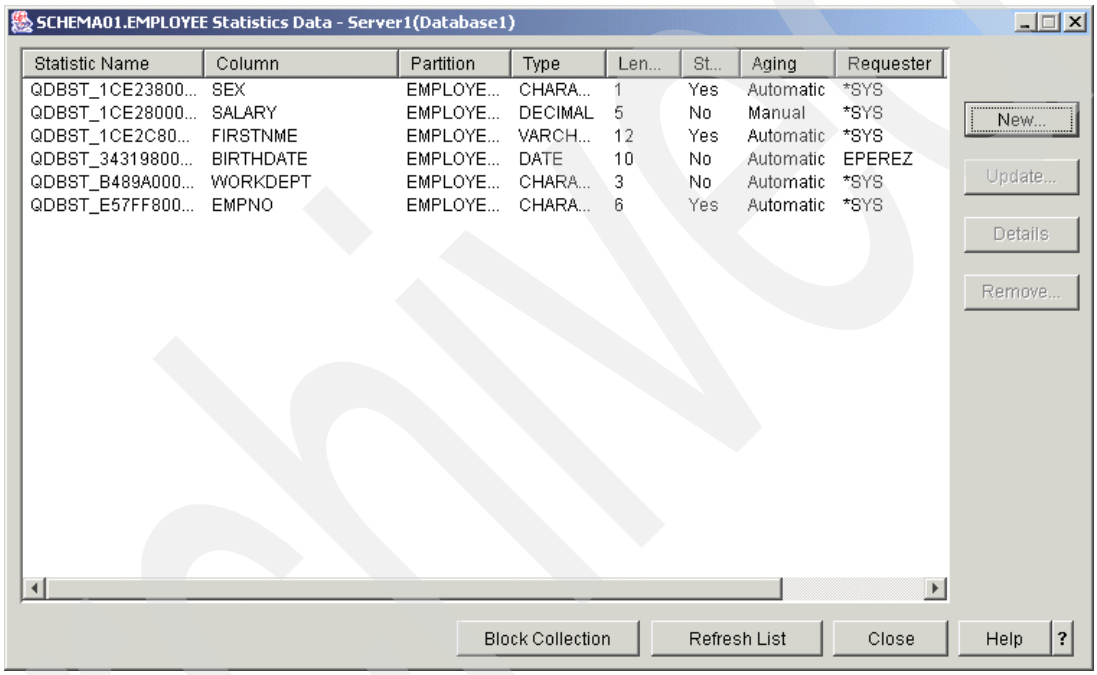


Figure 8-11 Statistic Data window

8.5.3 Controlling the diagram level of detail

You can select how much detail you want to see on the Visual Explain graphs. The menu options enable you to change the level of detail.

Click **Options** →**Graph Detail** →**Basic** to see only the icons that are directly related to the query. Or click **Options** →**Graph Detail** →**Full** to see the icons that are indirectly related to the query, such as table scans performed to build temporary indexes. If you select Full, a white cross is displayed next to some icons where you can right-click the icon and select **Expand** or **Expand to window** (Figure 8-12) to view a subgraph of the query operation.

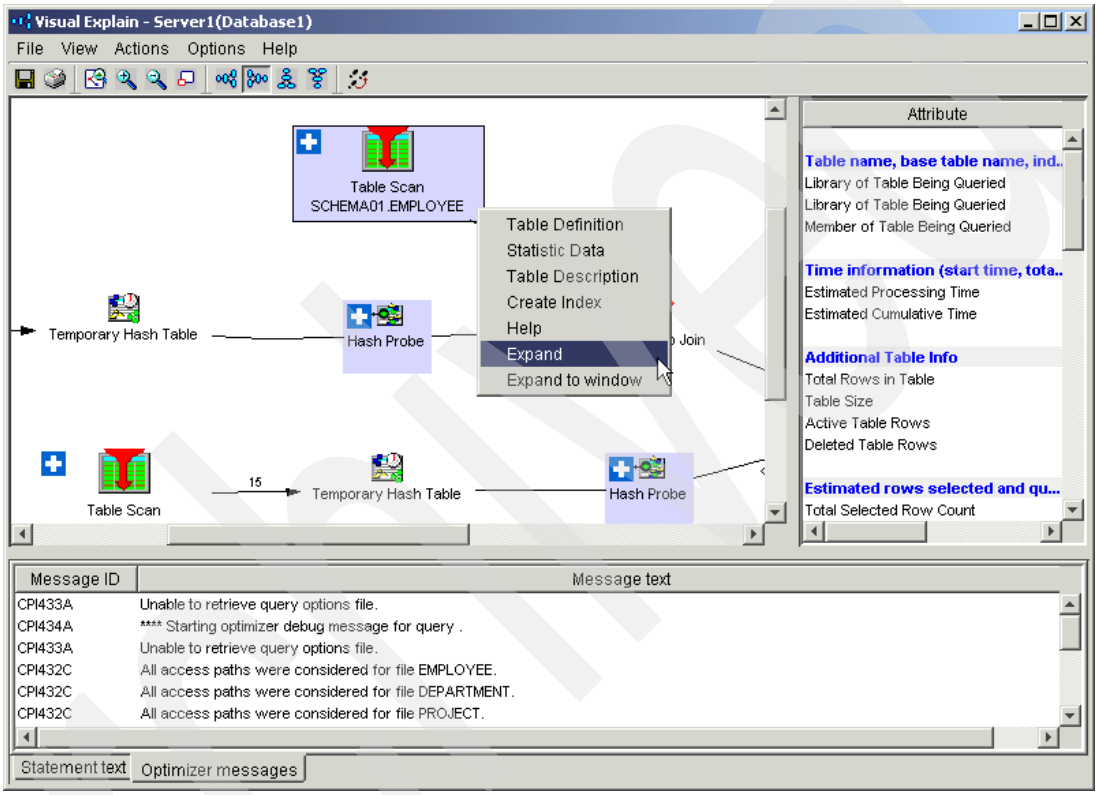


*Figure 8-12 Controlling the diagram level of detail: *Full level (Part 1 of 2)*

Figure 8-13 shows a subgraph of the query operation. You can right-click the subgraph and select the Collapse option to see the graph in the original way.

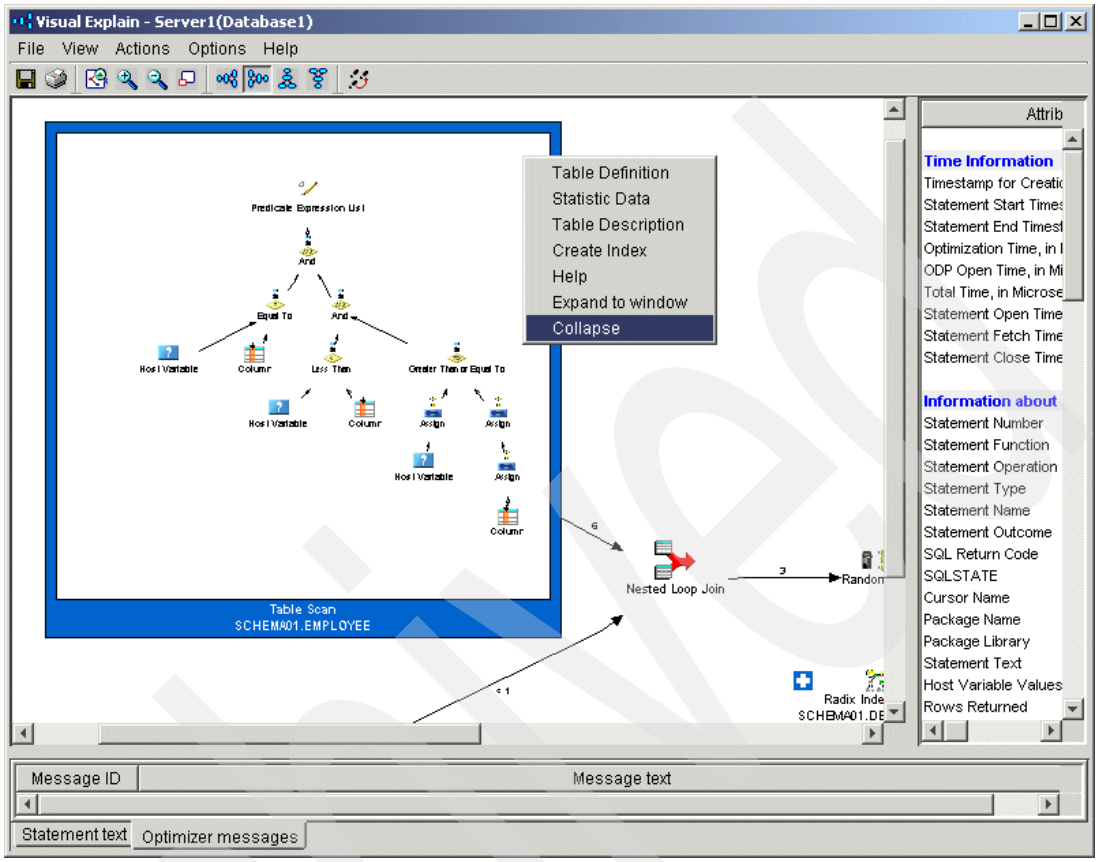


*Figure 8-13 Controlling the diagram level of detail: *Full level (Part 2 of 2)*

Most users will be satisfied with the *BASIC diagram, while others, with more performance tuning experience, may prefer the *FULL diagram.

Additionally, you can view attributes in Visual Explain in two detail settings: Basic and Full (If Possible). Select the Basic setting to retrieve only the basic subset of attributes for each icon. Select the Full (If Possible) setting to retrieve a full set of attributes for each icon. The full set of attributes for each icon are shown, unless there is too much data to be returned in Visual Explain. If this happens, basic attributes are retrieved.

8.5.4 Displaying the query environment

The query environment is available as a fast path from the Final Results icon. It shows the work management environment (Figure 8-14) where the query was executed. You can also obtain this information from the Query Attributes and Values displays.

Attribute	Value
System Name	AS23
Job Name	QZDASOINIT
Job User	QUSER
Job Number	004855
Query Options Library	*N
Query Options Table	DATABASE1
Memory Pool Size	151759536
Date Format	ISO
Date Separator	-
Time Format	ISO
Time Separator	.
Decimal Point	.

Figure 8-14 Environment

8.5.5 Visual Explain query attributes and values

The query attributes and values show more information about the optimizer implementation of the query. If you select an icon from the Query Implementation graph, you obtain information about that icon, as well as that part of the query implementation.

We selected a few of the query implementation icons to show you the query attributes and values. This way, you can see exactly how much information Visual Explain collects. Prior to Visual Explain, the information was often available, but never in one place.

Table name, base table name, index name

The section in Figure 8-15 shows the name and library or schema of the table being selected.

Attribute	Value
Table name, base table name, index name	
Library of Table Being Queried	EMPLOYEE
Library of Table Being Queried	SCHEMA01
Member of Table Being Queried	EMPLOYEE

Figure 8-15 Table name

Estimated processing time and table info

The estimated processing time (Figure 8-16) shows the time that the optimizer expects to take from this part of the query. The Additional Table Info section shows information about the rows and table size.

Attribute	Value
Time information (start time, total time)	
Estimated Processing Time	1.336E-1
Estimated Cumulative Time	1.336E-1
Additional Table Info	
Total Rows in Table	42
Table Size	4032
Active Table Rows	42
Deleted Table Rows	0

Figure 8-16 Estimated processing time and table info

Estimated rows selected and query join info

The estimated rows selected (Figure 8-17) shows the number of rows that the optimizer expects to output from this part of the query. If the query is only explained, it shows an estimate of the number of rows. If it is run and explained, it shows the number of rows that are selected.

Attribute	Value
Estimated rows selected and query join info	
Total Selected Row Count	5.75
Total Row Count	42
Optimize For N Rows	4.645
Optimize For N Sets	1
Selectivity	1.37E-1
Cumulative Selectivity	1.37E-1
Fetch N Rows	ALL

Figure 8-17 Estimated rows selected and query join info

Estimated Cost Information About the Plan Performed

This section (Figure 8-18) indicates whether the query is CPU or I/O bound. Queries can be CPU-intensive or I/O-intensive. When a query's constraint resource is the CPU, it is called *CPU bound*. When a query's constraint resource is the I/O, it is called *I/O bound*. A query that is either CPU or I/O bound gives you the opportunity to review the query attributes being used when the query was processing. If a symmetric multiprocessor (SMP) is installed on a multi-processor system, you should review the DEGREE parameter to ensure that you are using the systems resources effectively.

Attribute	Value
Estimated Cost Information About The Plan Performed	
Plan Name	Table Scan
Estimated Processing Time	1.336E-1
CPU Or IO Bound	CPU Bound
IO Cost	0.0
Cpu Cost	1.336E-1
IO Count	0.0
PreLoad Relation	No
Estimated Memory Used	3925
Estimated Memory Available	151759536
Is Memory Constrained	No
Is Cumulative Memory Constrained	No

Figure 8-18 Estimated Cost Information

Information about the index scan performed

This display 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 a description about the different reason codes, refer to the manual *DB2 UDB for iSeries Database Performance and Query Optimization* for V5R2.

SMP parallel information

The SMP information (Figure 8-19) 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. This information is relevant only when the DB2 SMP licensed feature is installed.

SMP parallel information	
Parallel Degree Requested	0.0
Max Capable SMP Parallel Degree	1
Parallel Pre Fetch Capable	Yes
Statement Text	CREATE HASHTABLE(SELECT *DEP...
Hash Table Node Name	Node_244

Figure 8-19 SMP parallel information

Index advised information

The Index advised section (Figure 8-20) tells you whether the query optimizer is advising the creation of a permanent index. If an index is being advised, the number and names of the columns to create the index are suggested. This is the same information that is returned by the CPI432F optimizer message. If the Highlight Index Advised option is set, advised index information, such as base table name, library, and involved columns, are easily identifiable.

Attribute	Value
Index advised information	
Creation of an Index is Advised	Yes
Number of Primary Key Columns	1
Library of Table Being Queried	SCHEMA01
Name Of Base Table	EMPLOYEE
List of Key Columns for Advised Index	SEX
Type of Index Created	Binary Radix
Number of Unique Index Values	Not Available
ACS Table Name	*HEX
ACS Table Library	*N

Figure 8-20 Index advised

It is possible for the query optimizer to not use the suggested index, if one is created. This suggestion is generated if the optimizer determines that a new index may improve the performance of the selected data by one microsecond.

Additional information about SQL statement

The display in Figure 8-21 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 and JDBC driver settings.

Statement is Explainable specifies whether the SQL statement can be explained by the Visual Explain tool.

Additional information about SQL statement	
CLOSQLCSR Value	
ALWCPYDTA Value	AnyTime
Pseudo Open	No
Pseudo Close	No
Hard Close Reason Code	Not Available
ODP Implementation	Reusable
Dynamic Replan Reason Code	Not Available
Timestamp of Last Replan	0001-01-01-00.00.00.000
Data Conversion Reason Code	Not Available
Blocking Enabled	Not Available
Delay Prep	Not Available
Statement is Explainable	Yes
Naming Convention	Not Available
Type of Dynamic Processing	Not Available
SQL Path	Not Available

Figure 8-21 Additional information

8.6 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 iSeries Navigator or with a CL command. With iSeries Navigator, the SQL Performance Monitors component is used to collect Database Monitor data. If you want to use Visual Explain with the data collected with an SQL Performance Monitor, then you must choose the detailed monitor collection when setting up the Database Performance Monitor in iSeries Navigator.

If you intend to use Visual Explain on the Database Monitor data collected with the CL commands, the data must be imported into iSeries Navigator as detailed data.

Using Visual Explain

In iSeries Navigator, click **Databases** and expand the database that you want to use. Click **SQL Performance Monitors** to obtain a list of the SQL Performance Monitors that are currently on the system.

Right-click the name that you want, and select **List Explainable Statements**. An *explainable statement* (Figure 8-22) is an SQL statement that can be explained by Visual Explain. Because Visual Explain does not process all SQL statements, some statements may not be selected.

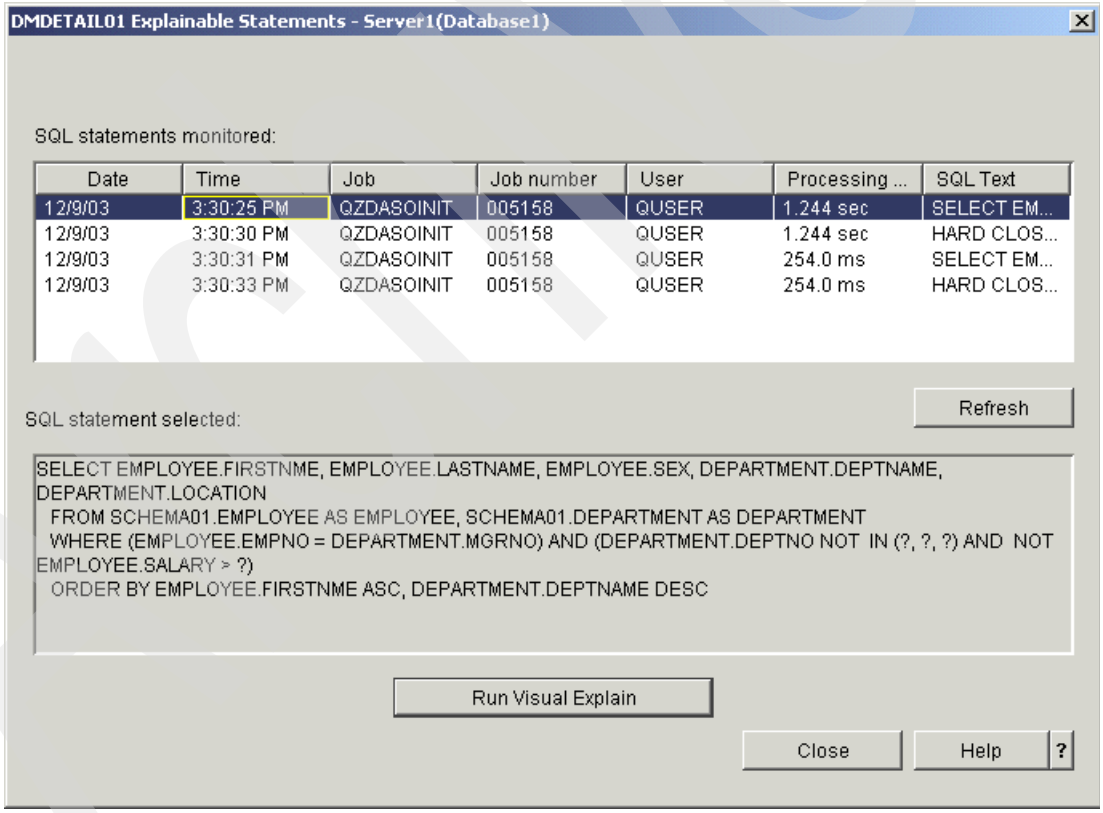


Figure 8-22 SQL explainable statements

The explainable SQL statements that were optimized by the job are now listed. If you were monitoring an SQL Script window, this is the SQL statement that was entered.

Note: Query optimizer information is generated only for an SQL statement or query request when an open data path (ODP) is created. When an SQL or query request is implemented with a reusable ODP, then the query optimizer is not invoked. Therefore, there is no feedback from the query optimizer in terms of monitor data or debug messages. Also, the statement is not explainable in Visual Explain. The only technique for analyzing the implementation of a statement in reusable ODP mode is to look for an earlier execution of that statement when an ODP was created for that statement.

To use Visual Explain on any of the statements, select the statement from the display. The full SQL statement appears in the lower part of the display for verification. Click **Run Visual Explain** (Figure 8-22) 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.

8.7 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 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. They 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 that 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 (Cobol, Java, or RPG)
- SQL statements processed through an ODBC or JDBC interface

The statements that do not generate SQL and, therefore, that cannot be used with Visual Explain via the SQL Performance Monitor include:

- Native database access from a high level language, for example, Cobol, RPG, etc.
- Query
- Open Query File (OPNQRYF) command
- OS/400 Create Query API (QQQQRY)

8.7.1 Query for iSeries and Visual Explain

Query/400, now renamed Query for iSeries, 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 (STRQMQRY) CL command runs 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 partitioned table, performance data is not collected for the second and subsequent partitions. Instead, you need to use an SQL supported interface, such as an alias for partitions.

To use this SQL statement with Visual Explain, either start an SQL Performance Monitor for this job in advance of issuing the STRQMQRY command, or use the native STRDBMON CL command to collect data for the job.

8.7.2 The Visual Explain icons

Table 8-1 lists the icons that you may find on the Visual Explain query implementation chart.

Table 8-1 Visual Explain icons

	The *Final Select* icon displays the original text and summary information about how the query was implemented.
	The *Table* icon indicates that a table was accessed. See the Select icon for more details.
	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 that meet the selection criteria were retrieved. To obtain the result in a particular sequence, you must specify the ORDER BY clause.
	The *Table Probe* icon indicates that data from the table needed to be processed and selected for this query. The table is probed using a key derived from the ordinal number or relative record number associated with each row in the table. The ordinal number is used to calculate what pages of data need to be retrieved and brought into main memory to satisfy this probe request. The ordinal number used for the probe operation was provided by another data access method previously processed for this query.
	The *Index* icon indicates that an index object was used to process this query.

	The *Index Scan* icon indicates that the entire index will be scanned, which causes all of the entries in the index to be paged into main memory to be processed. Any selection criteria, whose predicates match the key columns of the index, can then be applied against the index entries. Only those key entries that match the specified key selection will be used to select rows from the corresponding table data.
	The *Index Probe* icon indicates that the selection criteria whose predicates matched the leading key columns of the index were used to probe directly into the index. The probe selection minimizes the number of key entries that must be processed and paged into main memory. Additional key selection can be applied against the non-leading key columns of the index to further reduce the number of selected key entries. Only those key entries that match the specified probe and key selection will be used to select rows from the corresponding table data.
	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 Scan* icon indicates that a skip sequential table scan access method was used and multiple tasks were used to fill the rows in parallel. The table was partitioned, and each task was given a portion of the table to use.
	The *Derived Column Selection* icon indicates that a column in the row selected had to be mapped or derived before selection criteria could be applied against the row. Derived column selection is the slowest selection method.
	The *Parallel Derived Column Selection* icon indicates that derived field selection was performed, and the processing was accomplished using multiple tasks. The table was partitioned, and each task was given a portion of the table to use.
	The *Index Key Positioning* icon indicates that only entries of the index that match a specified range of key values were “paged in”. The range of key values was determined by the selection criteria whose predicates matched the key columns of the index. Only selected key entries were used to select rows from the corresponding table data.
	The *Parallel Index Key Positioning* icon indicates that multiple tasks were used to perform the key positioning in parallel. The range of key values was determined by the selection criteria, whose predicates matched the key columns of the index. Only selected key entries were used to select rows from the corresponding table data.
	The *Index Key Selection* icon indicates that all entries of the index were paged in. Any selection criteria, whose predicates match the key columns of the index, was applied against the index entries. Only selected key entries were used to select rows from the table data.
	The *Parallel Index Key Selection* icon indicates that multiple tasks were used to perform key selection in parallel. The table was partitioned, and each task was given a portion of the table to use.
	The *Encoded-vector Index* icon indicates that access was provided to a database file by assigning codes to distinct key values, and then representing these values in an array (vector). Because of their compact size and relative simplicity, encoded-vector indexes (EVIs) provide for faster scans.

	The *Parallel Encoded-vector Index* icon indicates that multiple tasks were used to perform the EVI selection in parallel. This allows for faster scans that can be more easily processed in parallel.
	The *Encoded-vector Index Scan* icon indicates that the entire EVI will be scanned causing all of the distinct values represented in the index to be processed. Any selection criteria, whose predicates match the key columns of the EVI, can then be applied against the distinct values represented in the index. Only those distinct values that match the specified key selection will then be used to process the vector and generate either a temporary row number list or temporary row number bitmap.
	The *Encoded-vector Index Probe* icon indicates that the selection criteria whose predicates matched the leading key columns of the EVI were used to probe directly into the distinct values represented in the index. Only those distinct values that match the specified probe selection will then be used to process the vector and generate either a temporary row number list or temporary row number bitmap.
	The *Sort Sequence* icon indicates that selected rows were sorted using a sort algorithm.
	The *Grouping* icon indicates that selected rows were grouped or summarized. Therefore, duplicate rows within a group were eliminated.
	The *Nested Loop Join* icon indicates that queried tables were joined together using a nested loop join implementation. Values from the primary file were joined to the secondary file by using an index whose key columns matched the specified join columns. This icon is usually after the method icons used on the underlying tables (that is, index scan-key selection and index scan-key positioning).
	The *Hash Join* icon indicates that a temporary hash table was created. The tables queried were joined together using a hash join implementation where a hash table was created for each secondary table. Therefore, matching values were hashed to the same hash table entry.
	The *Temporary Index* icon indicates that a temporary index was created, because the query either requires an index and one does not exist, or the creation of an index will improve performance of the query.
	The *Temporary Hash Table* icon indicates that a temporary hash table was created to perform hash processing.
	The *Temporary Table* icon indicates that a temporary table was required to either contain the intermediate results of the query, or the queried table could not be queried as it currently exists and a temporary table was created to replace it.
	The *Dynamic Bitmap* icon indicates that a bitmap was dynamically generated from an existing index. It was then used to determine which rows were to be retrieved from the table. To improve performance, dynamic bitmaps can be used in conjunction with a table scan access method for skip sequential or with either the index key position or key selection.
	The *Bitmap Merge* icon indicates that multiple bitmaps were merged or combined to form a final bitmap. The merging of the bitmaps simulates boolean logic (AND/OR selection).

	The *DISTINCT* icon indicates that duplicate rows in the result were prevented. You can specify that you do not want any duplicates by using the DISTINCT keyword, followed by the selected column names.
	The *UNION Merge* icon indicates that the results of multiple subselects were merged or combined into a single result.
	The *Subquery Merge* icon indicates that the nested SELECT was processed for each row (WHERE clause) or group of rows (HAVING clause) selected in the outer level SELECT. This is also referred to as a *correlated subquery.*
	The *Hash Table Scan* icon indicates that the entire temporary hash table will be scanned and all of the entries contained with the hash table will be processed. A hash table scan is generally considered when optimizer is considering a plan that requires the data values to be collated together but the sequence of the data is not required. The use of a hash table scan allows the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary hash table.
	The *Hash Table Probe* icon indicates that the selection criteria that match the key columns used to create the temporary hash table will be probed to find all of the matching values stored within the hash table. A hash table probe is generally considered when determining the implementation for a secondary table of a join. The use of a hash table probe allows the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary hash table. An additional benefit of using a hash table probe is that the data structure of the temporary hash table usually causes the table data to remain resident within main memory after creation, reducing paging on the subsequent probe operation.
	The *Temporary Distinct Hash Table* icon indicates that a temporary distinct hash table was created in order to perform hash processing. A distinct hash table is a data structure that is identical to the temporary hash table, except all duplicate data is compressed out of the temporary being created. The resulting hash table can then be used to perform distinct or aggregate operations for the query
	The *Distinct Hash Table Scan* icon indicates that the entire temporary distinct hash table will be scanned and all of the entries contained with the hash table will be processed. A distinct hash table scan is generally considered when optimizer is considering a plan that requires the data values to be collated together and all duplicate removed but the sequence of the data is not required. The use of a distinct hash table scan will allow the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary distinct hash table. An additional benefit of using a distinct hash table scan is that the data structure of the temporary distinct hash table usually causes the table data within the distinct hash table to remain resident within main memory after creation, reducing paging on the subsequent scan operations.
	The *Distinct Hash Table Probe* icon indicates that the selection criteria that match the key columns used to create the temporary distinct hash table will be probed to find all of the matching values stored within the hash table. The use of a distinct hash table probe allows the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary distinct hash table. An additional benefit of using a distinct hash table probe is that the data structure of the temporary distinct hash table usually causes the table data to remain resident within main memory after creation, reducing paging on the subsequent probe operation.

	The *Temporary Sorted List* icon indicates that a temporary sorted list was created in order to perform a sequencing operation. A sorted list is a data structure where the table data is collated and sorted based upon the value of a column or columns referred to as the sort key. The sorted list can then be used to return the data in a specified sequence or to perform probe operations using the sort key to quickly retrieve all of the table data that matches a particular sort key.
	The *Sorted List Scan* icon indicates that the entire temporary sorted list will be scanned and all of the entries contained with the sorted list will be processed. A sorted list scan is generally considered when optimizer is considering a plan that requires the data values to be sequenced based upon the sort key of the sorted list. The use of a sorted list scan will allow the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary sorted list. An additional benefit of using a sorted list scan is that the data structure of the temporary sorted list usually causes the table data within the sorted list to remain resident within main memory after creation, reducing paging on the subsequent scan operations.
	The *Sorted List Probe* icon indicates that the selection criteria that match the key columns used to create the temporary sorted list will be probed to find all of the matching values stored within the sorted list. A sorted list probe is generally considered when determining the implementation for a secondary table of a join when either the join condition uses an operator other than equal or a temporary hash table is not allowed in this query environment. The use of a sorted list probe allows the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary sorted list. An additional benefit of using a sorted list probe is that the data structure of the temporary sorted list usually causes the table data to remain resident within main memory after creation, reducing paging on the subsequent probe operation.
	The *Temporary List* icon indicates that a temporary list was created. The temporary list was required to either contain the intermediate results of the query, or the queried table could not be queried as it currently exists and a temporary list was created to replace it. The list is an unsorted data structure with no key. The data contained within the list can only be retrieved by a scan operation.
	The *List Scan* icon indicates that the entire temporary list will be scanned and all of the entries will be processed.
	The *Temporary Row Number List* icon indicates that a temporary row number list was created in order to help process any selection criteria. A row number list is a data structure used to represent the selected rows from a table that matches any specified selection criteria. Since the selected rows are represented by a sorted list of row numbers, multiple lists can be merged and combined to allow for complex selection and processing to be performed without having any paging occur against the table itself.
	The *Row Number List Scan* icon indicates that the entire row number list will be scanned and all of the entries will be processed. Scanning a row number list can provide large amounts of savings for the table data associated with the temporary row number list. Since the data structure of the temporary row number list guarantees that the row numbers are sorted, it closely mirrors the row number layout of the table data, ensuring that the paging on the table will never revisit the same page of data twice.
	The *Row Number List Probe* icon indicates that a row number list was used to verify that a row from a previous operation in the query matches the selection criteria used to create the temporary row number list. The use of a row number list probe allows the optimizer to generate a plan that can process the rows in the table in any manner regardless of any specified selection criteria. As the rows are processed, the ordinal number from the row is used to probe into the row number list to determine if that row matches the selection criteria. This is generally found when an index is used to satisfy the ORDER BY from a query and a separate viable index exists to process the selection criteria.

	The *Bitmap Scan* icon indicates that the entire bitmap will be scanned and all of the entries that represent selected rows will be processed. Scanning a bitmap can provide large amounts of savings for the table data associated with the temporary bitmap. Since the data structure of the temporary bitmap mirrors the row number layout of the table data, the bitmap can be used to efficiently schedule paging of the table for all selected rows.
	The *Bitmap Probe* icon indicates that a bitmap was used to verify that a row from a previous operation in the query matches the selection criteria used to create the temporary bitmap. The use of a bitmap probe allows the optimizer to generate a plan that can process the rows in the table in any manner regardless of any specified selection criteria. As the rows are processed, the ordinal number from the row is used to probe into the bitmap to determine if that row matches the selection criteria. This is generally found when an index is used to satisfy the ORDER BY from a query and a separate viable index exists to process the selection criteria.
	The *Index Scan* icon indicates that the entire temporary index will be scanned causing all of the entries in the index to be paged into main memory to be processed. Any selection criteria, whose predicates match the key columns of the index, can then be applied against the index entries. Only those key entries that match the specified key selection are used to select rows from the corresponding table data.
	The *Index Probe* icon indicates that the selection criteria whose predicates matched the leading key columns of the index were used to probe directly into the temporary index. The probe selection minimizes the number of key entries that must be processed and paged into main memory. Additional key selection can be applied against the non-leading key columns of the temporary index to further reduce the number of selected key entries. Only those key entries that matched the specified probe and key selection are used to select rows from the corresponding table data.
	The *Temporary Correlated Hash Table* icon indicates that a temporary correlated hash table was created in order to perform hash processing. A hash table is a data structure where the table data is collated based upon the value of a column or columns referred to as the *hash key*. The hash table can then be used to perform probe operation using the hash key to quickly retrieve all of the table data that matches a particular hash value. Because this is a correlated hash table, the hash table needs to be rebuilt prior to any scan or probe operations being performed.
	The *Correlated Hash Table Scan* icon indicates that the entire temporary hash table will be scanned and all of the entries contained with the hash table will be processed. A correlated hash table scan is generally considered when optimizer is considering a plan that requires the data values to be collated together but the sequence of the data is not required. In addition, the some of the values used to create the correlated hash table can change from one scan to another. The use of a correlated hash table scan will allow the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary correlated hash table. An additional benefit of using a correlated hash table scan is that the data structure of the temporary correlated hash table usually causes the table data within the hash table to remain resident within main memory after creation thus reducing paging on the subsequent scan operations.
	The *Correlated Hash Table Probe* icon indicates that the selection criteria that match the key columns used to create the temporary correlated hash table will be probed to find all of the matching values stored within the hash table. A correlated hash table probe is generally considered when determining the implementation for a secondary table of a join. The use of a hash table probe allows the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary correlated hash table. An additional benefit of using a correlated hash table probe is that the data structure of the temporary correlated hash table usually causes the table data to remain resident within main memory after creation, reducing paging on the subsequent probe operation.

	The *Temporary Correlated List* icon indicates that a temporary correlated list was created. The temporary correlated list was required to either contain the intermediate results of the query, or the queried table could not be queried as it currently exists and a temporary correlated list was created to replace it. The list is an unsorted data structure with no key that must be rebuilt prior to any scan operation being performed.
	The *Correlated List Scan* icon indicates that the entire temporary list will be scanned and all of the entries will be processed.
	The *Temporary Buffer* icon indicates that a temporary buffer was created to store the intermediate rows of an operation. The temporary buffer is generally considered at a serialization point within a query to help facilitate operations such as parallelism. The buffer is an unsorted data structure but it differs from other temporary data structures in that the buffer does not have to be fully populated in order allow its results to be processed.
	The *Buffer Scan* icon indicates that the entire temporary buffer will be scanned and all of the entries will be processed.
	The *Table Random Pre-Fetch* icon indicates that the pages required for the table probe operation will be requested synchronously in the background prior to the actual table probe operation being performed. The system attempts to manage the paging for the table probe to maintain that all of the pages of data necessary to perform the table probe operation stay resident within main memory until they are processed. The amount of pre-fetch paging that is performed by this data access method is dynamically controlled by the system based upon memory consumption and the rate at which rows continue to be processed.
	The *Table Clustered Pre-Fetch* icon indicates that the pages required for the table probe operation will be requested synchronously in the background prior to the actual table probe operation being performed. The system attempts to manage the paging for the table probe to maintain that all of the pages of data necessary to perform the table probe operation stay resident within main memory until they are processed. The amount of pre-fetch paging that is performed by this data access method is dynamically controlled by the system based upon memory consumption and the rate at which rows continue to be processed.
	The *Index Random Pre-Fetch* icon indicates that the pages required for the index probe operation will be requested synchronously in the background prior to the actual index probe operation being performed. The system attempts to manage the paging for the index probe to maintain that all of the pages of data necessary to perform the index probe operation stay resident within main memory until they are processed. The amount of pre-fetch paging that is performed by this data access method is dynamically controlled by the system based upon memory consumption and the rate at which rows continue to be processed.
	The *Logic* icon indicates that the query needed to perform an operation or test against the data in order to generate the selected rows.
	The *Fetch N Rows* icon indicates that a limit was placed upon the number of selected rows. The fetch n rows access method can either be used to implement a user specified limit on the selected rows or it can be combined with other access methods by the optimizer to satisfy complex implementation plans.

	The *Lock Row for Update* icon indicates that an update lock was acquired for the associated table data in order to perform an update or delete operation. To minimize contention between queries, the optimizer attempts to place the lock row for update operation such that the lock is not acquired and held for a long duration.
	The *User-defined table* function icon indicates that a user-defined function that returns a table was used. A table function can be referenced in an SQL FROM clause in the same way that a table or view can be referenced.
	The *Select* icon indicates a point in the query where multiple results are brought together into a single result set. For example, if a query is the union of two different select statements, at the point before the union occurs, the Select icon indicates the points where the select statements finished and the union is about to occur. This icon also represents the default information for an operation that is unknown or not defined elsewhere with Visual Explain. It can help to represent tables or insert, update and delete operations for a query. The summary information for this icon contains any available information to help describe the operation being performed and what the icon represents.
	The *Incomplete Information* icon indicates that a query could not be displayed due to incomplete information.

8.8 SQL performance analysis using Visual Explain

This section presents a brief example about SQL performance analysis using Visual Explain. A complete explanation about performance analysis is beyond the scope of this redbook, but you can find extensive information on the Web at:

http://www-1.ibm.com/servers/eserver/iseries/library/

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.

iSeries Navigator SQL Performance Monitor provides you with tools to gather and analyze SQL performance information. After you collect the SQL performance data, you can use the predefined queries to look for specific queries that have large table scans or that are evidencing some lack of indexes.

You can access those predefined queries by right-clicking the specific SQL Performance Monitor collected and selecting Analyze Results as shown in Figure 8-23.

The Basic Statement Information predefined query gives you a general idea about the queries that are being monitored, as well as the kind of access methods used by these queries. This report provides information that is related to execution time per each execution, total execution time, advised indexes, whether table scan or temporary index creation was used, and more.

After you detect a query or set of queries that needs further analysis, you can use a detailed query analysis tool, such as Visual Explain, to explore them in detail. Query analysis is iterative in nature. Try 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. A list of explainable statements appears and you can choose those in which you are interested.

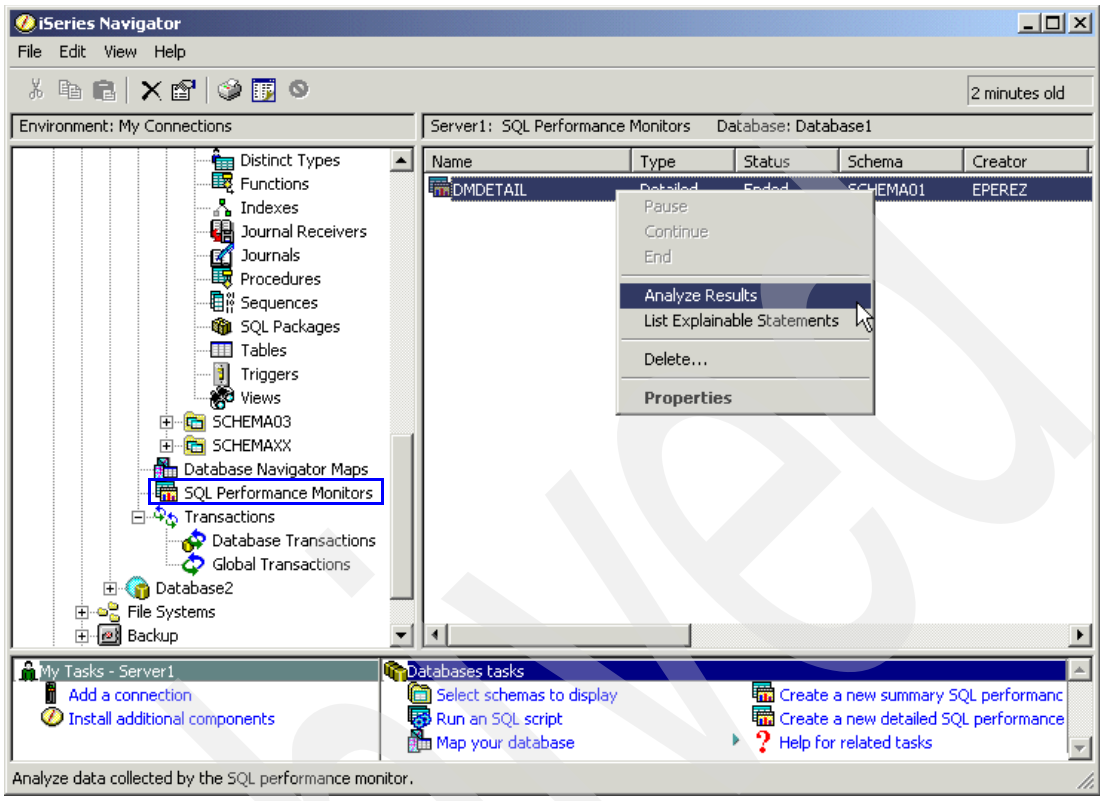


Figure 8-23 Analyzing SQL performance results

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 244. Note that some of the documents referenced here may be available in softcopy only.

- *Advanced Functions and Administration on DB2 Universal Database for iSeries*, SG24-4249
- *Managing OS/400 with Operations Navigator V5R1 Volume 1: Overview and More*, SG24-6226
- *Managing OS/400 with Operations Navigator V5R1 Volume 2: Security*, SG24-6227
- *Managing OS/400 with Operations Navigator V5R1 Volume 3: Configuration and Service*, SG24-5951
- *Managing OS/400 with Operations Navigator V5R1 Volume 4: Packages and Products*, SG24-6564
- *Managing OS/400 with Operations Navigator V5R1 Volume 5: Performance Management*, SG24-6565
- *Stored Procedures, Triggers and User Defined Functions on DB2 Universal Dabase for iSeries*, SG24-6503
- *Preparing for and Tuning the V5R2 SQL Query Engine on DB2 Universal Database for iSeries*, SG24-6598

Other publications

These publications are also relevant as further information sources:

- *AS/400 National Language Support*, SC41-5101
- *Backup and Recovery*, SC41-5304
- *SQL Reference*, SC41-5612
- *Database Programming*, SC41-5701
- *DDS Reference*, SC41-5712
- *Indexing and statistics strategies for DB2 UDB for iSeries* whitepaper

  http://www-1.ibm.com/servers/enable/site/education/abstracts/indxng_abs.html
- *Table Partitioning Strategies for DB2 UDB for iSeries* whitepaper

  http://www-1.ibm.com/servers/enable/site/education/abstracts/2c52_abs.html

Online resources

These Web sites and URLs are also relevant as further information sources:

- Information Center

  http://www.iseries.ibm.com/infocenter

- DB2 Universal Database for iSeries

  http://www.ibm.com/iseries/db2

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Q

R

S

T

U

V

DB2 Universal Database for iSeries Administration: The Graphical Way on V5R3

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

DB2 Universal Database for iSeries Administration

The Graphical Way on V5R3

Unleash the new features of iSeries Navigator for database in V5R3

Discover database maps, constraints, and Generate SQL

Learn the secrets of Run SQL Scripts and Visual Explain

iSeries Navigator for IBM i5/OS and OS/400 is *the* graphical interface to manage your IBM eserver iSeries runtime environment. V5R3 iSeries Navigator contains major functions and interface enhancements over previous releases.

This IBM Redbook presents an overview of all V5R3 iSeries Navigator functions to manage and administer DB2 Universal Database (UDB) for iSeries. It serves as a companion guide for the database administrator to use DB2 UDB for iSeries. This redbook includes the latest announced features in V5R3 that iSeries Navigator provides for the database administrator of DB2 UDB for iSeries.

This redbook discusses the new graphical interface provided for database in V5R3. In addition, it covers the following topics:

- Definition of tables, indexes, views, and other database objects
- Constraint definition and management
- Database Navigator maps
- Reverse engineering and Generate SQL
- Run SQL Script center
- Visual Explain

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6092-00 ISBN 0738491373