DB2 DataJoiner ®

Administration Supplement

Version 2 Release 1 Modification 1

SC26-9146-01

DB2 DataJoiner ®

Administration Supplement

Version 2 Release 1 Modification 1

SC26-9146-01

Note

Before using this information and the product it supports, please read the general information under “Notices” on page 171.

Second Edition (July 1998)

This edition replaces and makes obsolete the previous edition, SC26-9146-00. The technical changes for this edition are summarized under “What’s New in DataJoiner Version 2?” on page xiii, and are indicated by a vertical bar to the left of a change.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

A form for readers’ comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, BWE/H3
P. O. Box 49023
San Jose, CA 95161-9023
U.S.A.

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

Contents

About This Book

This book focuses on the administration of a heterogeneous, multidatabase environment with DataJoiner. Among other topics, it contains DataJoiner-specific information about:

- Security
- Identifying data sources
- Data access considerations
- Distributed unit of work considerations
- Tuning DataJoiner systems and queries
- Utilities
- System monitoring
- SQL explain facilities

This book focuses on information specific to DataJoiner. Complete information on the administration of the local database workstation environment is in the *DATABASE 2 Administration Guide*.

Who Should Use This Book

This book is intended for database administrators, system administrators, security administrators, and system operators who need to design, implement, and maintain a heterogeneous database environment. It can also be used by programmers and other users who require an understanding of the administration and operation of DataJoiner.

Terms for Products

Some product names in the documentation refer to more than one product, some refer to specific product levels, and some are shortened versions of full names. These product names are:

DataJoiner
Refers to DB2 DataJoiner Version 2. References specific to or including DataJoiner Version 1 will include the version.

DB2 By itself, refers to any one or all of the DB2 for common server Version 2 database server products on all platforms, which includes DataJoiner.

If a DB2 reference is qualified with a specific operating system or version, the reference applies only to that particular version.

DB2 Family
Refers to all DataJoiner-supported versions of DATABASE 2 (DB2) database server products on all platforms (DB2 for OS/390, DB2 for VM, DB2 for

common servers, DataJoiner, and so on). Supported versions are listed in the *DataJoiner Planning, Installation, and Configuration Guide* for your platform.

DB2 for CS
Refers to any DB2 for common servers Version 2 database server product. This term is often used when describing DataJoiner and DB2 for common servers functional differences.

RDB Refers to Oracle RDB Version 6 or above.

SQL Anywhere
Refers to Sybase SQL Anywhere Version 5.

Highlighting Conventions

This book uses these highlighting conventions:

Boldface type
Indicates commands and graphical user interface (GUI) controls (for example, names of fields, names of folders, menu choices). Boldface type also indicates examples of SQL keywords in the *Application Programming and SQL Reference Supplement.*

`Monospace type`
Indicates examples of coding or of text that you type.

Italic type
Indicates variables that you should replace with a value. Italic type also indicates book titles and emphasizes words.

UPPERCASE TYPE
Indicates SQL keywords and names of objects (for example, tables, views, and servers).

How to Read the Syntax Diagrams

The following rules apply to the syntax diagrams used in this book:

Arrow symbols
Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

►►— Indicates the beginning of a statement.

—► Indicates that the statement syntax is continued on the next line.

►— Indicates that a statement is continued from the previous line.

—►◄ Indicates the end of a statement.

Diagrams of syntactical units other than complete statements start with the ►— symbol and end with the —► symbol.

Conventions

- SQL commands appear in uppercase.
- Variables appear in italics (for example, *column-name*). They represent user-defined parameters or suboptions.
- When entering commands, separate parameters and keywords by at least one blank if there is no intervening punctuation.
- Enter punctuation marks (slashes, commas, periods, parentheses, quotation marks, equal signs) and numbers exactly as given.
- Footnotes are shown by a number in parentheses, for example, (1).
- A ␣ symbol indicates one blank position.

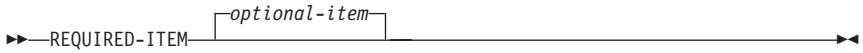
Required items

Required items appear on the horizontal line (the main path).

Optional items

Optional items appear below the main path.

If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.



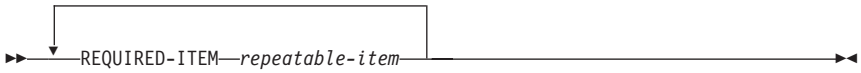
Multiple required or optional items

If you can choose from two or more items, they appear vertically in a stack. If you *must* choose one of the items, one item of the stack appears on the main path.

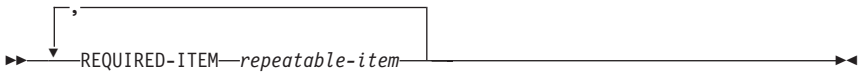
If choosing one of the items is optional, the entire stack appears below the main path.

Repeatable items

An arrow returning to the left above the main line indicates that an item can be repeated.



If the repeat arrow contains a comma, you must separate repeated items with a comma.



A repeat arrow above a stack indicates that you can specify more than one of the choices in the stack.

Default keywords

IBM-supplied default keywords appear above the main path, and the remaining choices are shown below the main path. In the parameter list following the syntax diagram, the default choices are underlined.

What's New in DataJoiner Version 2?

DataJoiner Version 2 offers new features and enhancements. They include:

DB2 Version 2 functionality

DataJoiner is built on the DB2 Version 2 code base, which means that DataJoiner provides all the major functional enhancements provided by DB2, including:

- Extended SQL capabilities
- An enhanced SQL optimizer
- Improved database performance
- Systems management support
- Robust integrity and data protection
- Object relational capabilities
- National language support (NLS)
- Support for the Java Development Kit (JDK) 1.1 for the Java Database Connectivity (JDBC) API

For detailed information about many of these features, see the *DB2 Administration Guide.*

DataJoiner for Windows NT

DataJoiner has extended its reach to provide industrial strength heterogeneous database management on Windows NT systems. DataJoiner for Windows NT supports the same SQL and features as DataJoiner for UNIX-based platforms.

Support for Oracle 8, RDB, and SQL Anywhere

With Version 2, DataJoiner continues to increase the number of natively-supported data sources. The most recent additions are:

- Oracle 8 (on any system that DataJoiner accesses from AIX or Windows NT)
- Oracle RDB Version 6 and above (on any system that DataJoiner accesses from Windows NT)
- Sybase SQL Anywhere Version 5.0 (on any system that DataJoiner accesses from Windows NT)

Spatial Extender

DataJoiner now supports geographic information system (GIS) data (also known as spatial or geographic data). New data types, spatially-enabled columns, and spatial join capability allow you to take advantage of geographic data in your applications. Included are powerful two-dimensional functions that allow you to create specific relationships among the geographic objects you define. Included with the spatial extender are the following components:

- A set of spatial data types
- A set of spatial operations and predicates

- A set of spatial index data types
- An administration tool suite for the spatial extender
- Sample programs

You can also take advantage of existing geographic data stores using the load and transform capability of the Spatial Extender.

Expanded DataJoiner SQL support

This version of DataJoiner contains many new and modified SQL statements. New DDL statements provide greater flexibility and safety in defining your DataJoiner environment—users can create, alter, and drop mappings for data sources, users, user-defined and built-in functions, and data types. Additionally, new SQL DML statements provide enhanced functions for local and distributed queries; an example is the CASE expression, which is useful for selecting an expression based on the evaluation of one or more conditions.

DataJoiner SQL for creating, altering, and deleting data source tables

Version 2 includes a new DataJoiner SQL statement for creating tables in different types of data sources. If the native SQL for creating tables in these data sources includes a unique option—for example, the option in DB2 for OS/390 for specifying what database you want a table to reside in—you can code this option in the new DataJoiner statement. If you create a data source table with this new statement, you can also alter and delete it with DataJoiner SQL.

Heterogeneous data replication

DataJoiner now provides replication administration as an integrated component. You can define, automate, and manage replication data from a single control point across your enterprise. The replication administration tool provides administrative support for the replication environment, with objects and actions that define and manage source and target table definitions. DataJoiner's Apply component performs the actual replication, tailoring and enhancing data as you specify, and serving as the interface point to and from your various data sources. DataJoiner also supplies an executable, IBM DB2 DataPropagator for Microsoft Jet, that allows you to replicate server data for browsing and updating in LAN, occasionally connected, and mobile environments.

Distributed heterogeneous update support

DataJoiner now allows you to update multiple heterogeneous data sources within a distributed unit of work while maintaining transaction atomicity. This task is accomplished through adherence to the two-phase commit model. Supported data sources include most versions of the DB2 Family and, with the appropriate XA libraries, various other data sources as well.

New graphical installation, configuration, and administration tools

A variety of new tools is available to help you accomplish administrative chores. Wizards walk you through data source configuration. And the Administrator's Toolkit provides a collection of tools designed to assist you with the day-to-day operation of DataJoiner. It includes the following components:

The Database Director
Allows you to perform configuration, backup and recovery, directory management, and media management tasks.

Visual Explain
A tool for graphically viewing and navigating complex SQL access plans.

The DB2 Performance Monitor
Monitors the performance of your DB2 system for tuning purposes.

Stored procedures
DataJoiner now supports stored procedures at remote data sources as well as the local DataJoiner database. Use stored procedures to speed application performance. For example, applications that process huge amounts of data at a server but return smaller result sets should run faster as stored procedures. Another benefit is that stored procedures usually reduce network traffic between clients and databases.

DataJoiner stored procedures can augment standard data security. For example, in a 3-tier environment, data can be retrieved from a remote server and then processed at the DataJoiner server; only a subset of data needs to be available to the client.

System catalog information available in views
DataJoiner provides views from which you can access system catalog information about each DataJoiner database. Some of these views contain data—for example, data about tables, indexes, and servers—that was accessible only from tables in previous versions of DataJoiner. Other views contain data—for example, data about stored procedures, server options, and server functions—that is now available in Version 2.

Performance enhancements
In addition to general engine performance improvements, this latest version offers new query rewrite capabilities, improved pushdown performance, and remote query caching.

Chapter 1. Overview

DataJoiner is a multidatabase server that provides a new level of remote data access. DataJoiner provides client access to diverse data sources that reside on different platforms, both IBM and multi-vendor, relational and nonrelational. DataJoiner integrates access to distributed data and presents a single database image of a heterogeneous environment to its users. With DataJoiner, you can access all data, remote and diverse, as though the data were local.

Figure 1 shows clients and data sources that are supported by DataJoiner.

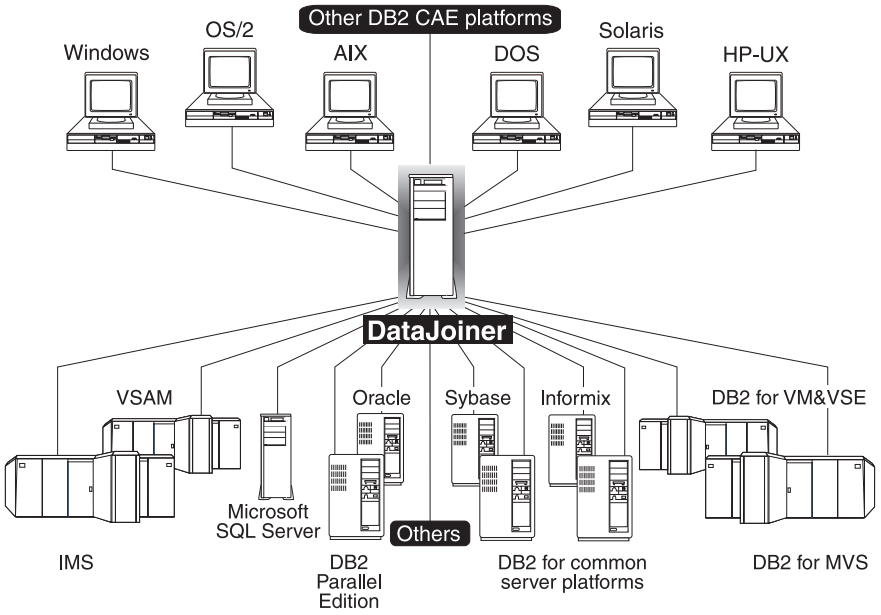


Figure 1. DataJoiner and Its Supported Clients and Data Sources

The DataJoiner Environment

A typical DataJoiner environment consists of a database server, one or more clients (local or remote), and one or more data sources. Users submit queries via clients to DataJoiner; results are returned from data sources. Figure 2 on page 2 provides an overview of DataJoiner query processing.

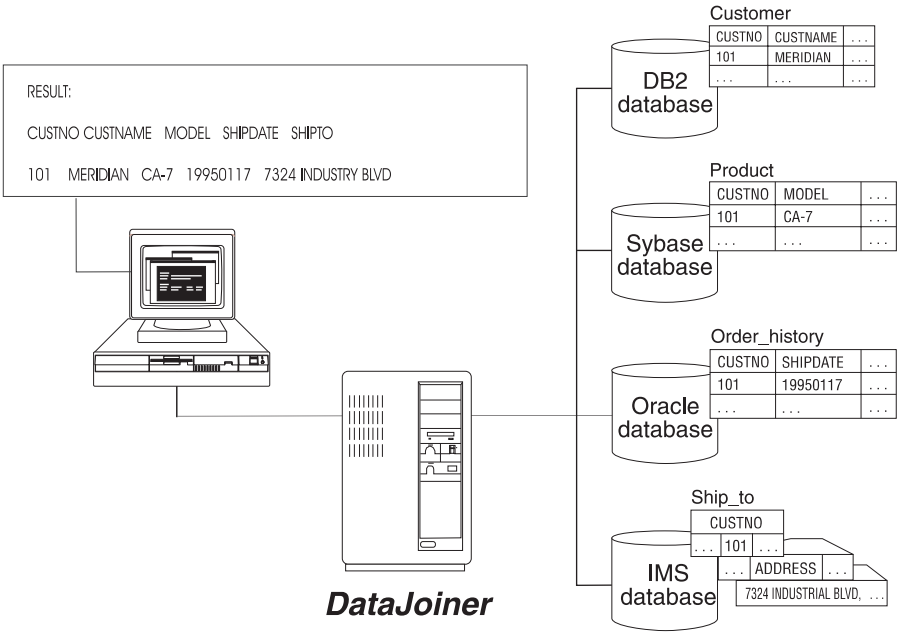


Figure 2. DataJoiner Environment

DataJoiner supports functional compensation and distributed application processing. Functional compensation occurs, for example, when a query contains a function that a data source does not support. For example, many data sources do not support recursive SQL. In this case, data source data is collected at DataJoiner for processing. In other cases, operations are pushed-down to data sources. Pushdown capabilities are one way to distribute processing across data sources and improve query performance. For example, performing sort operations at data sources can reduce the flow of data across a network and speed query results.

From a traditional DB2 for CS client/server environment perspective, DataJoiner is at one node. Applications running on different platforms, via DB2 for CS clients, can transparently and concurrently access data on data sources via this node. From a Distributed Relational Database Architecture (DRDA) perspective, DataJoiner can serve as both a DRDA Application Requestor (DRDA AR) or a DRDA Application Server (DRDA AS). DataJoiner's ability to serve as a DRDA AS lets DRDA AR clients on OS/390, MVS, VSE, and VM access the DataJoiner node and request data from data sources.

Product Components

This section describes DataJoiner components and the relationships between those components.

Databases

DataJoiner's local database capability provides the local data store with full relational database management system support. Typical operations (back-up, restore, transaction processing, and so on) are all provided in addition to DataJoiner remote data access capabilities.

In a DataJoiner environment, databases can be located at the DataJoiner instance itself (*local* database) and at data sources (*data source* databases). Data source databases are managed by their data source (for example, by DB2 for MVS and Sybase** SQL Server), but they can be accessed by DataJoiner.

Each DataJoiner instance must have at least one local database. This local database is the database to which DataJoiner users connect. The local database or databases contain nicknames for data and stored procedures located in data sources. The local DataJoiner database, together with the data source databases, appears to end users as a single database.

Nicknames

DataJoiner uses a nickname scheme that allows users to map a two or three-part table (such as `SERVER.REMOTE_AUTHID.TABLENAME`), view, or stored procedure name to a nickname. Users can subsequently use the nickname in an SQL statement whenever access to the remote table, view, or stored procedure is required. Nicknames are used to provide *location transparency*—the ability to separate the data source location from the address for that data source.

System Catalog Scope and Views

DataJoiner maintains its own system catalog of metadata, which describes the data objects in DataJoiner's integrated database. The catalog contains information about objects in local databases as well as objects that are stored at data sources. Because the catalog contains information about objects in multiple systems, it is often referred to as a *global catalog*.

Certain column information in the system catalog can be updated. This information can be updated with data definition language (DDL) statements. The system catalog can be easily queried through system catalog views. DataJoiner creates and maintains a set of system catalog views for each DataJoiner database. They contain information about indexes, views, tables, packages, columns, and so on. These views are similar to the system catalog views documented in the *DATABASE 2 SQL Reference*. DataJoiner, however, has several additional catalog views and in many cases has extended DB2 for CS views to accommodate DataJoiner functions. The unique and modified views, and their columns, are documented in the *DataJoiner Application Programming and SQL Reference Supplement*.

System catalog views are changed during normal database operations and also by users through DDL commands (DataJoiner SQL commands are also documented in the

DataJoiner Application Programming and SQL Reference Supplement).

Directories

Directories identify DB2 Family databases to which users can directly connect. Directories are necessary for accessing both local and remote DB2 Family databases. When the directory information is complete, access to the database is transparent to the user or application, regardless of where the database physically resides.

DataJoiner uses two types of *database directories* plus a *node directory*. A database directory is a file that identifies the location of one or more databases. A node directory is a file that contains network connection information for remote databases. The *node directory* contains entries for all DB2 Family nodes to which DataJoiner can connect. Only TCP/IP and APPC nodes for the DB2 Family must be cataloged. Each entry contains the remote node's name and its communication information.

DataJoiner also contains a *DCS directory*, which is used exclusively by Distributed Database Connection Services (DDCS). See the *DDCS User's Guide* for more information on the DCS directory.

These directories are useful for DB2 Family access and IBM replication software operations. For example, by using DataJoiner DDCS or DB2 Connect with the Apply and Capture replication programs, DataJoiner can be used as a DDCS data source for data propagation.

See the *DDCS User's Guide* for information on DDCS connections between clients and host databases. See the *DataJoiner Planning, Installation, and Configuration Guide* for your platform for information about configuring DataJoiner to access data sources. See "Chapter 3. Node and Database Directories" on page 41 for additional information about directories.

Configuration Files

DataJoiner has two configuration files that define resources owned by DataJoiner but not associated with configured data sources. The configuration files contain parameter values that define the resources allocated to a DataJoiner instance and to the local databases created under that instance. The two types of configuration files are DataJoiner's *database manager configuration file*, for a DataJoiner instance, and the *database configuration file*, for each local DataJoiner database. To access these two configuration files, use the following DB2 commands:

```
GET DATABASE MANAGER CONFIGURATION
GET DATABASE CONFIGURATION
```

See the *DATABASE 2 Command Reference* for more information on these commands.

DataJoiner's database manager configuration file is created when an instance of DataJoiner is created. The file affects the system resources allocated to DataJoiner for an individual instance. The file's parameters have global applicability, independent of

any one database stored on the system. You can change many of the parameters from the system default values to improve performance or increase capacity, depending on the workstation configuration.

The database configuration file for a local DataJoiner database is created when the database is created and resides where the local database physically resides. (There is one configuration file per database.) This file's parameters specify the amount of resources to be allocated to that database. You can change many of the parameters to improve performance or increase capacity. Different changes might be required depending on the type of activity in that specific local database.

DataJoiner Data Management

In any database, there is data that needs to be managed—imported, exported, backed up, restored, and more. There are two types of data you can manage in DataJoiner:

Catalog data The DataJoiner catalog contains most of the data that describes your multidatabase environment. This data includes server definitions, nicknames, and security information.

User data This is the data stored in any table created in a DataJoiner local database.

There *is* a third type of data available to DataJoiner: the data available at a DataJoiner data source (the servers defined in SYSCAT.SERVERS). In some cases, DataJoiner does provide a way to manage the data in these data sources; however, this data source management capability is limited. Utilities supported at data sources are noted in Table 1 on page 6.

DataJoiner provides utilities to help you manage and maintain the catalog and user data. You can access these utilities through:

- DataJoiner commands run from the command line processor (CLP).

  You invoke the command line processor with the `DB2` command, which is described in the *DATABASE 2 Command Reference*.
- Application Program Interfaces (APIs) run from an application program.
- The Database Director

  This utility provides a graphical user interface for accomplishing common database administration tasks. It is described in the *DATABASE 2 Command Reference*.

  You cannot use the Database Director to configure a data source. See the *DataJoiner Planning, Installation, and Configuration Guide* for your platform to learn about configuring data sources.

Table 1 on page 6 lists data management utilities and indicates where they are supported.

Table 1. DataJoiner Utilities

Utility	Supported at	Description
BACKUP DATABASE	DataJoiner only	Makes a backup copy of a local database.
EXPORT	All data sources	Copies data from a DataJoiner table or view to another database or spreadsheet program.
LOAD	DataJoiner only	Loads data from files, tapes, or named pipes into a DataJoiner table.
IMPORT	All data sources that support LOCK TABLE, DELETE, INSERT, and UPDATE SQL statements	Copies data from another database or spreadsheet program into a DataJoiner table or view.
REORG TABLE	DataJoiner only	Rearranges the data of a table in a local database into a physical sequence according to a specified index. This reorganization can provide faster access to the data, and thereby improve performance. A table can become fragmented due to many updates, causing performance to deteriorate. You should run this utility if you notice a performance impact when accessing a table.
REORGCHK	All data sources	Calculates statistics on the database to determine if tables require reorganization.
RESTORE DATABASE	DataJoiner only	Rebuilds a local database to the state it was in when the backup copy was made. Also, copies a local database to another file system or workstation. You cannot use this utility to restore a data source database.
ROLLFORWARD DATABASE	DataJoiner only	Takes the changes made to the tables that DataJoiner has been recording in a log, and applies these changes to the local database after it is restored with the RESTORE DATABASE utility.
RUNSTATS	All data sources	Updates statistics about the physical characteristics of a table and its indexes. Also, it can gather statistics about data source tables by querying a nickname and then storing statistical data derived from that nickname locally. This utility updates only DataJoiner statistics.

See “Chapter 8. Data Movement and Analysis Utilities” on page 103 for additional information on these utilities in a DataJoiner environment. See the *DATABASE 2 Command Reference* for information on running the utilities, and the *DATABASE 2 Administration Guide* for general information on these utilities.

Application Program Interfaces (APIs)

APIs provide access to DataJoiner facilities from compiled application programs running on the client, the DataJoiner server, or both. APIs typically manipulate the DataJoiner database environment, not the data in the DataJoiner local database.

Environment APIs allow manipulation of the DataJoiner environment, including creating and deleting DataJoiner local databases and scanning database directories.

Utility APIs are used to import and export data, back up and restore local DataJoiner databases, gather statistics for the optimizer, and reorganize the data.

Configuration APIs are used to display or change the DataJoiner configuration file or individual DataJoiner database configuration files.

Client/Server APIs support client-server specific operations, such as stored procedures. They are also known as database application remote interfaces.

Runtime Service APIs provide the run-time interface for precompiled SQL statements. These APIs are not usually called directly by the application programmer; instead, they are inserted into the modified source file by the precompiler after processing. They can also be used by programmers who want to write their own precompilers.

Database System Monitor APIs allow applications to gather statistical information regarding the operation of the local DataJoiner database and DataJoiner itself. Information gathered through these APIs can be used to assess the status of individual databases, tables, and individual applications.

See the *DATABASE 2 Command Reference* for more information on APIs.

Transaction Support

For IBM relational database products, a transaction is commonly called a *unit of work*. A unit of work is a recoverable sequence of operations within an application process. It is the basic building block used to ensure that a database is in a consistent state. Any reading or writing to the database is done within a unit of work. A *point of consistency* (or commit point) is a time when all recoverable data that an application accesses is consistent with related data.

A unit of work (transaction) can involve one or more databases. If a transaction involves two or more databases, it is a *distributed unit of work* (DUOW) transaction.

The DataJoiner database provides a single database image that is composed of nicknames that can represent many different tables at different data sources. It is possible that an application connecting to the DataJoiner database and then reading or updating data sources via nicknames will start a DUOW transaction.

Applications that successfully connect to a DataJoiner database must end transactions by issuing either a COMMIT or a ROLLBACK statement. The COMMIT statement makes permanent all changes made within the transaction. The ROLLBACK statement restores data to the state it was in at the prior commit point. If an application ends normally, without a COMMIT or ROLLBACK statement, a COMMIT will be issued implicitly. If an application ends abnormally while in the middle of a unit of work, the unit

of work is automatically rolled back. Once issued, a COMMIT or ROLLBACK cannot be stopped. A COMMIT makes all updates to the data sources referenced in the transaction permanent, whether it is the local DataJoiner database or the data source databases.

DataJoiner also provides functionality to guarantee *transaction atomicity* when there are updates to multiple data sources (because it provides a single database image of data in one or more data sources). All data sources must commit the transaction or all data sources must rollback the transaction. The functionality DataJoiner uses to guarantee transaction atomicity is called *two-phase commit*.

Two-phase commit protocol ensures atomicity by splitting the commit processing into two phases: prepare and commit. During the prepare phase, DataJoiner polls data sources to ensure they are ready to commit the transaction. The responses to the first phase drive the second phase. In phase two, if all data sources voted that they can commit the transaction, it is committed. If one or more data sources voted that they cannot commit the transaction, the transaction (and associated data source updates) is rolled-back.

For more information on DUOW transactions, see “Chapter 6. Distributed Unit of Work (DUOW) Transactions” on page 59. For additional information on general transaction support, see the *DATABASE 2 Administration Guide*.

Concurrency Control

Concurrent processing means that multiple processes or applications can access the same database at the same time. The locking component in DataJoiner is used locally to maintain data integrity during concurrent processing. Locking

- Guarantees that a transaction maintains control over a DataJoiner database row until it has finished
- Prevents another application from changing a row before the ongoing change is complete

DataJoiner relies on the concurrency control protocols of its data sources for concurrency control at the data source.

DataJoiner provides both table-level and row-level locking for its local database. Row-level locking provides finer granularity and better concurrency support; however, it requires more overhead. If concurrent access is not a prime concern, use the LOCK TABLE statement instead to lock the entire table until the transaction is committed or rolled back.

DataJoiner tables and rows can be locked in either share or exclusive mode. If you choose share mode, other applications can retrieve data as read-only.

DataJoiner uses the isolation-level option to determine the basic locking scheme for cursors in an application. When pushing queries to data sources, DataJoiner attempts

to ensure that the isolation level you request is mapped correctly on the data source database. DataJoiner provides the same levels of isolation as DB2 for CS by formulating requests to the data sources to provide logically “close” degrees of isolation. For information on isolation levels, see “Isolation Levels and Locking” in the *DataJoiner Application Programming and SQL Reference Supplement*. For information on using locks in application programs, see the *DATABASE 2 Application Programming Guide*.

A *deadlock* can occur in a DataJoiner database when two or more transactions are each waiting for data locked by the other. A *deadlock detector* is periodically activated in the background to check the locks currently in the system to determine if there is a deadlock situation. If a deadlock is present, the deadlock detector randomly selects one of the transactions involved in the deadlock and stops it. This transaction is rolled back, and the other transactions can proceed. The frequency of the deadlock detector activity is controlled by a parameter in the database configuration file.

DataJoiner can detect deadlocks involving only local data. DataJoiner relies on its data sources to detect deadlocks between their own resources, and on data source timeout mechanisms to break multidatabase deadlocks.

Data Integrity

Data integrity refers to the accuracy of the values within database tables.

In general, DataJoiner relies on the recovery mechanisms inherent in data source databases to maintain data integrity. DataJoiner also relies on data sources for the roll-forward recovery support of their data.

For DUOW transactions, DataJoiner relies on its two-phase commit functionality and the DUOW support provided in each data source. See “Chapter 6. Distributed Unit of Work (DUOW) Transactions” on page 59 for more information.

DataJoiner ensures data integrity of local data through transaction support, concurrency control, and roll-forward recovery. Roll-forward recovery provides the ability to recover lost online data due to a media failure, such as a hard disk crash. Roll-forward recovery applies log journal information against the restored database. Log journals contain the changes made to the database since the last backup. After these journals are applied, the database is in the same state as it was prior to the failure.

Pass-Through Support

DataJoiner’s *pass-through* function, lets you pass an SQL statement directly to a data source. You might use this function for:

- Data Definition Language, where the types of objects supported vary, depending on the data source
- Data Manipulation Language, where the statement is not supported by DataJoiner

For additional information on pass-through operations, see the *DataJoiner Application Programming and SQL Reference Supplement*.

System Management Facilities

DataJoiner provides several facilities for managing your DataJoiner environment:

- Database Director
- Visual Explain
- DB2 Explain
- Performance Monitor

DataJoiner also provides facilities for diagnosing problems. Each facility is summarized in following sections.

Database Director

The Database Director helps you find and select database objects. It also provides information about the relationships between objects. Visual Explain and other utilities can be started from the Database Director.

A key component of the Database Director is the DBA Utility. It helps you perform common DataJoiner administrative tasks against objects selected in the Database Director. You can configure instances, backup databases, recover table spaces, and so on.

Most of the information for the Database Director is provided in its help system. For more information, start the tool (use the **db2dd** command or select the Database Director icon) and invoke the online help facility.

Visual Explain

Visual Explain helps you analyze and tune SQL statements in a GUI environment. Use it to view the access plan chosen by the DataJoiner optimizer for an explained SQL statement selected from the Database Director. You can also call this utility from the command line (use the **db2vexp** command) to explain a dynamic SQL statement.

Most of the information for Visual Explain is provided in help. For more information, start the Database Director tool (use the **db2dd** command or select the Database Director icon) and invoke the online help facility.

DB2 Explain

DB2 explain (SQL explain) describes the access plan selected for static SQL statements in the packages stored in the system catalog. It is useful for obtaining an explanation of the chosen access plan for packages that do not have existing explain data.

DB2 explain is also used to analyze remote plans from data sources. Remote plans are the access plans chosen by data sources to evaluate queries sent to data sources by DataJoiner. DB2 explain (use the **db2expln** command) and dynamic explain (use the **dynexpln** file) are the only tools that can retrieve remote plans from data sources.

DB2 Explain information is provided in text. For more information about DB2 explain, see “Appendix A. SQL Explain Utilities” on page 145.

Performance Monitor

The performance monitor helps you gather information about DataJoiner and database application performance.

Invoke the database system monitor from the Database Director. For more information, start the Database Director tool (use the **db2dd** command or select the Database Director icon) and invoke the online help facility.

“Chapter 9. Database System Monitor” on page 107 describes system monitor elements applicable for a heterogeneous database environment.

Problem Diagnosis Facilities

DataJoiner problem diagnosis facilities include:

- Online messages
- Error logging facilities
- An independent trace facility
- A process status utility

Some diagnostic facilities are designed to be used under the direction of IBM Service in diagnosing problems.

For more information on diagnosing problems, see the *DataJoiner Messages and Problem Determination Guide*.

Managing Multiple Instances

An individual DataJoiner server is called an *instance*. Each instance can contain one or more databases, with users connected at each instance.

Only one DB2 installation/version can exist on an NT operating system; however, one or more instances can exist for each installation. On UNIX operating systems, one or more DB2 installations can exist with one or more DB2 instances.

See the *DataJoiner Planning, Installation, and Configuration Guide for AIX* for information on creating, configuring, and managing several instances of DataJoiner on

one workstation. See the *DATABASE 2 Application Programming Guide* for information on programming applications to access a particular instance of DataJoiner.

Security

DataJoiner provides authentication and privilege-based security facilities. It also provides additional flexibility to cover database security needs in a distributed environment.

Authentication is the process that DataJoiner uses to ensure that *the user is really the person they claim to be*. Authentication occurs when users attempt to access a database. DataJoiner authorization IDs (or *user names*) and passwords can be authenticated at:

- The client
- DataJoiner
- Data sources
- A combination of the client, DataJoiner, and data sources

DataJoiner server authentication settings, along with the CREATE USER MAPPING statement and server options, can be used to create a secure environment appropriate for your distributed data.

DataJoiner lets administrators define privileges for local objects (such as tables, views, and nicknames). Users must have the appropriate privileges to perform operations against nicknames (similar to table privileges).

DataJoiner privileges apply only to local objects—they do not override or replace data source object privileges. Authorized users can grant privileges for their own objects to other users, but these local privileges do not override privileges on data source objects.

It is possible to use a combination of nickname privileges and pass-through facility privileges to limit access to data source tables. Additional information is provided in “Chapter 2. Security” on page 13

Chapter 2. Security

This chapter describes how to control access to DataJoiner, how DataJoiner controls access within itself, and how to control access to other data sources. It also describes how you can customize access to the databases in your DataJoiner system.

The chapter topics are:

Overview

As shown in "Chapter 1. Overview" on page 1, a traditional, three-tier DataJoiner environment typically involves clients, the DataJoiner database management system, and one or more data sources. Data sources are usually accessed when clients connect to a DataJoiner database containing nicknames. When a user performs SQL operations against a nickname, the object represented by that nickname (such as a table or view) is accessed at the data source. Data sources are typically remote database systems that are managed by other IBM or non-IBM database management systems. Typically, you will have one or more remote data sources and possibly one or more local DataJoiner databases (containing nicknames and other data).

Understanding DataJoiner security requires that you understand two broad topics: managing client access to DataJoiner and managing client access to remote data sources through DataJoiner.

Managing client access to DataJoiner is similar to managing client access to DB2 for CS. First, a user is authenticated. Then, the database manager determines if the user has sufficient privileges to perform operations (SELECT, ALTER, and so on) against database objects. At this point, a user can perform operations against DataJoiner databases and issue SQL statements against DataJoiner objects. This level of distinction is important. A nickname is a DataJoiner object; the object at a data source that is represented by a nickname is not a DataJoiner object.

Managing client access to data sources requires some additional planning and may require a bit more work. After you enable users to access DataJoiner databases, the next step is to consider the security in place for your remote data sources. Most require a user ID and password to access data. Additionally, most require that the user ID and password have a specific authority level or privileges required for viewing and manipulating data after a user is authenticated. Additionally, if you are accessing DB2 for OS/390 or other DRDA data sources using advanced program-to-program communications (APPC), APPC network security rules must be followed.

In summary, you must consider:

- Authentication processing between clients and DataJoiner
- Authentication processing between DataJoiner clients and data sources
- DataJoiner database privileges
- Object privileges at data sources
- Additional requirements (APPC security or security elements unique to your environment)

The rest of this chapter provides information on authentication processing and privileges from a DataJoiner perspective.

DataJoiner and Data Source Authentication

The first step is to ensure that users can access DataJoiner. The process of validating users is called authentication.

Authentication

Access to an instance is first validated outside the database manager. This process, known as *Authentication*, verifies that users are really who they claim to be. All users/applications must be authenticated before database requests are processed.

Figure 3 on page 15 shows how user information is passed to DataJoiner (and possibly on to data sources).

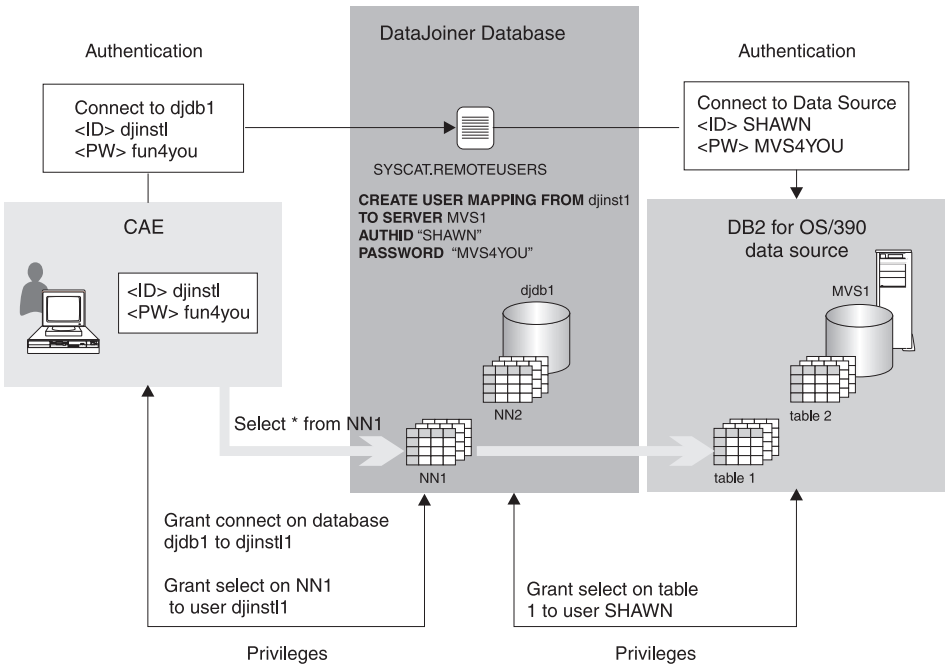


Figure 3. Authentication and Privilege Processing: Overview

Authentication can take place at multiple locations (client, DataJoiner, data sources, or a combination).

Where authentication takes place depends on:

- The client authentication type.
- The DataJoiner authentication type. The authentication type of the DataJoiner database must match the authentication type that is specified when the client catalogs the DataJoiner database.
- Whether a data source is configured to expect a password.
- Whether the DataJoiner password server option is set to send a password to a data source
- APPC security (that is specified for the APPC connections between the client and DataJoiner, and between DataJoiner and the data source).

DataJoiner offers functions helping you enable end-to-end authentication processing (user/client to DataJoiner to data sources). Consider the situation when a user's ID and password are the same for accessing DataJoiner and also for accessing a non-DataJoiner data source. DataJoiner can send the user ID and password used for a connection to DataJoiner directly to the data source. If the password is similar (perhaps only the case changes) a server option (fold_pw) can handle this situation. Of course, if

the user ID and password for DataJoiner and the data source are completely different, that situation can be handled as well. Remote user IDs and passwords can be created with CREATE USER MAPPING statements.

In all cases, the DataJoiner password server option needs to be consistent with how the data source is configured. If the a data source is configured to validate passwords, the password server option needs to be set so that the password will flow to the data source. Data source password options are set in DataJoiner using the CREATE SERVER OPTION statement.

Authentication Types

System administrators can specify the *authentication type*. It determines where and how users are verified. The authentication type:

- Is assigned to the DataJoiner instance when it is created or cataloged (at the client).
- Works with the APPC security type to determine where validation takes place.

The formal definitions for DataJoiner authentication types are:

SERVER Authentication occurs at DataJoiner. Authentication can also occur at the data source, depending on whether the data source was configured to validate users. The ″password″ server option must be set so that it is consistent with the data source. SERVER is the default setting.

Because the authorization name and password flow from the client to DataJoiner, they are available to flow from DataJoiner to the data source. See “What Authorization names and Passwords Flow from DataJoiner to Data Sources?” on page 23 for more information.

CLIENT Authentication occurs at the client (on the node where the application is started) and can also occur at the data source. The authorization name and password specified during a connection attempt are compared with the valid authorization name and password combinations on the client node. They determine if the authorization name is permitted access to the database. An example is when the application issues a CONNECT TO statement. DataJoiner assumes that the user is authenticated at the location they first sign on to. No further authentication takes place at DataJoiner.

When authentication is set to CLIENT, the password used to authenticate the client does not flow to DataJoiner. If a password is required for authentication at a data source, a password must exist in SYSCAT.REMOTEUSERS for the data source. You can add entries to SYSCAT.REMOTEUSERS with CREATE USER MAPPING statements.See “What Authorization names and Passwords Flow from DataJoiner to Data Sources?” on page 23 for more information.

DCS A DCS setting is similar to SERVER. The difference is that a DCS setting causes authentication to take place where the data is located

(at a data source). Authentication does not take place at DataJoiner. Similar to SERVER, a password is expected. The password can be passed from the client to the data source (via DataJoiner); or, a password stored at DataJoiner can be sent. Use the CREATE USER MAPPING statement to store passwords at DataJoiner.

OS/2 Client Note: Client workstations running under Extended Services for OS/2 2.0 and DB2 for OS/2 Version 1 cannot specify an authentication type; therefore, DataJoiner assumes that the authentication type for these clients is SERVER.

DOS or WINDOWS Client Note: You cannot authenticate authorization names or passwords at the client. Authorization names and passwords that are assigned to the DB2USERID and DB2PASSWORD environment variables are sent to DataJoiner unvalidated.

Specifying Authentication for an Instance and Databases

Before you create instances or change authentication settings, consider operating system group and user security for the instance and associated databases. One user must own an instance, and that user's primary group becomes the group that is used to designate the administrators for that instance. This group is critical. Members can perform all possible tasks for that instance and databases within that instance (for example, starting and stopping the instance).

Local operating system commands create the users and groups that will access DataJoiner instances. For specific examples, see the *DataJoiner Planning, Installation, and Configuration Guide* for your platform.

The authentication type for a DataJoiner instance is set when the instance is created. Databases created under an instance inherit instance authentication settings. Use the **db2icrt** command to create instances.

If you need to change the authentication setting, use the **UPDATE DATABASE MANAGER COMMAND**. See the *DATABASE 2 Command Reference* for more information about updating an instance.

Matching Authentication Settings: Server and Clients

Use the CATALOG DATABASE command at client workstations to store information about remote or local database authentication settings. This command catalogs information about databases the client will connect to.

The authentication type specified when the client database is cataloged must match the authentication type specified for the DataJoiner instance databases the client will connect to. The client and DataJoiner authentication types are compared every time the client connects to DataJoiner. DataJoiner refuses the connection request if the authentication types do not match.

Determining Security Requirements

This section is a work sheet to help you determine your security requirements. It answers three important security questions:

- Do I need authorization names and passwords at the DataJoiner server for all my DataJoiner users?
- Should I configure my DataJoiner server with authentication set to CLIENT, SERVER, or DCS?
- Do I need to use CREATE SERVER MAPPING statements to maintain entries in SYSCAT.REMOTEUSERS?

Security at the DataJoiner Server

Questions in this section apply to your DataJoiner server. The answers to these questions can help you determine if your DataJoiner server must be password protected. Use the prefix area before the question to mark 'Y' or 'N'. You will need these answers in “Putting It All Together” on page 19.

__A1. Will your DataJoiner server contain user data that must be password protected? That is, will you use the local data store option of DataJoiner to store user data?

__A2. If the answer to A1 is Yes, will that data require password protection or can anyone with access to the server read it? In most cases, if it contains user data, that data must be password protected.

__A3. Will any of the system data stored in the DataJoiner catalog require password protection? Is it acceptable to allow DataJoiner users to select from the DataJoiner system catalog and see table and view names, column names and attributes, statistics, and so on?

Authorization names at the Data Sources

Questions in this section apply to your authorization names at the data sources. Include your DataJoiner server as a data source for the purpose of these questions, which can help you determine if SYSCAT.REMOTEUSERS needs to be maintained. Use the prefix area before the question to mark 'Y' or 'N'. You will need these answers in “Putting It All Together” on page 19.

__B1. Are authorization names consistent across all data sources? That is, for all users, are their authorization names exactly the same on all data sources?

__B2. If the answer to B1 is No, is the only difference case (entirely lowercase names versus uppercase names)? For example, DB2 for OS/390 data sources require uppercase authorization names; Sybase SQL Server data source authorization names are usually in lowercase. If all authorization names for a given user were converted to uppercase, would they be exactly the same?

Passwords at the Data Sources

Questions in this section apply to password requirements for your data sources. Do not include your DataJoiner server as a data source for the purpose of these questions (unless instructed to do so later in Step 2). Answers to these questions can help you determine if SYSCAT.REMOTEUSERS needs to be maintained. The answers will also help determine the impact of this maintenance on answers to the previous questions. Use the prefix area before the question to mark 'Y' or 'N'. You will need these answers in “Putting It All Together”.

__C1. Are passwords needed on any data source?

__C2. If the answer to C1 is Yes, then considering only those servers in which passwords are required, are passwords consistent across all those servers? That is, for all users, are their passwords exactly the same on all data sources that require passwords?

__C3. If the answer to C2 is No, then is the only difference case (entirely lowercase passwords versus entirely uppercase ones)? For example DB2 for OS/390 data sources require uppercase passwords; Sybase SQL Server data source authorization names are usually in lowercase. If all passwords for a given user were converted to uppercase, would they be exactly the same?

Putting It All Together

Use the results of the questions from the previous sections to determine the correct authentication approach, and whether entries are needed in SYSCAT.REMOTEUSERS.

1. If you answered Yes to C1 and No to both C2 and C3, then the same password provided to DataJoiner cannot be used for all data sources. DataJoiner must use SYSCAT.REMOTEUSERS to identify the correct password for those users. Set your answer to A2 to Yes if it is not currently Yes. Your users must maintain SYSCAT.REMOTEUSERS to allow DataJoiner to select the correct password for each user/data source pair.
2. If you answered Yes to A1, A2, or both, change your answers to questions C1 through C3. This time include your DataJoiner server as a data source. Your answer to question C1 should be Yes, so you should also answer questions C2 and C3. Reread Step 1 with your new answers to C1, C2, and C3 in mind.
3. If your answers to questions B1 and B2 were both No, then the authorization name provided to DataJoiner cannot be used for all data sources. Your users must maintain SYSCAT.REMOTEUSERS to allow DataJoiner to identify the correct authorization name for each user/data source pair.
4. If you answered Yes to questions A1, A2, or both, your DataJoiner database must be configured with authentication set to SERVER. Skip directly to step 8.
5. If you answered No to question C1, then your DataJoiner database must be configured with authentication set to CLIENT. You are finished.
6. If you answered Yes to question C2, C3, or both, your DataJoiner database must be configured with authentication set to DCS. You are finished.

7. Your DataJoiner database must be configured with authentication set to SERVER. Continue with 8.
8. If your DataJoiner database has been configured with authentication set to SERVER then a local authorization name and password is required on the DataJoiner server for each DataJoiner user.

ID and Password Validation and Flow Considerations

This section shows how passwords are validated and then passed to data sources in a three-tier environment.

Password Validation Matrixes

In this section, Table 2 through Table 5 describe several authentication (password validation) scenarios. The scenarios differ, based on the communication protocol used. In the tables, an 'x' indicates that authentication takes place at the client, DataJoiner server, or data source. An 'm' indicates that authentication might take place at the DataJoiner server or a data source. A blank entry indicates that authentication does not take place at the client, DataJoiner server, or data source.

- Table 2 shows how password validation occurs for a non-SNA data source with a TCP/IP client.
- Table 3 shows how password validation occurs for a non-SNA data source with an APPC client.
- Table 4 shows how password validation occurs for a SNA data source with a TCP/IP client.
- Table 5 shows how password validation occurs for a SNA data source with an APPC client.

Table 2. Password Validation: Non-SNA Data Source with TCP/IP Client

	Client		**DataJoiner Server**				**Data Source**
Authentication	**Verify**	**TCP/IP———>**	**Authentication**	**Verify**	**Password**	**SEND**	**Verify**
CLIENT	x	authorization name ————————>	CLIENT		N	authorization name ————————>	
CLIENT	x	authorization name ————————>	CLIENT		Y	authorization name ————————> password	m
SERVER		authorization name ————————> password	SERVER	x	N	authorization name ————————>	

Table 2. Password Validation: Non-SNA Data Source with TCP/IP Client (continued)

Client			DataJoiner Server				Data Source
Authentication	**Verify**	**TCP/IP———>**	**Authentication**	**Verify**	**Password**	**SEND**	**Verify**
DCS		authorization name ————————> password	DCS		N	authorization name ————————>	
SERVER		authorization name ————————> password	SERVER	x	Y	authorization name ————————> password	m
DCS		authorization name ————————> password	DCS		Y	authorization name ————————> password	m

Table 3. Password Validation: Non-SNA Data Source with APPC Client

Client				DataJoiner Server				Data Source
Authentication	**Verify**	**Node Directory Security**	**APPC/ ———> APPN**	**Authentication**	**Verify**	**Password**	**SEND**	**Verify**
CLIENT	x	None or Same	authorization name ————————>	CLIENT		N	authorization name ————————>	
CLIENT	x	None or Same	authorization name ————————>	CLIENT		Y	authorization name ————————> password	m
SERVER		None, Same, or Program	authorization name ————————> password	SERVER	x	N	authorization name ————————>	
DCS		None, Same, or Program	authorization name ————————> password	DCS		N	authorization name ————————>	
SERVER		None, Same, or Program	authorization name ————————> password	SERVER	x	Y	authorization name ————————> password	m
DCS		None, Same, or Program	authorization name ————————> password	DCS		Y	authorization name ————————> password	m

Table 4. Password Validation: SNA Data Source with TCP/IP Client

Client			DataJoiner Server					Data Source
Authentication	**Verify**	**TCP/IP ———>**	**Authentication**	**Verify**	**Password**	**Node Directory Security**	**APPC/ ———> APPN**	**Verify**
CLIENT	x	authorization name ———>	CLIENT		N	Same	authorization name ———>	
CLIENT	x	authorization name ———>	CLIENT		Y	Program	authorization name ———> password	m
SERVER		authorization name ———> password	SERVER	x	N	Same	authorization name ———>	
DCS		authorization name ———> password	DCS		N	Same	authorization name ———>	
SERVER		authorization name ———> password	SERVER	x	Y	Program	authorization name ———> password	m
DCS		authorization name ———> password	DCS		Y	Program	authorization name ———> password	m

Table 5. Password Validation: SNA Data Source with APPC Client

Client				DataJoiner Server						Data Source
Authentication	**Verify**	**Node Directory Security**	**APPC/ ———> APPN**	**Authentication**	**Verify**	**Password**	**Node Directory Security**	**APPC/ ———> APPN**	**Verify**
CLIENT	x	None or Same	authorization name ———>	CLIENT		N	Same	authorization name ———>	
CLIENT	x	None or Same	authorization name ———>	CLIENT		Y	Program	authorization name ———> password	m
SERVER		None, Same, or Program	authorization name ———> password	SERVER	x	N	Same	authorization name ———>	

Table 5. Password Validation: SNA Data Source with APPC Client (continued)

Client				DataJoiner Server					Data Source
Authentication	**Verify**	**Node Directory Security**	**APPC/ ———> APPN**	**Authentication**	**Verify**	**Password**	**Node Directory Security**	**APPC/ ———> APPN**	**Verify**
DCS		None, Same, or Program	authorization name ————> password	DCS		N	Same	authorization name ————>	
SERVER		None, Same, or Program	authorization name ————> password	SERVER	x	Y	Program	authorization name ————> password	m
DCS		None, Same, or Program	authorization name ————> password	DCS		Y	Program	authorization name ————> password	m

What Authorization names and Passwords Flow from DataJoiner to Data Sources?

DataJoiner can send either the authorization name used to connect to DataJoiner or an authorization name defined at DataJoiner. By default, the authorization name used to connect to DataJoiner is sent to data sources. However, if you define a remote authorization ID for a particular user and server, that authorization ID is sent instead. You can define remote authorization IDs with the CREATE USER MAPPING statement.

Password flow logic is similar to authorization flow logic. The exception is that you can decide if passwords flow to data sources in addition to specifying a remote authorization password.

If the DataJoiner password server option is set to 'Y' for a particular data source server, a password can be sent to that data source. The default value is 'N' (no). You can change server options with the ALTER SERVER OPTION statement.

If a password is sent to a data source, the default action is to send the password that is associated with the authorization name that connected to DataJoiner. However, if you define a remote password for a particular user and server, that password will be sent instead. You can define remote passwords with the CREATE USER MAPPING statement.

When a DataJoiner database has an authentication setting of CLIENT, this means that a password does not flow from the client to DataJoiner. If a password is required for a data source, you must define a password at DataJoiner and set the password server option to 'Y'. Define remote passwords with the CREATE USER MAPPING statement. See the *DataJoiner Application Programming and SQL Reference Supplement* for more information on DataJoiner DDL statements.

When Are Entries Required in SYSCAT.REMOTEUSERS?

Entries in SYSCAT.REMOTEUSERS are required when:

- The authorization name at the data source differs from the authorization name at DataJoiner (other than the case).
- The password at the data source differs from the password at DataJoiner (other than the case).

If the authorization name or password differences are simply a matter of case (uppercase or lowercase), you don't have to add an entry in SYSCAT.REMOTEUSERS. DataJoiner can fold authorization names and passwords to uppercase, as described in "Folding Authorization Names and Passwords".

Folding Authorization Names and Passwords

Authorization names and passwords flow from DataJoiner to the data sources. In many cases, no action is required on your part; however, in some cases the names and passwords may need to change. Different data sources have different authorization name and password requirements (regarding the use of uppercase or lowercase).

DataJoiner provides two server options that can help you resolve naming differences (you can view them in the SYSCAT.SERVER_OPTIONS catalog view). The option names are **fold_id** and **fold_pw**, and their settings are:

'U' DataJoiner folds the authorization name or password to uppercase before sending it to the data source.

'N' DataJoiner does not fold the authorization name or password.

'L' DataJoiner folds the authorization name or password to lowercase before sending it to the data source.

null DataJoiner first sends the authorization name or password as uppercase; if that fails, DataJoiner folds it to lowercase and sends it again.

The null setting may seem attractive because it covers many possibilities. However, from a performance perspective, it is best to set these options so that only one attempt is made for connections. If both the fold_id and fold_pw options are set to null, it is possible that DataJoiner will make four attempts to send the authorization name and password:

1. Both authorization name and password in uppercase.
2. Authorization name in uppercase and password in lowercase.
3. Authorization name in lowercase and password in uppercase.
4. Both authorization name and password in lowercase.

There are some special situations that affect your use of these options. For DUOW transactions, the user ID and password used to access a Sybase SQL Server data source (specified with a CREATE USER MAPPING statement) must match the actual user ID and password at the data source server (same case). Consider setting the server options (using SET SERVER OPTION SQL statements) fold_pw and fold_id for this server to N. Alternatively, if the user ID and password are all in one case, then you could set fold_pw and fold_id to L or U, depending on the case.

Additional information on setting the fold_id and fold_pw server options is in the *DataJoiner Application Programming and SQL Reference Supplement*.

General Operational Considerations for Authentication

Operational considerations for authentication include:

- The default setting for the password server option is 'N'. This server option must be changed if a data source expects a password.
- Exercise extreme caution when using CLIENT authentication. Consider this form of authentication only for secure networks. A user has SYSADM authority for the DataJoiner database when the following conditions are met:
  - Authentication is set to CLIENT
  - The user has root status at the client.
  - The user knows the SYSADM's authorization name.
  - The user defines an authorization name on the client that is the same as the SYSADM's on DataJoiner.

- Exercise caution when authentication is set to DCS. Authentication is not done at either the client or at DataJoiner. Any user who knows the SYSADM authentication name can assume SYSADM authority for theDataJoiner database.
- An authorization name can be defined with the same name as a group name. DBADM authority is given to the user and not the group if both exist at run time. To avoid this situation, make sure that the group name is not the same as an authorization name.
- A DB2 for OS/2 application that connects to several databases can be ported to DataJoiner, as is, with certain restrictions. Authentication must be set to CLIENT, and the user must be signed on prior to running the application.
- The effective authorization name is used to verify the user's authorization to perform those functions that do not require a prior connection to a database. Examples include FORCE, CATALOG and CREATE commands. The connection ID is used once a connection has been established.
- Because the Database Application Remote Interface (DARI) requires a connection, the connection ID is used for all functions whether or not the function itself requires a prior connection.
- You cannot perform SYSADM functions with the DARI when authentication is set to CLIENT. Users, when using DARI, can manipulate databases other than the one to which they are connected. The other database might have an authentication type

different from the authentication type of the database to which the user is connected. This restriction helps maintain a secure environment.

- On AIX operating systems, if a server is a Network Information Service (NIS) or Network Information Service Plus (NIS +) client, the SYSADM group should be defined at the NIS or NIS + server and not at the client. To tell if NIS or NIS + is running, use the `ypwhich` command. `ypwhich` reports the name of the NIS or NIS + server if NIS or NIS + is running.

Operating System Considerations for Authentication

This section addresses operating system group and user security considerations.

Groups

DataJoiner can consider group membership when determining what authorization a particular user has. DataJoiner group support is optional and varies by operating system.

Windows NT Operating Systems

DataJoiner group support on NT matches DB2 for CS group support. Users in the NT Administrators group have SYSADM authority for the instance.

AIX Operating Systems

Group support for DataJoiner on AIX operating systems is available. It is controlled using the DB2GROUPS setting in the db2profile shell script. By default, group support is set to OFF because the AIX **mkgroup** command (by default) is available to any user.

To ensure only authorized groups are recognized by DataJoiner, either:

- Change the permission on the mkgroup command so that only trusted users can make groups. To do this:
  1. login as root
  2. enter

     ```
     cd /usr/bin
     ```

     (or equivalent)
  3. enter

     ```
     chmod u-s mkgroup
     ```
- Leave DataJoiner group support OFF

User and Group Requirements

Users accessing DataJoiner, when authentication is set to SERVER, must have an authorization name and password defined using the operating system security facilities.

Authorization names and passwords are not required if the authentication setting is CLIENT or DCS.

If group names are used to grant authorizations to database resources (see “Groups” on page 26), then these groups must be defined and managed using the operating system security facilities. For group authorizations, authorization names are required even if authentication is set to CLIENT or DCS.

Any user or group names defined at DataJoiner must be defined in lowercase within the DataJoiner operating system security facilities. Don’t use delimited identifiers when referring to these names.

For information on situations in which DataJoiner converts the case of authorization names, group names, and passwords sent to data sources, see “Folding Authorization Names and Passwords” on page 24.

OS/2 and DOS Client Note: Because those client’s security facilities always convert authorization names and passwords to uppercase, adhere to the following rules when you create a user:

- Create the user with an authorization name that is the lowercase version of the OS/2 or DOS client.
- Assign this user a password that is the uppercase version of the password for the OS/2 or DOS client authorization name.

Data Source-Specific Considerations for Authentication

This section covers authentication information specific to particular data sources.

Informix 5 Considerations for AIX Operating Systems

Specify a login and password in the .netrc file of any UNIX operating system user running applications that connect to Informix 5 databases. Either the .netrc file or the /etc/hosts and /etc/hosts.equiv files must be set up. This section assumes you are using a .netrc file.

A DataJoiner instance is an Informix application if it access an Informix database. Therefore, when accessing an Informix 5 database, an instance needs an entry in its .netrc file for the system where the Informix 5 database resides. The impact of this relationship is:

- DataJoiner does not use the entries in SYSCAT.REMOTEUSERS for Informix 5 databases because Informix 5 only uses the information from .netrc.
- All DataJoiner users share the same .netrc login and password when accessing an Informix 5 database from a given DataJoiner instance.
- The DataJoiner authentication type has no impact on password flow to Informix 5.

- All DataJoiner users have the same authorizations at Informix because there can be only one entry in .netrc related to a given Informix 5 database.

  This situation could pose a potential security problem; however, DataJoiner provides authorization control on nicknames and PASSTHRU SQL statements. Use REVOKE PASSTHRU SQL statements to ensure that users cannot pass-through to an Informix 5 database. Then, have all Informix authorizations granted to the login specified in .netrc. The *DataJoiner Application Programming and SQL Reference Supplement* contains more information about DataJoiner SQL statements.

See your Informix books for more information about the security requirements of an Informix 5 database.

Classic Connect Considerations

Sessions running under unguarded resident services are handled with no authorization checking. If you configure a DMSI to run as an unguarded service, do not create DataJoiner databases with authentication set to DCS. With authentication set to DCS, neither DataJoiner or Classic Connect will check authorizations for your data.

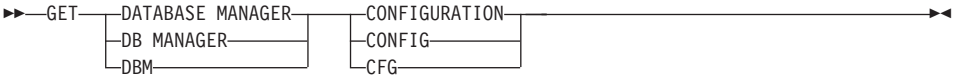
For more information, see the *DataJoiner Classic Connect Planning, Installation, and Configuration Guide*.

Authentication and Connection Commands and Syntax

Querying for the Authentication Type

You can retrieve the authentication type by using the following command line processor command:

GET DB MANAGER CONFIGURATION



The authentication type is returned.

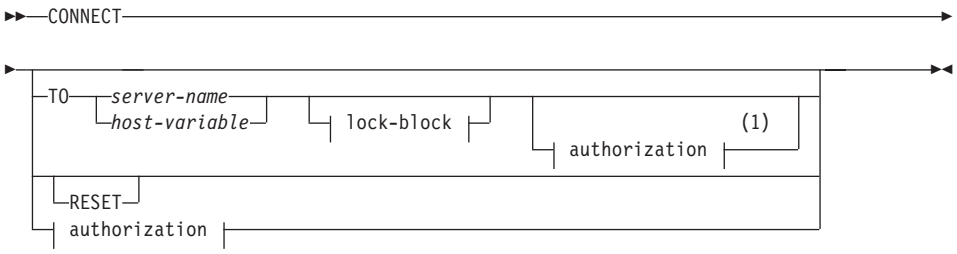
See the *DATABASE 2 Command Reference* for a command output sample.

Connecting to the DataJoiner Database

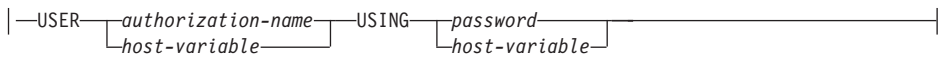
To use DataJoiner's database, you must connect to it either implicitly or explicitly. An implicit connection occurs when you issue an SQL statement without being connected to any database (and implicit connect is enabled). DataJoiner connects you to the default database. To connect to a database other than the default, you must use the CONNECT command to explicitly connect to the database of your choice.

CONNECT

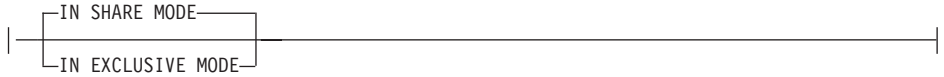
CONNECT



authorization



lock-block



Notes:

1. This form is only valid if implicit connect is enabled.

Table 6 on page 30 shows various forms of CONNECT and how they relate to authentication. Table 6 on page 30 also lists the *connect type* of each statement. There are two connect types:

Connect Type 1 — DataJoiner uses the authorization name and password specified on the CONNECT statement.

Connect Type 2 — DataJoiner uses the local authorization name on the CONNECT statement. See “What Authorization names and Passwords Flow from DataJoiner to Data Sources?” on page 23 for information on authorization names that flow from DataJoiner to a data source.

Table 6. Connection and Authentication

Statement	Notes	Authentication Type Supported	Connect Type
CONNECT TO *server-name* **USER** *authorization name* **USING** *password*	• Server name explicitly supplied. • Authorization name and password explicitly supplied. • User may or may not be signed on.	• SERVER • CLIENT • DCS	Type 1
CONNECT USER *authorization name* **USING** *password*	• Server name not supplied. • Authorization name and password explicitly supplied. • If there is not an existing connection, the default database is used (as specified by the environment variable `DB2DBDFT`). • If there is an existing connection, the existing database is connected to again.	• SERVER • CLIENT • DCS	Type 1
CONNECT TO *server-name*	• Authorization name or password are not specified. • SERVER and DCS cannot be supported if the client is remote since the password is not available. SERVER and DCS are supported if the client is local. • CLIENT is supported but only if the user is signed on.	• CLIENT • SERVER and DCS can be supported if the client is local.	Type 2

Table 6. Connection and Authentication (continued)

Statement	Notes	Authentication Type Supported	Connect Type
CONNECT	• User is not connected to any database. • Implicit connect must be enabled. • Default database is used. • Authentication information is retrieved from the security system. • SERVER and DCS cannot be supported because the password is not available. • CLIENT supported if the user is signed on.	• CLIENT • SERVER and DCS can be supported if the client is local.	Type 2
CONNECT	• If user is connected, this form is used to query who is connected. • No actions related to authentication are taken.	N/A	N/A
CONNECT	• User is not connected. • The implicit connect environment variable (`DB2DBDFT`) is not defined. • No action related to authentication needs to be taken.	N/A	N/A

Authorization, Authorities, and Privileges

Authenticated users must also have the appropriate *authorization* to access DataJoiner databases and objects (nicknames, tables, views, and so on). Authorization to access objects is managed by authority levels and privileges. *Authority levels* provide a method of grouping privileges and higher level database manager operations. Named *authorities* are used to group specific sets or privileges (the SYSADM authority is the highest authority level). *Privileges* are set at a lower level and manage authorization to specific database objects and resources. Individual privileges are granted to users.

One way to understand the relationship between authentication, local DataJoiner authorities and privileges, and data source privileges is to consider security from four levels:

DataJoiner Access can be controlled with the workstation user name and password. User must meet authentication requirements.

DataJoiner data The user must have access to both the DataJoiner database and the appropriate data (tables *and* nicknames) within that database. An authenticated user must have privileges (such as SELECT) for DataJoiner objects or the appropriate authority level.

Data sources The user may need to authenticate their user name/password at the data source as well as at DataJoiner.

Data source data The user must have been granted access to the appropriate data source data to perform operations against that data. Privileges granted at DataJoiner do not imply that a user has the same level of privileges at the data source. If a user has the SELECT privilege at DataJoiner for a nickname but lacks privileges for the data source table referenced by that nickname, the query will fail.

Privileges and authorities combine to form a controlled access system for database objects. Given access to a database, a user has access to the database objects only as permitted by the privileges granted to the user.

A user can be authorized for any combination of individual privileges or administrative authorities. When a privilege is associated with a resource, that resource must already exist. For example, a user cannot be given the SELECT privilege on a table unless that table has previously been created.

See the *DATABASE 2 Command Reference*, the *DATABASE 2 Application Programming Guide*, or the *DataJoiner Application Programming and SQL Reference Supplement* for information about required authorities or privileges for a particular command or SQL statement.

Authorities

Administrative authorities are sets of privileges covering a set of objects. Four administrative authorities are supported: SYSADM, SYSCTRL, DBADM and SYSMAINT. Users with these authorities can grant or revoke privileges to or from other users.

There are four administrative authorities (that are fully documented in the *DATABASE 2 Administration Guide*). DataJoiner-specific concepts for each authority are:

SYSADM authority SYSADM is the highest level, with control over all resources created and maintained by DataJoiner. This authority level is assigned to the group specified by the *sysctrl_adm* configuration

parameter. See the *DataJoiner Planning, Installation, and Configuration Guide* for your platform for instructions on adding users to groups.

A single user can have SYSADM authority for a number of instances of DataJoiner on the same node. Only users with SYSADM, SYSCTRL, or SYSMAINT authority can start and stop DataJoiner.

SYSCTRL authority

SYSCTRL is the highest level of system control authority. Give users this authority when they must perform utility and maintenance operations against a database manager and databases containing sensitive information. This authority level is assigned to the group that is specified by the *sysctrl_group* configuration parameter. See the *DataJoiner Planning, Installation, and Configuration Guide* for your platform for instructions on adding users to groups.

SYSMAINT authority

SYSMAINT is the second level of system control authority. Grant users this authority when they must perform a subset of utility and maintenance operations (such as running a trace) against a database manager and databases containing sensitive information. This authority level is assigned to the group specified by the *sysmaint_group* configuration parameter. See the *DataJoiner Planning, Installation, and Configuration Guide* for your platform for instructions on adding users to groups.

DBADM authority

DBADM is the second highest level of administrative authority. It is specific to a single DataJoiner database. DBADM authority can be given to any user or group.

Privileges

The capability to create or access a resource is a privilege. A privilege permits an authorization name to act on a type of object. There are several privilege types:

- Database privileges control the access and creation of databases.
- Table privileges control the access and creation of tables.
- View privileges control the access and creation of views. You can use views to control access to a table. By granting access to a view instead of to the underlying tables, you can restrict access to the rows and columns selected by the view definition.
- Nickname privileges control the access and creation of nicknames.
- Package privileges control the creation, modification, and execution of packages.

- Index privileges control the creation of indexes.
- The pass-through privilege controls direct access to data sources.

Database resources are hierarchical, and so are the access privileges of those resources. Figure 4 illustrates the hierarchical relationship of access authorities and privileges.

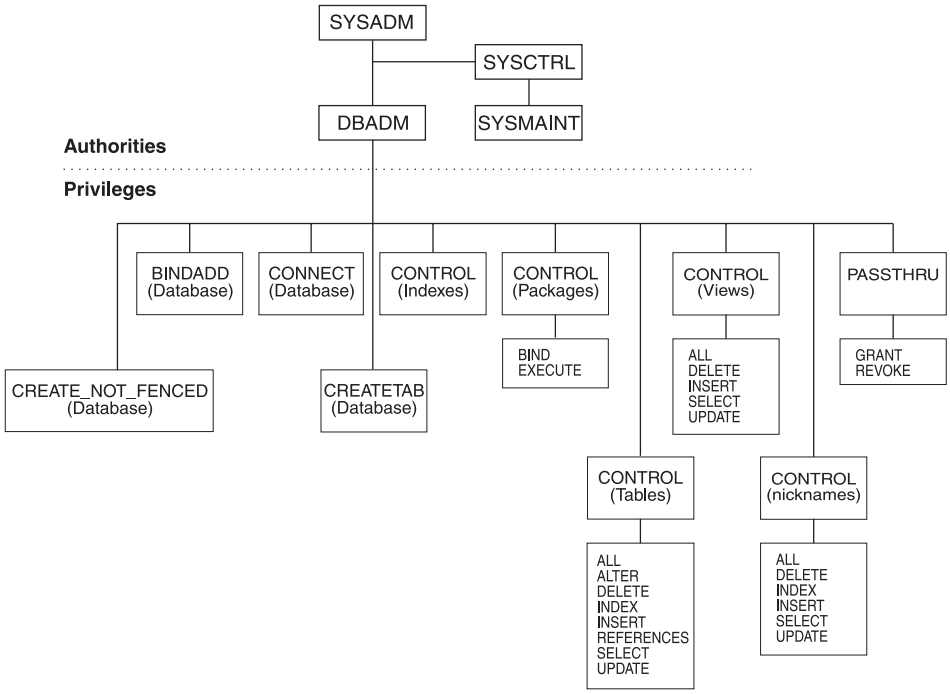


Figure 4. Hierarchy of Authorities and Privileges

See the *DATABASE 2 Administration Guide* for more information on authorities and privileges.

Ownership (CONTROL) privileges

The CONTROL privilege is automatically granted to the creator of an object (although some views are exceptions to this rule). This privilege is equivalent in all respects to ownership of the object. It includes the right to access an object in any way and to grant privileges on the object. Privileges are controlled by users with ownership or administrative authority. These users can grant or revoke privileges to or from other users.

Individual privileges

Individual privileges allow a specific function, sometimes on a specific object.

Implicit privileges Users can be implicitly allowed to exercise privileges if they have the privilege to execute a package that requires those privileges. This kind of authorization is controlled by privileges to create and execute packages.

Two SQL statements control privileges. The GRANT statement gives privileges to a user, and REVOKE takes them away.

It is possible to use DataJoiner privileges to control access to data sources. See “Options for Controlling Access to Distributed Database Objects” on page 37 for details.

Nickname, Stored Procedure Nickname, Table, and View Privileges

Nickname, stored procedure nickname, table, and view privileges limit the actions authorized users can perform on those objects. Accessing objects requires the appropriate privileges at both the data source and at DataJoiner.

Additional information on authorization requirements is in the *DATABASE 2 Administration Guide*.

For stored procedure nicknames, the authorized user creating the nickname must have the SELECT privilege on the data source’s system catalogs. The user must also have authorized access to the stored procedure itself. Sybase SQL Server, in addition, requires that users have the EXECUTE privilege for remote procedures.

Some table privileges are not valid for views or nicknames, as shown in Figure 4 on page 34 . For more information on creating nicknames, see the *DataJoiner Application Programming and SQL Reference Supplement*.

Package Privileges

Package privileges govern the ability to control, bind, and execute packages. DataJoiner package privileges and overall package behavior is similar to information documented in the *DATABASE 2 Administration Guide* with some differences.

If a package contains nicknames, the package owner must have the appropriate privileges or authority level for the nicknames within the package. At a minimum, the package owner requires SELECT authority.

Users or applications that use a package containing nicknames don’t need additional privileges or an authority level for the nicknames within the package. However, package users connecting to DataJoiner to process the package must have the EXECUTE privilege for that package.

The package owner does not need privileges or an authority level for objects referenced by the nicknames at data sources. However, package users must have the appropriate privileges or authority levels for data source objects at the data source.

It is possible that DataJoiner packages may require additional authorization steps because DataJoiner uses dynamic SQL when communicating with DB2 data sources. The authorization ID running the package at the data source must have the appropriate authority to execute the package dynamically at that data source. See the *DataJoiner Application Programming and SQL Reference Supplement* for more information about how DataJoiner processes static and dynamic SQL.

Index Privileges

Index privileges involve the ability to create and drop indexes. Index privileges can be defined for index information maintained by DataJoiner that was retrieved from a data source when a nickname was created. Note that the indexes do not exist at DataJoiner.

Pass-through Privileges

DataJoiner pass-through privileges control which authorization IDs can issue SQL statements directly to data sources. Use them to control the ability to pass DDL and DML statements directly to data sources.

DataJoiner provides four SQL statements to support pass-through operations:

- GRANT PASSTHRU, which grants the authority to issue PASSTHRU SQL statements against a data source
- REVOKE PASSTHRU, which revokes the authority to issue PASSTHRU SQL statements against a data source
- SET PASSTHRU, which establishes a pass-through session
- SET PASSTHRU RESET, which ends a pass-through session

A sample statement granting pass-through authorization to the user Shawn for the server ORACLE1 is:

```
GRANT PASSTHRU ON SERVER oracle1 TO USER shawn
```

For complete information on the syntax of these statements, see the *DataJoiner Application Programming and SQL Reference Supplement*.

Pass-through Operations and Security

Within any pass-through session:

- DataJoiner processes all static SQL statements.
- Dynamic SQL statements prepared within the pass-through session are processed at the data source.

  If an SQL statement prepared within the pass-through session executes *after* the session ends, the statement ends in SQLCODE -901.
- Dynamic SQL statements not prepared within the pass-through session are processed by DataJoiner.

Options for Controlling Access to Distributed Database Objects

The DataJoiner environment introduces several new security concepts that extend standard DB2 for CS database object control. Some options/ideas to consider include:

- Limiting authorization for pass-through sessions. If your environment requires that many applications have direct (pass-through) access to a data source (complex DDL operations, unique SQL syntax), consider limiting pass-through privileges. Set up one ID at the data source with the privileges required by all incoming applications. Then, restrict access to that data source ID with PASSTHRU statements at DataJoiner.
- Single user approach. One possible security approach is to ensure that only one user at the DataJoiner database has SELECT privileges on nicknames. Then, revoke PASSTHRU privileges for all other users. That one user (excepting SYSADM and DBADM users) is the only user that can access all the DataJoiner data sources.
- Providing access to system catalog views. Users may need access to DataJoiner catalog views (or perhaps selected columns within those views) when creating applications. Although the method for granting access to DataJoiner catalog views is similar to that for DB2 for CS catalog views, there are several new views and new columns. See the *DataJoiner Application Programming and SQL Reference Supplement* for information on the catalog views unique to, or modified by, DataJoiner.
- Creating multi-location views. Because DataJoiner can access data in multiple data sources, a typical practice is to create *multi-location views*, where the view definition is composed of metadata from one or more data sources. Such views are useful when joining information in columns of sensitive tables across a distributed environment.

Examples

The following examples provide an overview of two security scenarios and outline required steps to ensure user access to data.

Two Remote Data Sources With Similar IDs and Passwords

In this scenario, the task is to enable user Shawn to perform a UNION operation against two nicknames (SYREM1 and DB2REM1). The nicknames represent two tables at different data sources. One data source is a Sybase SQL Server where Shawn's ID and password are expected in lower case. The other data source is DB2 for OS/390 where Shawn's ID and password are expected in upper case. DataJoiner authentication is set to SERVER. The DataJoiner database name is DJ1. Shawn will access DataJoiner from a Windows NT client across a TCP/IP connection. The connection from DataJoiner to DB2 for OS/390 is TCP/IP.

Assuming that Shawn has the proper privileges at both data sources, first ensure that DataJoiner is expecting a password and that a password is being sent. Also, ensure

that the client and server authentication types match. Check the DataJoiner server authentication type by issuing the command:

```
GET DATABASE MANAGER CONFIGURATION
```

from the DataJoiner server. Check the client authentication type by issuing the command:

```
LIST DATABASE DIRECTORY
```

from the client. In both cases, you would look for authentication set to SERVER. If the setting for the client is DCS or CLIENT, you can change it by using the UNCATALOG DATABASE and CATALOG DATABASE commands.

Next, ensure that passwords will be sent to the data sources. Issue the commands:

```
CREATE SERVER OPTION password FOR SERVER SYBASE1 SETTING 'Y'
CREATE SERVER OPTION password FOR SERVER MVS1 SETTING 'Y'
```

Assume that Shawn logs in to Windows NT each day with a valid user ID and password. The next step is to define privileges allowing Shawn to connect to DJ1 and select the specified nicknames. Login at DataJoiner with a user ID that has one of the administrative authorities (SYSADM or DBADM) and issue the SQL statements:

```
GRANT CONNECT ON DATABASE DJ1 To SHAWN;
GRANT SELECT ON syrem1 TO USER SHAWN;
GRANT SELECT ON db2rem1 TO USER SHAWN;
```

Now, enable access to the DB2 for OS/390 data source. There are several steps required to set up access to a DB2 for OS/390 data source. Details are in the *Planning, Installation, and Configuration Guide* for your platform. For now, realize that you must catalog the TCP/IP node at DataJoiner using correct security settings. An example catalog command is:

```
DB2 CATALOG TCPIP NODE db2node REMOTE DB2TCP SERVER MVS1
```

Now, create server options to change the case of Shawn's ID and password. You set them with the commands:

```
CREATE SERVER OPTION fold_id TO 'L' FOR SERVER SYBASE1
CREATE SERVER OPTION fold_pw TO 'L' FOR SERVER SYBASE1
CREATE SERVER OPTION fold_id TO 'U' FOR SERVER MVS1
CREATE SERVER OPTION fold_pw TO 'U' FOR SERVER MVS1
```

At this point, Shawn can access data on both data sources.

Two Remote Data Sources: Mixed IDs and Passwords

In this scenario, the task is to enable user Shawn to perform a JOIN operation against two nicknames (MSREM1 and ORAREM1). They represent two tables at different data sources. Microsoft SQL Server is the first data source. It expects Shawn's ID and password in lower case. The other data source is managed by an Oracle 7 database

manager. For Oracle, an ID and password are required and both are different from the ID and password that are used for DataJoiner. DataJoiner authentication is set to SERVER. The DataJoiner database name is DJ1. Shawn will access DataJoiner from a Windows NT client across a TCP/IP connection.

Assuming that Shawn has the proper privileges at both data sources, ensure that DataJoiner is expecting a password and that a password is being sent. Also, ensure that the client and server authentication types match. Check the DataJoiner server authentication type by issuing the command:

```
GET DATABASE MANAGER CONFIGURATION
```

from the DataJoiner server. Check the client authentication type by issuing the command:

```
LIST DATABASE DIRECTORY
```

from the client. In both cases, you would look for authentication set to SERVER. If the setting for the client is DCS or CLIENT, you can change it by using the UNCATALOG DATABASE and CATALOG DATABASE commands.

Next, ensure that passwords will be sent to the data sources. Issue the commands:

```
CREATE SERVER OPTION password FOR SERVER ORA1 SETTING 'Y'
CREATE SERVER OPTION password FOR SERVER MS1 SETTING 'Y'
```

Now, define privileges allowing Shawn to connect to DJ1 and select the specified nicknames. Login at DataJoiner with a user ID that has one of the administrative authorities (SYSADM or DBADM) and issue the SQL statements:

```
GRANT CONNECT ON DATABASE DJ1 To SHAWN;
GRANT SELECT ON MSREM1 TO USER SHAWN;
GRANT SELECT ON ORAREM1 TO USER SHAWN;
```

Enable DataJoiner to map Shawn's DataJoiner ID and password to the correct password for Oracle.

```
CREATE USER MAPPING FROM "SHAWN" TO SERVER ORA1 AUTHID "scott"
PASSWORD "tiger"
```

Create server options to control the case of Shawn's ID and password when they are sent from DataJoiner to the data sources. Use the commands:

```
CREATE SERVER OPTION fold_id TO 'N' FOR SERVER ORA1
CREATE SERVER OPTION fold_pw TO 'N' FOR SERVER ORA1
CREATE SERVER OPTION fold_id TO 'L' FOR SERVER MS1
CREATE SERVER OPTION fold_pw TO 'L' FOR SERVER MS1
```

At this point, Shawn can access data on both data sources.

Chapter 3. Node and Database Directories

This chapter describes the organization of the DataJoiner node and database directories, and their relationship to the SYSCAT.SERVERS and SYSCAT.REMOTEUSERS catalog views. Node and database directories provide communication information used by clients when accessing DataJoiner and DataJoiner when accessing data sources.

The focus of this chapter is communication flows between DataJoiner and data sources. For information about setting up your communications protocol to support database clients, see the section about configuring clients to DataJoiner in the *DataJoiner Planning, Installation, and Configuration Guide* for your platform.

Node Directory

Each DataJoiner instance maintains one node directory. The node directory contains entries for all nodes containing DB2 Family database directories that DataJoiner can access. The node directory is used to obtain communication information for network connections when the database being accessed is remote. DataJoiner creates or updates the node directory when remote nodes are cataloged. In a DataJoiner environment, the 'NODE' column in SYSCAT.SERVERS references the node directory for information on DB2RA [1] and DRDA protocols. Only TCP/IP and APPC nodes for the DB2 Family must be cataloged.

DB2 Data Source Access Note: DataJoiner provides access to DB2RA and DRDA data sources (DB2) in two ways: using nicknames and pass-through statements or using DRDA Application Requester (DRDA AR) functionality provided in DataJoiner. Your decision to use one or both methods depends on application requirements. Using DRDA AR functionality, for example, is appropriate for use with existing DDCS applications. Configuration information is provided in the *DataJoiner Planning, Installation, and Configuration Guide* for your platform.

DataJoiner automatically creates the node directory when the first remote node is cataloged. The node directory file is placed in the a directory owned by the DataJoiner instance. The user does not need to be connected to a database when cataloging a remote node.

Node Note: Nodes for data source databases *other than* the DB2 Family of databases (for example Sybase and Oracle) should *not* be cataloged in the node directory, even if DataJoiner will reference them.

1. DB2RA refers to the DB2 format and protocol sent between clients and servers. It is proprietary in nature and analogous to Oracle's SQL*NET and Sybase's Open Client/Open Server formats and protocols, which are also proprietary. DataJoiner uses DB2RA to natively communicate between clients and DataJoiner, and between DataJoiner and DB2 for CS data sources.

Cataloging Remote Nodes

Cataloging a remote node creates an entry in the node directory of the local instance of DataJoiner. The node entry contains the information required to establish the connection between DataJoiner and a remote IBM database. Parameters specified in the node directory must correspond to the parameters specified in the communication definitions. Only a user with SYSADM authority can catalog a remote node.

To establish a connection between the DataJoiner node and the data source node, you only need to catalog the data source node in DataJoiner's node directory. You do not need to catalog the DataJoiner nodes in the data source's node directory.

Uncataloging Remote Nodes

Uncataloging a remote node deletes the node name from the node directory on a file system. After a remote node has been uncataloged, any IBM databases residing on the remote node can no longer be accessed from DataJoiner. A user must have SYSADM authority to uncatalog a remote node.

Database Directories

Database names and locations must be stored at each system that accesses the databases. The database directories maintain this information. Database directories contain entries for all databases that can be accessed from the local system.

In general, only DataJoiner databases need to cataloged in the database directory. The exception is the DB2 Family databases that need to be accessed using DataJoiner DRDA AR functionality. If data source databases are going to be accessed using nicknames, they do not need to be cataloged. Data source databases other than the DB2 Family (such as Sybase and Oracle) never need to be cataloged in the database directory.

DataJoiner creates database directories when creating databases or processing catalog database commands. These directories are maintained outside of the database. In a DataJoiner environment, there are two types of database directories:

- System
- Local

System Database Directory

The *system database directory* is created in a directory owned by the DataJoiner instance. There is one system database directory per DataJoiner instance. It is used to access databases created for that instance of DataJoiner. It is also used to access remote databases that are using the DataJoiner DRDA AR functionality. Entries are not required for DB2 Family data source databases accessed through nicknames or pass-through sessions.

Each system database directory entry contains the database name, the database alias name, the entry type, where the local database directory is located (if indirect), the name of the node where the database is located (if remote), and other system information.

Local Database Directory

The *local database directory* resides in every subdirectory that contains a locally created database. It contains the name of the database and its physical location. An entry is added to it when a local database is created.

It is used to access the databases in that subdirectory. Each local database directory entry contains the database name, the database alias name, the entry type, the name of the file system directory where the database files are stored, and other system information. The database alias name is always the same as the database name in this directory. The entry type is always specified as “Home.”

Relationships between Node and Database Directories

The first two sections in this chapter described node directories and database directories, and how they are used in a DataJoiner environment. This section illustrates how these directories are related.

Client and DataJoiner Nodes and Directories

Figure 5 on page 44 shows the relationship between the node and database directories at the client, and the database directories at the DataJoiner instance.

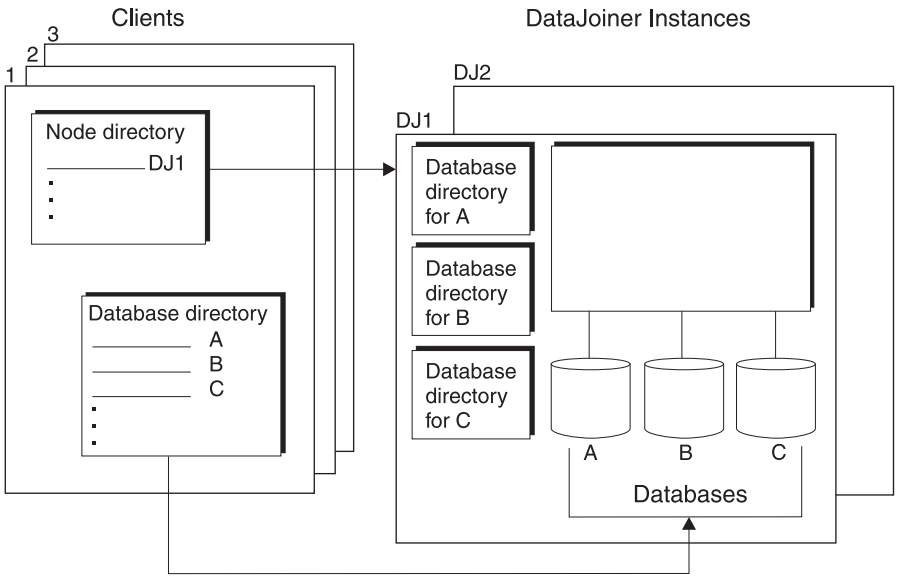


Figure 5. Client and DataJoiner Node and Database Directories

Figure 5 Notes:

- The client's node directory identifies which DataJoiner instances this client can talk to.
- The client's database directory contains an entry for each DataJoiner database to which this client can connect.
- There is a 1:*n* ratio between entries in the client's node directory and the client's database directory. In this example, for Client 1, the node directory shows one entry for DJ1, and three entries in the database directory for databases A, B, and C.
- DataJoiner's database directories contain information about the databases on this instance. There is usually a 1:1 ratio of DataJoiner database directories and databases.

DataJoiner and Data Source Nodes and Directories

Figure 6 on page 45 shows the relationship between DataJoiner's node and database directories, and the database directories at IBM data sources.

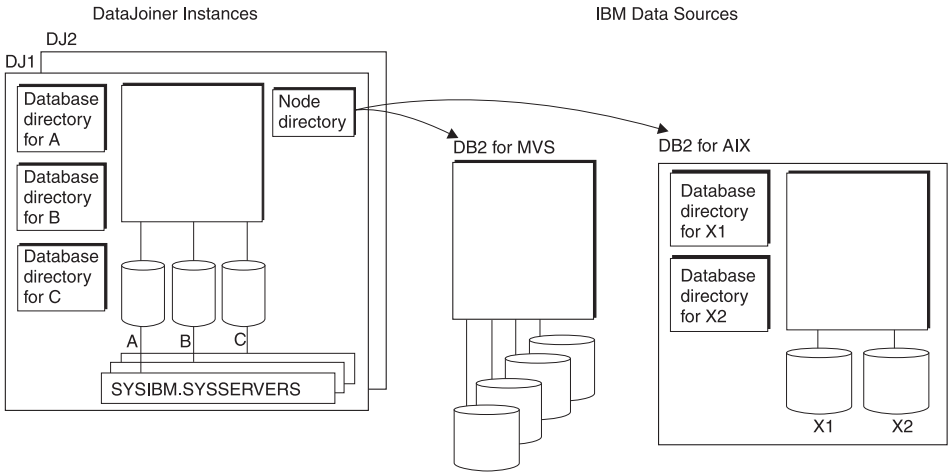


Figure 6. DataJoiner and Data Source Nodes and Directories

Figure 6 Notes:

- DataJoiner's node directory identifies which IBM data sources this DataJoiner instance can talk to.
- DB2 for OS/390, DB2 for OS/400, and DB2 for VM and VSE data sources do not have database directories. After you connect to one of these systems, you can access any database within that system. Therefore, database directory entries are not required for those data sources.
- The SYSCAT.SERVERS catalog view describes which databases this DataJoiner instance can connect to. For example, if Database C on DJ1 connects to the databases on DB2 for MVS and the two databases on DB2 for AIX, Database C's SYSCAT.SERVERS catalog view would contain three entries: one for the entire DB2 for MVS system and two for the databases located at the DB2 for AIX instance.
- The 'NODE' column in SYSCAT.SERVERS identifies the node directory entry to use for that server.

DB2 for OS/390 Note: Do *not* confuse DataJoiner's node and database directories with the DB2 for MVS directory, which is a database and not related to DataJoiner's directories.

Managing Your Database Directories

To add or delete entries to your database directories, use the following commands:

- CATALOG DATABASE
- UNCATALOG DATABASE

The following sections discuss cataloging and uncataloging databases in your system database directory. For more information about the commands listed above, see the *DATABASE 2 Command Reference*.

Cataloging a Database

When you catalog a database, information about the database is stored in the database directories. The database might have been created at another location. When a database is created, the database is cataloged on the local node automatically. Therefore, the only time you need to catalog a database is:

- When the database was not created on the local node
- To catalog a database with a different alias name
- To catalog a database entry that was previously deleted

You must have SYSADM authority at the node you are cataloging on to catalog a database.

Cataloging a database creates an entry in the system database directory. If the system database directory does not exist when you catalog the database, this directory file is created.

DataJoiner creates the system database directory automatically when the first database is created or cataloged on a specific node. You do not need to be connected to a database to catalog a database. When databases are cataloged on a node, the databases can be referred to by an alias. Two databases with the same alias cannot exist within the same instance of DataJoiner. DataJoiner must identify each database by a unique identifier. When you catalog a database at a node, the alias you assign is the database's unique identifier. By specifying aliases, users can distinguish between databases originally created with the same name but on different nodes. Users with SYSADM authority can use this feature to assign their own naming scheme to the various databases. DataJoiner uses the database alias as the database name for remote databases.

Uncataloging a Database

Uncataloging a database deletes a database entry from the system database directory on a specific node. You must have SYSADM authority to uncatalog a database. A database is uncataloged only in the system database directory. An entry in the local database directory can be deleted only when the database is erased by the drop database command.

Chapter 4. Identifying Existing Nicknames and Data Sources

This chapter provides SQL examples of how to identify nicknames and data sources in your DataJoiner environment.

Some of the steps involved in installing DataJoiner include:

- Configuring clients and data sources
- Creating nicknames
- Updating the catalogs with DataJoiner SQL statements, such as CREATE SERVER MAPPING and CREATE USER MAPPING.

After these tasks have been completed, there will likely be times when a DBA or other administrator must identify to which data source a given nickname corresponds; or, perhaps the administrator wants to identify all nicknames at a given data source.

The sections that follow provide examples on how you can query the communications control views to obtain this information.

Identifying a Nickname and Its Data Source

This example assumes that you know the nickname (*PAYROLL*) and who created it (*acctg*), but need additional information about the data source. Use the following SQL statement to first obtain information about what *PAYROLL* is known as at its data source (REMOTE_SERVER).

```
SELECT REMOTE_SERVER, REMOTE_TABSCHEMA, REMOTE_TABNAME
  FROM SYSCAT.TABLES
  WHERE TABNAME='PAYROLL' AND TABSCHEMA='acctg';
```

The answer set from this statement is `DB2_MVS, FINANCE, DEPTJ35_PAYROLL`. We now know that *PAYROLL* is the nickname for the table called DEPTJ35_PAYROLL owned by FINANCE at the server named DB2_MVS. We can use this information in a subsequent SELECT statement as follows:

```
SELECT NODE, DBNAME, SERVER_TYPE
  FROM SYSCAT.SERVERS
  WHERE SERVER='DB2_MVS';
```

The answer set from this statement is `REGIONW, DB2MVSDB3, DB2/MVS`. We now know that the table DEPTJ35_PAYROLL is located on a DB2 for MVS system, in a database named DB2MVSDB3, on a node called REGIONW.

With this information, you can use the `LIST NODE DIRECTORY` command to obtain information about the REGIONW node, such as the communications protocol and security type used. If the node had been for a data source other than the DB2 Family, you would need to check that data source's configuration files to find similar

information. For example, if the node had been an Oracle or Sybase data source, you would get similar information from the Oracle `tnsnames.ora` file and the Sybase interfaces file.

For information on the `tnsnames.ora` and interfaces files, see the *DataJoiner Planning, Installation, and Configuration Guide* for your platform. For information on the system catalog views, see the *DataJoiner Application Programming and SQL Reference Supplement*.

Identifying All Nicknames Known to DataJoiner

The following SQL statement provides a list of all nicknames known to DataJoiner, including the creator of the nickname and the remote server information.

```
SELECT TABNAME, DEFINER, REMOTE_SERVER, REMOTE_TABSCHEMA, REMOTE_TABNAME
   FROM SYSCAT.TABLES
   WHERE REMOTE_SERVER IS NOT NULL;
```

You can modify this statement to identify all nicknames at a particular data source, when you know the data source. Simply change the `WHERE` clause to `WHERE REMOTE_SERVER IS 'remote_server'`.

Chapter 5. Data Access Considerations and Restrictions

This chapter contains DataJoiner-specific administration information applicable to accessing information at data sources and then retrieving or using data source information. Considerations and restrictions are organized by DataJoiner components.

Large Objects (LOBs)

DataJoiner supports three types of LOBs: character large objects (CLOBs), double-byte character large objects (DBCLOBs) and binary large objects (BLOBs). For general information about these LOBs, see the following DB2 books:

- *DATABASE 2 Application Programming Guide*
- *DATABASE 2 SQL Reference*
- *DATABASE 2 Administration Guide*

DataJoiner provides additional support so that DB2 functionality to access and manipulate LOBs works for similar objects at remote data sources.

Because LOBs can be very large, the transfer of LOBs from a remote data source can be time consuming. DataJoiner attempts to minimize the transfer of LOB data between the data source and DataJoiner and also attempts to deliver requested LOB data directly from the data source to the requesting application without materializing the LOB at DataJoiner.

This section discusses:

- How DataJoiner retrieves LOBs
- How applications can use LOB handles
- How DataJoiner supports remote inserts, updates, and deletions of LOBs
- How pass-through supports LOBs
- Mappings between LOB and non-LOB data types

How DataJoiner Retrieves LOBs

DataJoiner uses three mechanisms to retrieve LOBs:

- LOB streaming
- LOB deferred retrieval
- LOB materialization

LOB Streaming

In LOB streaming, LOB data is retrieved piecemeal. DataJoiner uses LOB streaming for data in result sets of queries that are completely pushed down. For example, consider the query:

```
SELECT EMPNAME,PICTURE FROM O_T1 WHERE EMPNO = '01192345'
```

where PICTURE is a LOB column and O_T1 is a nickname referencing an Oracle table. The DataJoiner optimizer would mark the picture column for streaming if it decides to execute this query in its entirety at the Oracle data source. At execution time, when DataJoiner notes that a LOB is marked for streaming, it retrieves the LOB piecemeal from the data source. DataJoiner then transfers the data to either the application memory space or a file (as requested by the application).

LOB Deferred Retrieval

In LOB deferred retrieval, retrieval of a LOB is postponed until the LOB is assigned to a location in user space—a host variable or a file on disk. For example, in a join between two remote tables where LOB deferred retrieval is enabled, DataJoiner retrieves LOB values only for the rows that meet the join criteria. This approach can substantially boost query performance while reducing network traffic.

Remote deferred LOB retrieval is valid for a LOB column if all the following conditions are true:

- The deferred_lob_retrieval server option for the data source is set to 'y' (yes). For all data sources, this option's initial setting is 'n'. This setting is the default because deferred LOB retrieval cannot be guaranteed by most data sources. If the correctness of the deferred LOB retrieval can be guaranteed by the data source, change the server option to 'y'. To change it for a single session between an application and the data source, use the SET SERVER OPTION statement. To change it so that it remains in effect indefinitely over multiple sessions, use the CREATE SERVER OPTION statement. For information about these statements, see the *Application Programming and SQL Reference Supplement.*

  Note on static LOB data: If your LOB data at a data source is relatively static, you can set the deferred_lob_retrieval server option for that data source to 'y' even if the correctness of deferred LOB retrieval for that data source cannot be guaranteed.
- The LOB column is not marked for streaming.
- No local DataJoiner functions are being applied on the remote LOB column.
- The remote LOB column is uniquely identified on the remote table by either a ROWID or a unique index. For example, in Oracle data sources, the ROWID is used as a remote LOB locator.

If LOB columns can be retrieved on a deferred basis, DataJoiner retrieves LOB locators for the columns rather than the columns themselves. Each locator uniquely identifies its respective column. DataJoiner then does SQL processing, such as joins, predicate evaluation, and so on, on the columns; this processing is based on the plan generated by the DataJoiner global optimizer. When DataJoiner needs to transfer the LOB values

from the columns to the application space, it uses the LOB locators to retrieve these values. If the values are being transferred to a file on the application side, they are streamed piecemeal from the remote data source directly to the file without being stored at DataJoiner.

LOB Materialization

In LOB materialization, remote LOB data is retrieved by DataJoiner and stored locally. LOB materialization occurs when:

- A local function must be applied to a LOB column, which happens when DataJoiner compensates for unavailable functions at a remote data source. For example, Sybase SQL Server does not provide a SUBSTR function for LOB columns. To compensate, DataJoiner materializes the LOB column locally and applies the DataJoiner SUBSTR function to the retrieved LOB.
- The LOB column cannot be deferred or streamed.

How Applications Can Use LOB Handles

Applications can request LOB handles for LOBs stored in remote data sources. See the *DATABASE 2 Application Programming Guide* for general information about LOB handles.

DataJoiner can retrieve LOBs from remote data sources, store them at DataJoiner, and then issue a LOB handle against the stored LOB. LOB handles are released when:

- Applications issue ″FREE LOCATOR″ SQL statements.
- Applications issue COMMIT statements.
- DataJoiner is restarted.

How DataJoiner Supports LOB Operations at Data Sources

DataJoiner supports operations (inserts, updates, deletes) on LOBs at Informix, Microsoft SQL Server, Oracle (Version 7.2 or lower), and Sybase data sources.

When DataJoiner is ready to insert a LOB into a data source table, or to update a LOB in a data source table, the new or updated LOB will be transferred to the table in one of two ways. If the LOB is stored in a file, DataJoiner attempts to transfer the LOB directly from the application space. If the LOB is stored in the application space, it is transferred to DataJoiner and then to the table. The transfer to the table is done piecemeal if possible.

When DataJoiner is ready to append data to an existing remote LOB, DataJoiner can perform the append either at the data source (provided that the data source supports appends) or in DataJoiner's own environment. In the second case, the LOB is materialized at DataJoiner, the append is performed, and then the LOB, now enlarged by the append, is inserted back into the data source.

When you want to insert, update, or delete remote LOBs, you need to be aware of the following data source restrictions and requirements:

Microsoft SQL Server

For Microsoft SQL Server data sources accessed with the dblib protocol, you need to observe the requirements that apply to Sybase data sources accessed with dblib (see "Sybase"). For Microsoft SQL Server data sources accessed with ODBC, there are no requirements for, or restrictions on, LOB operations.

Oracle 7.2 and Previous Versions

For Oracle data sources version 7.2 and lower, there is no mechanism to insert or update LOBs in a piecemeal fashion.

Sybase

For an application to insert a LOB into a Sybase table, or to update a LOB in this table or in a view based on the table, the following requirements must be met:

- If the table resides at a data source accessed with the ctlib protocol:
  - There must be a unique index over one or more of the table's columns.
  - This index must be locally defined to DataJoiner.
  - When the application inserts or updates a LOB in the table, it must also insert or update an associated value in the indexed column or columns.
- If the table resides at a data source accessed with the dblib protocol:
  - There must be a unique index over one or more of the table's columns.
  - This index must be locally defined to DataJoiner.
  - The table must have a column for time stamps.
  - When the application inserts or updates a LOB in the table, it must also insert or update an associated value in the indexed column or columns. When the insertion or update is made, a time stamp is automatically generated in the time stamp column.

For example, suppose that Table T1 resides at a Sybase server accessed with the ctlib protocol. T1 has a column, PICTURE, for photographs of employees, but no unique index. So that an application can populate PICTURE, you:

- Define a unique index for T1. Assume that you define it over two columns, SOC_SEC_NO and EMP_NO.
- Program the application so that when it inserts data for an employee's photo into PICTURE, it also inserts the employee's social security and employee numbers into SOC_SEC_NO and EMP_NO.

How Pass-Through Supports LOBs

LOBs are supported in pass-through mode. LOB functionality is limited by the functionality supported by the remote data source.

LOB handles are not supported in pass-through mode.

Mappings between LOB and Non-LOB Data Types

There are few cases in which you can map a DataJoiner (that is, a DB2 for CS) LOB data type to a non-LOB data type at a data source. When you need to create a mapping between a DataJoiner LOB type and a counterpart at a data source, we recommend that you use a LOB type as the counterpart if at all possible.

Nicknames

This section contains nickname information and restrictions.

General

DataJoiner uses a nickname scheme that allows users to map a two or three-part table or view name (such as `SERVER.REMOTE_AUTHID.TABLENAME`) or a stored procedure name to a data source. Users can subsequently use the nickname in an SQL statement whenever the remote table, view or stored procedure is referenced.

You can define more than one nickname for the same table or view.

When a nickname is defined, catalog data from the remote server is retrieved and stored in DataJoiner's local catalog. For tables and views, this catalog data includes table and column definitions, and, if available, index definitions and statistics.

Nickname privileges can be used as ″initial line″ of defense for securing access to data sources. See "Options for Controlling Access to Distributed Database Objects" on page 37 for more information.

SQL DDL statements are provided to CREATE, ALTER, and DROP nicknames. Information about the SQL statements is in the *DataJoiner Application Programming and SQL Reference Supplement*.

Considerations and Restrictions

There are several considerations and restrictions to bear in mind when you want to:

- Define, change, and drop nicknames
- Reference objects by their nicknames
- Perform operations on objects that are referenced by nicknames

Defining, Changing, and Dropping Nicknames

- The objects for which you can define nicknames include tables, views, and stored procedures. To define a nickname associated with a table or view, use the CREATE NICKNAME statement. To define a nickname associated with a stored procedure, use the CREATE STORED PROCEDURE NICKNAME statement.
- You can define more than one nickname for the same table, view, or stored procedure.
- The ALTER TABLE statement cannot be used with a nickname. To change a nickname, use the ALTER NICKNAME statement.
- Dropping a nickname causes any views defined using the nickname to be dropped and invalidates any plans that are dependent upon it.

Referencing Objects by Nickname

- If an object is identified by a nickname, DDL statements can reference the object by the nickname, with one exception. A trigger definition can reference a table by its name or alias, but not by its nickname.
- Any reference to a remote table must use the defined nickname (except within a pass-through session). For example, if you define the nickname DEPT to represent the remote table DB2MVS1.PERSON.DEPT, the statement SELECT * FROM DEPT is allowed, but SELECT * FROM DB2MVS1.PERSON.DEPT is not allowed.

Performing Operations on Objects That Have Nicknames

- COMMENT ON, IMPORT, and EXPORT statements are valid against a nickname or columns defined on nicknames. The COMMENT ON statement updates the system catalog at the DataJoiner database; it doesn't update data source catalogs.
- INSERT, UPDATE, and DELETE statements are valid against a nickname whose source permits update.
- GRANT and REVOKE statements are valid against a nickname for all privileges and users. However, DataJoiner does not issue a corresponding GRANT or REVOKE against the underlying remote table or view. Therefore, the overall desired result might not be accomplished by a nickname GRANT or REVOKE alone. For example, a GRANT DELETE statement on a nickname causes DataJoiner to accept a delete statement against the nickname, but the data source might deny access if a corresponding GRANT DELETE statement was not issued for the remote table represented by the nickname.
- You can use the LOCK TABLE statement with a nickname only if the data source supports the LOCK TABLE statement.
- The LOAD and REORGANIZE TABLE utilities cannot be used with a nickname.
- A view with UNION ALL statements for multiple nicknames cannot be updated. Attempts to update such views can cause unpredictable behavior.

Stored Procedures

When working with stored procedures:

- Ensure that the number of DARI processes (MAXDARI) for the DataJoiner database is set to a value permitting stored procedure processing. The default setting is 0.
- When executing a stored procedure against a DRDA or DB2RA data source, it is possible that the CALL statement will return an error indicating that a package is not found. If this occurs, BIND the package sqllib/bnd/db2cliv2.bnd against the data source using the SQLERROR CONTINUE bind option. Then, retry executing the stored procedure.

Triggers

Triggers will work on local tables; however, you cannot define a trigger on a nickname.

Triggers defined on local database tables that affect nicknames will work. In other words, a trigger defined on a local DataJoiner table that updates a nickname is allowed.

DataJoiner triggers maintain database integrity differently than DB2 for CS. DataJoiner will rollback the entire unit of work associated with a trigger if the trigger fails; DB2 for CS rolls-back only the trigger and the events caused by that trigger (other statements are not affected).

User-Defined Functions (UDFs) and User-Defined Types (UDTs)

The following sections introduce:

- User-defined functions (UDFs) and the way to make them and new built-in functions accessible from DataJoiner
- User-defined data types (called user-defined types [UDTs] for short) and the way to make them known to DataJoiner

UDFs

This section provides an overview of UDFs and discusses the mappings through which DataJoiner accesses them and new built-in functions from data sources.

Overview of UDFs

Application developers often need to create their own suite of functions specific to their application or domain. They can use user-defined scalar functions for this purpose.

For example, a retail store could define a PRICE data type for tracking the cost of items that it sells. This store might also want to define a SALES_TAX function, which would take a given price value as input, compute the applicable sales tax, and return this data to the requesting user or application.

These functions can operate over all database types, including large object types and distinct types. UDFs allow queries to contain powerful computation and search predicates to filter irrelevant data close to the source of the data, thereby reducing response time. The SQL optimizer treats UDFs exactly like built-in functions such as SUBSTR and LENGTH. Applications can be developed using different application language environments, such as C, C++, COBOL, and FORTRAN, while sharing a set of SQL UDFs.

UDFs can not only manipulate data but also perform actions. For example, a UDF might be enabled to send an electronic message or to update a flat file.

In DB2, UDFs can include:

- Functions that you define from scratch.
- Functions in the SYSFUN schema. Examples include mathematical functions such as SIN, COS, and TAN; scientific functions such as RADIANS, LOG10, and POWER; and general purpose functions such as LEFT, DIFFERENCE, and UCASE.

For information on how to create new UDFs and how to make use of the UDFs in SYSFUN, see the *DB2 SQL Reference*.

Enabling DataJoiner to Access UDFs and New Built-In Functions at Data Sources

You can use DataJoiner in connection with UDFs when:

- Under DataJoiner, you want to directly invoke a UDF at a data source. You can do this in a pass-through session.
- You want DataJoiner to access either a UDF at a data source or a built-in function that resides at a data source and that's unknown to DataJoiner.

Before you can use DataJoiner to invoke a user-defined or unknown built-in function at a data source, DataJoiner must associate this function with a function specification stored in the DataJoiner database. The signature in this specification must correspond to the signature of the function that you want to invoke. A *signature* is the combination of a function's name and input parameters. Signatures *correspond* if they contain the same names and the same number of parameters, and if the data type of each parameter in one signature is the same as, or can be converted to, the data type of the corresponding parameter in the other signature.

There are two conditions under which DataJoiner can associate a function specification at its database with a user-defined or unknown built-in function at a data source:

- If the DataJoiner database contains a function whose signature corresponds to that of the signature of the user-defined or built-in function, you can map one function to the other.
- If the DataJoiner database doesn't contain a function with the requisite signature, you can define to the database a UDF template that contains this signature. (A *template*, in this context, is a minimal specification without any associated executable code.) Then you map the template to the function that you want to invoke.

To define a UDF template to the DataJoiner database, use the CREATE FUNCTION statement. To map a function or a UDF template at the DataJoiner database to a user-defined or built-in function at a data source, use the CREATE FUNCTION MAPPING statement.

UDTs

This section provides an overview of UDTs and discusses the mappings that enable DataJoiner to recognize UDTs at data sources.

Overview of UDTs

A UDT is a distinct user-defined data type that shares its internal representation with an existing type, but is considered to be a separate and incompatible type for semantic purposes. For example, a user might want to define a PICTURE type, a TEXT type, and an AUDIO type, all of which have quite different semantics, but which all use the predefined data type binary large object (BLOB) for their internal representation.

One of the benefits of UDTs is strong typing. Strong typing guarantees that only functions and operations defined on the distinct type can be applied to the type. For example, the system would not allow you to directly compare a PICTURE type with an AUDIO type even though they share the same underlying type. If you did want to do such a comparison, you would need to first convert values of one type to values of the other. For information about this process, called *casting*, see *SQL Reference for common servers.*

User-defined types, like built-in types, can be used for columns of tables as well as parameters of functions. For example, a user can define a data type such as ANGLE (which varies between 1 and 360) and a set of UDFs to act on it, such as SINE, COSINE and TANGENT.

Enabling DataJoiner to Recognize UDTs at Data Sources

In some cases, the definition of a table, view, or function at a data source might include a UDT that DataJoiner doesn't recognize. So that DataJoiner can recognize the UDT (and consequently access the table, view, or function), you must map the UDT to a corresponding one at the DataJoiner database. If the DataJoiner database doesn't contain a corresponding UDT, you can create one with the DB2 CREATE DISTINCT TYPE statement. To create the mapping, use the DataJoiner CREATE TYPE MAPPING statement.

Chapter 6. Distributed Unit of Work (DUOW) Transactions

This chapter covers DUOW transaction topics:

This chapter assumes that you understand basic transaction and DUOW transaction concepts. If you don't, read "Transaction Support" on page 7 first for a basic overview of transactions. Also, If needed, additional conceptual information about transactions is in the *DATABASE 2 Administration Guide*.

Terminology and Concepts

DataJoiner uses two-phase commit protocol for its DUOW transaction support. Specifically, DataJoiner uses the industry standard X/Open XA protocol (two-phase commit) to coordinate DUOW transaction processing between data sources.

Terminology

The XA model has several key terms:

Transaction Identifier
A *transaction identifier* (xid) uniquely identifies a distributed transaction. An xid is used by the transaction manager and one or more resource managers to synchronize the work they do on behalf of a DUOW.

Transaction Manager
The *transaction manager* (TM) coordinates a distributed transaction. The TM generates the DUOW transaction xid and is responsible for coordinating the decision to commit or rollback the DUOW transaction. The TM also coordinates failure recovery—it knows about all the resource managers (RMs) involved within the TM's branch of a global transaction. The TM for any one DUOW transaction

is identified by the TM_DATABASE variable (database manager configuration variable) or, if the keyword 1ST_CONN is used, the TM is the first database connected to by the application. More information about database manager variables is documented in the *DATABASE 2 Administration Guide*.

DataJoiner is the TM for a DUOW transaction if the name of the DataJoiner database is specified in the TM_DATABASE variable and the application SYNCPOINT is TWOPHASE.

Depending on the specifics of a DUOW transaction, DataJoiner can be the TM, a sub-TM, or both.

Sub-Transaction Manager

A *sub-transaction manager* (sub-TM) helps coordinate DUOW transactions on behalf of the TM. A sub-TM is actually a resource manager (RM) performing some TM functions on behalf of the TM. DataJoiner often plays the role of a sub-TM because even when the TM_DATABASE variable does not specify the DataJoiner database that is accessing nicknames, DataJoiner must guarantee the data integrity and atomicity of its data sources. For example, DataJoiner must log information about the remote data sources. DataJoiner must also track data source information in the event of a transaction failure and help coordinate failure recovery steps. DataJoiner performs these functions because the TM is unaware of the data sources connected to DataJoiner.

Resource Manager

A *resource manager* (RM) manages a portion of the data accessed by the DUOW transaction. For example, if the DataJoiner database has a nickname that refers to a table at a remote data source, the process that manages that remote data source is an RM. These RM processes associate xids with the work done on behalf of the DUOW transaction. RMs are not usually aware of the entire transaction. RMs follow the instructions provided by TMs to commit or rollback a transaction.

Resource Manager Note: An RM becomes a sub-TM if it coordinates one or more RMs.

Transaction Branch

A *transaction branch* represents a portion of a DUOW controlled by a sub-TM. The top part of the branch is the sub-TM; the bottom part is an RM controlled by that sub-TM. The sub-TM is responsible for maintaining transaction atomicity for

the RMs and sub-TMs under its control. This structure is shown in Figure 7

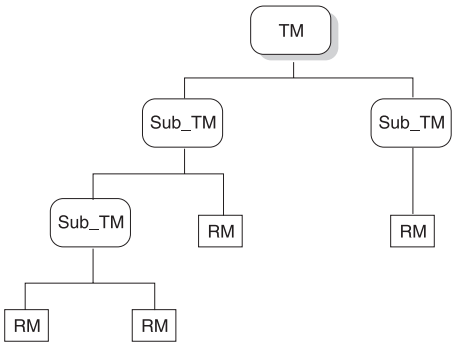


Figure 7. Transaction Processing Tree Structure

Clients connect to the DataJoiner database using one of two connection types:

TYPE 1 An application can connect to only one database.

TYPE 2 An application can connect to one or more databases. This connection type is typical for DUOW transactions.

Connections also have syncpoint settings:

SYNCPOINT NONE Specifies that no TM is to be used to perform two-phase commit. Single update, multiple read rules are not enforced. A commit is sent to each participating database. The application is responsible for error recovery if any commits fail.

SYNCPOINT ONEPHASE DUOW transactions with updates to multiple data sources are not supported. TYPE 1 connections are always set to SYNCPOINT ONEPHASE. In this case, a TM will not be used for two-phased commit processing.

If an application connects to DataJoiner with SYNCPOINT ONEPHASE or NONE, and the CONNECT TYPE is either 1 or 2, two-phase commit processing is not used externally to DataJoiner; however, DataJoiner will maintain (internally) transaction atomicity and use two-phase commit processing between DataJoiner and its data sources to maintain data integrity.

SYNCPOINT TWOPHASE DUOW transactions with updates to multiple data sources are supported. A TYPE 2 connection can be set to SYNCPOINT ONEPHASE or SYNCPOINT

TWOPHASE. It is also possible to have a TYPE 1 connection with SYNCPOINT TWOPHASE.

SYNCPOINT Note: In effect, SYNCPOINT settings apply to the connection from the client to DataJoiner. These settings do not apply to the connections made by DataJoiner to data sources. DataJoiner DUOW processing between data sources is driven by two_phase_commit server option settings. In other words, the stated rules apply to databases connected to by the application; however, DataJoiner will enforce single update, multiple read rules for the data sources it connects to on behalf of applications.

Two-Phase Commit Processing Concepts

DataJoiner tries to guarantee data consistency and atomicity of the data sources it's managing. Therefore:

- If an application with a TYPE 2 connection connects to DataJoiner, then there's the possibility that a data source outside of DataJoiner's control will get updated; therefore DataJoiner will only allow updates to data sources that support two-phase commit. Only read operations are allowed against one-phase commit data sources.
- If an application does not connect with a TYPE 2 connection, then the only possibility for multi-site update is within the control of DataJoiner. In that case, DataJoiner will allow the update of one-phase commit data source, as long as it is the only data source updated.
- DataJoiner takes a conservative approach when considering possible ″update″ requests. An update is:
  - SQL DML
  - SQL DDL
  - Remote stored procedure calls, because the contents of the remote stored procedure could contain an update operation

  One exception on handling possible update requests is UDF processing. A remote UDF could cause updates. DataJoiner could treat every remote UDF as an update call, but this restriction is somewhat severe because remote UDFs can be used in SELECT SQL statements. Therefore, UDFs are not treated as updates.

UDF Note: Ensure that UDFs used in read operations do not cause updates to the data source. If a UDF does update a data source object, issue a COMMIT before performing other update operations.

In summary: when a client uses a TYPE 1 phase connection (DataJoiner is the TM), DataJoiner controls DUOW activity within its own processes. All DUOW considerations are then controlled by DataJoiner. When a client uses a TYPE 2 connect, it means that the DUOW transaction could involve updates to other two-phase commit databases (other than the data sources controlled by DataJoiner). The DataJoiner database will be the TM if the TM_DATABASE variable specifies the DataJoiner database; otherwise, DataJoiner is the sub-TM. The connection type affects the way DataJoiner can update data sources. For example, DataJoiner will not allow any updates to one-phase commit

data sources under its control if a client uses a TYPE 2 connection.

Typical Configurations

With DataJoiner, there are two typical DUOW configuration scenarios. Each one is described in this section.

A key point for both configurations: DataJoiner CREATE SERVER OPTION or SET SERVER OPTION SQL statements dictate how two-phase commit processing is done between DataJoiner and its data sources . An application provides connection information—but the DataJoiner server option settings determine if a data source can be updated. The connection data supplied by the application is useful when DataJoiner must determine if one of its 1PC data sources can be updated; however, this information does not determine if a 1PC or a 2PC connection is used for that data source.

DataJoiner as a Sub-TM

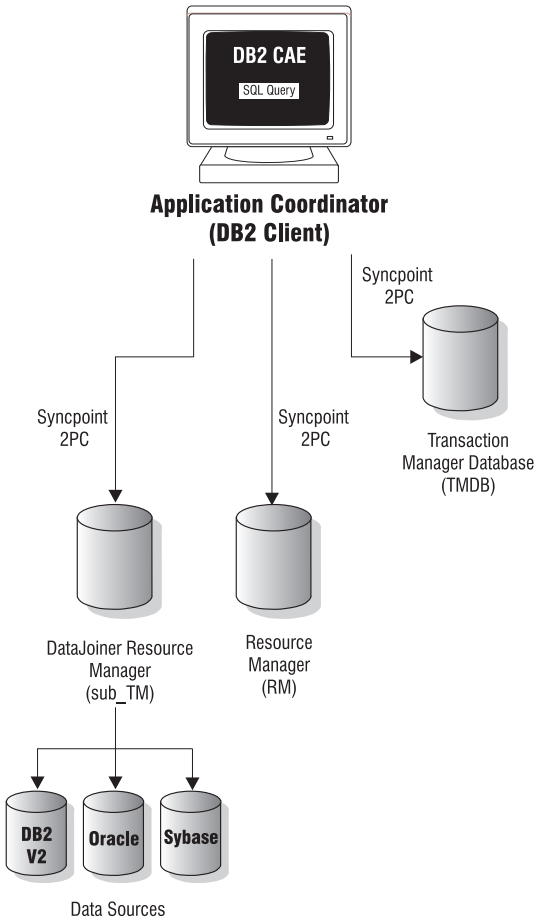


Figure 8. DataJoiner as a Sub-TM

If a client connects to the DataJoiner database using a TYPE 2 connect with SYNCPOINT TWOPHASE, the client must have specified a TM for the application (using the TM_DATABASE variable). DataJoiner is a sub-TM whenever the DataJoiner database is not specified in the TM_DATABASE variable for the client.

This configuration is the most typical. The TM database is usually a separate database whose primary role is to perform DUOW transaction logging and to coordinate failure recovery.

DataJoiner as a TM

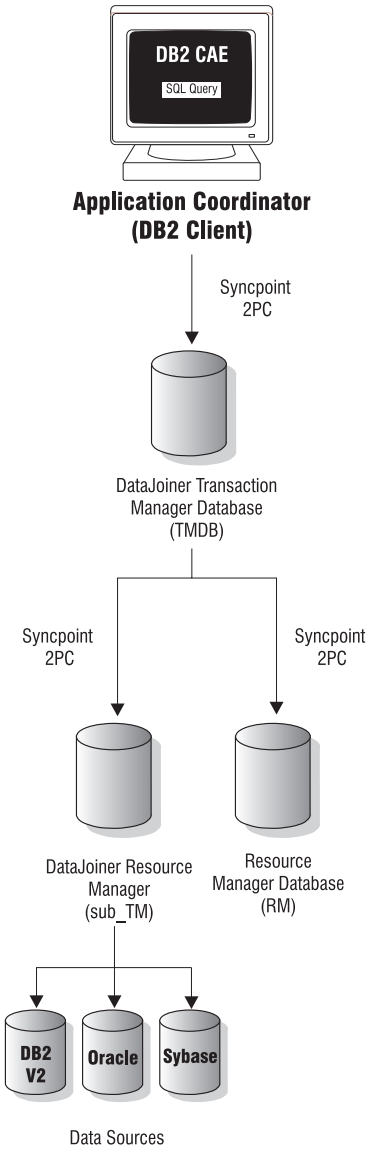


Figure 9. DataJoiner as a TM

If a client connects to the DataJoiner database using a TYPE 1 connect or a TYPE 2 connect with SYNCPOINT ONEPHASE, the DataJoiner database is the TM. Another case is when a TYPE 2 connection is made to DataJoiner with SYNCPOINT TWOPHASE and the DataJoiner database is specified as the TM (using the

TM_DATABASE variable). As the TM, DataJoiner coordinates all commit processing for RMs and also for data sources that are updated or read. The entire process is transparent to the application: DataJoiner provides a single database image even though multiple data sources may be updated.

Costs, Considerations, and Prerequisites

Two-phase commit processing provides substantial application flexibility. Multiple data sources can be read and updated in a single DUOW transaction. Transaction atomicity is maintained throughout the process.

Two-phase commit processing, however, does incur processing overhead, particularly during commit operations. If you know that your applications will not need DUOW transaction support to update data sources, then set the two_phase_commit server option for each data source server to 'N' (for no). If DUOW update support is needed for only a few data sources, set the two-phase commit server option for each data source server to 'Y' (for yes). Setting data source server options for DUOW processing is discussed in "Preparing Data Sources for DUOW Transactions" on page 71.

Before setting a data source two_phase_commit server option to 'Y', ensure that your target data source supports two-phase commit and that you have all the necessary product features. Table 7 shows which data sources support two-phase commit and additional features required for certain data sources. The second column indicates if a data source, with the listed constraints, can participate in a DUOW transaction. The third column indicates if the data source supports the creation of new objects at the data source (using a pass through session) while participating in a DUOW transaction.

Table 7. Data Sources Supporting Two–Phase Commit

Data Source	Supports two-phase commit?	Supports two-phase commit DDL?
Classic Connect	No	No
Cross Access	No	No
DB2 CS and DataJoiner, Version 1.2	Yes	Yes
DB2 CS and DataJoiner, Version 2	Yes	Yes
DB2 for OS/390, Version 3 and higher (with APARs PN67179 and PN70102)	Yes	Yes
DB2 for OS/400, before Version 3.1	No	No
DB2 for OS/400, Version 3.1 and higher	Yes	Yes
EDA/SQL	No	No
Generic Access API data sources	No	No
Informix Version 5	Yes	Yes
Informix, Versions 7.1 and 7.2, with TP/XA library	Yes	Yes
MS SQL Server (DBLIB)	No	No

Table 7. Data Sources Supporting Two–Phase Commit (continued)

Data Source	Supports two-phase commit?	Supports two-phase commit DDL?
MS SQL Server 4.2 (CTLIB) with XA library	No	No
MS SQL Server 6.5 (ODBC), using DataJoiner on NT	No	No
MS SQL Server 6.5 (ODBC), using DataJoiner V2 on UNIX operating systems	No	No
Oracle V7 and higher, without the XA library distributed database option	No	No
Oracle V7 and higher, with the XA library and the distributed database option on UNIX operating systems	Yes	No
Oracle V7.3 and higher, on Windows NT 3.51 and 4.0 operating systems (requires V7.3.3 Oracle Client at DataJoiner on NT operating systems)	Yes	No
Oracle RDB (all levels)	No	No
SQL Anywhere	No	No
SQL/DS, before Version 4.1	No	No
SQL/DS, Version 4.1 and higher	Yes	Yes
Sybase SQL Server (CTLIB) with XA library	Yes	No
Sybase SQL Server (DBLIB)	No	No

Additional technical requirements include:

- XA support/libraries for data source client software is required at the DataJoiner server.
- If you are connecting to a DRDA data source for 2PC processing, and your DataJoiner instance is running on AIX, ensure that you are using AIX SNA Server V2.1 with APAR fixes IX50393 and IX51831 or a later version. You can use Communications Server on AIX instead of AIX SNA Server. These fixes are required for the SNA Syncpoint Manager.
- The previous SNA requirement also applies to users connecting to DB2 for CS data sources over SNA using the DRDA protocol.

DataJoiner 1PC and 2PC Processing Rules

Several factors determine how a DUOW transaction is processed by DataJoiner. Critical factors include application SYNCPOINT settings, actions done for other data sources accessed during the transaction (for example, has a data source that does not support two-phase commit been updated?), and data source server option settings for each server that corresponds to a nickname touched during a DUOW transaction.

Server option settings refer to the values (Y, N, or D) that correspond to the two-phase-commit server option. If the two-phase-commit server option is set to Y for a server, it is referred to as a 2PC database for that server (two-phase commit is

enabled). If the two-phase-commit server option is set to N or D for a server, it is referred to as a 1PC server (two-phase commit is disabled).

DataJoiner allows a mix of 1PC and 2PC data sources. In order to guarantee the atomicity of transactions, several rules are enforced (see Figure 10).

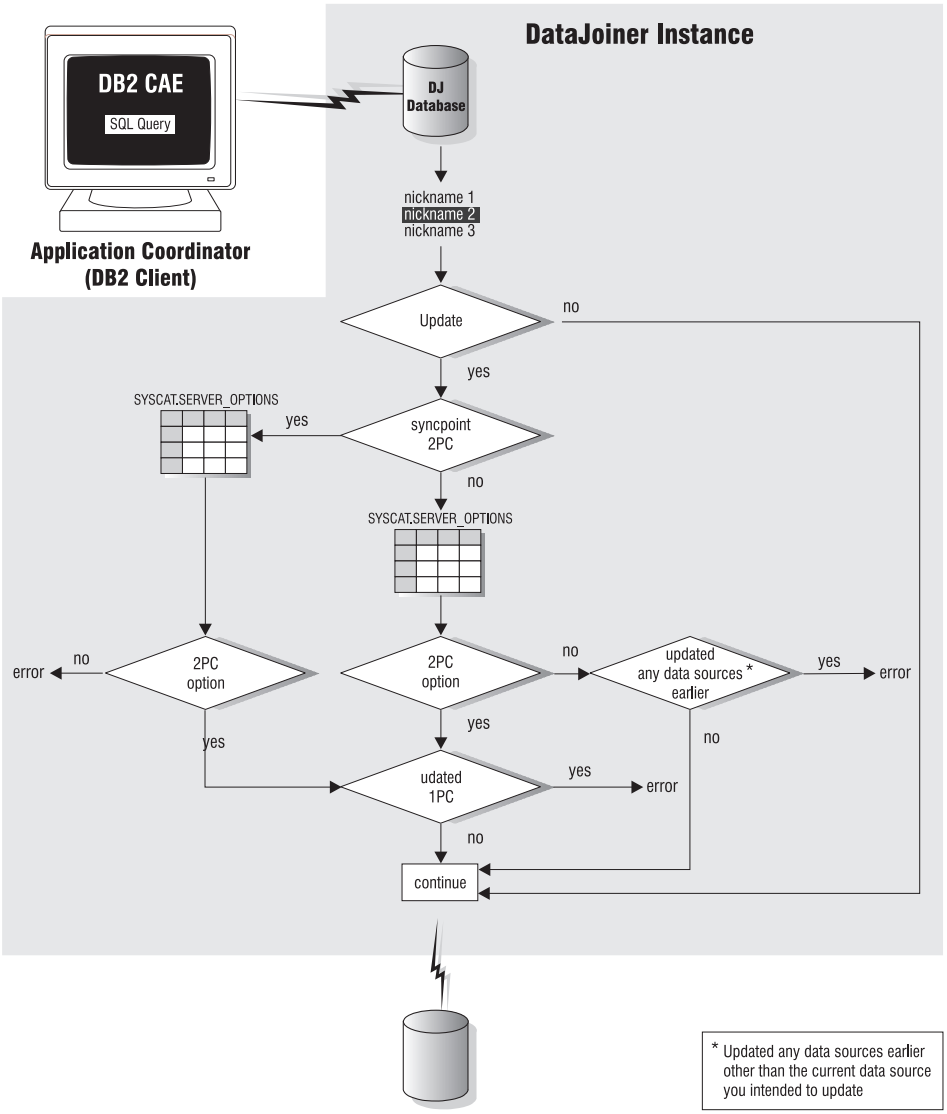


Figure 10. DataJoiner DUOW Transaction Atomicity Logic

As the flow chart shows, DataJoiner can support transactions accessing a mix of data sources set to either 1PC or 2PC. The processing rules are:

- Read operations are always permitted for 1PC and 2PC data sources.
- 1PC and 2PC database updates cannot be mixed in one DUOW transaction. If a SYNCPOINT 1 connection is made to the DataJoiner database, because DataJoiner is the only database referenced in the transaction, one 1PC data source can be updated as long as no other 1PC or 2PC data source is updated. However, if SYNCPOINT TWOPHASE is used, because another data source could be updated without DataJoiner's knowledge, DataJoiner will prevent all 1PC updates.
- Two 1PC data sources cannot be updated in one DUOW transaction.
- The local DataJoiner database is always considered a 2PC data source.

Special conditions influence how processing rules are evaluated:

- Although triggers are not supported on nicknames, DataJoiner supports the firing of triggers that refer to a nickname as a result of local table updates; however, if the data source (trigger target) is set to 1PC (the server option two-phase-commit is set to N or D), it will fail during the base table update. Similarly, if a 1PC data source fires a trigger that updates a different 1PC data source, the update will fail.
- DataJoiner supports the use of user-defined functions (UDFs). In general, because local DataJoiner UDFs cannot contain SQL or call stored procedures, update requests are not caused by UDFs. However, DataJoiner supports the mapping of a local UDF to a nickname server UDF. The remote UDF could cause an update—;and DataJoiner does not police this use of remote UDFs.

  Remote UDF Note: Do not map remote UDFs to local UDFs if they contain update statements—database integrity could be compromised. Only map remote UDFs to local UDFs when they are read-only functions.

Data Source Requirements, Restrictions, and Considerations

This section contains general and data source-specific information for two-phase commit operations.

All Data Sources

Common requirements include:

- A transaction resolution password and ID for resynch operations. Specifically, a special user name and password with the authority to COMMIT and ROLLBACK transactions must exist at the data source instance, and this ID and password must be identified to DataJoiner using the CREATE USER MAPPING SQL statement.

  DataJoiner will always use the special ID, SYSTMDB, for the resynch agent that connects to RMs during resynch processing. This ID must be mapped to a remote user ID and password that has the authority to commit or rollback the transaction initiated on remote tables by users through DataJoiner.

  For information on using CREATE USER MAPPING SQL statements, see the *DataJoiner Application Programming and SQL Reference Supplement.*

Common restrictions include:

- DDL restrictions. See Table 7 on page 66.
- You cannot use cursor with hold statements for remote tables in 2PC data sources.
- A view with UNION all statements for multiple nicknames cannot be updated. Attempts to update such views can cause unpredictable behavior.
- Do not access the data source via multiple paths (in the same transaction) with two-phase commit processing. Results can be unpredictable.
- Multi-site update restrictions are enforced even when one update is done by the database manager on your behalf. For example, it is possible for an application to receive an SQLCODE -752, which indicates that a multi-site update occurred even though the application did not explicitly issue updates to more than one location in a single transaction. The -752 can appear because if a plan or package is invalidated (a table, view, or nickname associated with the plan or package was dropped or created after the application was bound to DataJoiner), DataJoiner will perform an auto-bind on behalf of the application. The auto-bind updates the system catalog at the local database. Therefore, subsequent SQL statements that update other locations cause DataJoiner to detect a multi-site update and issue a rollback.

  One solution is to manually bind the invalid packages again. After the manual bind, verify the validity of packages by issuing the statement:

  ```
  SELECT pkgname, valid FROM syscat.packages
  ```

  For each package name, the column VALID should contain a 'Y'. Now, run the application again.

DRDA Data Sources

DRDA data source restrictions include:

- Transactions must go through a DDCS gateway to access two-phase commit data sources. The DDCS gateway can be a separate gateway or DataJoiner itself (configured as a DDCS gateway). For either case, the DDCS Sync Point Manager must be enabled. More information on DDCS configuration is in the *DataJoiner Planning, Installation, and Configuration Guide* for your platform.
- DataJoiner cannot serve as a two-phase commit DRDA2 application server (AS)—it cannot support 2PC as a DRDA AS.
- When you are using DataJoiner as a DRDA application requestor (AR), you cannot use two-phase commit protocol for applications sent to a DRDA AS via TCP/IP.

DRDA data source considerations include: when enforcing 2PC across DRDA AS data sources (two_phase_commit is set to 'Y'), DataJoiner will return an error if the actual data source supports only 1PC transactions or if the path to the data source only supports a 1PC connection. DataJoiner does not attempt to continue the transaction by switching the remote data source designation to 1PC and then re-trying the transaction.

Informix Data Sources

DataJoiner cannot access an Informix data source with a mix of one-phase and two-phase commit settings (see “Set Data Source two_phase_commit Option Values” on page 72) with the same data access module. To change the setting for an Informix data source from one-phase to two-phase commit, DataJoiner must be stopped (to unload the Informix data access module) and then restart the DataJoiner instance.

An entry must exist in the .netrc file for storing the ID and password (used for two-phase commit).

Sybase SQL Server Data Sources

Ensure that the user ID and password used to access a Sybase SQL Server data source (specified with a CREATE USER MAPPING SQL statement) match the actual user ID and password at the data source server (same case). Also, set the server options (using SET SERVER OPTION SQL statements) fold_pw and fold_id for this server to N. Alternatively, if the user ID and password are all in one case, then you could set fold_pw and fold_id to L or U, depending on the case.

For a Sybase SQL Server data source to act as an RM to DataJoiner, a logical resource manager (LRM) entry must exist in the xa_config file under the Sybase directory that maps the RM name to the Sybase SQL Server name (more information is available in Sybase SQL Server XA documentation). The LRM name is used by DataJoiner in the XA Open string. DataJoiner will use the Sybase SQL Server node name for the LRM name. If the DataJoiner user has write authority on the xa_config file, then DataJoiner will automatically create an LRM entry in the xa_config file during CREATE SERVER MAPPING SQL execution. If the DataJoiner user does not have write authority on xa_config, then the Sybase administrator will need to create LRM entries manually

If errors are encountered during XA processing, the error information is logged in the sybase_xa.log file located in the sqllib/db2dump directory.

Oracle Data Sources

Oracle must be linked with the XA code turned on (XA library distributed database option).

Preparing Data Sources for DUOW Transactions

Before you run DUOW applications, it is essential that you enable your DataJoiner instance logs and set data source 2PC options.

Start Logs at DataJoiner and Data Sources

Transaction logs are required for recording COMMIT/ROLLBACK status for, and at, data sources. If you intend to use 2PC processing, ensure that:

- Standard transaction logging is enabled at DataJoiner
- Standard transaction logging is enabled at the 2PC data sources controlled by DataJoiner (as a sub-TM or the TM)

Set Data Source two_phase_commit Option Values

There are two ways to initially set two_phase_commit values. To set a value that persists for multiple connections (creating a server default value), use CREATE SERVER OPTION SQL statements to set values for each data source server. Alternatively, to set a value that persists only for the duration of individual application client connections (dynamic setting), use SET SERVER OPTION SQL statements. You can also use ALTER SERVER OPTION SQL statements to update values set by CREATE SERVER OPTION SQL statements. All three statements are documented in the *DataJoiner Application Programming and SQL Reference Supplement*.

DataJoiner supports three values for a server's two_phase_commit option. The two_phase_commit option values are:

'Y' Yes. Two-phase commit is used to connect to the remote server.

'N' No. Two-phase commit is not used to connect to the remote server.

'D' Disabled. Indicates that two_phase_commit cannot be used to connect to the remote database.

Use this option value in CREATE SERVER OPTION SQL statements to ensure that applications cannot dynamically change a server's two_phase_commit option value to 'Y' for a connection. This value is different from 'N' because an 'N' value can be overridden by SET SERVER OPTION SQL statements. In any case, if a database server option is set to disabled, applications that try and access a data source with a 2PC connection will fail when the nickname for that data source is accessed.

DataJoiner internally sets the two-phase commit option to disabled if it doesn't support two-phase commit access to a remote data source.

Using CREATE SERVER OPTION SQL Statements to Enable a 2PC Setting

After you identify a data source to DataJoiner with the CREATE SERVER MAPPING SQL statement, you can set options for that server with the CREATE SERVER OPTION SQL statement. Current option settings are stored in SYSCAT.SERVER_OPTIONS.

To change the two_phase_commit option, use the ALTER SERVER OPTION SQL statement to define the option being changed and the new setting for that option. For example, to enable 2PC processing for the server DB2V21, the SQL statement is:

```
ALTER SERVER OPTION two_phase_commit FOR SERVER DB2V21 SETTING 'Y'
```

Additional information is in the *DataJoiner Application Programming and SQL Reference Supplement*.

Using SET SERVER OPTION SQL Statements to Enable/Disable a 2PC Setting for a Specific Connection

DataJoiner also supports a connection-specific method for enabling/disabling 2PC access to a data source; however, this method is not supported for data sources that currently have their two-phase-commit option set to 'D'.

Use SET SERVER OPTION SQL statements, for example, when:

- Your application must issue DDL against a data source that does not support DDL within an 2PC transaction
- You wish to leave the data source 2PC setting ″as is″ for the majority of your transactions

To issue SET SERVER OPTION SQL statements:

- Connect to the DataJoiner database.
- Issue a SET SERVER OPTION SQL statement to set 2PC access for this transaction. For example, to change the data source DB2V21 from 2PC to 1PC processing for the duration of the connection. The SQL statement is:

  ```
  SET SERVER OPTION two_phase_commit to 'N' FOR SERVER server-name
  ```

- Issue your application SQL statements.

  Note: Issue SET SERVER OPTION SQL statements before any other SQL statements; otherwise, an error is returned.

Additional information is in the *DataJoiner Application Programming and SQL Reference Supplement*.

Performance Considerations

There are facets of two-phase commit performance: its general impact on commit processing and two-phase commit-specific process performance:

- For information on general impact to query performance, see “Costs, Considerations, and Prerequisites” on page 66.
- For two-phase commit-specific process performance information, see the *DATABASE 2 Administration Guide*. Examine the sections on the resync_interval and spm_max_resync parameters.

DUOW Error Recovery

The following topics introduce DUOW error recovery concepts, critical processes, and options for resolving indoubt transactions:

Recovering from Problems

DataJoiner standard recovery procedures closely match DB2 for CS recovery procedures. If a problem occurs, both DB2 for CS and DataJoiner will attempt to recover from the error automatically through resynchronization processing. A general discussion is provided in the *DATABASE 2 Administration Guide*.

There are, however, some high-level differences between DB2 for CS DUOW error handling and DataJoiner DUOW error handling:

- When enforcing 2PC across DRDA AS data sources (two_phase_commit is set to 'Y'), DataJoiner will return an error if the actual data source supports only 1PC transactions or if the path to the data source only supports a 1PC connection. DB2 for CS, however, attempts to continue the transaction by switching the remote data source designation to 1PC and then attempting the transaction.
- DB2 for CS DUOW error handling assumes that all involved data sources are DB2 Family data sources. This assumption is not correct for DataJoiner. DataJoiner's support for non-IBM data sources means:
  - If DataJoiner is the TM, DB2 data sources that are DataJoiner RMs will inquire about the status of a failed DUOW. Other data sources, such as Oracle, will not come back to the DataJoiner TM.
  - DataJoiner administrators attempting to manually recover indoubt transactions may need to manually trace xids to determine transaction states for non-DB2 data sources.

Resynchronization Processing

Resynchronization processing tracks and resolves DUOW transaction states. It takes necessary actions, when possible, to commit or rollback DUOW transactions in the background without user intervention.

When a DUOW transaction fails during commit processing, such as losing communication to a database server after prepare processing but before commit processing, the transaction cannot complete. From the perspective of the RM that has lost contact with the application, the transaction is now indoubt because the RM is prepared to commit the transaction but has not been told by the TM to actually commit it. At this point, resynchronization processing can start.

RM Note: RMs can actively or passively participate in resynchronization processing. An RM can initiate resynchronization (ask the TM if it should commit or rollback the DUOW transaction provided that communication with the TM is restored); or, it can wait for the TM and keep the resources associated with this transaction branch locked.

Initialization

Resynchronization processing starts when:

- During normal commit or rollback processing, the commit or rollback to one or more data sources fails during the second phase.
- During database restart processing, the database logs indicate that transactions are not complete.

In both cases, transaction information is copied to the resync list. The transaction xid and state are recorded.

The actual transaction state could be indoubt (the decision to commit or rollback the transaction is unknown) or known, depending upon where resynchronization processing is taking place. At the TM level, the transaction state is known. At the sub-TM level, the transaction state might be known. The transaction is indoubt if the sub-TM has completed the PREPARE phase but has not heard from the TM if it should COMMIT or ROLLBACK the transaction. The transaction is not indoubt if the sub-TM received a decision from the TM but was unable to propagate that decision to all its RMs.

If the decision to commit the DUOW transaction was recorded in DataJoiner's log, the list of DataJoiner RMs associated with the transaction are added to the resync list. This RM resync list helps DataJoiner maintain a single database image for the transaction even if it involved multiple data sources.

Processes

The resynchronization process periodically checks the resync list. This interval is set by the database manager configuration parameter resync_interval. It attempts to resolve indoubt transactions and then commit or rollback transactions whose state is resolved.

If the failed transaction has a known state (not indoubt), the resync process connects to the RM database for that transaction and issues a commit or a rollback. If the RM is a DataJoiner database (actually, a sub-TM), DataJoiner issues commit or rollback requests to the data sources that participated in that DUOW transaction.

For transactions that are indoubt, and the TM is a DB2 or DataJoiner database, the resynchronization process contacts the TM database of that transaction to resolve the state of the transaction. The TM database responds by indicating if the transaction should be committed or rolled-back based on its log. If the TM database indicates that the transaction is to be committed, the resync manager process updates the transaction state and waits for the TMDB to drive the resync commit processing. However, if the TM database indicates that the transaction is to be rolled-back, the resync process updates the transaction state and connects to the DataJoiner sub-TM that has the indoubt transaction and issues a rollback. The DataJoiner database then goes through the list of 2PC databases that it has updated as a part of this transaction and issues a rollback.

TM Processing Note: If the TM database is a non-DB2 database, the resynchronization process waits for the TM to drive resync actions. The RMs and sub-TMs do not actively request TMDB transaction state information.

Manually Determining Transaction States

When a transaction is sent through DataJoiner, DataJoiner does not externalize the actual data sources that are a part of the transaction. If you are tracing an indoubt transaction, the only way you can determine the actual data source that failed is to capture the xid for the failed transaction and then look for that xid in all the data source database managers that DataJoiner may have accessed as a part of the DUOW transaction.

This section provides information about tracing transaction states and xids across data sources.

Listing Transaction Data

Use the LIST INDOUBT TRANSACTIONS command documented in the *DATABASE 2 Command Reference* against your application database, the DataJoiner database, and any DB2 Family data sources involved in the DUOW transaction to discover the xid and current transaction state for each indoubt transaction.

Use similar commands for each data source that might have been a part of the transaction. Each data source that supports two-phase commit might use a different command; search for the term ″xid″ in your data source command reference index.

Tracing xids Across DB2 Family and Other Data Sources

When DataJoiner is either the TM or a Sub-TM, it passes the xid to data sources during transaction processing. Before DataJoiner sends an xid to a non-DB2 Family data source, it modifies the xid so that the it conforms to the data source's xid format. For non-DB2 Family data sources, the changes are:

- Updating the branch qualifier length section of the xid
- Adding the branch qualifier section of the xid

These changes are similar for Oracle, Sybase SQL Server, Informix, and so on.

It is important that you know about these changes because you may compare xids across several data sources. You need to know which part of the xid you can safely compare across your environment when tracing an xid.

As an example, if the TM for a transaction is a DB2 system, entering the LIST INDOUBT command at the TM database would return a string (hexadecimal representation) for the transaction xid similar to:

```
53514C2000000027 0000000050455246 000000000033C827 2833B40678000000
00010F0000000000 0000000000000000 000000
```

This long string is actually composed of several distinct parts:

```
FORMAT    TID     BRANCH              TRANSACTION IDENTIFIER
ID       LENGTH   QUALIFIER
                  LENGTH
53514C20 00000027 00000000      5045524600000000033C827 2833B40678000000
                                00010F0000000000 0000000000000000 000000
```

Again, the values listed above are hexadecimal. For example, the TID LENGTH (hexadecimal 27) represents the decimal value 39.

If that same xid will get passed along to a non-DB2 Family RM via DataJoiner (acting as a sub-TM), the xid string is changed. For example, the xid passed to an Oracle data source RM changes to:

```
53514C2000000027 0000000150455246 000000000033C827 2833B40678000000
00010F0000000000 0000000000000000 00000001
```

Examine the branch qualifier length and branch qualifier sections reformatted below:

```
FORMAT    TID     BRANCH              TRANSACTION IDENTIFIER               BRANCH
ID       LENGTH   QUALIFIER                                               QUALIFIER
                  LENGTH
53514C20 00000027 00000001      5045524600000000033C827 2833B40678000000
                                00010F0000000000 0000000000000000 000000    01
```

You can see that the branch qualifier length section has a 1 and that the branch qualifier section was added.

You can still trace the xid across different data sources by limiting your search string input to the transaction identifier section.

Manually Recovering Indoubt Transactions (Heuristic Processing)

If you cannot wait for the TM to automatically resolve indoubt transactions, you can use the LIST INDOUBT TRANSACTIONS command to list and then manually resolve the transactions. This process is called ″heuristic processing.″

Consider using heuristic processing when you know why the transaction failed and there is a pressing need to free locked resources. For example, if your communications link between the TM and a sub-TM fails in the midst of a transaction, and you are familiar with the transaction, you could free local resources by heuristically rolling back the transaction.

Heuristic Processing Note: Heuristic decisions made at a TM or a sub-TM are not propagated to RMs. Each TM, sub-TM, and RM must be handled individually. For non-DB2 heuristic processing information, see the documentation that came with the data source. For DB2 heuristic processing information, see the *DATABASE 2 Administration Guide*.

Using DataJoiner with a Non-DB2 Transaction Manager

See the *DATABASE 2 Administration Guide* for information about running a DataJoiner instance under another transaction manager.

Chapter 7. System and Query Tuning

This chapter covers performance considerations. It has information on tools used for monitoring your system and methods for improving performance. Improving query performance is the overall focus and comprises most of the chapter. The major topics are:

- "DataJoiner System Monitoring and Tuning"
- "Tuning Query Processing" on page 85
- "Tuning DataJoiner Network Usage" on page 99
- "Tuning Data Source Configurations" on page 101

DataJoiner System Monitoring and Tuning

Tuning a DataJoiner system involves

- Monitoring DataJoiner database operations
- Tuning DataJoiner for maximum performance (that is, tuning the basic database engine contained in DataJoiner)
- Setting the DataJoiner collating sequence

A bottleneck at either DataJoiner or any data source accessed by DataJoiner can degrade performance. This section addresses local system performance. Be aware that system problems can originate at either DataJoiner, the data source, or both.

DataJoiner Monitoring

A set of database system monitor APIs are provided that allow applications to be written that gather statistical information regarding the operation of the database. Information gathered through these APIs can be used to assess the status of DataJoiner, individual databases, tables, and individual applications.

The information gathered by the monitor is generated by the internal components of DataJoiner. The information gathered contains status information regarding the current state of the database manager and activity information such as counters and other measurements of database processing.

Note: The DataJoiner monitor only tracks the state of DataJoiner and its interactions with data sources. It does not monitor the state of data sources beyond very simple tracking of interactions with them.

The database system monitor will record point-in-time information (*snapshot*) on the following components:

- Database connections

- Locks
- Buffer pool activity
- SQL statement activity
- Sorting
- Cursors
- Deadlocks
- Table activity
- Unit-of-work status
- Database status
- Communications activity
- DataJoiner activity with remote data sources
- DataJoiner use of data sources.

To set up the database system monitor, see “Chapter 9. Database System Monitor” on page 107 .

DataJoiner Tuning

The DataJoiner database engine has the same tuning parameters as DB2 CS Version 2. See the *DATABASE 2 Administration Guide* for information on tuning the basic database engine contained within DataJoiner. In general, you should start with the engine defaults and adjust them later for specific tuning issues.

Some areas deserve special consideration. The following topics greatly affect engine performance:

- Row blocking
- RUNSTATS utility
- REORG utility
- Configuration parameters

Each is topic is described in following sections.

Row Blocking

Row blocking, also called block fetch, is the process of fetching more than one row per call from the data source. Applications can indirectly affect when DataJoiner can use row blocking; however, DataJoiner makes the final determination of when to block fetch.

One of the greatest impacts on the performance of your system (and queries) is the extent to which DataJoiner can utilize block fetch, and the size of the block to be fetched. DataJoiner controls the size of the block used by the DRDA/DB2RA and SQL*NET protocols. Block fetch is also used by the dblib and ctlib protocols, but DataJoiner has no control over these. For DataJoiner, the *svrioblk* configuration parameter (described in the *DATABASE 2 Administration Guide*) determines the size of

the block. The parameter default is 4096 (a 4K byte block of data is fetched during each block fetch operation). When adjusting that parameter, remember that each cursor in your application can result in many data sources being accessed. Each cursor in your application can result in multiple blocked cursors going to data sources; therefore, multiple blocks of storage, of the size described in *svrioblk*, are allocated to service your request.

DataJoiner typically tries to block under the same conditions as DB2 for CS blocks; therefore, you should explicitly code your queries as FOR FETCH ONLY whenever possible, and use the BLOCKING ALL BIND option whenever possible. The *DATABASE 2 Administration Guide* provides more details regarding when DB2 for CS will block.

The RUNSTATS Utility

DataJoiner relies on catalog statistics for nicknames in the same way that DB2 for CS relies on catalog statistics for tables. These statistics are extremely important in the query optimization phase.

The catalog statistics for nicknames are gathered in two different ways:

- If you create a nickname, the CREATE NICKNAME process will verify the presence of the object at the data source and then attempt to gather existing data source statistical data. Information useful to the DataJoiner optimizer is read from the data source catalogs. Because some or all of the data source catalog information might be used by the DataJoiner optimizer, it is advisable to update statistics (using the data source command equivalent to RUNSTATS) at the data source before you create a nickname.
- If you issue the RUNSTATS command against a nickname, updated statistical data about the object referenced by the nickname is derived by first sending a query to the data source (retrieves the table) and then running the RUNSTATS command locally against the copy of the retrieved table. No statistical data from the data source is used.

DataJoiner does not automatically detect changes to objects referenced by nicknames. You may notice that an object has changed, indirectly, because of slower performance (the optimizer is making decisions based on data that is no longer true) or incorrect results (queries are not retrieving data from new columns added after the nickname was created).

If the object schema has not changed, issuing the RUNSTATS command against the nickname will update the statistical data used by the optimizer. If the object schema has changed (for example, some columns were deleted and others were altered), run the RUNSTATS utility equivalent at the data source. Then, drop the current nickname. Re-create the nickname. The nickname will now have updated statistical information consistent with the object schema.

The REORG TABLE Utility

Queries submitted to a data source from DataJoiner are processed by the data source like queries received from any other application. For this reason, DataJoiner is as susceptible to unorganized data on the data source as your normal applications. Therefore, frequent reorganization of the data stored in DataJoiner, and reorganization of the data stored in any accessed data source, is recommended.

Tuning Configuration Parameters

DataJoiner has a number of instance and database configuration parameters you can tune. These parameters are similar to those found in DB2 for CS and are described in the *DATABASE 2 Administration Guide*.

Some parameters, however, have special considerations in the DataJoiner environment as compared to the DB2 for CS environment or have additional considerations that are operating system dependent. This section discusses the special considerations for DataJoiner that are not operating system dependent. The *DataJoiner Planning, Installation, and Configuration Guide* for your platform lists operating system-specific parameter information specific to DataJoiner (see the memory and disk requirements sections).

Before You Change the Following Configuration Parameters: Most of the following suggestions involve reducing resources available to the DataJoiner database. It is recommended that you analyze your queries, and consider your requirements for spatial data processing, before reducing resources. You might discover that:

- Some queries will complete faster if tables are materialized locally. The DataJoiner optimizer will consider local materialization of tables if that is the least cost route. If tables are materialized locally, you might have to increase the resources available to the local DataJoiner database.
- Some queries will complete faster if tables are sorted locally. In some cases, sorts must occur locally. The DataJoiner optimizer will consider local materialization of tables if that is the least cost route. If tables are materialized locally, you might have to increase the resources available to the local DataJoiner database. If tables are sorted locally, you also might have to increase the resources available to the local DataJoiner database.
- Spatial operations (enabling and then using spatial data at DataJoiner) might require default or higher database resource settings to complete in a timely manner. Specifically, the user defined memory size parameter should be 4096 or higher. See the *DB2 Spatial Extender Administration Guide and Reference* for more information.

The DataJoiner Buffer Pool (BUFFPAGE): The DataJoiner buffer pool is only used to cache pages from the local DataJoiner database. Information retrieved from data sources is not cached here. Therefore, if you do not access local data, you probably will require significantly less buffer pool storage than a standard DB2 for CS database. You can lower the value of the buffer pool size configuration parameter (*buffpage*). See the *DATABASE 2 Administration Guide* for more information on tuning this parameter.

DataJoiner Recovery Log: The only recovery information logged in the DataJoiner recovery log is from operations that affect the local DataJoiner database. Therefore, if you do not access local data, you will require significantly less log space than a standard DB2 for CS database. You can lower the following configuration parameters:

- Size of log files (*logfilsiz*)
- Number of primary log files (*logprimary*)
- Log records to write before soft checkpoint (*softmax*)
- Log buffer size (*logbufsz*)
- Number of commits to group (*mincommit*)
- Location of log files (*logpath*)

Each of these parameters is described in the *DATABASE 2 Administration Guide*.

Locking: DataJoiner locks nicknames once it accesses them (one lock per usage of any given nickname). Locking prevents the nickname from getting dropped when it is still in use. All other local locking operations are unaffected. When a nickname is accessed, a single local lock list entry is used, but multiple locks might be acquired at the data source. This one to many relationship has performance implications. For example, if you don't access local data, you probably require significantly less lock list space than a standard DB2 for CS database. You can lower the following configuration parameters:

- Maximum storage for lock list (*locklist*)
- Maximum percent of lock list before escalation (*maxlocks*)
- Time interval for checking deadlock (*dlchktime*)

Each of these parameters is described in the *DATABASE 2 Administration Guide*.

Locks are acquired at data sources according to their concurrency control paradigms.

Sort (SORTHEAP): Unlike the logging, locking, and buffer-pool processes, DataJoiner's use of the sort facility is identical to DB2 for CS's use of the sort facility. All order-dependent operations that cannot be fully processed at the data source require some sort space on the DataJoiner server. Monitor sort space as described in the *DATABASE 2 Administration Guide*, regardless of whether or not you access local data.

Collating Sequence Considerations

The DataJoiner collating sequence is determined when a DataJoiner database is created. It can be one of two values:

SYSTEM Indicates that the collating sequence is based on the current country code (the default)

IDENTITY Indicates that the collating sequence is the identity sequence, where strings are compared byte for byte.

General Collating Sequence Performance Information

Your choice of collating sequence might affect DataJoiner performance. If a data source uses the same collating sequence as the DataJoiner database, DataJoiner can push down order-dependent processing involving character data to the data source. If a data source collating sequence does not match DataJoiner's, DataJoiner must retrieve the relevant data and do all order-dependent processing on character data (which can slow performance).

To determine if a data source and DataJoiner have the same collating sequence, consider the following factors:

- National language support

  The collating sequence is related to the language supported on a server. Compare the NLS information in the *DataJoiner Planning, Installation, and Configuration Guide* for your platform to the data source NLS information.
- Data source characteristics

  Some data sources are created using case-insensitive collating sequences, which can yield different results from DataJoiner in order-dependent operations.
- Customization

  Some data sources provide multiple options for collating sequences or allow the collating sequence to be customized.

Choose the collating sequence for a given DataJoiner database based on the mix of data sources that will be accessed from that database. For example:

- If a DataJoiner database will access mostly Oracle databases with the same code page (NLS) as DataJoiner, you might want to use DataJoiner's IDENTITY collating sequence because Oracle uses an equivalent collating sequence.
- If a DataJoiner database will access a number of DB2 for CS databases whose collating sequence is SYSTEM and where the code page (NLS) is the same as DataJoiner's, you might want to use DataJoiner's SYSTEM collating sequence.

Colseq Option of SYSCAT.SERVER_OPTIONS

The colseq option of SYSCAT.SERVER_OPTIONS tells DataJoiner if the collating sequence of a given data source matches the collating sequence of the DataJoiner database. It is used when data source collating sequences differ from DataJoiner's or when the data source collating operations might be case insensitive. For example, in a case insensitive data source with an English code page, TOLLESON, ToLLeSoN, and tolleson would all be considered equal.

If you set the colseq option to "Y", you are telling DataJoiner that the data source collating sequence matches the DataJoiner collating sequence. This setting allows the DataJoiner optimizer to consider order dependent processing at a data source, which can improve performance; however, if the data source collating sequence is not the same as the DataJoiner database collating sequence, you can receive incorrect results. For example, if your plan uses merge joins, the DataJoiner optimizer will push down

ordering operations to the data sources as much as possible. If the data source collating sequence is not the same, the join results may not have a correct result set. Set the colseq option to “N” if you are not sure that the collating sequence at the data source is identical to the DataJoiner collating sequence.

Tuning Query Processing

This section describes:

- “The DataJoiner SQL Compiler”
- “Pushdown Analysis” on page 87
- “Global Optimization” on page 92

The DataJoiner SQL Compiler

Figure 11 on page 86 shows a simplified view of the DataJoiner SQL compiler.

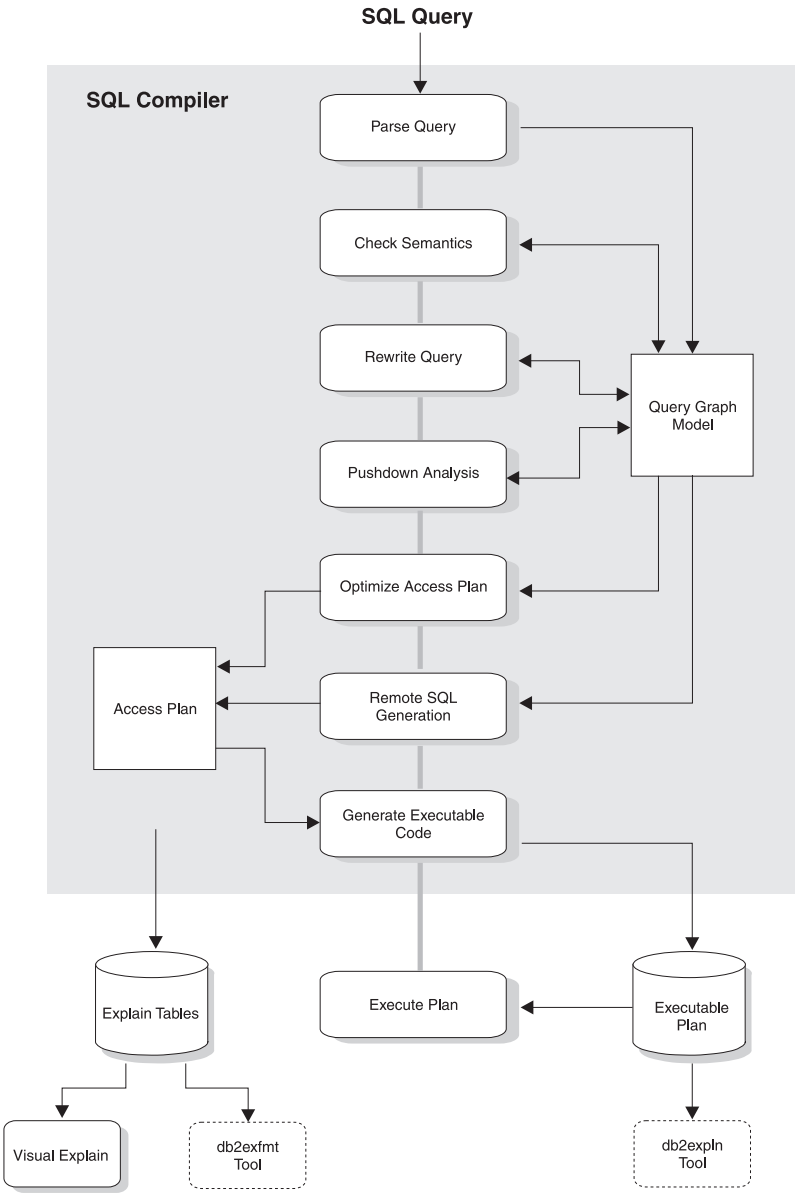


Figure 11. Steps Performed by the DataJoiner SQL Compiler

Most of the steps in the figure match query processing steps documented in the *DATABASE 2 Administration Guide* (SQL Compiler chapter). DataJoiner adds additional steps to the process:

- Pushdown analysis. The major task of this step is to recommend to the DataJoiner optimizer whether an operation can be remotely evaluated. It is just a recommendation because the DataJoiner optimizer may choose to not perform an operation directly on a remote data source because it is less cost effective.

  A secondary task is to attempt to transform the query into a form that can be better optimized by both the DataJoiner optimizer and remote query optimizers.
- Remote SQL generation. The final plan selected by the DataJoiner optimizer can consist of a set of steps that might operate on the remote data source. For those operations that will be performed by each data source, the remote SQL generation step creates an efficient SQL statement based on the data source SQL dialect. This step helps produce a globally (all data sources) optimal plan for the query.

The remainder of this section discusses each step in detail.

Pushdown Analysis

Pushdown analysis tells the DataJoiner optimizer if an operation can be performed at a remote data source. An operation can be a function, such as relational operator, system or user functions, or an SQL operator (GROUP BY, ORDER BY, and so on).

Functions that cannot be pushed-down can significantly impact query performance. Consider the effect of forcing a selective predicate to be evaluated locally instead of at the remote data source. This approach could require DataJoiner to retrieve the entire table from the remote data source and then filter it locally against the predicate. If your network is constrained—and the table is large—query performance could suffer.

Operators that are not pushed-down can also significantly impact query performance. For example, having a GROUP BY operator aggregate remote data locally could, once again, require DataJoiner to retrieve the entire table from the remote data source.

In general, the goal is to ensure that functions and operators can be pushed-down to data sources. Many factors can affect whether a function or an SQL operator is evaluated at a remote data source. The key factors are discussed in three groups: server characteristics, nickname characteristics, and query characteristics.

Server Characteristics Affecting Pushdown Opportunities

The following sections contain data source-specific factors that can affect pushdown opportunities. In general, these factors exist because DataJoiner lets you use the rich DB2 for CS SQL dialect to submit queries. This dialect may offer more functionality than the SQL dialect supported by a data server accessed during a DataJoiner query. DataJoiner can compensate for the lack of function at a data server, but doing so may require that the operation take place at DataJoiner.

SQL Capabilities: Each data source supports a variation of the SQL dialect and different levels of functionality. For example, consider the GROUP BY list. Most data sources support the GROUP BY operator; but, some have restrictions on the number of items on the GROUP BY list or restrictions on whether an expression is allowed on the

GROUP BY list. If there is a restriction at the remote data source, DataJoiner might have to perform the GROUP BY operation locally.

SQL Restrictions: Each data source can have different SQL restrictions. For example, some data sources require parameter markers to bind in values to remote SQL statements. Therefore, parameter marker restrictions must be checked to ensure that each data source can support such a bind mechanism. If DataJoiner cannot determine a good method to bind in a value for a function, this function must be evaluated locally.

SQL Limits: DataJoiner might allow the use of larger integers than its remote data sources; however, limit-exceeding values cannot be embedded in statements sent to data sources. Therefore, the function or operator that operates on this constant must be evaluated locally.

Server Specifics: Several factors fall into this category. One example is sorting NULL values (highest, lowest, or depending on the ordering). For example, if the NULL value is sorted at a data source differently from DataJoiner, ORDER BY operations on a nullable expression cannot be remotely evaluated.

DataJoiner Server Options: The previously listed server characteristics are not usually changed or set by users. The following options, however, can be set by users at DataJoiner and in some cases can affect query performance:

- colseq (collating sequence). If a data source has a collating sequence that differs from DataJoiner's collating sequence, any operation depending on DataJoiner's collating sequence cannot be remotely evaluated at a data source. An example is executing MAX column functions against a nickname character column at a data source with a different collating sequence. Because results might differ if the MAX function is evaluated at the remote data source, DataJoiner will perform the aggregate operation and the MAX function locally.
- varchar_no_trailing_blanks. For varying-length character strings that contain no trailing blanks, Oracle's non-blank-padded comparison semantics return the same results as DataJoiner's comparison semantics. If you are certain that all VARCHAR/VARCHAR2 table/view columns at an Oracle data source contain no trailing blanks, consider creating this server option for a data source. Ensure that you consider all objects that can potentially have nicknames (including views).

DataJoiner Type and Function Mapping Factors: The default local data type mappings provided by DataJoiner (in SYSCAT.SERVER_DATATYPES) are designed so that sufficient buffer space is given to each data source data type (to avoid runtime buffer overflow). A user can choose to customize the type mapping for a specific data source to suit specific applications. For example, if you are accessing an Oracle data source column with a DATE data type (which by default is mapped to the DataJoiner TIMESTAMP data type), you could change the local data type to the DataJoiner DATE data type. This change bypasses the use of a SCALAR function to extract a subset of the total data stored in the TIMESTAMP data type.

For function mapping, DataJoiner relies on entries in SYSCAT.SERVER_FUNCTIONS to identify which functions are supported by a remote data source. DataJoiner will compensate for functions not supported by a data source. There are three cases where function compensation will occur:

- This function simply does not exist at the remote data source. The SYSFUN functions, for example, do not exist on DB2 for OS/390 data sources, and thus require local compensation.
- The function does exist; however, the characteristics of the operand violate function restrictions. An example is the IS NULL relational operator. Most data sources support it, but some may have restrictions, such as only allowing a column name on the left hand side of the IS NULL operator.
- The function, if evaluated remotely, may return a different result. An example is the '>' (greater than) operator. For those data sources with different collating sequences, the greater than operator may return different results than if it is evaluated locally by DataJoiner.

Nickname Characteristics Affecting Pushdown Opportunities

The following sections contain nickname-specific factors that can affect pushdown opportunities.

Local Data Type of a Nickname Column: Ensure that the local data type of a column does not prevent a predicate from being evaluated at the data source. As mentioned earlier, the default data type mappings are provided to avoid any possible overflow. However, a joining predicate between two columns of different lengths might not be considered at the data source whose joining column is shorter, depending on how DataJoiner binds in the longer column. This situation can affect the number of possibilities in a joining sequence evaluated by the DataJoiner optimizer. For example, Oracle data source columns created using the INTEGER or INT data type are given the type NUMBER(38). A nickname column for this Oracle data type will be given the local data type FLOAT because the range of a DataJoiner integer is from 2**31 to (-2**31)-1, which is roughly equal to NUMBER(9). In this case, joins between a DB2 for CS integer column and an Oracle integer column cannot take place at the DB2 for CS data source (shorter joining column); however, if the domain of this Oracle integer column can be accommodated by the DataJoiner INTEGER data type, change its local data type with the ALTER NICKNAME statement so that the join can take place at the DB2 for CS data source.

VARCHAR/VARCHAR2 Columns at Oracle Data Sources: The ALTER NICKNAME SQL statement can indicate to DataJoiner pushdown analysis that a specific Oracle column contains no trailing blanks. The pushdown analysis step will then take this information into account when checking all operations performed on columns so indicated.

Based on this indication, DataJoiner may generate a different but equivalent form of a predicate to be used in the remote SQL statement sent to an Oracle data source. A user might see a different predicate being evaluated against the Oracle data source, but the net result should be equivalent.

Query Characteristics Affecting Pushdown Opportunities

A query can reference an SQL operator that might involve nicknames from multiple data sources. When DataJoiner must combine the results from two referenced data sources using one operator, such as a set operator (e.g. UNION), the operation must take place at DataJoiner. The operator cannot be evaluated at a remote data source directly.

Pushdown Analysis Trouble-Shooting

Rewriting SQL statements can provide additional pushdown opportunities for DataJoiner query processing. This section introduces tools for determining where a query is evaluated, lists common questions (and suggested areas to investigate) associated with query analysis, and addresses data source upgrade issues.

Determining Where a Query is Evaluated: There are two utilities provided with DataJoiner that show where queries are evaluated:

- Visual explain. Start it with the **db2dd** or the **db2vexp** command. Use it to view the query access plan graph. The execution location for each operator is included in the detailed display of an operator. You can also find the remote SQL statement generated for each data source in the RQUERY (select operation), RINSERT (insert into a nickname), RUPDATE (update a nickname) and RDELETE (delete a nickname) operators. If a statement is completely evaluated at the remote data source, you should see a SHIP operator on top of one of the above operators (based on your statement) in the access plan graph with no other major operator.
- SQL explain. Start it with the **db2expln** or the **dynexpln** command. Use it to view the access plan strategy as text. SQL explain does not provide a graphical user interface; however, it can be used to verify the remote plan chosen by the remote optimizer.

Understanding Why a Query is Evaluated at a Data Source or at DataJoiner: This section lists typical plan analysis questions and areas to investigate to increase pushdown opportunities. Key questions include:

- Why isn't this predicate being evaluated remotely?

  This question arises when a predicate is very selective and thus could be used to filter rows and reduce network traffic. Remote predicate evaluation also affects whether a join between two tables of the same data source can be evaluated remotely.

  Areas to examine include:

  – Subquery predicates. Does this predicate contain a subquery that pertains to another data source? Does this predicate contain a subquery involving an SQL operator that is not supported by this data source? Not all data sources support set operators in a predicate

  – Predicate functions. Does this predicate contain a function that cannot be evaluated by this remote data source? Relational operators are classified as functions.

  – Predicate bind requirements. Does this predicate, if remotely evaluated, require bind-in of some value? If so, would it violate SQL restrictions at this data source?

  - Global optimization. The optimizer may have decided that local processing is more cost effective. See “Global Optimization” on page 92 for more information.

- Why isn’t the GROUP BY operator evaluated remotely?

  There are several areas you can check:

  - Is the input to the GROUP BY operator evaluated remotely? If the answer is no, examine the input.
  - Does the data source have any restrictions on this operator? Examples include:
    - Limited number of GROUP BY items
    - Limited byte counts of combined GROUP BY items
    - Column specification only on the GROUP BY list
  - Does the data source support this SQL operator?
  - Global optimization. The optimizer may have decided that local processing is more cost effective. See “Global Optimization” on page 92 for more information.

- Why isn’t the set operator evaluated remotely?

  There are several areas you can check:

  - Are both of its operands completely evaluated at the same remote data source? If the answer is no and it should be yes, examine each operand.
  - Does the data source have any restrictions on this set operator? For example, are large objects or long fields valid input for this specific set operator?

- Why isn’t the ORDER BY operation evaluated remotely?

  Consider:

  - Is the input to the ORDER BY operation evaluated remotely? If the answer is no, examine the input.
  - Does the ORDER BY clause contain a character expression? If yes, does the remote data source not have the same collating sequence as DataJoiner?
  - Does the data source have any restrictions on this operator? For example, is there a limited number of ORDER BY items? Does the data source restrict column specification to the ORDER BY list?

- Why is a remote INSERT with a subselect statement not completely evaluated at the remote data source?

  Consider:

  - Could the subselect be completely evaluated on the remote data source? If no, examine the subselect.
  - Does the subselect contain a set operator? If yes, does this data source support set operators as input to an INSERT?
  - Does the subselect reference the target table? If yes, does this data source allow this syntax?
  - Does the data source support such syntax (remote INSERT with a subselect statement)?

- Why is a remote INSERT statement not completely evaluated at the remote data source?

Consider:

- Is there an expression in the VALUES list? Does this data source support an expression in the VALUES list?
- Does the expression involve a scalar subquery? Is that syntax supported?
- Does the expression reference the target table? Is that syntax supported?

- Why is a remote, searched UPDATE statement not completely evaluated at the remote data source?

  Consider:

  - Can the SET clause be completely evaluated at the remote data source? In other words, does an update expression contain a function not supported by the remote data source?
  - Can the search condition be completely evaluated at the remote data source? If the answer is no, examine the search condition instead.
  - Does the search condition reference the target table? Does the data source allow this syntax?

- Why is a positioned UPDATE statement not completely evaluated at the remote data source?

  This happens when DataJoiner chooses to evaluate the update expression locally before sending the UPDATE statement to the data source. This approach should not significantly affect performance.

- Why is a remote, searched DELETE statement not completely evaluated at the remote data source?

  Consider:

  - Can the search condition be completely evaluated at the remote data source? If the answer is no, examine the search condition instead.
  - Does the search condition reference the target table? Does the data source allow this syntax?

Data Source Upgrades and Customization: Although the DataJoiner SQL compiler has much information about native data source SQL support, this data may need adjustment over time because data sources can be upgraded and/or customized. In such cases, make enhancements known to DataJoiner by changing local catalog information. Use DataJoiner SQL DDL statements (such as CREATE FUNCTION MAPPING and CREATE SERVER OPTION) to update the catalog. See the *DataJoiner Application Programming and SQL Reference Supplement* for information on system catalog tables and the columns that can be updated.

Global Optimization

Global optimization produces a globally optimal access strategy to evaluate a query. For a heterogeneous query, the access strategy may involve breaking down the original query into a set of remote query units and then combining the results.

Using the output of pushdown analysis as a recommendation, the DataJoiner optimizer decides whether each operation will be evaluated locally at DataJoiner or remotely at

the data source. The decision is based on the output of its sophisticated cost model, which includes not only the cost to evaluate the operation but also the cost to transmit the data or messages between DataJoiner and data sources.

The goal is to produce an optimized query; however, many factors can affect the output from global optimization and thus affect query performance. The key factors are discussed in two groups: server characteristics and nickname characteristics.

Server Characteristics Affecting Global Optimization

Data source-specific factors that can affect global optimization include the:

- Relative ratio of CPU speed

  This value indicates how much faster or slower the data source CPU speed is compared with the DataJoiner CPU. A low ratio indicates that the data source workstation CPU is faster than the DataJoiner workstation CPU. For low ratios, the DataJoiner optimizer will consider pushing-down CPU-intensive operations to the data source. See the *DataJoiner Application Programming and SQL Reference Supplement* for more information about this ratio.

- Relative ratio of I/O speed

  This value indicates how much faster or slower the data source I/O speed is compared with the DataJoiner I/O speed. A low ratio indicates that the data source workstation I/O speed is faster than the DataJoiner workstation I/O speed. For low ratios, the DataJoiner optimizer will consider pushing-down I/O-intensive operations to the data source. See the *DataJoiner Application Programming and SQL Reference Supplement* for more information about this ratio.

- Communication rate between DataJoiner and the data source

  Lower communication rates (slow network communication between DataJoiner and the data source) encourage the DataJoiner optimizer to reduce the number of messages sent to or from this data source. If the rate is set to 0, the optimizer produces a query requiring minimal network traffic. See the *DataJoiner Application Programming and SQL Reference Supplement* for more information about this setting.

- Data source collating sequence

  If a data source collating sequence does not match the local DataJoiner database collating sequence, the optimizer considers the data retrieved from this data source as unordered.

  To encourage sorting at data sources, set the DataJoiner system catalog collating sequence value for a data source to Y (a data source has the same collating sequence as DataJoiner) when one of the following is true:

  1. All character data is upper case
  2. All character data is lower case
  3. All character data is composed of numbers

  See “Collating Sequence Considerations” on page 83 for more information about collating sequence performance issues.

- Information in the DataJoiner optimizer knowledge base

  DataJoiner has an optimizer knowledge base that contains data about native data sources. The DataJoiner optimizer does not generate remote access plans that cannot be generated by specific DBMSs. In other words, DataJoiner avoids generating plans that optimizers at remote data sources cannot understand or accept.

Server Options Affecting Global Optimization

Server options that can affect global optimization include:

- deferred_lob_retrieval

  The deferred lob_retrieval option setting enables DataJoiner to fetch LOBs after a query is processed. This option can prevent LOBs from being materialized in temporary relations at DataJoiner. Only those LOBs that are in the result set are then fetched from the data source.

  See the *DataJoiner Application Programming and SQL Reference Supplement* for more information about this option.

- remote_query_caching

  This option can improve query performance by reducing or omitting the need to re-fetch result sets. Queries that normally re-fetch result sets when the ″bind in″ host variable has the same value as a previous fetch can use the result set held within the cache.

  See “Remote Query Caching” on page 98 for more information about this option.

Nickname Characteristics Affecting Global Optimization

The following sections contain nickname-specific factors that can affect global optimization.

Index Considerations:

DataJoiner uses information about indexes at data sources to optimize queries. For this reason, it is important that the index information available to DataJoiner is current. The index information for nicknames is initially acquired at create nickname time.

While DataJoiner can retrieve the statistical data held at a data source, it cannot automatically detect updates to existing statistical data at data sources. Furthermore, DataJoiner has no mechanism for handling schema changes.

If the statistical data for an object has changed, but the object schema has not changed, issuing the RUNSTATS command against the nickname will update the statistical data used by the optimizer. If the statistical data for an object has changed, and the object schema has changed (for example, some columns were deleted and others were altered), run the RUNSTATS utility equivalent at the data source. Then,

drop the current nickname. Re-create the nickname. The nickname will now have updated statistical information consistent with the object schema.

Index Management: Applications do not decide when an index should be used: DataJoiner makes this decision based on the available table and index information. However, DBAs play an important role in the process by creating the necessary indexes that can improve performance.

Specific instructions for index management will differ for each remote data source. Check the appropriate product documentation for details. The following points are general guidelines for an enterprise indexing strategy:

- Define *primary keys* wherever they apply. Each primary key has a unique index. A primary key is a unique key that is part of the table definition. It is good practice to create one for a new table.
- Use indexes to optimize frequent queries to tables with more than 15 data pages. In DataJoiner's local database, one *data page* is a unit of storage within a table or index. The size of a data page is 4KB. Indexes also should be used for tables with more than 10 data pages that are primarily used as read only.
- Create an index on any columns you will use when joining tables.
- Create an index on any column from which you will be searching for particular values on a regular basis.
- Consider *not* using an index for tables smaller than six pages.
- Avoid creating indexes that are partial keys of other index keys on the columns. For example, if there is an index on columns x, y, and z, then a second index on columns y and z is not necessarily useful.
- Use indexes on foreign keys to improve the performance of delete and update operations on the parent table.
- It is more efficient to create an index after the table is loaded with data. This avoids the overhead of maintaining the index during data load. This approach also minimizes index space requirements.

Creating Indexes on Nicknames: Use CREATE INDEX statements against table nicknames for data source tables; don't issue CREATE INDEX statements against a nickname for a view.

For table nicknames, creating indexes is useful when:

- DataJoiner is unable to retrieve any index information from a data source, either via system catalogs or system stored procedures regarding a nickname, during nickname creation. SYSCAT.INDEXES would not contain any information about the nickname for the DataJoiner optimizer to consider. Classic Connect, for example, utilizes data sources that do not externalize index information.
- A user wants to encourage the DataJoiner optimizer to use a specific nickname as the inner table of a nested loop join. The user can create an index on the joining column if none exists.

The syntax for creating an index on a nickname is identical to the syntax for creating an index on a local DataJoiner table.

Catalog Statistics (in SYSCAT.TABLES and SYSCAT.COLUMNS) Considerations: This set of catalog information describes the overall size of tables/views and the range of values in associated columns. The information includes the:

- Number of rows in a nickname object
- Number of pages that a nickname occupies
- Highest/lowest values of a column

DataJoiner provides the RUNSTATS utility to generate local statistics. These statistics can be updated directly in DataJoiner. The RUNSTATS utility is described in "RUNSTATS (Update Statistics)" on page 104. See the *DataJoiner Application Programming and SQL Reference Supplement* for information on the DataJoiner system catalog and the columns that can be updated.

Use caution when providing statistics manually or issuing RUNSTATS on a nickname over a remote view. The statistical information, such as the number of rows this nickname will return, might not reflect the real cost to evaluate this remote view and thus might mislead the DataJoiner optimizer. Situations that can benefit from statistics include remote views defined on a single base table with no column functions applied on the SELECT list. Most complex views, however, might require more complex tuning process which unfortunately might require tuning for each query. Consider ceating views over nicknames instead so the DataJoiner optimizer knows how to derive the cost of the view more accurately.

Index Information Considerations: Like catalog statistics, index information for a table nickname is retrieved during nickname creation. Index information is not gathered for view nicknames. DataJoiner allows the CREATE INDEX statement to be issued against nicknames (both tables and views) so that additional information can be made available to the DataJoiner optimizer when such information is not externalized by the data source.

Be cautious when issuing CREATE INDEX statements against a nickname for a view. In one case, if the view is a simple SELECT on a table with an index, creating indexes on the nickname (locally) that match the indexes on the table at the data source can significantly improve query performance. However, if indexes are created locally over views that are not simple select statements (for example, a view created by joining two tables), query performance may suffer. For example, if an index is created over a view that is a join of two tables, the optimizer may choose that view as the inner element in a nested loop join. The query will have poor performance because the join will be evaluated several times.

Global Optimization Trouble-Shooting

This section introduces tools for analyzing query optimization and presents common questions (and suggested areas to investigate) associated with query analysis.

Analyzing Query Optimization: There are two utilities provided with DataJoiner that show global access plans:

- Visual explain. Start it with the db2dd or db2vexp command. Use it to view the query access plan graph. The execution location for each operator is included in the detailed display of an operator. You can also find the remote SQL statement generated for each data source in the RQUERY (select operation), RINSERT (insert into a nickname), RUPDATE (update a nickname) and RDELETE (delete a nickname) operators. By examining the details of each operator, you can see the number of rows estimated by the DataJoiner optimizer as input to and output from each operator. You can also see the estimated cost to execute each operator including the communications cost.
- SQL explain. Start it with the db2expln or dynexpln command. Use it to view the access plan strategy as text. SQL explain does not provide a graphical user interface. SQL explain does not provide cost information; however, you can get the access plan generated by the remote optimizer for those data sources supported by the remote explain function. See “Appendix A. SQL Explain Utilities” on page 145 for more information on the supported data source types and the prerequisites.

Understanding DataJoiner Optimization Decisions: This section lists optimization questions and key areas to investigate to improve performance. Key questions include:

- Why isn’t a join between two nicknames of the same data source being evaluated remotely?

  Areas to examine include:

  – Join operations. Can the data source support them?

  – Join predicates. Can the join predicate be evaluated at the remote data source? If the answer is no, examine the join predicate. See “Understanding Why a Query is Evaluated at a Data Source or at DataJoiner” on page 90 for more information.

  – Number of rows in the join result (with visual explain). Does the join produce a much larger set of rows than the two nicknames combined? Do the numbers make sense? If the answer is no, consider updating the nickname statistics with the RUNSTATS utility.

- Why isn’t the GROUP BY operator being evaluated remotely?

  Areas to examine include:

  – Operator syntax—verify that the operator can be evaluated at the remote data source. See “Understanding Why a Query is Evaluated at a Data Source or at DataJoiner” on page 90 for more information.

  – Number of rows. Check the estimated number of rows in the GROUP BY operator input and output using visual explain. Are these two numbers very close? If the answer is yes, the DataJoiner optimizer considers it more efficient to evaluate this GROUP BY locally. Also, do these two numbers make sense? If the answer is no, consider updating the nickname statistics using RUNSTATS.

- Why is the statement not being completely evaluated by the remote data source?

  The DataJoiner Optimizer performs cost-based optimization. Even if pushdown analysis indicates that every operator can be evaluated at the remote data source, the optimizer still relies on its cost estimate to generate a globally optimal plan. There

are a great many factors that can contribute to that plan. For example, even though the remote data source can process every operation in the original query, its CPU speed is much slower than DataJoiner's and thus it may turn out to be more beneficial to perform the operations at DataJoiner instead. If results are not satisfactory, verify your server statistics in SYSSTAT.SERVERS.

- Why does a plan generated by the optimizer, and completely evaluated at a remote data source, have much worse performance than the original query executed directly at the remote data source?

  Areas to examine include:

  – The remote SQL statement generated by the DataJoiner optimizer. Ensure that it is identical to the original query. Check for predicate ordering changes. A good query optimizer should not be sensitive to the predicate ordering of a query; unfortunately, not all DBMS optimizers are identical, and thus it is likely that the optimizer of the remote data source may generate a different plan based on the input predicate ordering. If this is true, this is a problem inherent in the remote optimizer. Consider either modifying the predicate ordering on the input to DataJoiner or contacting the service organization of the remote data source for assistance.

    Also, check for predicate replacements. A good query optimizer should not be sensitive to equivalent predicate replacements; unfortunately, not all DBMS optimizers are identical, and thus it is possible that the optimizer of the remote data source may generate a different plan based on the input predicate. For example, some optimizers cannot generate transitive closure statements for predicates.

  – The number of returned rows. You can get this number from visual explain. If the query returns a large number of rows, network traffic is a potential bottleneck.

  – Additional functions. Does the remote SQL statement contain additional functions compared with the original query? Some of the extra functions may be generated to convert data types. Ensure that they are necessary.

Remote Query Caching

Remote query caching refers to a server option that, for certain query types, improves query performance by reducing or omitting the need to re-fetch result sets. By enabling this server option, queries that normally re-fetch result sets when the ″bind in″ host variable has the same value as a previous fetch can use the result set held within the cache. Query performance should improve when you are submitting queries that require host variable binding. Examples include nested loop joins or correlated subqueries.

Remote query caching is enabled by setting the remote_query_caching server option value to “Y”. Set values for this option with either the CREATE SERVER OPTION or the SET SERVER OPTION SQL statements. For example, to enable remote query caching for the server oracle1, enter:

```
SET SERVER OPTION remote_query_caching to 'Y' FOR SERVER oracle1
```

The cache size is set to 2 MB (cannot be changed by user). The default setting is ″N″.

In general, if you are performing nested loop joins or other queries that reference host variables, enabling this server option should improve run-time performance. Caching applies to both inter-query and intra-query processing. The cache is freed up at commit time.

Remote query caching should not be used when the DataJoiner isolation level is set below Read Stability (see the *DATABASE 2 Administration Guide* for more information on isolation levels) and the application requires the most current data on the remote table. Also, remote query caching can consume up to 2 MB of memory per application. Consider disabling it for memory-constrained systems.

Remote Plan Hints

Remote plan hints are statement fragments that provide extra information for data source optimizers. This information can, for certain query types, improve query performance. The plan hints can help the data source optimizer decide whether to use an index, which index to use, or which table join sequence to use.

If plan hints are enabled, the query sent to the data source contains additional information. For example, a statement sent to an Oracle optimizer with plan hints could look like this:

```
SELECT /*+ INDEX (table1, t1index)*/
   col1
   FROM table1
```

The plan hint is the string `/*+ INDEX (table1, t1index)*/`.

Plan hints are supported by Oracle and Sybase SQL Server data sources.

Enable the use of plan hints by setting the server option plan_hints on. Server options can be set using either the SET SERVER OPTION or CREATE SERVER OPTION SQL statements. For example, to temporarily enable the user of plan hints for the Oracle server ORASEB1, issue the statement:

```
SET SERVER OPTION plan_hints to 'Y' FOR SERVER ORASEB1
```

Tuning DataJoiner Network Usage

This section provides a few suggestions that might improve query performance by optimizing DataJoiner's use of your network.

fold_id and fold_pw Options of SYSCAT.SERVER_OPTIONS

The fold_id and fold_pw options of SYSCAT.SERVER_OPTIONS control the transformations that occur to the user name and password sent to the specified data source. Depending on the settings of these options, multiple attempts can be made to access the data source with different user name/password combinations. In a worst

case scenario DataJoiner will attempt to access each data source 3 times before finding the correct combination of values (this is described in “Folding Authorization Names and Passwords” on page 24). Choosing a correct setting will minimize network load and increase performance. It is recommended that you set fold_id and fold_pw as follows:

- **fold_id**

  ’L’ Folds to lower case if the user name at the data source is lower case (typically true of Sybase)

  ’U’ Folds to upper case if the user name at the data source is upper case (typically true for the DB2 family and Oracle)

  ’N’ Doesn’t fold if the user name is already in the correct case.

  Use NULL minimally, if at all.

- **fold_pw**

  L Folds to lower case if the password at the data source is lower case (typically true of Sybase)

  U Folds to upper case if the password at the data source is upper case (typically true for the DB2 family and Oracle)

  N Doesn’t fold if the user name is already in the correct case.

  Use NULL minimally, if at all.

Match Client I/O Block Size and Packet Size

Settings for the client I/O block (rqrioblk) database manager parameter can affect performance. First, ensure that the current setting is a value that can divide evenly into the value set for your TCP/IP packet size. A mismatch can slow performance.

Also, consider the size of data being transmitted by single SQL statements. Large objects may require an increase from the default setting (4096 bytes).

See the *DATABASE 2 Administration Guide* for additional information.

Match Sybase TDS Packet Size to Network TCP/IP Packet Size

You can use CONNECTSTRING entries to ensure a match between the Open Client packet size and the packet size for your network. This information is stored in the DataJoiner catalog (SYSCAT.SERVER_OPTIONS). CONNECTSTRING syntax is documented in the *DataJoiner Planning, Installation, and Configuration Guide* for your platform.

Increase Network Operating System Prioritization

Some operating systems allow you to assign specific resources a higher or lower processing priority. Consider increasing the priority for network operations and lowering

the priority of DataJoiner processes. This approach can yield substantial query performance increases when multiple data sources are involved and local table materialization is not required for join or sort operations.

Tuning Data Source Configurations

In some cases, you can change configuration information at a remote data source to improve performance. This section is organized by data sources.

Classic Connect

It is possible to significantly enhance the performance of queries involving Classic Connect data sources (IMS, VSAM). For detailed technical information on factors affecting query performance, see the *Classic Connect Planning, Installation, and Configuration Guide*.

Microsoft SQL Server

By default, the initial Microsoft SQL Server collating sequence setting is case insensitive. This setting can yield unacceptable query performance because DataJoiner will not let the data source evaluate alphabetic character comparisons. If possible, avoid this collating sequence setting by not using the default. Specify that the database be sensitive to case.

SQL Anywhere

By default, the initial SQL Anywhere collating sequence setting is case insensitive. This setting can yield unacceptable query performance because DataJoiner will not let the data source evaluate alphabetic character comparisons. All comparisons will take place at DataJoiner (no pushdown opportunities). If possible, avoid this collating sequence setting by not using the default. Specify that the database be sensitive to case.

SQL Anywhere also specifies, by default, that trailing blanks are significant. If possible, specify that the database ignores trailing blanks. This step enables additional pushdown opportunities for DataJoiner.

Chapter 8. Data Movement and Analysis Utilities

This chapter provides overview information on the utilities that allow you to manipulate data in a DataJoiner environment, including:

- IMPORT, which inserts data from an input file into a table or view, with the input file containing data from another database or spreadsheet program.
- EXPORT, which copies data from a table or view to output files for use by another database or spreadsheet program.
- RUNSTATS, which updates statistics on a nickname, table, or view.
- REORGCHK, which determines whether to reorganize a nickname or table.
- BACKUP and RESTORE, which help to recover a DataJoiner database.

Complete information on how to use these utilities, including syntax, is in the *DATABASE 2 Command Reference*. See also the *DATABASE 2 Administration Guide*, which provides additional information on using these utilities in a local database environment.

This chapter does not discuss other vendor's products that might be available to manipulate or recover data.

IMPORT

The IMPORT utility inserts data into a local table, a local view, or a nickname. If the table or view receiving the imported data already contains data, you can Replace, Insert, or Insert_Update the data in the existing table or view with the data in the file.

Notes:

1. The Create keyword is not supported for nicknames
2. Data in an existing table cannot be replaced if the existing table is the parent table containing a primary key that is referenced by a foreign key in a dependent table
3. Some data sources may require a primary key defined on the table referenced by the nickname
4. The COMPOUND clause will not substantially increase performance when importing data to a nickname

In a heterogeneous database environment, IMPORT works against all data sources that support all four of the following SQL statements:

- LOCK TABLE
- DELETE
- INSERT
- UPDATE

For example, you can import to a nickname on Oracle and on DB2 databases; however, because Sybase does not support the LOCK TABLE statement, you cannot import data to a Sybase nickname.

The following information is required when importing data to a table or view:

- The path and the input file name in which the data to be imported is stored.
- The name of the table within the database to which the data should be imported.
- The format of the data in the input file. This format can be IXF, WSF, DEL, or ASC. See *DATABASE 2 Administration Guide* for a discussion of these formats.
- The options to use when copying the data in the input file. You can insert, update, replace or append the data in the table or view with the imported data.
- A message file name.

To import data into a new table, you must have SYSADM authority, DBADM authority, or CREATETAB privilege for the database. To replace data in an existing nickname, table, or view, you must have SYSADM authority, DBADM authority, CONTROL privilege for that object. To append data to an existing nickname, table or view, you must have SELECT and INSERT privileges for that object.

Information on how to run the IMPORT utility is in *DATABASE 2 Command Reference* Additional administrative information on using IMPORT in a local database environment is in *DATABASE 2 Administration Guide*.

EXPORT

The EXPORT utility exports data into an operating system file from a database. The output file has the format specified by the data format parameter.

EXPORT can be used against all nicknames in a heterogeneous database environment.

Information on how to run the EXPORT utility is in *DATABASE 2 Command Reference* Additional administrative information on using EXPORT in a local database environment is in *DATABASE 2 Administration Guide*.

RUNSTATS (Update Statistics)

The RUNSTATS utility updates statistics about the physical characteristics of database nicknames, tables, and indexes. DataJoiner uses these statistics to determine the optimal access paths to tables.

While DataJoiner can retrieve the statistical data held at a data source (when creating nicknames), it cannot automatically detect updates to existing statistical data at data sources. Furthermore, DataJoiner has no mechanism for handling nickname schema changes.

If the statistical data for an object has changed, but the object schema has not changed, issuing the RUNSTATS command against the nickname will update the statistical data used by the optimizer. If the statistical data for an object has changed, and the object schema has changed (for example, some columns were deleted and others were altered), run the RUNSTATS utility equivalent at the data source. Then, drop the current nickname. Re-create the nickname. The nickname will now have updated statistical information consistent with the object schema.

Information on how to use the RUNSTATS utility is in the *DATABASE 2 Command Reference*.

REORGCHK (Determine Whether to Reorganize a Table or Nickname)

REORGCHK is used to determine if a table or nickname should be reorganized. You can use the REORGCHK utility with nicknames; however, it will use DataJoiner's knowledge of clustering data to determine if the nickname or table must be reorganized. It will not use information from the data source.

REORG, however, cannot be used against a nickname. If you use REORG against a nickname, you will receive the message SQL2212N.

BACKUP and RESTORE

DataJoiner provides the means to backup and restore a DataJoiner local database. There are two types of data included in any backup or restore:

- **Catalog data**, which includes standard DB2 catalog data and catalog data unique to DataJoiner (for example, data source data in SYSCAT.SERVERS, SYSCAT.REMOTEUSERS, and so on).
- **User data**, which is stored in tables created in the local DataJoiner database.

The DataJoiner backup and restore facilities work with nickname metadata; however, they do not backup and restore any data at data sources. In addition, DataJoiner does not provide facilities for recovery of data source databases. For information on recovery methods for a data source, refer to the product documentation provided by the data source's vendor.

For complete information on backing up and recovering a DataJoiner local database, see the *DATABASE 2 Administration Guide*.

Chapter 9. Database System Monitor

This chapter provides information about using the database system monitor in a heterogeneous database environment.

- Database system monitor overview information
- Data source database status information
- Data source application status information

Purpose of the Database System Monitor

The database system monitor provides a wide variety of statistical information about the operation of DataJoiner. This information can be used to help you:

- Better understand how DataJoiner works
- Improve database and application performance
- Tune database and DataJoiner configuration parameters
- Determine the source and cause of problems
- Understand user and application activity within DataJoiner

You can control and access the monitor information by:

- Coding your own programs which call the APIs provided
- Using the commands provided with the command line processor interface

In both cases, the syntax and examples required to access the monitor are provided in the *DATABASE 2 Database System Monitor Guide and Reference*.

The information gathered by the database system monitor is generated by the internal components of DataJoiner. The DataJoiner level of information is first recorded when DataJoiner is started and is recorded until the DataJoiner instance is stopped. Database information is first recorded when the first connection is made to the database and is recorded until the database is quiesced, which occurs after the last application disconnects from the database. Application information is recorded starting when the application connects to a database and is recorded until it disconnects from the database. The recorded information contains status information (the current state of DataJoiner) and activity information (counters and other measurements of database processing).

Some *basic* monitoring information will always be collected. The database system monitor also allows you to selectivity collect other statistical information. Monitoring many or all elements can slow database performance, so it is important to assess your monitoring requirements.

Element Summary by DataJoiner Object

The tables in this section provide a summary of the database system monitor data elements that apply to a heterogeneous database environment. The elements are grouped in these tables by the API structure in which they are contained. Since the command line processor output follows the API structures to a fairly large extent, you can also use these tables to help you find the detailed information for a specific element while analyzing the command line processor output.

In the column titled **Resettable** in the following tables, the elements whose contents can be reset are indicated with 'Yes'.

The next section, “Data Elements Detail by Function” on page 110, provides detailed information about the database system monitor data elements that apply to a heterogeneous database environment.

General Information

The following elements provide general information about the data collected by the database system monitor. (API information is returned in the *sqlm_collected* data structure.)

Element Title	Element API Name	Monitor Group	Resettable
“Data Source Database Names” on page 111	dbase_remote	Basic	No
“Data Source Applications” on page 112	appl_remote	Basic	No

Data Source Database Status

The following elements provide information about the data source database for which status information is being collected. (API information is returned in the *sqlm_dbase* data structure.)

Element Title	Element API Name	Monitor Group	Resettable
“Data Source Name” on page 113	name	Basic	No
“Data Source Database Name” on page 113	dbase	Basic	No
“Connects” on page 114	connects	Basic	Yes
“Disconnects” on page 114	disconnects	Basic	Yes
“Commits” on page 115	commits	Basic	Yes
“Rollbacks” on page 115	rollbacks	Basic	Yes
“Queries” on page 116	queries	Basic	Yes
“Inserts” on page 117	inserts	Basic	Yes
“Updates” on page 117	updates	Basic	Yes
“Deletes” on page 118	deletes	Basic	Yes

Element Title	Element API Name	Monitor Group	Resettable
“Create Nicknames” on page 118	createNickname	Basic	Yes
“Pass-Through” on page 119	passThrus	Basic	Yes
“Stored Procedures” on page 120	storedProcs	Basic	Yes
“Rows Returned” on page 120	rowsReturned	Basic	Yes
“Rows Updated” on page 121	rowsUpdated	Basic	Yes
“Rows Deleted” on page 121	rowsDeleted	Basic	Yes
“Rows Inserted” on page 122	rowsInserted	Basic	Yes
“Rows Returned by Stored Procedures” on page 123	sprowsReturned	Basic	Yes
“Failed Statements” on page 123	failedStatements	Basic	Yes
“Query Response Time” on page 124	queryTime	Basic	Yes
“Insert Response Time” on page 125	insertTime	Basic	Yes
“Update Response Time” on page 125	updateTime	Basic	Yes
“Delete Response Time” on page 126	deleteTime	Basic	Yes
“Create Nickname Response Time” on page 126	createNicknameTime	Basic	Yes
“Pass-Through Time” on page 127	passthruTime	Basic	Yes
“Stored Procedure Time” on page 128	storedProcTime	Basic	Yes

Data Source Application Status

The following elements provide information about a particular application at a data source. For DataJoiner, an application exists only during the time it is connected to the data source database. (API information is returned in the *sqlm_appl* and *sqlm_appl_id* data structures.)

Note: Several new elements (Agent ID through Input Database Alias) were added for DB2 for CS V2 application snapshots that are not documented in this manual. See the *DATABASE 2 Database System Monitor Guide and Reference* for details.

Element Title	Element API Name	Monitor Group	Resettable
“Application Identification” on page 129	sqlm_appl_id_info	Basic	No
“Data Source Name” on page 130	name	Basic	Yes
“Data Source Database Name” on page 130	dbase	Basic	Yes
“Commits” on page 130	commits	Basic	Yes
“Rollbacks” on page 131	rollbacks	Basic	Yes
“Queries” on page 132	queries	Basic	Yes
“Inserts” on page 132	inserts	Basic	Yes
“Updates” on page 133	updates	Basic	Yes
“Deletes” on page 134	deletes	Basic	Yes

Element Title	Element API Name	Monitor Group	Resettable
“Create Nicknames” on page 134	createNickname	Basic	Yes
“Pass-Through” on page 135	passThrus	Basic	Yes
“Stored Procedures” on page 135	storedProcs	Basic	Yes
“Rows Returned” on page 136	rowsReturned	Basic	Yes
“Rows Updated” on page 136	rowsUpdated	Basic	Yes
“Rows Deleted” on page 137	rowsDeleted	Basic	Yes
“Rows Inserted” on page 138	rowsInserted	Basic	Yes
“Rows Returned by Stored Procedures” on page 138	sprowsReturned	Basic	Yes
“Failed Statements” on page 139	failedStatements	Basic	Yes
“Query Response Time” on page 139	queryTime	Basic	Yes
“Insert Response Time” on page 140	insertTime	Basic	Yes
“Update Response Time” on page 141	updateTime	Basic	Yes
“Delete Response Time” on page 141	deleteTime	Basic	Yes
“Create Nickname Response Time” on page 142	createNicknameTime	Basic	Yes
“Pass-Through Time” on page 143	passthruTime	Basic	Yes
“Stored Procedure Time” on page 143	storedProcTime	Basic	Yes

Data Elements Detail by Function

The following sections contain details about the information available from the database system monitor. It is organized by how you might use it, rather than how it is returned to you. For example, locking information is returned in a number of different ways, but all locking information is discussed in the same section.

Detailed information for each element includes:

- Level of information
  - DataJoiner
  - Database
  - Table
  - Application
  - DCS Application
  - Lock
- API element name
- API data type
- API structures in which element is returned
- Monitor group
  - Basic (always monitored)

  - Sorts
  - Locks
  - Table activity
  - Buffer pool activity
  - Unit of Work
  - SQL Statements
- Indicator of whether or not the data element is resettable
- Related database system monitor elements
- Description of the element contents
- Usage (how you can use the contents of this element, if appropriate)

API Note: Information about the APIs and the API structures is in the *DATABASE 2 Database System Monitor Guide and Reference*.

The following sections provide detailed information about the database system monitor elements:

- "General Information"
- "Data Source Database-Related Information" on page 112
- "Data Source Application-Related Information" on page 128

General Information

The following elements provide general information in a heterogeneous database environment:

Data Source Database Names

Information Level	DataJoiner
API Element Name	dbase_remote
Data Type	long
API Structure(s)	sqlm_collected
Monitor Group	Basic
Resettable	No
Related Elements	None

Description: When DataJoiner connects to a data source, it gathers statistics about that data source and its databases.

Data Source Applications

Information Level	DataJoiner
API Element Name	appl_remote
Data Type	long
API Structure(s)	sqlm_collected
Monitor Group	Basic
Resettable	No
Related Elements	None

Description: When DataJoiner connects to a data source database, it gathers statistics about the application at the data source.

Data Source Database-Related Information

The following elements list information about the total access to a data source by applications running in a DataJoiner heterogeneous database environment. They include:

- “Failed Statements” on page 123
- “Query Response Time” on page 124
- “Insert Response Time” on page 125
- “Update Response Time” on page 125
- “Delete Response Time” on page 126
- “Create Nickname Response Time” on page 126
- “Pass-Through Time” on page 127
- “Stored Procedure Time” on page 128

Data Source Name

Information Level	DataJoiner
API Element Name	name
Data Type	char [SQLM_IDENT_SZ]
API Structure(s)	sqlm_dbase_remote
Monitor Group	Basic
Resettable	No
Related Elements	“Data Source Name” on page 130

Description: This element contains the name of the data source whose remote access information by DataJoiner is being displayed. This element corresponds to the 'SERVER' column in SYSCAT.SERVERS.

Usage: Use this element to identify the data source whose access information has been collected and is being returned.

Data Source Database Name

Information Level	DataJoiner
API Element Name	dbase
Data Type	char [SQLM_IDENT_SZ]
API Structure(s)	sqlm_dbase_remote
Monitor Group	Basic
Resettable	No
Related Elements	“Data Source Database Name” on page 130

Description: This element contains the name of the database at the data source whose remote access information by DataJoiner is being displayed. This element corresponds to the 'DBNAME' column in SYSCAT.SERVERS.

Usage: Use this element to identify the database at the data source whose access information has been collected and is being returned.

Connects

Information Level	**DataJoiner**
API Element Name	**connects**
Data Type	**integer**
API Structure(s)	**sqlm_dbase_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**None**

Description: This element contains a count of the total number of times DataJoiner has connected to this data source on behalf of any application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine the total number of times DataJoiner has connected to this data source on behalf of any application. It provides information about the relative frequency of access to the data source from this DataJoiner instance.

Disconnects

Information Level	**DataJoiner**
API Element Name	**disconnects**
Data Type	**integer**
API Structure(s)	**sqlm_dbase_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**None**

Description: This element contains a count of the total number of times DataJoiner has disconnected from this data source on behalf of any application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine the total number of times DataJoiner has disconnected from this data source on behalf of any application Together with the CONNECT count, this element provides a mechanism by which you can determine the number of applications this instance of DataJoiner believes is currently connected to a data source.

Commits

Information Level	DataJoiner
API Element Name	commits
Data Type	integer
API Structure(s)	sqlm_dbase_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	"Commits" on page 130

Description: This element contains a count of the total number of times DataJoiner has issued a commit to this data source on behalf of any application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine the total number of times DataJoiner has issued a commit to this data source on behalf of any application. A small rate of change in this counter during the monitor period can indicate that applications are not doing frequent COMMITs, which can lead to problems with data concurrency.

You can also use this element to calculate the total number of units of work created by DataJoiner on this data source by computing the sum of this element, and the corresponding rollback count field.

Rollbacks

Information Level	DataJoiner
API Element Name	rollbacks
Data Type	integer
API Structure(s)	sqlm_dbase_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	"Rollbacks" on page 131

Description: This element contains a count of the total number of times DataJoiner has issued a ROLLBACK to this data source on behalf of any application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine the extent to which DataJoiner applications conflict with other applications at the data source. Try to minimize the number of rollbacks, because a higher rollback activity results in lower throughput for the data source.

You can also use this element to calculate the total number of units of work created by DataJoiner on this data source by computing the sum of this element, and the corresponding commits count field.

Queries

Information Level	**DataJoiner**
API Element Name	**queries**
Data Type	**integer**
API Structure(s)	**sqlm_dbase_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**"Queries" on page 132**

Description: This element contains a count of the total number of times DataJoiner has issued a SELECT statement to this data source on behalf of any application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine the level of database activity directed against this data source by DataJoiner.

You can also use this element to determine the percentage of read activity against this data source, with the following formula:

```
read_activity =
   SELECT statements /
   (SELECT statements + INSERT statements + UPDATE statements +
   DELETE statements)
```

Inserts

Information Level	DataJoiner
API Element Name	inserts
Data Type	integer
API Structure(s)	sqlm_dbase_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	“Inserts” on page 132

Description: This element contains a count of the total number of times DataJoiner has issued an INSERT statement to this data source on behalf of any application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine the level of database activity directed against this data source by DataJoiner.

You can also use this element to determine the percentage of write activity against this data source by DataJoiner, with the following formula:

```
write_activity =
   (INSERT statements + UPDATE statements + DELETE statements ) /
   (SELECT statements + INSERT statements + UPDATE statements +
   DELETE statements)
```

Updates

Information Level	DataJoiner
API Element Name	updates
Data Type	integer
API Structure(s)	sqlm_dbase_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	“Updates” on page 133

Description: This element contains a count of the total number of times DataJoiner has issued an UPDATE statement to this data source on behalf of any application since the later of:

- The start of the DataJoiner instance, or

- The last reset of the database monitor counters.

Usage: Use this element to determine the level of database activity directed against this data source by DataJoiner.

You can also use this element to determine the percentage of write activity against this data source by DataJoiner, with the following formula:

```
write_activity =
  (INSERT statements + UPDATE statements + DELETE statements ) /
  (SELECT statements + INSERT statements + UPDATE statements +
  DELETE statements)
```

Deletes

Information Level	DataJoiner
API Element Name	deletes
Data Type	integer
API Structure(s)	sqlm_dbase_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	“Deletes” on page 134

Description: This element contains a count of the total number of times DataJoiner has issued a DELETE statement to this data source on behalf of any application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine the level of database activity directed against this data source by DataJoiner.

You can also use this element to determine the percentage of write activity against this data source by DataJoiner, with the following formula:

```
write_activity =
  (INSERT statements + UPDATE statements + DELETE statements ) /
  (SELECT statements + INSERT statements + UPDATE statements +
  DELETE statements)
```

Create Nicknames

Information Level	DataJoiner

API Element Name	**createNickname**
Data Type	**integer**
API Structure(s)	**sqlm_dbase_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**"Create Nicknames" on page 134**

Description: This element contains a count of the total number of times DataJoiner has created a nickname over an object residing on this data source on behalf of any application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine the amount of CREATE NICKNAME activity against this data source by this DataJoiner instance. CREATE NICKNAME processing results in multiple queries running against the data source catalogs; therefore, if the value of this element is high, you should determine the cause and perhaps restrict this activity.

Pass-Through

Information Level	**DataJoiner**
API Element Name	**passThrus**
Data Type	**integer**
API Structure(s)	**sqlm_dbase_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**"Pass-Through" on page 135**

Description: This element contains a count of the total number of SQL statements that DataJoiner has passed through directly to this data source on behalf of any application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine what percentage of your SQL statements can be handled natively by DataJoiner, and what percentage requires pass-through mode. If this value is high, you should determine the cause and investigate ways to better utilize native support.

Stored Procedures

Information Level	DataJoiner
API Element Name	storedProcs
Data Type	integer
API Structure(s)	sqlm_dbase_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	“Stored Procedures” on page 135

Description: This element contains a count of the total number of stored procedures that DataJoiner has called at this data source on behalf of any application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine how many stored procedure calls were made locally at the DataJoiner database.

Rows Returned

Information Level	DataJoiner
API Element Name	rowsReturned
Data Type	integer
API Structure(s)	sqlm_dbase_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	“Rows Returned” on page 136

Description: This element contains the number of rows sent from the data source to DataJoiner as a result of SQL SELECT operations since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: This element has several uses. You can use it to compute the average number of rows sent to DataJoiner from the data source, per SELECT statement, with the following formula:

```
rows per SELECT = rows returned / SELECT statements
```

You can also compute the average time to return a row to DataJoiner from the data source:

```
average time = rows returned / aggregate query response time
```

You can use these results to modify CPU speed or communication speed parameters in SYSCAT.SERVERS. Modifying these parameters can impact whether the optimizer does or does not send requests to the data source.

Rows Updated

Information Level	DataJoiner
API Element Name	rowsUpdated
Data Type	integer
API Structure(s)	sqlm_dbase_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	“Rows Updated” on page 136

Description: This element contains the number of rows that were updated at the data source as a result of SQL UPDATE statements sent from DataJoiner to the data source on behalf of any application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: This element has several uses. You can use it to determine how many rows are affected, on average, by each UPDATE statement issued to this data source, with the following formula:

```
rows_per_update = rows updated / update requests
```

You can also determine how long, on average, it took to update each row:

```
time_per_row = rows updated / aggregate update response time
```

Rows Deleted

Information Level	DataJoiner
API Element Name	rowsDeleted
Data Type	integer
API Structure(s)	sqlm_dbase_remote
Monitor Group	Basic

Resettable	**Yes**
Related Elements	**“Rows Deleted” on page 137**

Description: This element contains the number of rows that were deleted at the data source as a result of SQL DELETE statements sent from DataJoiner to the data source on behalf of any application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: This element has several uses. You can use it to determine how many rows are affected, on average, by each DELETE statement issued to this data source, with the following formula:

```
rows_per_delete = rows deleted / delete requests
```

You can also determine how long, on average, it took to delete each row:

```
time_per_row = rows deleted / aggregate delete response time
```

Rows Inserted

Information Level	**DataJoiner**
API Element Name	**rowsInserted**
Data Type	**integer**
API Structure(s)	**sqlm_dbase_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**“Rows Inserted” on page 138**

Description: This element contains the number of rows that were inserted at the data source as a result of SQL INSERT statements sent from DataJoiner to the data source on behalf of any application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: This element has several uses. You can use it to determine how many rows are affected, on average, by each INSERT statement issued to this data source, with the following formula:

```
rows_per_insert = rows inserted / insert requests
```

You can also determine how long, on average, it took to insert each row:

```
time_per_row = rows inserted / aggregate insert response time
```

Note that multiple rows can be inserted, per INSERT statement, because DataJoiner can push INSERT FROM SUBSELECT to the data source, when appropriate.

Rows Returned by Stored Procedures

Information Level	DataJoiner
API Element Name	sprowsReturned
Data Type	integer
API Structure(s)	sqlm_dbase_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	"Rows Returned by Stored Procedures" on page 138

Description: This element contains the number of rows sent from the data source to DataJoiner as a result of stored procedure operations for this application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: This element has several uses. You can use it to compute the average number of rows sent to DataJoiner from the data source, per stored procedure, with the following formula:

```
rows per stored procedure = rows returned / number of stored procedures invoked
```

You can also compute the average time to return a row to DataJoiner from the data source for this application:

```
average time = rows returned / aggregate stored procedure response time
```

Failed Statements

Information Level	DataJoiner
API Element Name	failedStatements
Data Type	integer
API Structure(s)	sqlm_dbase_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	"Failed Statements" on page 139

Description: This element contains the number of SQL statements sent to the data source that the data source could not process successfully since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to help you identify the cause of poor performance. If requests are sent to the data source, but are not processed, performance is impacted due to lower throughput.

Query Response Time

Information Level	**DataJoiner**
API Element Name	**queryTime**
Data Type	**long**
API Structure(s)	**sqlm_dbase_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**"Query Response Time" on page 139**

Description: This element contains the aggregate amount of time, in milliseconds, that it has taken this data source to respond to queries from all applications running on this DataJoiner instance since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

The response time is measured as the difference in time between the time DataJoiner requests a row from the data source, and the time the row is available for DataJoiner to use.

Note: Due to query blocking, not all attempts by DataJoiner to retrieve a row result in communication processing; the request to get the next row can potentially be satisfied from a block of returned rows. As a result, the aggregate query response time does not always indicate processing at the data source, but it usually indicates processing at either the data source or client.

Usage: Use this element to determine how much actual time is spent waiting for data from this data source. This can be useful in capacity planning and tuning the CPU speed and communication rates in SYSCAT.SERVERS. Modifying these parameters can impact whether the optimizer does or does not send requests to the data source.

Insert Response Time

Information Level	**DataJoiner**
API Element Name	**insertTime**
Data Type	**long**
API Structure(s)	**sqlm_dbase_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**"Insert Response Time" on page 140**

Description: This element contains the aggregate amount of time, in milliseconds, that it has taken this data source to respond to INSERTs from all applications running on this DataJoiner instance since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

The response time is measured as the difference in time between the time DataJoiner submits an INSERT statement to the data source, and the time the data source responds to DataJoiner, indicating that the INSERT has been processed.

Usage: Use this element to determine the actual amount of time that transpires waiting for INSERTs to this data source to be processed. This information can be useful for capacity planning and tuning.

Update Response Time

Information Level	**DataJoiner**
API Element Name	**updateTime**
Data Type	**long**
API Structure(s)	**sqlm_dbase_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**"Update Response Time" on page 141**

Description: This element contains the aggregate amount of time, in milliseconds, that it has taken this data source to respond to UPDATEs from all applications running on this DataJoiner instance since the later of:

- The start of the DataJoiner instance, or

- The last reset of the database monitor counters.

The response time is measured as the difference in time between the time DataJoiner submits an UPDATE statement to the data source, and the time the data source responds to DataJoiner, indicating the UPDATE has been processed.

Usage: Use this element to determine how much actual time transpires while waiting for UPDATEs to this data source to be processed. This information can be useful for capacity planning and tuning.

Delete Response Time

Information Level	**DataJoiner**
API Element Name	**deleteTime**
Data Type	**long**
API Structure(s)	**sqlm_dbase_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**"Delete Response Time" on page 141**

Description: This element contains the aggregate amount of time, in milliseconds, that it has taken this data source to respond to DELETEs from all applications running on this DataJoiner instance since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

The response time is measured as the difference in time between the time DataJoiner submits a DELETE statement to the data source, and the time the data source responds to DataJoiner, indicating the DELETE has been processed.

Usage: Use this element to determine how much actual time transpires while waiting for DELETEs to this data source to be processed. This information can be useful for capacity planning and tuning.

Create Nickname Response Time

Information Level	**DataJoiner**
API Element Name	**createNicknameTime**
Data Type	**long**
API Structure(s)	**sqlm_dbase_remote**

Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**"Create Nickname Response Time" on page 142**

Description: This element contains the aggregate amount of time, in milliseconds, that it has taken this data source to process CREATE NICKNAME statements from all applications running on this DataJoiner instance since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

The response time is measured as the difference between the time DataJoiner started retrieving information from the data source to process the CREATE NICKNAME statement, and the time it took to retrieve all the required data from the data source.

Usage: Use this element to determine how much actual time was used to create nicknames for this data source.

Pass-Through Time

Information Level	**DataJoiner**
API Element Name	**passthruTime**
Data Type	**long**
API Structure(s)	**sqlm_dbase_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**"Pass-Through Time" on page 143**

Description: This element contains the aggregate amount of time, in milliseconds, that it has taken this data source to respond to PASSTHRU statements from all applications running on this DataJoiner instance since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

The response time is measured as the difference between the time DataJoiner submits a PASSTHRU statement to the data source, and the time it takes the data source to respond, indicating that the statement has been processed.

Usage: Use this element to determine how much actual time is spent at this data source processing statements in pass-through mode.

Stored Procedure Time

Information Level	**DataJoiner**
API Element Name	**storedProcTime**
Data Type	**long**
API Structure(s)	**sqlm_dbase_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**"Stored Procedure Time" on page 143**

Description: This element contains the aggregate amount of time, in milliseconds, that it has taken this data source to respond to stored procedure statements from all applications running on this DataJoiner instance since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

The response time is measured as the difference between the time DataJoiner submits a stored procedure to the data source, and the time it takes the data source to respond, indicating that the stored procedure has been processed.

Usage: Use this element to determine how much actual time is spent at this data source processing stored procedures.

Data Source Application-Related Information

The following elements contain information about access to a data source by a given application running in a DataJoiner instance. They include:

- "Application Identification" on page 129
- "Data Source Name" on page 130
- "Data Source Database Name" on page 130
- "Commits" on page 130
- "Rollbacks" on page 131
- "Queries" on page 132
- "Inserts" on page 132
- "Updates" on page 133
- "Deletes" on page 134
- "Create Nicknames" on page 134
- "Pass-Through" on page 135

Agent Note: Several new elements (Agent ID through Input Database Alias) were added for DB2 for CS application snapshots that are not documented in this manual. See the *DATABASE 2 Database System Monitor Guide and Reference* for details.

Application Identification

Information Level	**DataJoiner**
API Element Name	**sqlm_appl_id_info**
Data Type	**sqlm_appl_id_info**
API Structure(s)	**sqlm_appl_remote**
Monitor Group	**Basic**
Resettable	**No**
Related Elements	**None**

Description: This element contains the name of the application currently being monitored.

Usage: Use this element to identify the application whose access information has been collected and is being returned.

Data Source Name

Information Level	DataJoiner
API Element Name	name
Data Type	char [SQLM_IDENT_SZ]
API Structure(s)	sqlm_appl_remote
Monitor Group	Basic
Resettable	No
Related Elements	“Data Source Name” on page 113

Description: This element contains the name of the data source whose remote access information by DataJoiner is being displayed. This element corresponds to the 'SERVER' column in SYSCAT.SERVERS.

Usage: Use this element to identify the data source whose access information has been collected and is being returned.

Data Source Database Name

Information Level	DataJoiner
API Element Name	dbase
Data Type	char [SQLM_IDENT_SZ]
API Structure(s)	sqlm_appl_remote
Monitor Group	Basic
Resettable	No
Related Elements	“Data Source Database Name” on page 113

Description: This element contains the name of the database at the data source whose remote access information by DataJoiner is being displayed. This element corresponds to the 'DBNAME' column in SYSCAT.SERVERS.

Usage: Use this element to identify the database at the data source whose access information has been collected and is being returned.

Commits

Information Level	DataJoiner
API Element Name	commits

Data Type	**integer**
API Structure(s)	**sqlm_appl_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**“Commits” on page 115**

Description: This element contains a count of the total number of times DataJoiner has issued a COMMIT to this data source on behalf of this application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine the total number of times DataJoiner has issued a commit to this data source on behalf of the application. A small rate of change in this counter during the monitor period can indicate that the application is not doing frequent COMMITs, which can lead to problems with data concurrency.

You can also use this element to calculate the total number of units of work created by DataJoiner on this data source for this application by computing the sum of this element, and the corresponding rollback count field.

Rollbacks

Information Level	**DataJoiner**
API Element Name	**rollbacks**
Data Type	**integer**
API Structure(s)	**sqlm_appl_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**“Rollbacks” on page 115**

Description: This element contains a count of the total number of times DataJoiner has issued a ROLLBACK to this data source on behalf of this application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine the extent to which this application conflicts with applications at the data source. Try to minimize the number of rollbacks, because a higher rollback activity results in lower throughput for the data source.

You can also use this element to calculate the total number of units of work created by DataJoiner on this data source for this application by computing the sum of this element, and the corresponding commits count field.

Queries

Information Level	DataJoiner
API Element Name	queries
Data Type	integer
API Structure(s)	sqlm_appl_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	“Queries” on page 116

Description: This element contains a count of the total number of times DataJoiner has issued a SELECT statement to this data source on behalf of this application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine the level of database activity directed against this data source by this application.

You can also use this element to determine the percentage of read activity against this data source by this application, with the following formula:

```
read_activity =
  SELECT statements /
  (SELECT statements + INSERT statements + UPDATE statements +
  DELETE statements)
```

Inserts

Information Level	DataJoiner
API Element Name	inserts
Data Type	integer
API Structure(s)	sqlm_appl_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	“Inserts” on page 117

Description: This element contains a count of the total number of times DataJoiner has issued an INSERT statement to this data source on behalf of this application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine the level of database activity directed against this data source by this application.

You can also use this element to determine the percentage of write activity against this data source by DataJoiner, with the following formula:

```
write_activity =
  (INSERT statements + UPDATE statements + DELETE statements ) /
  (SELECT statements + INSERT statements + UPDATE statements +
  DELETE statements)
```

Updates

Information Level	DataJoiner
API Element Name	updates
Data Type	integer
API Structure(s)	sqlm_appl_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	"Updates" on page 117

Description: This element contains a count of the total number of times DataJoiner has issued an UPDATE statement to this data source on behalf of this application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine the level of database activity directed against this data source by this application.

You can also use this element to determine the percentage of write activity against this data source by DataJoiner for this application, with the following formula:

```
write_activity =
  (INSERT statements + UPDATE statements + DELETE statements ) /
  (SELECT statements + INSERT statements + UPDATE statements +
  DELETE statements)
```

Deletes

Information Level	DataJoiner
API Element Name	deletes
Data Type	integer
API Structure(s)	sqlm_appl_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	“Deletes” on page 118

Description: This element contains a count of the total number of times DataJoiner has issued a DELETE statement to this data source on behalf of this application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine the level of database activity directed against this data source by this application.

You can also use this element to determine the percentage of write activity against this data source by DataJoiner for this application, with the following formula:

```
write_activity =
  (INSERT statements + UPDATE statements + DELETE statements ) /
  (SELECT statements + INSERT statements + UPDATE statements +
  DELETE statements)
```

Create Nicknames

Information Level	DataJoiner
API Element Name	createNickname
Data Type	integer
API Structure(s)	sqlm_appl_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	“Create Nicknames” on page 118

Description: This element contains a count of the total number of times DataJoiner has created a nickname over an object residing on this data source on behalf of this application since the later of:

- The start of the DataJoiner instance, or

- The last reset of the database monitor counters.

Usage: Use this element to determine the amount of CREATE NICKNAME activity against this data source by this application. CREATE NICKNAME processing results in multiple queries running against the data source catalogs; therefore, if the value of this element is high, you should determine the cause and perhaps restrict this activity.

Pass-Through

Information Level	**DataJoiner**
API Element Name	**passThrus**
Data Type	**integer**
API Structure(s)	**sqlm_appl_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**“Pass-Through” on page 119**

Description: This element contains a count of the total number of SQL statements that DataJoiner has passed through directly to this data source on behalf of this application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine what percentage of this application’s SQL statements can be handled natively by DataJoiner, and what percentage requires pass-through mode. If this value is high, you should determine the cause and investigate ways to better utilize native support.

Stored Procedures

Information Level	**DataJoiner**
API Element Name	**storedProcs**
Data Type	**integer**
API Structure(s)	**sqlm_appl_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**“Stored Procedures” on page 120**

Description: This element contains a count of the total number of stored procedures that DataJoiner has called at this data source on behalf of this application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to determine how many stored procedure calls were made by this application against this DataJoiner database.

Rows Returned

Information Level	**DataJoiner**
API Element Name	**rowsReturned**
Data Type	**integer**
API Structure(s)	**sqlm_appl_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**"Rows Returned" on page 120**

Description: This element contains the number of rows sent from the data source to DataJoiner as a result of SQL SELECT operations for this application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: This element has several uses. You can use it to compute the average number of rows sent to DataJoiner from the data source for this application, per SELECT statement, with the following formula:

```
rows per SELECT = rows returned / SELECT statements
```

You can also compute the average time to return a row to DataJoiner from the data source for this application:

```
average time = rows returned / aggregate query response time
```

You can use these results to modify CPU speed or communication speed parameters in SYSCAT.SERVERS. Modifying these parameters can impact whether the optimizer does or does not send requests to the data source.

Rows Updated

Information Level	**DataJoiner**
API Element Name	**rowsUpdated**
Data Type	**integer**
API Structure(s)	**sqlm_appl_remote**

Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**“Rows Updated” on page 121**

Description: This element contains the number of rows that were updated at the data source as a result of SQL UPDATE statements sent from DataJoiner to the data source on behalf of this application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: This element has several uses. You can use it to determine how many rows are affected, on average, by each UPDATE statement issued to this data source for this application, with the following formula:

```
rows_per_update = rows updated / update requests
```

You can also determine how long, on average, it took to update each row:

```
time_per_row = rows updated / aggregate update response time
```

Rows Deleted

Information Level	**DataJoiner**
API Element Name	**rowsDeleted**
Data Type	**integer**
API Structure(s)	**sqlm_appl_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**“Rows Deleted” on page 121**

Description: This element contains the number of rows that were deleted at the data source as a result of SQL DELETE statements sent from DataJoiner to the data source on behalf of this application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: This element has several uses. You can use it to determine how many rows are affected, on average, by each DELETE statement issued to this data source by this application, with the following formula:

```
rows_per_delete = rows deleted / delete requests
```

You can also determine how long, on average, it took to delete each row:

```
time_per_row = rows deleted / aggregate delete response time
```

Rows Returned by Stored Procedures

Information Level	DataJoiner
API Element Name	sprowsReturned
Data Type	integer
API Structure(s)	sqlm_appl_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	"Rows Returned by Stored Procedures" on page 123

Description: This element contains the number of rows sent from a data source to DataJoiner as a result of stored procedure operations for this application since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: This element has several uses. You can use it to compute the average number of rows sent to DataJoiner from the data source for this application, per stored procedure, with the following formula:

```
rows per stored procedure = rows returned / number of stored procedures invoked
```

You can also compute the average time to return a row to DataJoiner from the data source for this application:

```
average time = rows returned / aggregate stored procedure response time
```

Rows Inserted

Information Level	DataJoiner
API Element Name	rowsInserted
Data Type	integer
API Structure(s)	sqlm_appl_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	"Rows Inserted" on page 122

Description: This element contains the number of rows that were inserted at the data source as a result of SQL INSERT statements sent from DataJoiner to the data source on behalf of this application since the later of:

- The start of the DataJoiner instance, or

- The last reset of the database monitor counters.

Usage: This element has several uses. You can use it to determine how many rows are affected, on average, by each INSERT statement issued to this data source for this application, with the following formula:

```
rows_per_insert = rows inserted / insert requests
```

You can also determine how long, on average, it took to insert each row:

```
time_per_row = rows inserted / aggregate insert response time
```

Note that multiple rows can be inserted, per INSERT statement, because DataJoiner can push INSERT FROM SUBSELECT to the data source, when appropriate.

Failed Statements

Information Level	**DataJoiner**
API Element Name	**failedStatements**
Data Type	**integer**
API Structure(s)	**sqlm_appl_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**“Failed Statements” on page 123**

Description: This element contains the number of SQL statements sent to the data source on behalf of this application that the data source could not process successfully since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

Usage: Use this element to help you identify the cause of poor performance. If requests are sent to the data source, but are not processed, performance is impacted due to lower throughput.

Query Response Time

Information Level	**DataJoiner**
API Element Name	**queryTime**
Data Type	**long**
API Structure(s)	**sqlm_appl_remote**
Monitor Group	**Basic**

Resettable	**Yes**
Related Elements	**"Query Response Time" on page 124**

Description: This element contains the aggregate amount of time, in milliseconds, that it has taken this data source to respond to queries from this application running on this DataJoiner instance since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

The response time is measured as the difference in time between the time DataJoiner requests a row from the data source, and the time the row is available for DataJoiner to use.

Note: Due to query blocking, not all attempts by DataJoiner to retrieve a row result in communication processing; the request to get the next row can potentially be satisfied from a block of returned rows. As a result, the aggregate query response time does not always indicate processing at the data source, but it usually indicates processing at either the data source or client.

Usage: Use this element to determine how much actual time is spent waiting for data from this data source. This can be useful in capacity planning and tuning the CPU speed and communication rates in SYSCAT.SERVERS. Modifying these parameters can impact whether the optimizer does or does not send requests to the data source.

Insert Response Time

Information Level	**DataJoiner**
API Element Name	**insertTime**
Data Type	**long**
API Structure(s)	**sqlm_appl_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**"Insert Response Time" on page 125**

Description: This element contains the aggregate amount of time, in milliseconds, that it has taken this data source to respond to INSERTs from this application running on this DataJoiner instance since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

The response time is measured as the difference in time between the time DataJoiner submits an INSERT statement to the data source, and the time the data source responds to DataJoiner, indicating that the INSERT has been processed.

Usage: Use this element to determine the actual amount of time that transpires waiting for INSERTs to this data source to be processed for this application. This information can be useful for capacity planning and tuning.

Update Response Time

Information Level	DataJoiner
API Element Name	updateTime
Data Type	long
API Structure(s)	sqlm_appl_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	"Update Response Time" on page 125

Description: This element contains the aggregate amount of time, in milliseconds, that it has taken this data source to respond to UPDATEs from this application running on this DataJoiner instance since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

The response time is measured as the difference in time between the time DataJoiner submits an UPDATE statement to the data source, and the time the data source responds to DataJoiner, indicating the UPDATE has been processed.

Usage: Use this element to determine how much actual time transpires while waiting for UPDATEs to this data source to be processed for this application. This information can be useful for capacity planning and tuning.

Delete Response Time

Information Level	DataJoiner
API Element Name	deleteTime
Data Type	long
API Structure(s)	sqlm_appl_remote
Monitor Group	Basic

Resettable	**Yes**
Related Elements	**"Delete Response Time" on page 126**

Description: This element contains the aggregate amount of time, in milliseconds, that it has taken this data source to respond to DELETEs from this application running on this DataJoiner instance since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

The response time is measured as the difference in time between the time DataJoiner submits a DELETE statement to the data source, and the time the data source responds to DataJoiner, indicating the DELETE has been processed.

Usage: Use this element to determine how much actual time transpires while waiting for DELETEs to this data source to be processed for this application. This information can be useful for capacity planning and tuning.

Create Nickname Response Time

Information Level	**DataJoiner**
API Element Name	**createNicknameTime**
Data Type	**long**
API Structure(s)	**sqlm_appl_remote**
Monitor Group	**Basic**
Resettable	**Yes**
Related Elements	**"Create Nickname Response Time" on page 126**

Description: This element contains the aggregate amount of time, in milliseconds, that it has taken this data source to process CREATE NICKNAME statements from this application running on this DataJoiner instance since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

The response time is measured as the difference between the time DataJoiner started retrieving information from the data source to process the CREATE NICKNAME statement, and the time it took to retrieve all the required data from the data source.

Usage: Use this element to determine how much actual time was used to create nicknames for this data source.

Pass-Through Time

Information Level	DataJoiner
API Element Name	passthruTime
Data Type	long
API Structure(s)	sqlm_appl_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	“Pass-Through Time” on page 127

Description: This element contains the aggregate amount of time, in milliseconds, that it has taken this data source to respond to PASSTHRU statements from this application running on this DataJoiner instance since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

The response time is measured as the difference between the time DataJoiner submits a PASSTHRU statement to the data source, and the time it takes the data source to respond, indicating that the statement has been processed.

Usage: Use this element to determine how much actual time is spent at this data source processing statements in pass-through mode for this application.

Stored Procedure Time

Information Level	DataJoiner
API Element Name	storedProcTime
Data Type	long
API Structure(s)	sqlm_appl_remote
Monitor Group	Basic
Resettable	Yes
Related Elements	“Stored Procedure Time” on page 128

Description: This element contains the aggregate amount of time, in milliseconds, that it has taken this data source to respond to stored procedure statements from this application running on this DataJoiner instance since the later of:

- The start of the DataJoiner instance, or
- The last reset of the database monitor counters.

The response time is measured as the difference between the time DataJoiner submits a stored procedure to the data source, and the time it takes the data source to respond, indicating that the stored procedure has been processed.

Usage: Use this element to determine how much actual time is spent at this data source processing stored procedures.

Appendix A. SQL Explain Utilities

DataJoiner extends DB2 for CS SQL explain utility functions to a heterogeneous, distributed database environment. DataJoiner—specific requirements and output are described in this appendix. For basic information on the utilities, see the *DATABASE 2 Administration Guide*.

DB2 Explain (db2expln)

Using **db2expln** in a DataJoiner environment does not require special syntax or additional keywords. The location, path, and syntax for **db2expln** is documented in the *DATABASE 2 Administration Guide*.

If you are using **db2expln** in a DataJoiner environment with data sources, Table 8 describes the data sources that it supports (along with prerequisites and any usage restrictions).

Table 8. Prerequisites for Supported Data Sources

Data Source	Prerequisite	Usage Restrictions
DB2/PE, DB2 for CS, DB2 for AIX, DB2 UDB	The remote data source must have AUTHENTICATION set to CLIENT. There must not be a local database named 'DJ_TDB'.	Do not use 'DJ_TDB' for any other database name.
DB2 for OS/390	The plan table PLAN_TABLE must exist prior to running the Explain tool.	Reserve `queryno=999999999` for DataJoiner.
Oracle	A plan table named DATA_JOINER_PLAN must exist prior to running the Explain tool.	Reserve `statement_id='999999999'` for DataJoiner.
Sybase	None	Only English is supported
MS SQL Server (accessed via dblib)	None	Only English is supported

Note: DB2 for CS V1 and DB2 for AIX V1 are not supported.

db2expln Output

db2expln output from DataJoiner is similar to DB2 for CS **db2expln** output. For each query to be issued to a data source, its corresponding SQL statement is presented. If the DBMS is DB2 for OS/390, DB2 for CS V2/V5, DB2/PE, DB2 for AIX, Microsoft SQL Server (via dblib), Oracle, or Sybase, its remote access strategy is retrieved, if available, and presented using the local format. Because each data source provides different levels of output, the amount of information might not be consistent across all data sources. Note that all remote access strategies are retrieved at the time **db2expln** is run; therefore, data sources must be online. Each remote access strategy is surrounded by a pair of lines that indicate the beginning and the end of the remote plan.

Input host variables can cause the remote plan to be dynamic, because the remote access strategy can change based on the value of the input host variable. To enable explain processing at the data source, the host variables are replaced with default values. As a result, the remote plan for such queries might not be accurate.

See the *DATABASE 2 Administration Guide* for a general description of explain output.

The rest of this section contains three **db2expln** examples applicable to a DataJoiner environment.

Example 1

Figure 12 is an example of a query being completely evaluated on the remote data source.

```
-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:

  select distinct c1
  from oracle_nickname1


Access Table : Remote Access
|  Server: ORACLE73
|  Remote SQL Statement:
|  |
|  |  SELECT DISTINCT A0."C1"
|  |  FROM "J15USER1"."TABLE1" A0
|  |
|  ==  Remote Plan from Server ORACLE73 ==
|  |
|  |  Access Table Name = TABLE1
|  |  |  Relation Scan
|  |  Create/Insert Into Sorted Temp Table for Duplicate Reduction
|  |
|  ==  End of Remote Plan from Server ORACLE73 ==
|
End of Section
```

Figure 12. Query 1 Example

Example 2

Figure 13 on page 147 is an example of a query that cannot be completely evaluated on the remote data source (because ORACLE73 is an Oracle DBMS, which uses different VARCHAR comparison semantics).

```
-------------------- SECTION --------------------------------------
Section = 1
SQL Statement:

  select distinct c1
  from oracle_nickname1
  where c3 = 'San Jose'

Access Table : Remote Access
|  Server: ORACLE73
|  Remote SQL Statement:
|  |
|  |  SELECT A0."C3", A0."C1"
|  |  FROM "J15USER1"."TABLE1" A0
|  |
|  ==  Remote Plan from Server ORACLE73 ==
|  |
|  |  Access Table Name = TABLE1
|  |  |  Relation Scan
|  |
|  ==  End of Remote Plan from Server ORACLE73 ==
|
Residual Predicate Application
|  #Predicates = 1
Create/Insert Into Sorted Temp Table  ID = t1
|  #Columns = 1
|  #Sort Key Columns = 1
|  Sortheap Allocation Parameters:
|  |  #Rows     = 40
|  |  Row Width = 12
|  Piped
|  Duplicate Elimination
Access Temp Table  ID = t1
|  #Columns = 1
|  Relation Scan
|  |  Prefetch: Eligible
End of Section
```

Figure 13. Query 2 Example

The four major steps to this plan are:

1. The nickname object ORACLE_NICKNAME1 is retrieved via a remote query sent to ORACLE73. Its remote plan from ORACLE73 shows that a relational scan is used to retrieve the entire object.
2. For each row retrieved from the server ORACLE73, the predicate `c3 = 'San Jose'` is evaluated locally on DataJoiner. This process is indicated by the `Residual Predicate Application` section.
3. If a row satisfies the predicate condition, it is then inserted into a sorted temporary table with duplicate elimination.

db2expln Note: Even though the create statement of the sorted temporary table indicates that the result will be piped (that is, stay in memory), the access plan treats this case as if the result was written to a real table and provides a scan of the table to send the result to the user. This approach allows the access plan to be independent of whether or not the results are kept in memory at execution time. The temporary table accessed at execution time will either reside in memory or on disk.

Example 3

Figure 14 is an example of a merge join of two nicknames.

```
-------------------- SECTION --------------------------------------
Section = 1


SQL Statement:

  select *
  from oracle_nickname1 x, sybase_nickname1 y
  where x.c1 = y.c1


Access Table : Remote Access
|  Server: ORACLE73
|  Remote SQL Statement:
|  |
|  |  SELECT A0."C1", A0."C2", A0."C3"
|  |  FROM "J15USER1"."TABLE1" A0
|  |
|  ==  Remote Plan from Server ORACLE73 ==
|  |
|  |  Access Table Name = TABLE1
|  |  |  Relation Scan
|  |
|  ==  End of Remote Plan from Server ORACLE73 ==
|
```

Figure 14. Query 3 Example (Part 1 of 2)

```
Create/Insert Into Sorted Temp Table  ID = t1
|  #Columns = 3
|  #Sort Key Columns = 1
|  Sortheap Allocation Parameters:
|  |  #Rows     = 1000
|  |  Row Width = 23
|  Piped
Access Temp Table  ID = t1
|  #Columns = 3
|  Relation Scan
|  |  Prefetch: Eligible
Merge Join
|  Data Stream 1: Evaluate at Open
|  |  Not Piped
|  |  Access Table : Remote Access
|  |  |  Server: SYBASE11
|  |  |  Remote SQL Statement:
|  |  |  |
|  |  |  |  SELECT A0.c1, A0.c2, A0.c3
|  |  |  |  FROM j15user1.table1 A0
|  |  |  |
|  |  |  ==  Remote Plan from Server SYBASE11 ==
|  |  |  |
|  |  |  |
|  |  |  |  Access Table Name = j15user1.table1
|  |  |  |  |  Relation Scan
|  |  |  |  |  Scan Direction  = Forward
|  |  |  |
|  |  |  ==  End of Remote Plan from Server SYBASE11 ==
|  |  |
|  |  Create/Insert Into Sorted Temp Table  ID = t2
|  |  |  #Columns = 3
|  |  |  #Sort Key Columns = 1
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 1000
|  |  |  |  Row Width = 19
|  |  |  Not Piped
|  End of Data Stream 1
|  Access Temp Table  ID = t2
|  |  #Columns = 3
|  |  Relation Scan
|  |  |  Prefetch: Eligible
End of Section
```

Figure 14. Query 3 Example (Part 2 of 2)

The plan steps are:

1. The nickname object ORACLE_NICKNAME1 is retrieved and sorted locally at DataJoiner
2. The nickname object SYBASE_NICKNAME1 is retrieved and sorted locally at DataJoiner

3. The sorted output of these two nicknames is joined via the merge join methodology

Dynamic Explain (dynexpln) Sample Shell Script on AIX

A sample shell script (`dynexpln`) that provides a means for analyzing dynamic SQL is provided in the `INSTHOME/sqllib/misc` directory of the instance owner. Information about this sample is documented in the *DATABASE 2 Administration Guide*.

The sample has been slightly modified to allow for remote DB2 for AIX explain processing. Do not alter or move this shell script because it is used by DataJoiner executables.

Visual Explain (db2vexp)

Visual explain can be started from the Database Director or from the command line. Use the **db2vexp** command to start visual explain from the command line.

The location, path, and syntax for **db2vexp** is documented in the *DATABASE 2 Administration Guide*.

To see information on starting visual explain, issue the command:

```
db2vexp -h
```

To start visual explain (and load the sample output shown in “db2vexp Output”), use the command:

```
   db2vexp
-db sb -sql"SELECT *
FROM oracle_nickname1 x, sybase_nickname1 y WHERE x.c1 = y.c1"
```

db2vexp Output

db2vexp output from DataJoiner is similar to DB2 for CS **db2vexp** output. Query analysis results are displayed in a graphical user interface.

db2vexp displays local plans. It does not display remote access strategy information. For example, you can display a query intended for use against a nickname for an Oracle table to discover the global plan; however, **db2vexp** will not display, as an example, the join strategy that will be used at the Oracle data source.

See the *DATABASE 2 Administration Guide* for a general description of **db2vexp** output.

The rest of this section contains a **db2vexp** example applicable to a DataJoiner environment. Figure 15 on page 151 shows part of the information available when

analyzing a merge join of two nicknames (the same query shown previously in Figure 14 on page 148).

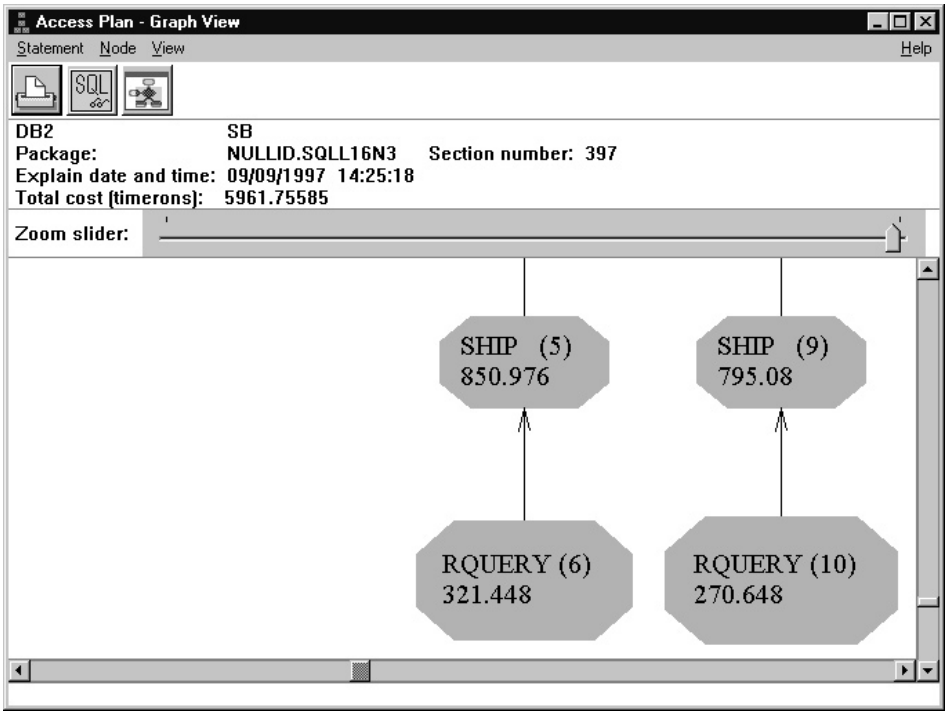


Figure 15. Query 3 Example Shown with Visual Explain: Bottom Elements

You can't see all of the query elements in this zoomed view, but the plan steps are:

1. The nickname object ORACLE_NICKNAME1 is retrieved and sorted locally at DataJoiner
2. The nickname object SYBASE_NICKNAME1 is retrieved and sorted locally at DataJoiner
3. The sorted output of these two nicknames is joined via the merge join methodology

The display can be scrolled upward. Figure 16 on page 152 shows the top part of the displayed information (join).

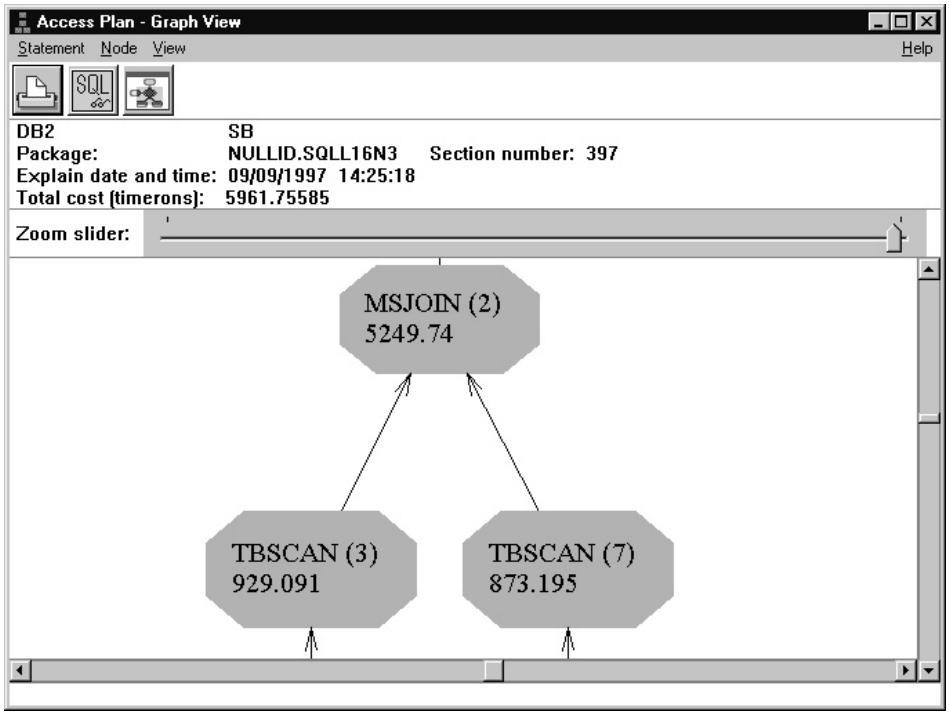


Figure 16. Query 3 Example Shown with Visual Explain: Top Element

Most of the elements displayed are common to elements displayed by DB2 for CS. Some elements are unique to DataJoiner:

SHIP Shows the cost, in timerons, of executing the remote query and fetching the results

RINSERT Shows the cost, in timerons, of executing the remote insert operation

RUPDATE Shows the cost, in timerons, of executing the remote update operation

RDELETE Shows the cost, in timerons, of executing the remote delete operation

RQUERY Shows the cost, in timerons, of executing the remote query

Additional operator details are available. Double-click on the top operator to display the Operator Details window (see Figure 17 on page 153).

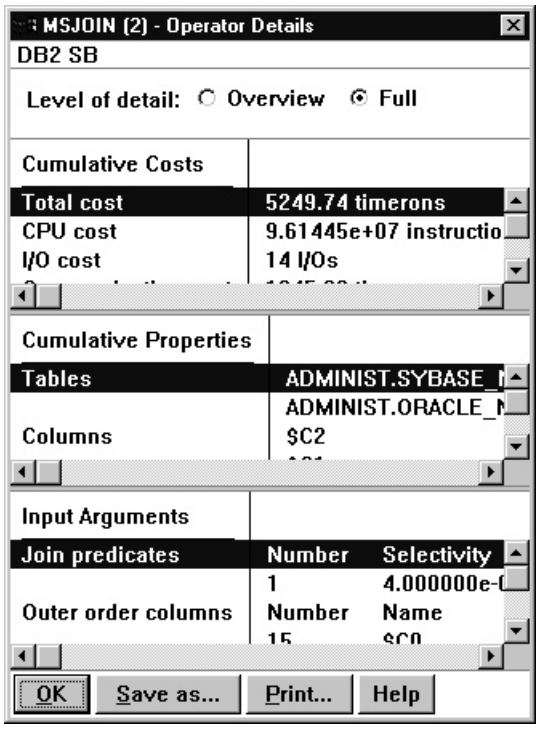


Figure 17. Operator Details Window: Visual Explain

The DataJoiner nicknames accessed by this query are listed in the Cumulative Properties section. If you scroll down to the bottom of the Cumulative Properties section, you can see a field called Comm Cost that displays the total cost of communication with the data sources accessed by this query.

Appendix B. Resolving Problems Encountered by Applications That Predate Version 2.1.1

This appendix explains how to resolve problems that arise when certain applications, such as those based on DataJoiner Version 1.2, try to perform operations that are no longer valid in Version 2.1.1, to query or modify catalog tables that were updated for Versions 2.1 and 2.1.1, or to query catalog views that were updated for Version 2.1.1.

The word *applications* here refers to a wide range of programs and instructions; for example:

- Application program code
- Third-party utilities
- Interactive SQL queries
- Commands
- API invocation

This appendix does not describe:

- DataJoiner operations that are less likely to generate an error in Version 2.1.1 than in Version 1.2. These operations can have only a positive impact on existing applications.
- Inter-version differences that are common to DataJoiner and DB2. For a discussion of problems that can result from them, see “Appendix I. Incompatibilities between Releases”, in the *DB2 SQL Reference for common servers.*

The problems that this appendix addresses are those that can arise when applications that predate DataJoiner Version 2.1.1 try to:

- Link DataJoiner libraries to certain clients and data sources in AIX
- Use the **db2start** and **db2stop** commands to start and stop Classic Connect processes
- Query DataJoiner Version 2.1.1 catalog tables, or query DB2 for CS catalog views that have been updated for DataJoiner Version 2.1.1
- Modify DataJoiner Version 2.1.1 catalog tables

Linking DataJoiner Libraries to Clients and Data Sources in AIX

This section indicates:

- How the method for linking DataJoiner libraries to clients and data sources has changed in Version 2.1.1
- What problem can result when a Version 1.2 application tries to link Version 2.1.1 libraries to certain clients and data sources
- How to resolve this problem

Change

When you use Version 1.2 in the AIX environment, you edit djxlink.makefile and run the djxlink shell script to link DataJoiner libraries to Oracle and Sybase client libraries or to DRDA data sources accessed through APPC. When you use Version 2.1.1 in the AIX environment, you do not need to edit djxlink.makefile for most data sources. Simply running djxlink.sh will link DataJoiner libraries to libraries of nearly all data sources.

Problem

Running the Version 1.2 djxlink shell script in Version 2.1.1 does not work. This script is not shipped with Version 2.1.1.

Resolution

Run the Version 2.1.1 djxlink.sh script. For the small number of data sources that cannot be link-edited using djxlink.sh, edit djxlink.makefile to contain the data source library information. Then use djxlink.makefile to link libraries.

Starting and Stopping Classic Connect Instances

This section indicates:

- How the method for starting and stopping Classic Connect instances has changed in Version 2.1.1
- What problems can result when a Version 1.2 application tries to start and stop Classic Connect instances
- How to prevent this problem

Change

In Version 1.2, the **db2start** command starts Classic Connect processes and the **db2stop** command stops these processes. In Version 2.1.1, the **djxstart** and **djxstop** commands start and stop Classic Connect processes.

Problem

If, in Version 2.1.1, you issue a **db2start** command, the Classic Connect processes will not start. If you issue **db2stop**, the Classic Connect processes will not stop.

Resolution

To start and stop Classic Connect processes for a DataJoiner instance in Version 2.1.1, issue the **djxstart** and **djxstop** commands.

Querying System Catalog Tables and Views

This section explains:

- How DataJoiner catalog tables and DB2 for CS catalog views have been updated to support DataJoiner Version 2.1.1
- What problems can result when certain applications, such as those based on DataJoiner Version 1.2, try to query these tables or views
- How to resolve these problems

Changes

Changes have been made to several DataJoiner system catalog tables, and to certain DB2 for CS catalog views that support DataJoiner. This section discusses:

- Changes that could cause problems for applications designed to access catalog tables that were used by DataJoiner Version 1.2
- Changes that could cause problems for applications designed to access DB2 for CS views that have been updated to support the Spatial Extender.

Changes in Tables Used by DataJoiner Version 1.2

DataJoiner Version 1.2 uses three DB2 for CS catalog tables—SYSCOLUMNS, SYSINDEXES, and SYSTABLES—and two tables specific to DataJoiner—SYSREMOTEUSERS and SYSSERVERS. The following changes, listed by table, were made for DataJoiner Version 2.1 and retained in Version 2.1.1:

The SYSCOLUMNS Table: The following changes, listed by column, were made to this table:

HIGH2KEY	Non-character values are now in printable format rather than binary format.
LOW2KEY	Non-character values are now in printable format rather than binary format.
NULLS	The value D (not null with default) has been changed to N (not nullable).
REMOTE_TYPE	In Version 1.2, values denoted data types of columns of data source tables that DataJoiner referenced by nickname. In Version 2.1.1, these values are stored in REMOTE_TYPENAME.

The SYSINDEXES Table: In Version 1.2, the value in the CLUSTERRATIO column of this table was -1 if statistics were not gathered. In Version 2.1.1, the value is -1 either if statistics are not gathered or if detailed index statistics are gathered. In the latter case, an appropriate value is added to the CLUSTERFACTOR column.

The SYSREMOTEUSERS Table: The data type for this table's AUTHID column was changed from CHAR to VARCHAR.

The SYSSERVERS Table: The following changes, listed by column, were made to this table:

COLSEQ	Deleted from SYSSERVERS. In Version 2.1.1, this server option is denoted by a value (colseq) in the OPTION column of the SYSCAT.SERVER_OPTIONS catalog view.
CONNECTSTRING	Deleted from SYSSERVERS. In Version 2.1.1, this server option is denoted by a value (connectstring) in the OPTION column of the SYSCAT.SERVER_OPTIONS catalog view.
CPURATIO	Data type changed from DOUBLE to FLOAT.
DATEFORMAT	Deleted from SYSSERVERS. In Version 2.1.1, this server option is denoted by a value (DATEFORMAT) in the OPTION column of the SYSCAT.SERVER_OPTIONS catalog view.
FOLDID	Deleted from SYSSERVERS. In Version 2.1.1, this server option is denoted by a value (fold_id) in the OPTION column of the SYSCAT.SERVER_OPTIONS catalog view.
IORATIO	Data type changed from DOUBLE to FLOAT.
PASSWORD	Deleted from SYSSERVERS. In Version 2.1.1, this server option is denoted by a value (password) in the OPTION column of the SYSCAT.SERVER_OPTIONS catalog view.
TIMEFORMAT	Deleted from SYSSERVERS. In Version 2.1.1, this server option is denoted by a value (TIMEFORMAT) in the OPTION column of the SYSCAT.SERVER_OPTIONS catalog view.
TIMESTAMPFORMAT	Deleted from SYSSERVERS. In Version 2.1.1, this server option is denoted by a value (TIMESTAMPFORMAT) in the OPTION column of the SYSCAT.SERVER_OPTIONS catalog view.

The SYSTABLES Table: The following changes, listed by column, were made to this table:

PACKED_DESC	Data type changed from LONGVARCHAR to BLOB.
REL_DESC	Data type changed from LONGVARCHAR to BLOB.
VIEW_DESC	Data type changed from LONGVARCHAR to BLOB.

Changes in DB2 for CS Views That Support the Spatial Extender

The following DB2 for CS catalog views were changed to support the Spatial Extender, an optional facility that became available with DataJoiner Version 2.1.1. For information about the Spatial Extender, see *DataJoiner Spatial Extender Administration Guide and Reference*.

The SYSCAT.DATATYPES View: The following columns were added to this view: EXTRA_LENGTH, TYPE_PRECEDENCE, and INSTANTIABLE.

The SYSCAT.FUNCPARMS View: The following columns were added to this view: PARMNAME, TYPE_PRESERVING, and MUTATED.

The SYSCAT.FUNCTIONS View: The following columns were added to this view: CONTAINS_SQL, DBINFO, RESULT_COLS, BODY, EFFECT, TYPE_PRESERVING, FUNC_PATH, and SELECTIVITY.

The SYSCAT.TRIGDEP View: A column named DTYPE was added to SYSCAT.TRIGDEP.

Problems

A variety of problems could occur. For example:

- If a DataJoiner Version 1.2 application does a qualified search on a column that takes a different value than it did before (for example, a search on NULLS in SYSIBM.SYSCOLUMNS for a value of D), the application might react differently than expected.
- If a DataJoiner Version 1.2 application queries a column whose data type has changed (for example, CPURATIO in SYSIBM.SYSSERVERS), too much or too little data might be returned.
- If a DB2 for CS application uses star notation (SELECT *) to query a view with new columns that the application doesn't recognize (for example, SYSCAT.DATATYPES, which has several new columns to support the Spatial Extender), the application will receive an error.

Resolution

Review the changes listed above to decide whether they affect your applications and, if so, what corrective action to take (for example, updating the application). So that any problems in accessing or maintaining catalog tables can be avoided, we strongly recommend that instead of querying these tables, you query the catalog views derived from them.

If you need a rough approximation of the degree of clustering, select both CLUSTERRATIO and CLUSTERFACTOR in the SYSCAT.INDEXES catalog view and choose the greater of the two values that you retrieve.

Modifying System Catalog Tables

This section explains:

- How the method for modifying system catalog tables changed in Version 2.1.1
- What problems can result when Version 1.2 applications try to modify Version 2.1.1 catalog tables
- How to resolve these problems

Change

For DataJoiner to perform operations on a specific data source, DataJoiner must associate an identifier (specifically, a server name) with that data source. In Version 1.2, you could create such an association by inserting appropriate values into the table SYSIBM.SYSSERVERS. You could also modify an association by updating SYSIBM.SYSSERVERS, and terminate an association by deleting a server name from SYSIBM.SYSSERVERS. In Versions 2.1 and 2.1.1, you use DDL to perform these same operations indirectly. Specifically, you create DataJoiner-to-data source associations with the CREATE SERVER MAPPING statement, modify them with the ALTER SERVER MAPPING statement, and terminate them with the DROP statement. These statements operate on SYSCAT.SERVERS, a catalog view derived from SYSIBM.SYSSERVERS. The changes that you make to the view are propagated to SYSIBM.SYSSERVERS.

For a user to access data sources from DataJoiner, DataJoiner must associate the ID under which the user connects to DataJoiner with the IDs under which the user connects to these data sources. In Version 1.2, you could create such an association by inserting appropriate values into the table SYSIBM.SYSREMOTEUSERS. You could also modify an association by updating SYSIBM.SYSREMOTEUSERS, and terminate an association by deleting an ID from SYSIBM.SYSREMOTEUSERS. In Versions 2.1 and 2.1.1, you use DDL to perform these same operations indirectly. Specifically, you create associations between IDs with the CREATE USER MAPPING statement, modify them with the ALTER USER MAPPING statement, and terminate them with the DROP statement. These statements operate on SYSCAT.REMOTEUSERS, a catalog view derived from SYSIBM.SYSREMOTEUSERS. The changes that you make to the view are propagated to SYSIBM.SYSREMOTEUSERS.

Problem

If you issue an INSERT, UPDATE, or DELETE statement against SYSIBM.SYSSERVERS, SYSIBM.SYSREMOTEUSERS, or any of DataJoiner's other system catalog tables, the statement will fail.

Resolution

To modify SYSIBM.SYSSERVERS or SYSIBM.SYSREMOTEUSERS, use the SERVER MAPPING or USER MAPPING DDLs, as described in "Change".

Appendix C. Where to Find Out More about DataJoiner, DB2 for CS, and Replication Products

This appendix lists IBM books about DataJoiner, DB2 for CS, and Replication Administration; states how to obtain these books; and tells you where to go on the Internet to learn more about DataJoiner.

DataJoiner, DB2 for CS, and Replication Publications

Table 9 lists the DataJoiner, DB2 for CS, and Replication books applicable to installing, configuring, administrating, using, and running applications against DataJoiner. The *DataJoiner for AIX Planning, Installation, and Configuration Guide* and the *DataJoiner for Windows NT Systems Planning, Installation, and Configuration Guide* are provided in hardcopy with DataJoiner. In addition, these two books and all other DataJoiner books are provided in softcopy formats (PostScript, HTML, and PDF) on the product CD-ROM. All other books in Table 9 are provided in PostScript; most are also provided in HTML (the two exceptions are the DB2 for CS Software Developer Kit publications). Additionally, most of the DB2 for CS books are provided in INF format (see Table 9).

To understand how the DataJoiner books in Table 9 are organized, it is important to understand how DataJoiner and DB2 for CS are interrelated. DataJoiner provides a “superset” of DB2 for CS. The two products share common functions and syntax; therefore, information that is common to DataJoiner and DB2 for CS is documented in the DB2 for CS books. The DataJoiner books listed in Table 9 document the function and syntax that DataJoiner has *in addition to* the function and syntax that it shares with DB2 for CS.

Table 9 does not list all of the DB2 for CS books. View or print a DB2 for CS book to see the publications list for all DB2 for CS books.

If you order Classic Connect, you will receive additional books (the *DataJoiner Classic Connect Planning, Installation, and Configuration Guide*, the *DataJoiner Classic Connect data mapper Sample for Windows Installing and Using Guide*, and the *DataJoiner Messages and Problem Determination Guide*) and a program directory.

Table 9. DataJoiner, DB2 for CS, and Replication publications applicable to DataJoiner

Book Name	Form Number	File Prefix	INF
DataJoiner Version 2.1.1 Books			
DataJoiner for Windows NT Systems Planning, Installation, and Configuration Guide	SC26-9150	DJXN2	no
This book covers capacity planning, resource management, installation, and configuration tasks for IBM DataJoiner on Microsoft Windows NT operating systems.			

Table 9. DataJoiner, DB2 for CS, and Replication publications applicable to DataJoiner (continued)

Book Name	Form Number	File Prefix	INF
DataJoiner for AIX Systems Planning, Installation, and Configuration Guide	SC26–9145	DJXG6	no
This book covers capacity planning, resource management, installation, and configuration tasks for IBM DataJoiner on AIX operating systems.			
DataJoiner Administration Supplement	SC26–9146	DJXD5	no
This book provides information that assists DBAs and other system administrators of DataJoiner with performing administrative tasks. It includes a product overview section, security considerations, data source identification steps, database utility notes, performance considerations, database system monitor reference data, large object information, and explain tool examples.			
DataJoiner Application Programming and SQL Reference Supplement	SC26–9148	DJXK5	no
This book provides SQL statements, descriptions of system catalog data, guidelines, and other information for application programmers. With this information, application programmers can use DataJoiner to perform multiple tasks in a distributed database environment—tasks such as creating nicknames by which to reference tables and views, invoking functions and stored procedures, passing SQL directly to databases for processing, and using server options to optimize query performance.			
DataJoiner Generic Access API Reference	SC26–9147	DJXM4	no
This book explains how to create a generic access module that allows you to use existing drivers or to create new drivers to gain access to an unlimited set of data sources.			
DataJoiner Classic Connect Planning, Installation, and Configuration Guide	GC26–8869	DJXC4	no
This book provides information on the DataJoiner Classic Connect for MVS product. The audience for this information includes application programmers, database administrators, network administrators, system administrators, and system programmers. The book documents key tasks required to set up Classic Connect in the MVS operating environment: planning your setup; installing components via SMP/E, configuring the kernel, DMSIs, and network communications; managing instances; and creating relational data maps for IMS and VSAM data.			
DataJoiner Classic Connect data mapper Sample for Windows Installing and Using Guide	GC26–8873	DJXZ2	no
This book provides information on the DataJoiner Classic Connect data mapper sample for Windows. The audience for this information includes system programmers, DBAs, or anyone that needs to produce relational maps (USE grammar) for IMS and VSAM data. The book documents key tasks required to set up and use the data mapper in the Windows environment: installing product files, starting the product, and generating USE grammar statements for input to DataJoiner Classic Connect projection utilities.			

Table 9. DataJoiner, DB2 for CS, and Replication publications applicable to DataJoiner (continued)

Book Name	Form Number	File Prefix	INF
DataJoiner Messages and Problem Determination Guide	SC26–9149	DJXP4	no
This book describes the messages and codes issued by DataJoiner and Classic Connect instances. For messages that report errors, the book explains the cause of the errors and recommends corrective actions. The book also provides guidelines on using diagnostic tools to isolate and understand problems.			
DB2 Spatial Extender Administration Guide and Reference	SC26–9316	DJXS1	no
This book provides instructions for spatially enabling a DataJoiner database, an introduction to spatial capabilities using geometry data types and functions, descriptions of spatial data exchange formats, an SQL and message reference for spatial data, and appendices containing the standard representations of spatial reference systems.			
DB2 for CS and Replication Books			
DB2 Information and Concepts Guide	SH20–4664	SQLG0	no
Provides product and conceptual information to anyone who needs a comprehensive overview of the DB2 products. It is useful when deciding which DB2 products suit your environment. It also includes a glossary of terms used in the book.			
DB2 Administration Guide	S20H-4580	SQLD0	yes
Contains information required to design, implement, and maintain a database to be accessed either locally or in a client/server environment.			
DB2 Database System Monitor Guide and Reference	S20H–4871	SQLF0	yes
Includes a description of how to use the Database System Monitor and a description of all the data elements for which information can be collected.			
DB2 Command Reference	S20H–4645	SQLN0	yes
Provides the reference information needed to use system commands and the DB2 command line processor to execute database administrative functions. Describes the commands that can be entered at an operating system command prompt or in a shell script to access the database manager. Explains how to invoke and use the command line processor, and describes the command line processor options. Provides a description of all the database manager commands.			
DB2 API Reference	S20H–4984	SQLB0	yes

Table 9. DataJoiner, DB2 for CS, and Replication publications applicable to DataJoiner (continued)

Book Name	Form Number	File Prefix	INF
Provides information about the use of application programming interfaces (APIs) to execute database administrative functions. Presents a description of APIs and the data structures used when calling APIs, as well as detailed information on the use of database manager API calls in applications written in the supported programming languages.			
DB2 SQL Reference	S20H–4665	SQLS0	yes
Is intended to serve as a reference for syntax and rules governing the use of SQL statements. Syntax diagrams, semantic descriptions, rules, and examples are provided for the SQL statements. Catalog views, product maximums, release-to-release incompatibilities, and a glossary are also included in this book.			
DB2 Application Programming Guide	S20H–4643	SQLA0	yes
Discusses the application development process and how to code, compile, and execute application programs that use embedded SQL to access the database. It includes discussions on programming techniques and performance considerations for the application programmer.			
DB2 Call Level Interface Guide and Reference	S20H–4644	SQLL0	yes
Is a guide and reference manual for programmers using the Call Level Interface. DB2 Call Level Interface is a callable SQL interface based on the X/Open CLI specification and is compatible with Microsoft Corporation's ODBC.			
DB2 Messages Reference	S20H–4808	SQLM0	yes
Lists messages and explanations. Each explanation includes the action to be taken when a message or code is issued.			
DB2 Problem Determination Guide	S20H–4779	SQLP0	yes
Provides information that helps in determining the source of errors, recovering from problems, and describing and reporting defects.			
DDCS User's Guide	S20H–4793	SQLC0	yes
Provides concepts, programming guidelines, and general information about the DDCS products.			
DB2 Replication Guide and Reference	S95H–0999	DB3E0	no
Describes how to plan, configure, administer, and operate IBM replication products, including the Apply and Capture programs.			
DB2 for CS Platform-Specific Books			
DB2 SDK for AIX Building Your Applications	S20H-4780	SQLA3	yes

Table 9. DataJoiner, DB2 for CS, and Replication publications applicable to DataJoiner (continued)

Book Name	Form Number	File Prefix	INF
This book provides environment setup information and step-by-step instructions to compile and link DB2 applications on the AIX operating system.			
DB2 SDK for Windows 95 and NT Building Your Applications	S33H-0310	SQLA6	yes
This book provides environment setup information and step-by-step instructions to compile and link DB2 applications on Windows 95 and NT operating systems.			

How to Order, View, and Print Publications

Use order number SBOF-5289 to request one hardcopy of each of the DataJoiner, DB2 for CS, and Replication books shown in Table 9 on page 161.

To view online documentation, follow the instructions located in the README files on the CD-ROM. Most of the books in Table 9 on page 161 are provided as HTML files and can be viewed with an HTML browser. You can also view INF versions of many DB2 for CS books. Instructions for installing the INF reader on AIX are provided in the DB2 README files; on NT operating systems, the INF reader is installed automatically. DataJoiner and Replication information is not provided in INF format.

To print individual books, follow the instructions provided in the README files on the CD-ROM. PostScript files for all the books are provided.

Internet Resources

The following Internet resources provide additional information about DataJoiner.

World Wide Web

The following DataJoiner-specific Web site contains general and technical (frequently asked questions) product information. The address of the site is:

http://www.software.ibm.com/data/datajoiner/

Also available online are the most current versions of books in the DB2 library. You can view books in the DB2 library by clicking the Library link from the following address:

http://www.software.ibm.com/data/pubs/techinfo.html

Internet Newsgroups

DataJoiner questions, answers, and discussions can be found in:

- bit.listserv.db2-l
- comp.databases

- comp.databases.ibm-db2

Appendix D. DataJoiner Classes and Services

This appendix describes:

- Classes you can take to learn about DataJoiner
- Services to help you plan to use DataJoiner, and to install and configure it

DataJoiner Classes

IBM offers classes that teach you how to install, use, and maintain DataJoiner. These classes are described in this section.

For more information, or to enroll in any IBM class, call 1-800-IBM-TEACH (1-800-426-8322) and refer to the IBM US Course Code. For locations outside the United States, contact your IBM representative.

Class descriptions will also be maintained at the DataJoiner Web site. The DataJoiner URL is:

http://www.software.ibm.com/data/datajoiner/

Using DataJoiner

IBM US Course Code DW202

Duration
2 days

Format Lecture with classroom exercises.

This course introduces the student to DataJoiner and its powerful multidatabase server capabilities. After completing this course, students should be able to effectively use DataJoiner to perform simple and complex distributed requests. They should also be able to monitor and tune SQL queries, accounting for the capabilities and characteristics of diverse DataJoiner data sources. Areas covered include:

- Global optimization
- Multi-vendor query considerations
- Nicknames
- Basic security
- An introduction to the DataJoiner catalog
- DataJoiner query performance
- The DataJoiner Explain tool
- The DataJoiner Database System Monitor

Who Should Take This Course
This course is appropriate for anyone who will be using, managing, installing, or maintaining a DataJoiner multiple database environment.

Prerequisite
SQL experience. You can obtain this experience by attending the “SQL Workshop,” IBM US Course Code CF120.

DataJoiner Administration

IBM US Course Code DW212

Duration
3 days

Format Lecture with classroom exercises.

This course trains the student to install, configure, and manage a secure DataJoiner multidatabase server environment. Areas covered include:

- Installing DataJoiner
- Generating and managing the DataJoiner database
- Configuring DataJoiner
- Enabling DataJoiner client access to remote data sources
- DataJoiner security
- DataJoiner server performance

Who Should Take This Course
This course is appropriate for anyone who will be managing, installing, or maintaining a DataJoiner multiple database environment.

Prerequisite
DataJoiner knowledge or experience. You can obtain this experience by attending “Using DataJoiner,” IBM US Course Code DW202.

DataJoiner Services

IBM provides services for DataJoiner that include assistance with planning, installing, and configuring the product. The assistance is customized to your individual environment and takes place in two phases.

First Phase: Planning

The first phase helps you plan the installation and configuration of DataJoiner, and to configure network systems so that DataJoiner can communicate optimally with all data sources and clients. This phase includes:

- Assessing general readiness
- Defining clients

- Defining data sources
- Assessing applications
- Defining backup and recovery strategies for DataJoiner
- Configuring DataJoiner database parameters
- Identifying test queries for system validation
- Defining security requirements

Second Phase: Implementation

The second phase focuses on implementing the plan developed in the planning phase. It includes:

- Installing DataJoiner
- Configuring data sources
- Providing access to data source tables and views
- Installing and configuring remote clients
- Validating and documenting the environment
- Providing final turnover to the customer

At the end of this phase, active remote and local clients can access multiple data sources through DataJoiner.

DataJoiner services can be combined with replication services if you are interested in replicating data across a heterogeneous database environment. For more information about DataJoiner and replication services, contact your IBM representative or see the DataJoiner Web page. The DataJoiner URL is:

http://www.software.ibm.com/data/datajoiner/

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
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

W92/H3
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

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.

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.

Trademarks

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

ADSTAR
Advanced Peer-to-Peer Networking
AIX
APPN
AS/400
AT
CICS
CICS/6000
Client Acces
Current
DATABASE 2
DataGuide
DataJoiner
DataPropagator
DataRefresher
DB2
DFSMS
Distributed Relational Database Architecture
DProp
DRDA
Extended Services for OS/2
HACMP/6000
IBM
IIN
IMS
IMS/ESA
Language Environment
MVS
MVS/ESA
MVS/XA
NetView
Operating System/2
Operating System/400
OS/2
OS/390
OS/400
RACF
RETAIN
RISC System/6000
RS/6000
RT
SP
SQL/DS
SQL/400
System/390
VisualAge
VTAM

Intel is a registered trademark of the Intel Corporation in the United States and other countries.

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

Microsoft, Windows, WindowsNT®, and the Windows logo are registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

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

Index

E

O

P

Q

R

S

T

U

V

W

Readers' Comments — We'd Like to Hear from You

DB2 DataJoiner ®
Administration Supplement
Version 2 Release 1 Modification 1

Publication No. SC26-9146-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC26-9146-01

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation, BWE/H3
PO Box 49023
San Jose, CA 95161-9945

Fold and Tape | **Please do not staple** | Fold and Tape

SC26-9146-01

Cut or Fold
Along Line

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-9146-01

Spine information:

DB2 DataJoiner

DB2 DataJoiner Version 2
Administration Supplement

Version 2
Release 1
Modification 1

SC26-
9146-01