DB2 DataJoiner®

Application Programming and SQL Reference Supplement

Version 2 Release 1 Modification 1

SC26-9148-01

DB2 DataJoiner®

Application Programming and SQL Reference Supplement

Version 2 Release 1 Modification 1

SC26-9148-01

Note

Before using this information and the product it supports, please read the general information under "Notices" on page 217.

Second Edition (July 1998)

This edition replaces and makes obsolete the previous edition, SC26-9148-00. The technical changes for this edition are summarized under "What's New in DataJoiner Version 2?" on page xi, 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.

Contents

About This Book

This book provides SQL statements, descriptions of system catalog data, pointers on optimizing queries, 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 submitting distributed queries to heterogeneous data sources, performing multi-database transactions, creating data source tables, and invoking stored procedures.

Who Should Use This Book

This book is for application programmers who develop applications to access heterogeneous data with DataJoiner. *Heterogeneous data* is data from a variety of data sources including members of the DB2 family of databases, Sybase, Oracle, and IMS. This book also provides reference information for database administrators and system administrators. For information specific to the data sources that you are using, refer to the documentation for those data sources.

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.

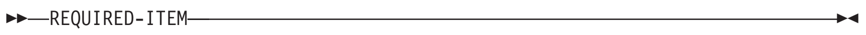
Diagrams of syntactical units other than complete statements start with the ▶—— symbol and end with the ——▶ symbol.

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 ƀ symbol indicates one blank position.

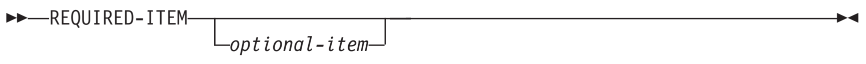
Required items

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

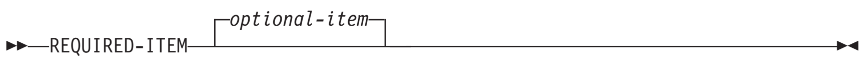


Optional items

Optional items appear below the main path.

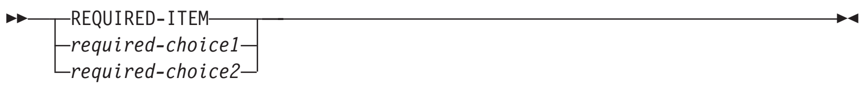


If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

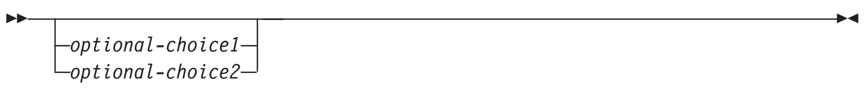


Multiple required or optional items

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.

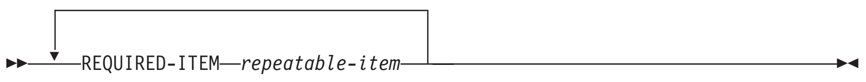


If choosing one of the items is optional, the entire stack appears below the main path.

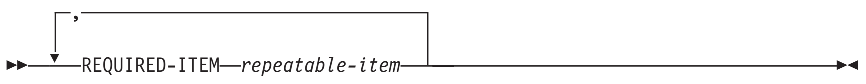


Repeatable items

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



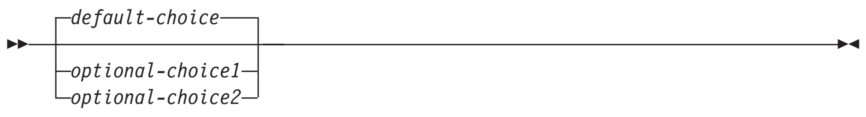
If the repeat arrow contains a comma, you must separate repeated items with a comma.



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. Accessing, Querying, and Configuring Data Sources

DataJoiner is a multi-database server that provides client access to diverse data sources, both relational and nonrelational, on multiple platforms. A *data source* is a logical database management system plus the database or databases that the system supports. The particulars of the system and supported databases vary with RDBMS type. For example, in DB2 for OS/390, each system is an instance of the database manager and supports multiple databases. In Oracle, each system is a combination of processes and memory buffers, and supports a single database. Each instance of DataJoiner presents a unified database image of the various data sources that it supports to its users.

Note on terms: Because you use DataJoiner to submit requests to data sources, you can regard data sources as servers. Accordingly, this book uses the terms *data source* and *server* interchangeably. Be aware that DataJoiner SQL uses *server* to refer to data sources. An example is the following statement, which creates a mapping between a data type defined to the DataJoiner database and a data type defined to an Oracle database:

```
CREATE TYPE MAPPING MY_ORACLE_DATE
  FROM SYSIBM.DATE
  TO SERVER ORACLE1
  TYPE DATE
```

This statement refers to ORACLE1 as a server; but in keeping with the convention of this book, you can refer to ORACLE1 also as a data source. (For more information about this statement, see “CREATE TYPE MAPPING” on page 112.)

This chapter provides information to help you to:

- Access data sources
- Submit queries to retrieve and manipulate their data
- Configure data sources to facilitate optimization and to enable certain kinds of processing, such as two-phase commit transactions

Accessing Data Sources

This section discusses:

- The interfaces through which you can access DataJoiner and the data sources that it supports
- The authorizations that are required to use these data sources
- Associations that DataJoiner establishes with data sources; for example, DataJoiner-to-data source mappings by which DataJoiner references specific data sources and specific data source tables.

- The way in which DataJoiner responds to requests for isolation levels for locks on data source data

Interfaces to DataJoiner

You can access the DataJoiner database and data sources supported by DataJoiner through:

- The command line processor (CLP)
- Embedded SQL in an application program
- The DB2 Call Level Interface (CLI)
- The Java Database Connectivity (JDBC) API

You can access data sources not supported by DataJoiner through:

- A third-party gateway such as CrossAccess.
- A custom data access module that you create. You can create this module with the Generic Access API. For more information, see the *DataJoiner Generic Access API Reference.*

The following sections discuss the CLP, embedded SQL, the DB2 CLI, and the JDBC API.

The CLP

Use the CLP to access and manipulate DataJoiner and data source databases from the system command prompt. With the CLP, you can:

- Issue SQL statements and DataJoiner commands
- Maintain a history file of all requests
- Redirect the output of a CLP request
- Access both local and remote databases
- Request syntax help for DB2 commands
- Request message help

The CLP is automatically installed with DataJoiner. For more information about it, see the *DB2 Command Reference*.

Embedded SQL

You can access and manipulate data by embedding SQL statements in application programs. DataJoiner provides support for compiled languages such as C, COBOL, and FORTRAN, and for interpreted languages, such as REXX. For a compiled language, an appropriate precompiler must be available to process the SQL statements. For a list of compilers, see the *DataJoiner Planning, Installation, and Configuration Guide* for your platform. You can use other programming languages if you write a customized precompiler by using the DataJoiner precompiler API. See the *DB2 Command Reference* for more information about the precompiler API.

The DB2 CLI

The DB2 CLI provides a set of standard function calls that invoke SQL statements and direct the processing of them. For example, the function call SQLTables() invokes the SQL for querying tables and views. The function SQLExecute initiates the processing of an SQL statement.

Unlike embedded SQL, the SQL that's invoked by the DB2 CLI doesn't need to be compiled. Therefore, an application that uses this interface is independent of any particular database server. Accordingly, the application doesn't need to be recompiled to access different database servers, but can select the appropriate server at run time.

The DB2 CLI is based on the evolving X/Open Call Level Interface standard, which is compatible with the Microsoft Core Open Database Connectivity (ODBC) standard.

The JDBC API

You can use function calls based on the Java Database Connectivity (JDBC) API to construct and execute SQL. You invoke the calls from Java applets and applications. DataJoiner supports the Java Development Kit (JDK) 1.1 implementation of JDBC.

Authorization

Authorizations to use databases and database objects reside at DataJoiner and at the clients and data sources that you use with DataJoiner. You need to carefully coordinate the authorizations between DataJoiner, the clients, and the data sources. A user must have authority (whether granted explicitly or implicitly) to perform an action at DataJoiner. The corresponding user ID at the data source must also have authority to perform the action. If matching authorizations do not exist both at DataJoiner and at the data source, the action fails.

For example, suppose that DataJoiner is connected to a DB2 for OS/390 data source and an Oracle data source. A user, Bob Smith, is assigned the authorization ID BOBSMITH on the DataJoiner server. The same user is assigned the authorization ID SMITHB on the DB2 for OS/390 data source and the authorization ID BOBS on the Oracle data source. Bob wants to join an Oracle table named JOBS with a DB2 for OS/390 table named EMPLOYEE. The DataJoiner nicknames for the tables are the same as their table names.

When BOBSMITH attempts to access DataJoiner, DataJoiner checks to see if BOBSMITH has connect authority. If so, DataJoiner gives BOBSMITH access to the server. When BOBSMITH submits a query, DataJoiner checks to see if this authorization ID has authority to access the nicknames or local tables contained in the query. If so, DataJoiner constructs the appropriate queries and sends them to the DB2 for OS/390 and Oracle data sources. For DataJoiner to access the data, the authorization IDs SMITHB and BOBS must have authority at their respective data sources, to first connect and then select the required data from the tables.

For Bob to perform his join, all the following requirements must be met:

- User BOBSMITH on the DataJoiner server must have:
  - Connect authority for the DataJoiner database
  - Select authority for the nicknames EMPLOYEE and JOBS
- User SMITHB on the DB2 for OS/390 data source must have:
  - Connect authority for the DB2 for OS/390 database
  - Select authority for the EMPLOYEE table
- User BOBS on the Oracle data source must have:
  - Connect authority for the Oracle database
  - Select authority for the JOBS table
- SYSCAT.REMOTEUSERS must contain entries that map BOBSMITH to:
  - SMITHB on the DB2 for OS/390 data source
  - BOBS on the Oracle data source
- SYSCAT.SERVERS must contain entries in the SERVER column that match the SERVER entries in SYSCAT.REMOTEUSERS for the DB2 for OS/390 and Oracle data sources.

System administrators and database administrators control who has access to DataJoiner and the data sources, and to what extent each user has access, and are responsible for the safety and integrity of the data.

DataJoiner provides four administrative authorities:

SYSADM	System administrator authority
DBADM	Database administrator authority
SYSCTRL	System control authority
SYSMAINT	System maintenance authority

For more information on security, see the *DataJoiner Administration Supplement*

DataJoiner-to-Data Source Associations

| For DataJoiner to operate on data sources, you need to create (or, in some cases,
| make use of) certain kinds of associations, or mappings, between DataJoiner and the
| data sources; for example, mappings between:

| - DataJoiner's identifiers for data sources and the data sources themselves
| - User IDs at DataJoiner and their counterparts at data sources
| - DataJoiner's identifiers for data source objects and the objects themselves
| - Function specifications at DataJoiner and corresponding functions at data sources
| - DataJoiner data types and data source data types

Identifiers for Data Sources

For DataJoiner to operate on a specific data source, DataJoiner must associate an identifier (specifically, a server name) with that data source. You form this association with the CREATE SERVER MAPPING statement (described on page 96). If you want, you can later modify the association (for example, assign a new server name to a data source) with the ALTER SERVER MAPPING statement (described on page 59).

Mappings between User IDs

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. You can create such an association with the CREATE USER MAPPING statement (described on page118) and modify it with the ALTER USER MAPPING statement (page 69).

Identifiers for Tables, Views, and Stored Procedures

So that DataJoiner can access a table, view, or stored procedure at a data source, you need to create a nickname by which DataJoiner can reference the table, view or stored procedure. You do this with the CREATE NICKNAME statement (described on page 86) and CREATE STORED PROCEDURE NICKNAME statement (page 103). To change a nickname associated with a table or view, use the ALTER NICKNAME statement. For more information, see “Nicknames” on page 23.

Mappings between Functions

If you want DataJoiner to invoke a function at a data source, and the function is user-defined or built-in but unknown to DataJoiner, you must map it to a function or function specification at the DataJoiner database. You do this with the CREATE FUNCTION MAPPING statement (described on page 80). For more information, see “UDFs” on page 38.

Mappings between Data Types

DataJoiner maintains two kinds of mappings between data source data types (called *remote types* here, for short) and data types defined to the DataJoiner database (called *local types* here, for short). In a *forward type mapping,* a remote type points to a comparable local type. For example, there’s a default forward type mapping in which the Sybase type int points to the local type INTEGER.

When you define a remote table or view to DataJoiner, DataJoiner includes in the definition the local types that the table’s or view’s types point to. For example, suppose that a Sybase table contains a column C1 with a data type of int. If you create a nickname for this table, the table will be defined to DataJoiner and, if you don’t override the default mapping of int to INTEGER, the definition will include INTEGER as the local counterpart to int for C1.

In a *reverse type mapping,* a local type points to a comparable remote type. For example, there’s a default reverse mapping in which the local type DECIMAL points to the SQL Anywhere type NUMERIC.

When you use DataJoiner’s CREATE TABLE statement to create a remote table, the table’s columns are assigned types that the local types in the statement point to. For example, suppose that you use the CREATE TABLE statement to define an RDB table with a column C2, and that you specify INTEGER for C2 in the statement. If you don’t override the default mapping of INTEGER to SQL_INTEGER, SQL_INTEGER will be defined for C2 at the RDB data source.

You can override a default forward type mapping, or create a new forward type mapping, with the CREATE TYPE MAPPING statement (described on page 112). You can override a default reverse type mapping, or create a new reverse type mapping, with the CREATE REVERSE TYPE MAPPING statement (page 89).

For more information, see “Data Type Mappings” on page 26.

Isolation Levels

When DataJoiner issues queries to a data source, it tries to ensure that the isolation level requested by the user is mapped correctly on the remote database. Table 1 lists:

- The isolation levels that the user can request from DataJoiner. They are:

CS	Cursor stability
RR	Repeatable read
RS	Read stability
UR	Uncommitted read

- Data source isolation levels that requested levels map to.

Table 1. Comparable Isolation Levels between DataJoiner, Sybase, Microsoft SQL Server, Oracle, Informix, and RDB

DataJoiner	CS	RR	RS	UR
Sybase	Default	HOLDLOCK	HOLDLOCK	Same as cursor stability
Microsoft SQL Server	Default	HOLDLOCK	HOLDLOCK	Same as cursor stability
Oracle	Default	Transaction read-only (when the transaction is read-only). Default (when the transaction is not read-only).	Transaction read-only (when the transaction is read-only). Default (when the transaction is not read-only).	Same as cursor stability

Table 1. Comparable Isolation Levels between DataJoiner, Sybase, Microsoft SQL Server, Oracle, Informix, and RDB (continued)

Informix	Cursor stability	Repeatable read	Repeatable read	Dirty read
RDB	Default	SQL_TXN_SERIALIZABLE	SQL_TXN_REPEATABLE _READ	Not supported

For more information about DB2 for CS isolation levels, see the DB2 *Administration Guide.*

Querying Data Sources

This section discusses:

- The language that you use to define and manipulate data at data sources: structured query language (SQL)
- How you can use DataJoiner's SQL to perform operations at a data source that the data source's native SQL can't perform
- How you can use DataJoiner to query a data source with the data source's own SQL
- What error codes DataJoiner issues when errors occur

Structured Query Language (SQL)

SQL is a standardized language for defining and manipulating data in a relational database. DataJoiner uses the SQL language of DB2. This section describes the SQL concepts for DataJoiner that differ from or add to the SQL used by DB2. For more information on general SQL concepts, see the *DATABASE 2 SQL Reference*.

DataJoiner allows for both dynamic and static SQL statements. The method of compiling an SQL statement and the persistence of its operational form distinguish static SQL from dynamic SQL.

Dynamic SQL statements are constructed and made operational at run time. This can be done in several ways; for example, through:

- The processing of two specialized embedded SQL statements, PREPARE and EXECUTE
- The DB2 CLI

For more information about dynamic SQL, see the *DB2 Application Programming Guide* and the *DATABASE 2 SQL Reference for common servers*.

Static SQL statements must be embedded within application programs. Static SQL statements are transformed to their operational form at precompile time, as opposed to application run time. In some cases, static SQL provides a performance advantage over dynamic SQL. The DataJoiner precompile process is identical to that of DB2.

How DataJoiner handles a static SQL statement that references a table or view depends on whether the table or view is local or remote:

- If the table or view is local, the statement is prepared before the program that contains it is run. In addition, the operational form of the statement persists beyond the execution of the program. This procedure is identical to how the DB2 family of databases handles static SQL statements.
- If the table or view is remote and the statement references it by a nickname, DataJoiner handles the statement dynamically, so that it becomes operational at run time and does not persist after the program containing it is executed. Even if a static SQL statement could be passed to a data source unchanged, DataJoiner passes it as a dynamic statement because many data sources do not support the concept of static SQL. Each data source is responsible for optimizing the dynamic SQL statements that it receives from DataJoiner.

Compensation

Compensation is the processing of SQL statements for RDBMSs that do not support those statements. Each type of RDBMS (DB2, Informix, Oracle, and so on) supports a subset of the international standard of SQL. In addition, some types support SQL constructs that exceed this standard; for example, DB2's rules for updating views surpass the standard. The totality of SQL that a type of RDBMS supports is called an *SQL dialect*. With DataJoiner, you can use DB2's SQL dialect to compensate for lacks in other RDBMS's SQL dialects.

For example:

- Only DB2 SQL includes the clause, common-table-expression. In this clause, you specify a name by which all FROM clauses in a fullselect can reference a table. With DataJoiner, you can process common-table-expression for a database in Informix, Sybase, or any other non-DB2 family RDBMS, even though the SQL dialect supported by this other RDBMS doesn't include common-table-expression.
- When connecting to a data source that does not support multiple open cursors within an application, DataJoiner can simulate this function by establishing separate, simultaneous connections to the data source. DataJoiner can use this same technique to simulate CURSOR WITH HOLD capability for a data source that does not provide that function.

Compensation makes it possible to use the DB2 SQL dialect to make all queries supported by DataJoiner. You don't need to use dialects specific to RDBMSs other than | DB2.

| ## Pass-Through Sessions for Querying Data Sources in Their Own SQL

| With the pass-through function (called *pass-through* for short), you can query a data
| source in the SQL that's native to that data source. This section:

| - Summarizes the SQL for using pass-through
| - Describes occasions for using pass-through

- States what kind of SQL statements DataJoiner and data sources process in pass-through sessions
- Lists considerations and restrictions to be aware of when you use pass-through

SQL for Using Pass-Through

DataJoiner provides the following SQL statements to manage pass-through sessions:

SET PASSTHRU
: Provides a way to communicate with a data source in the SQL dialect directly supported by that data source.

SET PASSTHRU RESET
: Terminates a pass-through session

GRANT PASSTHRU
: Grants a user, group, list of authorization IDs, or PUBLIC the privilege to initiate pass-through sessions to a specific server

  You use the GRANT PASSTHRU SQL statement if you want users (other than those having SYSADM, DBADM, SYSCTRL, or SYSMAINT authority) to issue SET PASSTHRU SQL statements. For example:

```
GRANT PASSTHRU ON SERVER ORACLE1 TO USER SHAWNB
```

REVOKE PASSTHRU
: Removes the privilege to initiate pass-through sessions to a specific server from a user, group, list of authorization IDs, or PUBLIC.

When to Use Pass-Through

You might use a pass-through session to submit:

- Data Definition Language (DDL) statements in a data source's own SQL dialect
- Data Manipulation Language (DML) statements that are not supported by DataJoiner

You can issue the SET PASSTHRU and SET PASSTHRU RESET SQL statements only as dynamic SQL. You can issue these SQL statements via the PREPARE, EXECUTE, or EXECUTE IMMEDIATE SQL statements.

SQL Processing in Pass-Through Sessions

The following rules specify whether an SQL statement is processed by the data source or by DataJoiner:

- DataJoiner processes all static SQL statements.
- The data source processes dynamic SQL statements that undergo a PREPARE within the pass-through session.
- DataJoiner processes dynamic SQL statements that do not undergo a PREPARE within the pass-through session.

| • The COMMIT, ROLLBACK, SET PASSTHRU, and SET PASSTHRU RESET
| statements are not “passed through” to data sources. These statements are
| processed by the local DataJoiner instance.

Considerations and Restrictions

There are a number of considerations and restrictions to bear in mind when you use pass-through. Some of these considerations and restrictions are of a general nature; others apply to specific data sources: namely, Microsoft SQL Server, Sybase, and Oracle.

General Information:

- An application can have several SET PASSTHRU statements in effect at the same time to different data sources. Although the application might have issued multiple SET PASSTHRU SQL statements, the pass-through sessions are not truly nested. DataJoiner will not pass through one server to access another server. DataJoiner accesses each server directly. A SET PASSTHRU RESET ends all pass-through sessions. For example:

  ```
  CONNECT TO DJDB
  SET PASSTHRU LOC1
  INSERT INTO TAB1 VALUES ('A')
  SET PASSTHRU LOC2
  SELECT * FROM TAB2
  SET PASSTHRU RESET
  ```

  In this example, the application passes through to LOC2 from DJDB, not from LOC1. After SET PASSTHRU RESET, no pass-through sessions remain in effect.
- Host variables defined in SQL statements within a pass-through session must take the form :H*n* where H is uppercase and *n* is a unique whole number. The values of *n* must be numbered consecutively beginning with zero.
- You cannot use WHERE CURRENT OF conditions in UPDATE and DELETE statements within a pass-through session.
- You cannot pass through to more than one data source at a time.
- Pass-through does not support stored procedure calls.
- Pass-through does not support Classic Connect and CrossAccess data sources.

Microsoft SQL Server and Sybase: The following information applies to both Microsoft SQL Server and Sybase data sources:

- You cannot use user-defined transactions for Sybase in pass-through mode, because Sybase restricts which SQL statements can be specified within a user-defined transaction. Because SQL statements that are processed in pass-through mode are not parsed by DataJoiner, it is not possible to detect whether the user specified an SQL statement that is permitted within a user-defined transaction.
- The COMPUTE clause is not supported on Sybase data sources.
- DDL statements are not subject to transaction semantics on Sybase data sources. The operation, when complete, is automatically committed by Sybase. If a rollback occurs, the DDL is not rolled back.

- If you run system defined stored procedures in Sybase, no immediate errors are returned. However, the subsequent SELECT statement in pass-through mode returns SQL30081N, which maps to Sybase error 20019.

Oracle: The following information applies to Oracle data sources:

- Remote clients that issue SELECT statements with a CLP against an Oracle data source using DataJoiner pass-through functionality will receive an SQLCODE –30090 with reason code 11 if the client code is a DB2 SDK prior to UDB Version 5. In order for remote clients to use a CLP in pass-through mode to issue SELECT statements against an Oracle data source, the remote clients must use a DB2 SDK that is at Version 5 or greater or use the DataJoiner SDK.
- Any DDL statement issued against an Oracle server is performed at parse time and is not subject to transaction semantics. The operation, when complete, is automatically committed by Oracle. If a rollback occurs, the DDL is not rolled back.
- When you issue a SELECT statement from raw data types, use the RAWTOHEX function to receive the hexadecimal values. When you perform an INSERT into raw data types, provide the hexadecimal representation.

Error Codes

DataJoiner issues the same SQLCODE error codes as DB2. DataJoiner maps the SQL error codes returned from the data sources to DB2 SQLCODE error codes whenever possible. If no mapping is available, DataJoiner reflects the SQL error code from the data source using the SQLCODE 1822. If more than one error message is associated with a statement, DataJoiner propagates the first error received to the user.

Using Server Options to Configure Data Sources

| This section discusses using configuration specifications, called *server options*, to:

- Configure data sources to facilitate optimization and to enable certain kinds of processing, such as two-phase commit transactions
- Configure data sources to enable two-phase commit transactions among multiple databases
- Indicate when DataJoiner and data sources use the same collating sequence—a correspondence that helps to optimize queries. This section also indicates how you can use an API to create such a correspondence.

This section concludes by summarizing the server options and their settings.

Server Options

You can specify options that affect how DataJoiner operates on data sources and that facilitate optimization. Data sources are denoted by the keyword SERVER in DataJoiner-specific SQL; accordingly, these options are called *server options.*

For example, you can set the two_phase_commit option to allow applications to update databases in multiple data sources in a single transaction. As a prerequisite, the data sources must support this type of transaction. For data sources that don't support it, you can set the two_phase_commit option to exclude them from it. (For more information about multiple updates in a single transaction, see "Multi-Database Transactions" on page 13.)

You can set the fold_id option so that before DataJoiner sends an authorization name to a data source, DataJoiner transforms the name to the case (upper or lower) that the data source requires. Alternatively, if you define the name to DataJoiner in the required case, you can set the fold_id option to prevent transformation and its associated overhead.

There are three SQL statements for setting server options: CREATE SERVER OPTION, ALTER SERVER OPTION, and SET SERVER OPTION. Use the CREATE SERVER OPTION statement to set an option to a value that persists indefinitely over time for multiple connections to a data source. With this statement, you can set an option to a value other than the default or, if an option has no default value, you can set it to one of the valid values allowed by DataJoiner. (To find out what the default and other allowable values are, see Table 2 on page 17.)

For example, the default setting for the two_phase_commit option is 'n' (no, do not enable two-phase commit processing). Suppose that this option has never been used at a data source called GOSHEN, and that GOSHEN contains a couple tables, T1 and T2, for which nicknames have been defined. Then imagine that on a table in the DataJoiner database, you define a trigger that causes T1 and T2 to be updated whenever the DataJoiner table is updated. This two-database update requires a two-phase commit. So, for GOSHEN, you need to supersede the default with 'y' (yes, enable two-phase commit processing). Because you want the trigger to operate indefinitely for multiple connections to GOSHEN, you use the CREATE SERVER OPTION statement to supersede the default:

```
CREATE SERVER OPTION TWO_PHASE_COMMIT
  FOR SERVER GOSHEN
  SETTING 'y'
```

If, after defining an option setting with the CREATE SERVER OPTION STATEMENT, you later need to set the option differently for multiple connections over time, you can do so with the ALTER SERVER OPTION statement. For example, suppose the tables T1 and T2 at GOSHEN are dropped, ending the need for two-phase commit processing at that data source. Accordingly, you decide to reset two_phase_commit for GOSHEN to 'n'. Because you want this new setting to remain in effect for the foreseeable future, you define it with the ALTER SERVER OPTION statement:

```
ALTER SERVER OPTION TWO_PHASE_COMMIT
  FOR GOSHEN
  SETTING 'n'
```

Regardless of whether you set an option with the CREATE SERVER OPTION statement or the ALTER SERVER OPTION statement, you can, except in one case,[1] override the option for the duration of a single connection between an application and a database. You do this with the SET SERVER OPTION statement. For example, Oracle data sources do not process DDL in two-phase commit mode. So if an application needs to submit DDL for processing at an Oracle data source for which two_phase_commit is set to 'y', the 'y' would need to be changed to 'n' for the duration of the processing. You would set the 'n' with the SET SERVER OPTION statement. For example, if the name of the data source is SEER, you'd specify:

```
SET SERVER OPTION TWO_PHASE_COMMIT
  TO 'n'
  FOR SERVER SEER
```

To permanently remove a server option, use the DROP statement.

For the syntax of the SQL statements for using and removing server options, see "Chapter 4. DataJoiner SQL Statements" on page 53.

Multi-Database Transactions

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 and 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 transaction can involve one or more databases. A transaction that involves two or more databases is a distributed unit of work (DUOW). In DataJoiner, you can write applications to perform DUOWs that involve combinations of databases such as:

- The DataJoiner database and one or more databases in one or more data sources
- Multiple databases within a data source
- Multiple databases within multiple data sources

Applications that connect to a DataJoiner database must end units of work by issuing either a COMMIT or a ROLLBACK statement. The COMMIT statement makes permanent all changes made within the transaction, whereas 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 is issued implicitly. If an application ends abnormally while in the middle of a unit of work, the unit of work is automatically rolled back. When issued, a COMMIT or ROLLBACK cannot be stopped. A COMMIT makes all updates to the databases referenced in the transaction permanent.

1. The exception is that if two_phase_commit is set to 'd', it can't be overridden by SET SERVER OPTION.

In a nondistributed unit of work, or in a DUOW involving reading from one or more databases to update another database, each COMMIT is processed in one operation. Accordingly, the operation is called a *one-phase commit*. In a DUOW involving updates of multiple databases, the updates are committed in two phases: prepare and commit. During the prepare phase, DataJoiner polls data sources to ensure that they are ready to commit the transaction. In phase two, if all data sources voted that they commit the transaction, it is committed. If one or more data sources voted that they cannot commit the transaction, the updates are rolled back. Taken together, prepare and commit are called a *two-phase commit.*

Not every data source is capable of participating in a two-phase commit transaction. For example, DB2 for MVS Version 2.3 is a valid data source, but it cannot support two-phase commit. The Generic Access API also does not support two-phase commit. For a list of data sources that support two-phase commit, and for rules and restrictions that govern the use of one-phase and two-phase commits under DataJoiner, see the *DataJoiner Administration Supplement.*

In any DUOW application that you write for DataJoiner, you need to specify whether a two-phase commit is to be enforced or disallowed by each data source that the application acts on. (Be aware that it's possible to have a mixed-mode transaction, in which one-phase and two-phase data sources are involved in a single transaction request.) To specify the sort of commit you want, you set the two_phase_commit server option to the appropriate value in the CREATE SERVER OPTION, ALTER SERVER OPTION, or SET SERVER OPTION statement. For an overview of these statements, see "Server Options" on page 11. For their syntax, see "Chapter 4. DataJoiner SQL Statements" on page 53. For guidelines on deciding what value to assign to *option-value*, see the *DataJoiner Administration Supplement.*

Collating Sequences

You can help to optimize queries by configuring a DataJoiner database to use the same collating sequence that a data source uses. In general terms, a *collating sequence* is a defined ordering for character data that determines whether a particular character sorts higher, lower, or the same as another. This section explains:

- How collating sequences determine sort orders
- How to set your local collating sequence to optimize queries of sorted data
- How to indicate whether local and remote collating sequences are the same

How Collating Sequences Determine Sort Orders

A collating sequence determines the sort order of the characters in a coded character set. A *character set* is the aggregate of characters that are used in a computer system or programming language. In a *coded* character set, each character is assigned to a different number within the range of 0 to 255 (or the hexidecimal equivalent thereof). The numbers are called *code points*; the assignments of numbers to characters in a set are collectively called a *code page*.

In addition to being assigned to a character, a code point can be mapped to the character's position in a sort order. In technical terms, then, a collating sequence is the collective mapping of a character set's code points to the sort order positions of the set's characters. A character's position is represented by a number; this number is called the *weight* of the character. In the simplest collating sequence, called an *identity* sequence, the weights are identical to the code points.

The order in which data in a database is sorted depends on the collating sequence defined for the database. For example, suppose that database A uses the EBCDIC code page's default collating sequence and that database B uses the ASCII code page's default collating sequence. Sort orders at these two databases would differ, as shown in Figure 1.

```
SELECT.....
  ORDER BY COL2

EBCDIC-Based Sort        ASCII-Based Sort

COL2                     COL2
----                     ----
V1G                      7AB
Y2W                      V1G
7AB                      Y2W
```

Figure 1. Example of How a Sort Order in an EBCDIC-Based Sequence Differs from a Sort Order in an ASCII-Based Sequence

Similarly, character comparisons in a database depend on the collating sequence defined for that database. So if database A uses the EBCDIC code page's default collating sequence and database B uses the ASCII code page's default collating sequence, the results of character comparisons at the two databases would differ. Figure 2 illustrates the difference.

```
SELECT.....
  WHERE COL2 > 'TT3'

EBCDIC-Based Results     ASCII-Based Results

COL2                     COL2
----                     ----
TW4                      TW4
X72                      X72
39G
```

Figure 2. Example of How a Comparison of Characters in an EBCDIC-Based Sequence Differs from a Comparison of Characters in an ASCII-Based Sequence

Setting the Local Collating Sequence to Optimize Queries

When a query from DataJoiner requires sorting, the place where the sorting is processed depends on whether DataJoiner's collating sequence is the same as that of the data source where the queried data is stored. If the two collating sequences are the same, the sorting can be performed at the data source. If the query also requires a comparison of character data, this comparison can also be performed at the data source.

But if DataJoiner's and the data source's collating sequences differ, DataJoiner retrieves the data to its own database, so that it can do the sorting and comparison locally. The reason is that users expect to see the query results ordered according to the collating sequence defined for DataJoiner; by ordering the data locally, DataJoiner ensures that this expectation is fulfilled.

Retrieving data for local sorts and comparisons usually decreases performance. Therefore, we recommend that you configure the DataJoiner database to use the same collating sequences that your data sources use. That way, performance can be optimal, because DataJoiner can avoid the retrieval and allow the sorts and comparisons to take place at the data sources. If your data sources use different collating sequences, then configure the DataJoiner database to use the collating sequence used by the data source that you access most often, or by the data source whose performance is most critical to you.

You set the DataJoiner database's collating sequence as part of the CREATE DATABASE API. Through this API, you can specify one of the following sequences:

- An identity sequence
- A *system* sequence (the sequence used by the operating system that supports the database)
- A *customized* sequence (a predefined sequence that DB2 supplies or that you define yourself)

For example, in DB2 for OS/390, sorts defined by ORDER BY clauses are implemented by a collating sequence based on an EBCDIC code page. If you want to use DataJoiner to retrieve DB2 for OS/390 data sorted in accordance with ORDER BY clauses, it is advisable to configure the DataJoiner database so that it uses the predefined collating sequence based on the EBCDIC code page CCS 500. For more information about the CREATE DATABASE API, see the *API Reference*.

If the collating sequences at DataJoiner and the data source differ, and you do need to see the data ordered in the data source's sequence, you can submit your query in pass-through mode, or define the query in a remote view.

Indicating Whether the Local and Remote Collating Sequences Are the Same

If the DataJoiner database and a data source use the same collating sequence, set the colseq server option for the data source to 'y'. If the DataJoiner database uses a different collating sequence than a data source uses, set the colseq server option for the data source to ‘n’ or ‘i’. Specify ‘n’ if the data source’s collating sequence is case-sensitive; specify ‘i’ if this collating sequence is case-insensitive.

To set the colseq server option as a default for all sessions with a data source, use the CREATE SERVER OPTION or ALTER SERVER OPTION statement. To set this option to be in effect during a specific session with a data source, use the SET SERVER OPTION statement. For the syntax of these statements, see “Chapter 4. DataJoiner SQL Statements” on page 53.

Summary of Server Options and Their Settings

Table 2 describes the server options and the values that you can set them to.

Table 2. Server Options and Their Settings

Option	Valid Settings	Default Setting
colseq	Specifies whether the data source uses the same default collating sequence as DataJoiner, based on the code set and the country information: ’y’ Data source’s collating sequence is the same as DataJoiner’s. ’n’ Data source’s collating sequence is not the same as DataJoiner’s. ’i’ Data source’s collating sequence is different from DataJoiner’s and is case-insensitive (for example, ’TOLLESON’ and ’TolLESon’ are considered equal).	’n’
connectstring	Use this option to set timeout thresholds to interrupt queries that run for too long a period of time. For more information, see the *Planning, Installation, and Configuration Guide* for your platform. This field is case-sensitive.	None.
DATEFORMAT (See note 1.)	The date format used by the data source. Enter the format using 'DD', 'MM', and 'YY' or 'YYYY' to represent the numeric form of the date. You should also specify the delimiter such as a space or comma. For example, to represent the date format for '1994-01-01', use 'YYYY-MM-DD'. This field is nullable.	None.
deferred_lob_retrieval	Specifies whether the retrieval of LOB data is to be deferred until the data is assigned to a location in user space—a host variable or a file on disk. Deferring retrieval can substantially boost query performance while reducing network traffic. ’y’ Deferred LOB retrieval is valid for a LOB column. ’n’ Deferred LOB retrieval is not valid for a LOB column. 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’.	’n’

Table 2. Server Options and Their Settings (continued)

Option	Valid Settings	Default Setting
fold_id (See note 2.)	Specifies if and how user IDs are folded. Valid values are: 'u' DataJoiner folds the user ID to uppercase before sending it to the data source. (See note 3.) 'n' DataJoiner does nothing to the user ID before sending it to the data source. (See note 3.) 'l' DataJoiner folds the user ID to lowercase before sending it to the data source. If none of these settings are used, DataJoiner first tries to send the user ID to the data source as uppercase. If the user ID fails in uppercase, DataJoiner tries sending it as lowercase.	None.
fold_pw (See notes 2 and 4.)	Specifies if and how passwords are folded. Valid values are: 'u' DataJoiner folds the password to uppercase before sending it to the data source. 'n' DataJoiner does nothing to the password before sending it to the data source. 'l' DataJoiner folds the password to lowercase before sending it to the data source. If none of these settings are used, DataJoiner first tries to send the password to the data source as uppercase. If the password fails in uppercase, DataJoiner tries sending it as lowercase.	None.

Table 2. Server Options and Their Settings (continued)

Option	Valid Settings	Default Setting
ignore_udt	Specifies whether DataJoiner should determine the built-in type that underlies a UDT without strong typing. Applies only to data sources accessed through the ctlib and dblib protocols. Valid values are: ’y’ Ignore the fact that UDTs are user-defined and determine what built-in types under lie them. ’n’ Do not ignore user-defined specifications of UDTs. When DataJoiner creates nicknames, it looks for and catalogs information about the objects (tables, views, stored procedures) that the nicknames point to. As it looks for the information, it might find that some objects have data types that it doesn’t recognize (that is, data types that don’t map to counterparts at the DataJoiner database). Such unrecognizable types can include: • New built-in types • UDTs with strong typing • UDTs without strong typing; that is, built-in types that the user has simply renamed. These types are supported only by certain data sources, such as Sybase and Microsoft SQL Server. When DataJoiner encounters data types that it doesn’t recognize, it returns the error message, SQL3324N. However, it can make an exception to this practice. For data sources accessible through the ctlib or dblib protocols, you can set the ignore_udt server option so that when DataJoiner encounters an unrecognizable UDT without strong typing, DataJoiner determines what the UDT’s underlying built-in type is. Then, if DataJoiner recognizes this built-in type, DataJoiner returns information about the built-in type to the catalog. To have DataJoiner determine the underlying built-in types of UDTs that do not have strong typing, set ignore_udt to ’y’.	’n’
password	Specifies whether the password is to be validated at a data source. ’y’ Passwords are always to be sent to the data source and validated. ’n’ Password are not sent to the data source (regardless of any user mappings) and not validated.	’n’

Table 2. Server Options and Their Settings (continued)

Option	Valid Settings	Default Setting
plan_hints	Specifies whether *plan hints* are to be enabled. 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. 'y' Plan hints are to be enabled at the data source if the data source supports plan hints. 'n' Plan hints are not to be enabled at the data source.	'n'
pushdown	'y' DataJoiner will consider letting the remote data source evaluate as many operations as possible. 'n' DataJoiner will retrieve only columns from the remote data source and will not let the data source evaluate other operations, such as joins.	'y'
remote_query_caching	Specifies whether the performance enhancement of remote query caching should be considered: 'y' DataJoiner will consider caching the result for a remote query so that subsequent execution of the same query can reuse the same result. This reuse can potentially save resources on the network and the remote data sources. 'n' DataJoiner will not cache the result for a remote query.	'y'
TIMEFORMAT (See note 1.)	The time format used by the data source. Enter the format using 'hh12', 'hh24', 'mm', 'ss', 'AM', or 'A.M'. For example, to represent the time format of '16:00:00', use 'hh24:mm:ss'. To represent the time format of '8:00:00 AM', use 'hh12:mm:ss AM'. This field is nullable.	None.
TIMESTAMPFORMAT (See note 1.)	The timestamp format used by the data source. The format follows that for date and time, plus 'n' for tenth of a second, 'nn' for hundredth of a second, 'nnn' for milliseconds, and so on, up to 'nnnnnn' for microseconds. For example, to represent the timestamp format of '1994-01-01-24:00:00.000000', use 'YYYY-MM-DD-hh24:mm:ss.nnnnnn'. This field is nullable.	None.

Table 2. Server Options and Their Settings (continued)

Option	Valid Settings	Default Setting
two_phase_commit	Specifies whether DataJoiner should participate with data sources in distributed, two-phase commit transactions. Valid values are: 'y' DataJoiner is to participate with the specified data source (or data sources) in two-phase commit transactions. 'n' DataJoiner is not to participate with the specified data source (or data sources) in two-phase commit transactions. Exception: If two_phase_commit is set to 'n' for a data source in the CREATE SERVER OPTION or ALTER SERVER OPTION statement and to 'y' for the same server in the SET SERVER OPTION statement, the 'y' overrides the 'n' when the SET SERVER OPTION statement is processed. 'd' In the CREATE SERVER OPTION and ALTER SERVER OPTION statement, 'd' (disabled) means: Do not participate in two-phase commit and do not allow the SET SERVER OPTION statement to override this option. If two_phase_commit is set to 'd' for a server in the CREATE SERVER OPTION or ALTER SERVER OPTION statement and to 'y' for the same server in the SET SERVER OPTION statement, an error is generated when the SET SERVER OPTION statement is processed. In the SET SERVER OPTION statement, 'd' means the same as 'n': Do not participate in two-phase commit.	'n'
varchar_no_trailing_blanks	Indicates whether trailing blanks are absent from all VARCHAR and VARCHAR2 columns: 'y' None of the VARCHAR columns at this data source have trailing blanks. 'n' One or more VARCHAR columns at this data source have trailing blanks. The optimizer's strategy for accessing Oracle data sources depends in part on whether there are trailing blanks in VARCHAR and VARCHAR2 columns of tables residing at the data sources. By default, the optimizer "assumes" that all Oracle VARCHAR and VARCHAR2 columns have trailing blanks. On this assumption, it develops an access strategy that involves modifying queries so that the values returned from these columns are the ones that the user expects. If, however, a VARCHAR or VARCHAR2 column has no trailing blanks, and you let the optimizer know this, it can develop a more efficient access strategy. To tell the optimizer that the VARCHAR and VARCHAR2 columns at an Oracle data source have *no* trailing blanks, set the varchar_no_trailing_blanks option to 'y' for that data source. To tell the optimizer that a specific column has no trailing blanks, specify that column in the ALTER NICKNAME statement (for guidelines, see "ALTER NICKNAME" on page 56).	'n'

Notes on Table 2 on page 17:

1. This option is used only when the value of SERVER_TYPE is GENERIC. This option is ignored for all other values of SERVER_TYPE.
2. This field is applied regardless of the SECURITY specification.
3. Because DataJoiner stores user IDs in uppercase, the values ‘n’ and ‘u’ are logically equivalent to each other.
4. The setting for fold_pw has no effect when the setting for password is ‘n’. Because no password is sent, case cannot be a factor.

Chapter 2. Referencing and Manipulating Database Objects

This chapter explains how you can use DataJoiner to reference and manipulate database objects. The first section focuses on:

- Defining and using data source tables
- Invoking data source stored procedures
- Accessing DataJoiner catalog views

The second section focuses on:

- Enabling DataJoiner to access user-defined functions (UDFs) and to recognize user-defined data types (UDTs)
- Retrieving and manipulating large objects (LOBs)

Working with Tables, Views, and Stored Procedures

This section provides information to help you access and use tables, views, and stored procedures. The section begins by explaining:

- Nicknames that you create so that DataJoiner can reference tables, views, and stored procedures
- Mappings between data types defined to DataJoiner and data types included in the definitions of data source tables, views, and stored procedures

Next comes information that pertains mainly to tables:

- How you can use DataJoiner to create, alter, and delete data source tables
- What referential integrity is; how DataJoiner enforces it locally, but not for data sources
- How you can use indexes to facilitate access to DataJoiner and to data source tables
- Requirements for tables accessed through the dblib protocol

This section also explains:

- What catalog data you can access and update from views
- How you can invoke stored procedures from DataJoiner

Nicknames

When a client connects to DataJoiner, that client issues SQL requests intended for data sources. A simple way to recognize requests intended for data sources is to require three-part names consisting of location, qualifier, and table name; however, this requirement provides no location transparency or independence to the client or application program.

DataJoiner uses nicknames to map two- and three-part table and view names to data sources. In the same way, it uses nicknames to map two- and three-part stored procedure names to data sources. Nicknames are not alternate names for tables, views, and stored procedures; rather, they are pointers by which DataJoiner references these objects.

The name of every remote table, view, or stored procedure that is referenced must have an associated nickname; however, you can reference anything explicitly if you establish a pass-through session.

When a nickname is created for a table at a data source, DataJoiner updates the DataJoiner catalog with data pertaining to the table. This data includes column definitions; you can access them from the catalog view, SYSCAT.COLUMNS. If the table has an index, the definition of this index might also be cataloged. If it is, you can access it from the view, SYSCAT.INDEXES.

Not all data about a table can be cataloged when a nickname is created for the table. However, there are ways to obtain or make up for missing data. To supply missing statistics, run the RUNSTATS utility against the nickname. If information about indexes is missing, you can compensate for this lack by creating a local definition of an index (not an actual index) for the table. You create this definition with the DataJoiner CREATE INDEX statement. For more information about obtaining statistics and index information for the catalog, see the *DataJoiner Administration 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. See the *DataJoiner Administration Supplement* for more information about nickname privileges.
- 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.

Using Nicknames with Views

You can create a nickname for a view on a data source, define views on nicknames for remote tables and views, and manipulate all such views. DataJoiner treats the nickname for a remote view the same way it treats the nickname for a remote table.

Views do not have statistics or indexes of their own because they are not actual tables located in a database. This statement is true even when a view is identical in structure and content to a single base table. For more information about statistics and indexes, see *DATABASE 2 Administration Guide.*

Because DataJoiner can accommodate a join of base tables at different locations, you can easily define global views from base tables that reside at different data sources. Multi-location views offer a high degree of data independence for a globally integrated database, just as views defined on multiple local tables do for centralized relational database managers. This global view mechanism is one way that DataJoiner offers a high degree of data independence.

When you modify data through views, be aware of the data source's rules for updates. For example, some types of data sources currently do not allow updates through views if the UPDATE statement references more than one table in the top subselect. These rules can change from release to release.

Data Type Mappings

DataJoiner uses mappings between data source data types and data types defined in the DataJoiner database to determine:

- What data types to define locally (that is, to the DataJoiner database) for columns of data source tables and views
- What data types to define at the data source for columns of data source tables created from DataJoiner
- What values are to be returned when you:
  - Query a data source table or view
  - Invoke a data source function or stored procedure

This section explains how DataJoiner makes these determinations and how you can override default data type mappings or create new ones.

How DataJoiner Determines What Data Types to Define Locally

When you create a nickname for a table, DataJoiner adds information about this table to the DataJoiner catalog. This information includes, but isn't limited to, the nickname, the table's name, all column names and, for each column:

- The data type that was defined for the column at the data source
- A corresponding data type that's supported by the DataJoiner RDBMS (DB2 for CS).

This section refers to the first data type as a remote type and to the second as a local, or locally-defined, type.

How does DataJoiner determine what local type to use for a remote column? It consults a pre-existing mapping between the column's type at the data source and a comparable local type, and chooses the latter. For example, in a default mapping supplied by DataJoiner, an Informix data type CHAR, which supports up to 254 bytes, points to the DB2 for CS data type CHARACTER. So if you're creating a nickname for an Informix table, and column C1 of the table has a data type of CHAR with a maximum length of 200, then, unless you override the default, C1's locally-defined type will be CHARACTER. A remote-to-local data type mapping from which DataJoiner determines what local type to use for a remote column is called a *forward type mapping.*

How DataJoiner Determines What Data Types to Define at Data Sources

From DataJoiner, you can use DataJoiner's CREATE TABLE statement to define data source tables. In pre-existing local-to-remote data type mappings, the types that you specify in this statement point to corresponding data source data types. These corresponding data types will be defined at the data source for the columns of the tables that you create with this statement.

For example, the DB2 for CS data type DOUBLE maps by default to the Oracle type FLOAT. So if you're creating an Oracle table from DataJoiner, and your CREATE TABLE statement specifies DOUBLE for column C1, then, unless you override the default, C1's data type at the data source will be FLOAT. Data type mappings used to determine what data source type to use for a data source column are called *reverse type mappings.*

How Type Mappings Determine What Values Are Returned

When you query a column of a data source table from DataJoiner, DataJoiner returns values that are common to this column's data type and its associated data type—that is, the one it maps to—at the DataJoiner database. Similarly, when you use DataJoiner to invoke a data source function that has an input or output parameter, or to invoke a stored procedure that has input, output, or result set parameters, DataJoiner returns values that are common to the parameters' data types and these data types' associated counterparts at the DataJoiner database.

Because of differences between RDBMSs, a mapping between a data source and DataJoiner data type usually isn't 1-to-1. However, the mapping can be close enough to ensure that all requested values are returned.

For example, there's a default forward type mapping between:

- The Oracle type NUMBER(9,0) (where 9 is the maximum precision and 0 the maximum scale)
- The DB2 for CS type INTEGER, with a maximum length of 4 bytes

Suppose that you create a nickname for an Oracle table that has a column C2 with a type of NUMBER(9,0). If you don't change the default mapping, the type for C2 will be locally defined as INTEGER. And because the 4 bytes of INTEGER support a maximum precision of 10, you can be sure that all values of C2 will be returned when C2 is queried from DataJoiner.

Similarly, there are default reverse type mappings between:

- DB2 for CS type TIMESTAMP and Sybase type datetime
- DB2 for CS type DATE and Sybase type datetime
- DB2 for CS type TIME and Sybase type datetime

TIMESTAMP and datetime are for time stamps; DATE is for dates; TIME is for times. Suppose that you're creating a Sybase table from DataJoiner, and that your CREATE TABLE statement specifies TIMESTAMP for column C3. If you don't change the default mapping, C3's data type at the data source will be datetime. Because time stamps are common to both TIMESTAMP and datetime, you can be sure that all timestamps in C3 will be returned when C3 is queried from DataJoiner.

If the CREATE TABLE statement specifies DATE for C3, C3's type at the data source will be datetime, and values common to DATE and datetime—that is, dates—will be returned for queries from DataJoiner. If the statement specifies TIME, C3’s type at the data source will still be datetime, and values common to TIME and datetime—that is, times—will be returned.

Overriding Type Mappings and Creating New Ones

Default forward and reverse type mappings are shipped with DataJoiner. As the preceding examples indicate, the local type and remote type in a default mapping are similar enough to ensure that when you query remote columns for which the remote type is defined, all values that conform to both types will be returned. But sometimes, you might require an alternative mapping. Consider these scenarios:

Defining a Forward Type Mapping That Applies to One or More Data Sources: An Oracle table T1 has a column C4 with a data type DATE, for time stamps. In a default forward type mapping, this type points to the local DB2 for CS type TIMESTAMP. So if you were to create a nickname for T1 without changing the default, TIMESTAMP would be defined locally for C4, and queries of C4 from DataJoiner would yield time stamps. But suppose that you want queries of C4 to yield times only. You could then map Oracle DATE to DB2 for CS TIME, overriding the default. That way, when you create the nickname, TIME, not TIMESTAMP, is defined locally for C4. As a result, when you subsequently query C4 from DataJoiner, only the time portion of the time stamps in C4 is returned.

To override or create a forward type mapping for data source tables and views that you want to define to DataJoiner, use the CREATE TYPE MAPPING statement. In this statement, you indicate whether the mapping is to apply to a specific data source (for example, a data source that a department in your organization uses) or to all data sources of a specific version or type (for example, all of your organization’s Informix 7.2 data sources, or all of its Informix data sources, regardless of version).

Defining a Reverse Type Mapping That Applies to One or More Data Sources: You want to use DataJoiner to create a UDB table with a column, C5, for dollar amounts. Because Oracle has no data type for monetary units, you create one at the UDB data source, and call it DOLLAR. And because there has to be a reverse type mapping between DOLLAR and a local counterpart before DOLLAR can be defined for C5, you create both the local counterpart, calling it DB2MONEY, and the required mapping. Thus, when you code the CREATE TABLE statement, you can specify DB2MONEY as the data type for C5, knowing that, because of the mapping, DOLLAR will be defined for C5 at the data source.

To override or create a reverse type mapping for data source tables that you want to create from DataJoiner, use the CREATE REVERSE MAPPING statement. In this statement, you indicate whether the mapping is to apply to a specific data source (for example, a data source that contains your organization’s payroll data) or to all data sources of a specific version or type (for example, all of your organization’s Oracle 8.0.3 data sources, or all of its Oracle data sources, regardless of version).

Changing a Type Mapping for a Specific Table: You can change the local type in a forward or reverse type mapping for a specific table. For example, Oracle

NUMBER(32,3) maps by default to DB2 for CS DOUBLE, a floating decimal data type. Suppose that in an Oracle table for employee information, a column BONUS was defined with a data type of NUMBER(32,3). Because of the mapping, a query of BONUS could return values that look like this:

```
5.0000000000000E+002
1.0000000000000E+003
```

where +002 signifies that the decimal point should be moved two places to the right, and +003 signifies that the decimal point should be moved three places to the right.

So that queries of BONUS can return values that look like dollar amounts, you could, for this particular table, remap NUMBER(32,3) to DB2 for CS DECIMAL(6,2). Under the constraint of this new mapping, a query of BONUS would return values like this:

```
000500.00
001000.00
```

To change the type mapping for a column of a specific table, use the ALTER NICKNAME statement (described in “ALTER NICKNAME” on page 56). With this statement, you can change the type defined locally for a column of a table for which a nickname has been defined.

Data Source Tables That You Create from DataJoiner

You can use the DataJoiner SQL statements CREATE TABLE, ALTER TABLE, and DROP to create, alter, and delete tables at these types of data sources: DB2 Family, Generic, Informix, Microsoft SQL Server, Oracle, SQL Anywhere, and Sybase. This section summarizes what you do to:

- Create a data source table and its nickname in one operation
- Alter this table
- Delete this table and its nickname in one operation

Creating Data Source Tables from DataJoiner

This section provides overviews of:

- The task of using DataJoiner to create data source tables and their nicknames
- The role that local-to-remote type mappings play in this task
- The steps for combining this task with the creation of any type mappings that you need

Creating Tables and Their Nicknames: With the DataJoiner CREATE TABLE statement, you can define a data source table’s name, columns, data types, and primary keys. You can also specify an option that’s specific to the data source. For example, if you’re creating a DB2 for OS/390 table, you can specify the IN DATABASE option, which is specific to DB2 for OS/390 tables.

When you run the CREATE TABLE statement, DataJoiner automatically creates a nickname for the table that you're creating. The nickname is the same as the table's name.

To give your table attributes that are supported by the data source but not by the DataJoiner CREATE TABLE statement—for example, foreign keys—open a pass-through session to access the table after it's created. Then, in this session, use the data source's native SQL to define the attributes that you want. If you also want the table to have an index at this point, you can create this index in the pass-through session as well. For information about pass-through, see "Pass-Through Sessions for Querying Data Sources in Their Own SQL" on page 8.

The DataJoiner CREATE TABLE statement is a specialized version of the DB2 for CS CREATE TABLE statement. For the syntax of the DataJoiner statement, see "CREATE TABLE" on page 107. For the combined syntax of the DataJoiner and DB2 for CS CREATE TABLE statements, see page 167.

Using Data Type Mappings to Set Data Types: DataJoiner supplies default mappings between the DB2 for CS data types that you can specify in the DataJoiner CREATE TABLE statement and comparable data types at the data sources. For example, DB2 for CS DOUBLE maps by default to Oracle FLOAT, and DB2 for CS CLOB maps by default to Informix TEXT. Such mappings are called *reverse type mappings.*

When, in the CREATE TABLE statement, you specify a data type for a column of a data source table, this type is included in DataJoiner's local definition of the column. In addition, the type that it maps to is defined for the column at the data source. For example, if you specify DOUBLE for column C1 of an Oracle table, and you don't override the default mapping between DOUBLE and FLOAT, then DOUBLE is defined for C1 locally and FLOAT is defined for C1 at the data source.

Before you run the CREATE TABLE statement, the data types that you specify in the statement must map to the data types that you want defined at the data source. So if there is no reverse mapping between a data type that you want to define at the data source and a DB2 for CS type, then you need to create such a mapping. You do this with the CREATE REVERSE TYPE MAPPING statement (described in "CREATE REVERSE TYPE MAPPING" on page 89).

For more information about reverse type mappings, see "Data Type Mappings" on page 26.

Steps for Creating Data Source Tables and Associated Type Mappings: To create a data source table and to provide type mappings, as needed, for it:

1. Determine whether there are reverse mappings between data types that you want defined at the data source and the data types that you can specify in DataJoiner's CREATE TABLE statement. To do this, you might check "Appendix B. Default Reverse Type Mappings" on page 155. Or you might query the SYSCAT.REVTYPEMAPPINGS view to obtain listings of all reverse mappings, both

default and non-default. For information about this view, see “SYSCAT.REVTYPEMAPPINGS” on page 184.

2. If you don’t find mappings that you need, specify them with the CREATE REVERSE TYPE MAPPING statement.
3. Code the CREATE TABLE statement so that it defines the table and specifies the data source. For guidelines, see page 107.
4. Run any CREATE REVERSE TYPE MAPPING statements that you coded. Then run the CREATE TABLE statement.

Altering Tables

You can use the DataJoiner ALTER TABLE statement to add columns and primary keys to a data source table that was created with the DataJoiner CREATE TABLE statement. Any column that you select for a primary key cannot be nullable. If you want to make additional changes to the table—for example, to drop a primary key or add a foreign key—you can do so in a pass-through session with the SQL that’s native to the table’s data source.

For the syntax of the DataJoiner ALTER TABLE statement, see “ALTER TABLE” on page 65.

Deleting Tables

You can use the DataJoiner DROP statement to delete a data source table that was created with the DataJoiner CREATE TABLE statement. The DROP statement also deletes the table’s nickname. For the syntax of this statement, see“DROP” on page 121.

Referential Integrity

Referential integrity is the state of a database in which all values of all foreign keys are valid. A *foreign key* is a key that is part of the definition of a referential constraint. A *referential constraint* is the rule that the values of the foreign key are valid only if they also appear as values of a primary key. The table containing the primary key is called the *parent table* of the referential constraint, and the table containing the foreign key is said to be a dependent of that table.

The *insert rule* of a referential constraint is that a non-null insert value of the foreign key must match some value of the primary key of the parent table. The *update rule* of a referential constraint is that a non-null update value of the foreign key must match some value of the primary key of the parent table. The value of a composite foreign key is null if any component of the value is null.

DataJoiner enforces referential constraints between tables within the DataJoiner database. It doesn’t enforce referential constraints between tables in the same data source, or tables in different data sources. DataJoiner does not compensate for referential integrity differences between data sources, nor does DataJoiner interfere with referential integrity enforcement at the data sources. However, referential integrity constraints at a data source can affect DataJoiner processing. For example, if an insert into a table at a data source violates a referential integrity constraint at that data source, DataJoiner maps the resulting data source error to a DataJoiner error. Referential integrity between data sources is the responsibility of the applications.

Indexes

An *index* is an ordered set of identifiers that is separate from the data in the table and identifying rows of a base table. A database manager builds indexes and uses them to expedite access to the data. Because indexes dramatically affect the performance of queries that are able to exploit them, DataJoiner needs accurate information about indexes. When a CREATE NICKNAME statement is run, DataJoiner tries to determine what indexes exist on the table to which the nickname being created refers. If DataJoiner finds any indexes, it records their characteristics in the DataJoiner catalog.

There are environments in which DataJoiner cannot determine whether an index exists on a table for which a nickname has been defined. In these situations, you can code a definition of an index in the DataJoiner CREATE INDEX statement, and then run this statement against the table's name or nickname. This action doesn't create an actual index at the data source. Rather, it populates the DataJoiner catalog with index information that helps DataJoiner to optimize queries of the table. To decide whether to issue CREATE INDEX for a table, consult the SYSCAT.INDEXES catalog view. If the table or its nickname isn't listed in the TABNAME column, you can conclude that DataJoiner either has determined that the table has no index, or has not determined whether the table has an index. For more information about CREATE INDEX, see "CREATE INDEX" on page 84 and the *DataJoiner Administration Supplement* .

Just as you can issue a CREATE INDEX statement against a table's nickname to create a local index definition for the table, so can you issue a DROP statement against the nickname to remove this definition.

Tables Accessed through the dblib Protocol

If you use dblib as the protocol for a positioned update or delete against a Sybase or Microsoft SQL Server table, or for an operation in which DataJoiner issues a positioned update or delete against a Sybase or Microsoft SQL Server table, the table must have:

- A unique index known to DataJoiner
- A column with a data type of timestamp

System Catalog Views

This section introduces you to DataJoiner's system catalog views and explains how you can update two of them to improve performance.

Introduction to DataJoiner's Catalog Views

DataJoiner maintains a set of system catalog views for each database created. Some of these views are identical to the catalog views maintained by DB2 for its databases, some are DB2 catalog views that were modified for use by DataJoiner, and some are unique to DataJoiner. The ones that are unique to DataJoiner contain information used to communicate with data sources. All system catalog views are created when you run the CREATE DATABASE statement.

When you run DataJoiner DDL (for example, statements to define data sources to DataJoiner and to create data source tables), the values specified in the DDL are added to the system tables from which the catalog views are derived. Then, with the proper authorization, you can access these values from the views.

DataJoiner ensures that the catalog contains accurate descriptions of the objects in the database with the exception of objects associated with nicknames. DataJoiner cannot ensure the accuracy of objects associated with nicknames because DataJoiner is not informed when an object changes at a remote data source.

For information about catalog views that are identical in DataJoiner and DB2, see the *DATABASE 2 SQL Reference for common servers*. For information about views that were modified specifically for DataJoiner Version 2.1.1, see “Changes in DB2 for CS Views That Support the Spatial Extender” on page 203. For complete information about the other DB2 catalog views that were modified for DataJoiner, and about the views that are unique to DataJoiner, see “Appendix E. System Catalog Views” on page 175.

Updating Catalog Views to Improve Performance

You can use two of the catalog views—SYSSTAT.SERVERS and SYSSTAT.SERVER_FUNCTIONS—not only to access values, but also to update them. These changes are propagated to the tables that underlie the views.

The values that you can update are statistics that the DataJoiner optimizer can use to develop access plans. To illustrate: When you code DDL (the CREATE SERVER MAPPING statement) to define a data source to DataJoiner, you can specify certain statistics—for example, a number denoting how much faster or slower the data source CPU is compared to the DataJoiner CPU, or a number denoting how the data source I/O device rate compares to the DataJoiner I/O device rate. When you run this DDL, these statistics are included in two catalog views: SYSCAT.SERVERS and SYSSTAT.SERVERS. If the statistics change (as might happen, for instance, if the data source CPU is upgraded), you can update SYSSTAT.SERVERS with the change. The optimizer then uses your update in developing its next access plan for the data source.

Similarly, the DDL for mapping definitions of local functions to data source functions—the CREATE FUNCTION MAPPING statement—can include such statistics as the estimated number of I/Os and instructions per invocation of a data source function. When you run the CREATE FUNCTION MAPPING statement, these statistics are added to the SYSCAT.SERVER_FUNCTIONS and SYSSTAT.SERVER_FUNCTIONS views. If the statistics change, you can update SYSSTAT.SERVER_FUNCTIONS accordingly, so that the optimizer can make a corresponding adjustment in its next access plan for the data source in question.

Stored Procedures

You can use DataJoiner to invoke stored procedures from DB2 for OS/390, Informix, Microsoft SQL Server, Oracle, and Sybase data sources. A *stored procedure* is a reusable block of code that performs a particular task. The language of the code depends on the data source where the stored procedure resides. For example, a stored procedure in a DB2 data source can include SQL statements as well as statements in other languages, such as C++, COBOL, and FORTRAN.

You can use stored procedures to perform tasks that need to be done often; for example, receiving a fixed set of data, performing the same multiple requests against a database, or returning a fixed set of data. Applications that process huge amounts of data at a server site, but return smaller amounts of data as the result of the processing, are strong candidates for being written as stored procedures. As stored procedures, they would allow for the amount of data shipped between client and server to be considerably reduced.

This section tells you:

- The format of data returned by stored procedures
- What SQL statements you issue to invoke stored procedures from DataJoiner
- Factors and restrictions to consider when you invoke stored procedures from DataJoiner

Data Returned by Stored Procedures

A stored procedure can be coded to return either single values or tuples of values called result sets. *Tuple* is a relational database term that refers to a row or record of data; a *result set* can consist of one or more tuples.

A stored procedure might return *uniform* result sets—that is, consecutive result sets that have the same number of values and the same data types; or *non-uniform* result sets—that is, consecutive result sets that have different numbers of values and different data types. For example, if stored procedure X returns two result sets with one value each and a data type of CHAR(1), then these result sets are referred to as uniform result sets. If stored procedure Y returns a result set with one value and a data type of INTEGER followed by a result set with two values and the data types of INTEGER and CHAR(1), these result sets are referred to as non-uniform result sets.

SQL for Invoking Stored Procedures from DataJoiner

To invoke a stored procedure at a data source from DataJoiner, an application must first issue the CREATE STORED PROCEDURE NICKNAME statement. Through this statement, DataJoiner associates a nickname with the stored procedure. The application then issues a CALL statement that references the stored procedure by its nickname.

If the stored procedure is coded to return a result set, the application must issue an ALLOCATE CURSOR statement. This statement associates a cursor with the stored procedure. A *cursor* is a control structure that is used to retrieve rows from an ordered set of rows, such as a result set.

After the stored procedure is executed and produces the result set, the application has the option of issuing a DESCRIBE CURSOR statement to obtain a description of the result set. Next, the application issues a FETCH statement. This statement causes the cursor to point to the rows of the result set; as it points to each row, that row is returned to the calling application.

When the application finishes processing the result set, it issues the CLOSE statement. The SQLCODE associated with this statement informs the application whether more result sets exist. If any more do exist, the application can issue the DESCRIBE CURSOR, FETCH, and CLOSE statements against each one. When the SQLCODE associated with the CLOSE statement indicates that no more result sets exist (SQLCODE 0), the cursor is closed.

For the syntax of the CALL, ALLOCATE CURSOR, and DESCRIBE CURSOR statements, see “Chapter 4. DataJoiner SQL Statements” on page 53. For examples of using these statements together, see “Notes” on page 71 and “Appendix D. Sample Program Fragment for Invoking a Stored Procedure” on page 173.

Considerations and Restrictions

This section discusses factors to consider and restrictions to observe when you plan to use data source stored procedures that yield result sets.

General Information: When you write code to retrieve result sets from a stored procedure at a data source, be aware that:

- The CALL statement returns an SQLCODE +466 if one or more result sets are associated with the stored procedure.
- Unlike a conventional cursor, a cursor associated with a stored procedure does not get opened. The reason is that the stored procedure at the data source does the processing that makes it possible for rows of data to be returned to the application.Therefore, if you issue the ALLOCATE CURSOR statement to associate a cursor with a stored procedure, you should not issue the OPEN statement. If you do, an SQLCODE will be returned. If you issue the OPEN statement after issuing a DESCRIBE CURSOR statement, this SQLCODE will be -502 (the cursor specified in an OPEN statement is already open). If you issue the OPEN statement without issuing a DESCRIBE CURSOR statement beforehand, the SQLCODE will be -517 (the cursor identifies a prepared statement that is not a SELECT or VALUES statement).
- When an application finishes processing a result set, the application can issue a CLOSE statement. If this statement returns an SQLCODE of +467, then another

result set needs to be returned to the application. The application can then process this result set by issuing a DESCRIBE CURSOR, FETCH, or CLOSE statement against the cursor.

- Define only one cursor to process all result sets—uniform or non-uniform—associated with a stored procedure.
- When a stored procedure is executed against a DRDA or DB2RA data source, the CALL statement might return an error indicating that a package is not found. If this error occurs, bind the package `sqllib/bnd/db2cliv2.bnd` against the data source using the SQLERROR CONTINUE bind option. Then, retry executing the stored procedure.
- Stored procedures currently do not support LOBs as input or output parameters or in result sets.
- For input parameters, output parameters, and result sets, the DataJoiner default data types for the data source are used.

| ***DB2 for OS/390 Stored Procedures:*** DataJoiner supports DB2 for OS/390 stored procedures that return output parameters only; it doesn't support DB2 for OS/390 stored procedures that return output parameters and result sets, or result sets only.

| ***Informix Stored Procedures:*** Informix stored procedures can return uniform result sets; they can't return non-uniform result sets.

| ***Microsoft SQL Server Stored Procedures:*** When you write code to invoke Microsoft SQL Server stored procedures, be aware that you can access them from DataJoiner in either of two ways:

- | Through the dblib protocol. If a stored procedure accessed in this way returns result sets, it cannot have output parameters.
- | By using an ODBC driver.

| When accessed through the dblib protocol, Microsoft SQL Server stored procedures can return uniform as well as non-uniform result sets. When accessed under ODBC, they can return non-uniform result sets but not uniform ones.

| ***Oracle Stored Procedures:*** An Oracle stored procedure is written as a PL/SQL procedure. In order for a DataJoiner application to access an Oracle stored procedure for which a nickname has been defined, the stored procedure should meet the following requirements:

- | It should fetch each record in its result set and fill in the Oracle array interface, one record at a time.
- The size of the Oracle array interface should be 1.
- | Each stored procedure must check for the end-of-data condition. In addition, whenever end-of-data is reached, the stored procedure must, through one of its parameters, return end-of-data information to the application. Therefore, after the application performs each fetch call, the application should check this parameter to determine whether end-of-data was reached (end-of-data is inferred through this parameter). In particular, the application should *not* expect the usual SQLCODE of

100 to indicate end-of-data for any form of data (that is, for input, output, or result set parameters).

When you write code to invoke Oracle stored procedures that can return result sets, be aware that:

- Oracle stored procedures can return uniform result sets; they can't return non-uniform result sets.
- In your application, all input, output, and result set parameters of an Oracle stored procedure should be specified in the CREATE STORED PROCEDURE NICKNAME statement as well as in the CALL statement. This distinction is unique to Oracle stored procedures (as opposed to stored procedures defined in other types of data sources, such as Sybase or Microsoft SQL Server).
- The maximum number of rows that can be obtained from an Oracle stored procedure call is currently fixed at 10,000.

Sybase Stored Procedures: When you write code to invoke Sybase stored procedures, be aware that:

- They can be accessed from DataJoiner only through the dblib protocol.
- They can return uniform as well as non-uniform result sets.
- If they return result sets, they cannot also return output parameters.

Working with User-Defined Functions, User-Defined Types, and Large Objects

This section provides information to help you access and use user-defined functions, user-defined data types, and large objects. The section explains:

- What user-defined functions are; how you can enable DataJoiner to access them and new built-in functions at data sources
- What user-defined data types are; how you can enable DataJoiner to recognize them at data sources
- How you can retrieve and manipulate large objects that reside at data sources

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. For information to help you set up the session, see “Pass-Through Sessions for Querying Data Sources in Their Own SQL” on page 8 and “SET PASSTHRU” on page 129.
- 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. For information about these statements, see "CREATE FUNCTION" on page 79 and "CREATE FUNCTION MAPPING" on page 80.

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. For information about the CREATE TYPE MAPPING statement, see “CREATE TYPE MAPPING” on page 112.

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 “SET SERVER OPTION” on page 132 and “CREATE SERVER OPTION” on page 100.

  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” on page 43). 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.

Chapter 3. Specifying Identifiers

An *identifier* is a token that is used to form a name. This chapter describes:

- The types, characteristics, and limits of the identifiers that users can specify in DataJoiner SQL statements
- How to ensure that case-sensitive identifiers and passwords are specified correctly at the data source
- How to find node identifiers that you need to specify in DataJoiner SQL statements

Identifiers Used By DataJoiner

In DataJoiner, you can use the same types of identifiers that you use in DB2 for CS. In addition, you need to use identifiers that are specific to DataJoiner. This section discusses:

- The two basic kinds of identifiers: SQL and host
- Characteristics of identifiers in DataJoiner SQL statements
- Limits that DataJoiner imposes on length of identifiers

For complete information about identifiers that are common to DataJoiner and DB2 for CS, see *DATABASE 2 SQL Reference for common servers*.

SQL and Host Identifiers

An identifier in an SQL statement is either an SQL identifier or a host identifier.

SQL Identifiers

The three types of SQL identifiers are:

Ordinary identifier
A letter followed by zero or more characters, each of which is an uppercase letter, a digit, or the underscore character. An ordinary identifier must not be identical to a reserved word. Examples might be `WKLYSAL` and `WKLY_SAL`. See the *DATABASE 2 SQL Reference* for a list of SQL reserved words.

Delimited identifier
A sequence of one or more characters enclosed within quotation marks ("). When you define an identifier—for example, when you define the name of a table that you're creating—you need to delimit it if it violates the guidelines for an ordinary identifier. Examples are reserved words, words in lowercase, and terms enclosed by quotation marks. Thus, to designate the reserved word UNION as a name, you'd specify it as `"UNION"`. To define wkly_sal as a name,

specify it as `"wkly_sal"`; and to define “Mickey” as a name, specify it as `""Mickey""`.

Remote identifier
A sequence of one or more characters that is used in several statements to identify a data source or an object at a data source. For example, the CREATE NICKNAME statement includes a parameter, *remote-object-name*, which can consist of either two or three parts. In the statement, you assign a remote identifier to each part. For example, if *remote-object-name* consists of *remote-table-name* and *data-source-name*, then you assign the name of the table for which you’re creating a nickname to *remote-table-name*, and you assign the name of the data source where the table resides to *data-source-name*.

Ordinary and delimited identifiers are also classified according to their maximum length. A *long identifier* has a maximum length of 18 bytes. A *short identifier* has a maximum length of 8 bytes. These limits do not include the quotation marks surrounding the delimited identifier. Remote identifiers are limited to 128 characters or the maximum for the data source.

Host Identifiers

A *host identifier* is a name declared in the host program. The rules for forming a host identifier are the rules of the host language. A host identifier cannot be greater than 30 characters and cannot begin with SQL.

Identifiers in DataJoiner SQL

This section describes the names and terms used in the syntax descriptions for the DataJoiner SQL statements described in “Chapter 4. DataJoiner SQL Statements” on page 53 . See the *DataJoiner Administration Supplement* for more information on naming rules in DataJoiner. See “Limits Imposed by DataJoiner” on page 49 for limits on the length of a term. The following list defines these terms:

authorization-name
For DataJoiner users, a short identifier that designates a DataJoiner user. When the authorization name is local to DataJoiner, the following restrictions apply:

- The underscore character (_) is not valid.
- The name must not begin with the characters SYS, IBM, or SQL, or the numbers 0 through 9.
- The name must not be PUBLIC.
- A delimited authorization name must not contain lowercase letters.
- Letters from the extended character set are not valid.

For selected examples of authorization names, see “GRANT PASSTHRU” on page 126 and “REVOKE PASSTHRU” on page 128.

column-name
A qualified or unqualified name that designates a column of a table or view. The unqualified form of a column name is a long identifier. The qualified form is a qualifier, followed by a period and a long identifier. The qualifier is a nickname, table name, view name, or correlation name. Column names have some restrictions specified in the *DB2 Administration Guide*.

| For selected examples of column names, see "COMMENT ON" on page 74,
| "CREATE INDEX" on page 84, and "CREATE TABLE" on page 107.

correlation-name
| A long identifier that designates a nickname, local table, or view. For examples,
| see *DATABASE 2 SQL Reference for common servers*.

cursor-name
| A long ordinary identifier that designates an SQL cursor. For examples, see
| "ALLOCATE CURSOR" on page 55 and "Appendix D. Sample Program
| Fragment for Invoking a Stored Procedure" on page 173.

data-source-name
| A long ordinary identifier that designates a data source. It is the name of a
| server listed in the SYSIBM.SYSSERVERS catalog. For selected examples,
| see "CREATE NICKNAME" on page 86 and "CREATE STORED PROCEDURE
| NICKNAME" on page 103.

host-variable
| A sequence of tokens that designates a host variable. A host variable includes
| at least one host identifier. For an example of a host variable, see "CALL" on
| page 71 . For more information about host variables, see *DATABASE 2 SQL*
| *Reference for common servers*.

index-name
A qualified or unqualified name that designates an index. The unqualified form of an index name is a long identifier. An unqualified index name in an SQL statement is implicitly qualified by the authorization ID of that statement. The qualified form is an authorization ID followed by a period and a long identifier. For an example of an index name, see "CREATE INDEX" on page 84.

nickname
| A qualified or unqualified name by which DataJoiner references a table, view,
| or stored procedure. The unqualified form of a nickname is a long identifier. An
| unqualified nickname in an SQL statement is implicitly qualified by the
| authorization ID of that statement. The qualified form is an authorization name
| followed by a period and a long identifier. For selected examples of nicknames,
| see "CREATE NICKNAME" on page 86.

package-name
| A qualified or unqualified name that designates a package. The unqualified
| form of a package name is a short SQL identifier. An unqualified package
| name in an SQL statement is implicitly qualified by the authorization ID of that
| statement. The qualified form is an authorization name followed by a period
| and a short SQL identifier. For examples of package names, see *DATABASE 2*
| *SQL Reference for common servers*.

remote-authorization-name
: For data source users, a remote identifier that designates a data source user. The rules for authorization names vary from data source to data source. For selected examples of remote authorization names, see “CREATE NICKNAME” on page 86 and “CREATE STORED PROCEDURE NICKNAME” on page 103.

remote-function-name
: Identifies the name of the function on the data source. For examples, see “CREATE FUNCTION MAPPING” on page 80.

remote-object-name
: A two- or three-part name used in CREATE NICKNAME and CREATE STORED PROCEDURE NICKNAME syntax. It describes a table or stored procedure at a remote data source. For examples of remote object names, see “CREATE NICKNAME” on page 86 and “CREATE STORED PROCEDURE NICKNAME” on page 103.

remote-table-name
: A two- or three-part name of a table or view at a data source. For examples, see “CREATE NICKNAME” on page 86.

remote-type-name
: Identifies the name and type of the remote type. Do not use the long form for system built-in types (for example, use CHAR for CHARACTER type). For examples of remote type names, see “CREATE REVERSE TYPE MAPPING” on page 89 and “CREATE TYPE MAPPING” on page 112.

stored-procedure-nickname
: A qualified or unqualified name by which DataJoiner references a stored procedure. The unqualified form of a stored procedure nickname is a short identifier. An unqualified stored procedure nickname in an SQL statement is implicitly qualified by the authorization ID of that statement. The qualified form is an authorization name followed by a period and a short identifier.

For selected examples of stored procedure nicknames, see “COMMENT ON” on page 74, “CREATE STORED PROCEDURE NICKNAME” on page 103, and “Appendix D. Sample Program Fragment for Invoking a Stored Procedure” on page 173.

table-name
: A qualified or unqualified name that designates a table. The unqualified form of a table name is a long identifier. An unqualified table name in an SQL statement is implicitly qualified by the authorization ID of that statement. The qualified form is an authorization name followed by a period and a long identifier.

For selected examples of table names, see “ALTER TABLE” on page 65 and “CREATE TABLE” on page 107.

view-name
: A qualified or unqualified name that designates a view. The unqualified form of a view name is a long identifier. An unqualified view name in an SQL statement is implicitly qualified by the authorization ID of that statement. The

| qualified form is an authorization name followed by a period and a long
| identifier. For an example of an unqualified form, see “DROP” on page 121.

Limits Imposed by DataJoiner

In general, DataJoiner SQL limits match the limits for DB2. See the *DB2 SQL Reference* for general limits information.

Some DB2 limits; however, might not apply to SQL used in a pass-through session. Database manager limits of the data source apply during a pass-through session.

Maximum Lengths

DataJoiner does have limits for its unique identifiers. The limits are in Table 3.

Table 3. Identifier Length Limits

Identifier Limits	**Limit in Bytes**
Longest DataJoiner authorization-name (can be only single-byte characters)	8
Longest remote-authorization-name	30
Longest constraint name	18
Longest correlation-name	18
Longest cursor-name	18
Longest external program name	8
Longest host identifier	30
Longest schema name	8
Longest server (database alias) name	8
Longest statement name	18
Longest unqualified column-name	18
Longest unqualified package-name	8
Longest unqualified nickname, table-name, view-name, alias-name, or index name	18
Longest remote-object-name	128
Longest nickname, table or view qualifier	8
Longest server-name	18
Longest remote-table-name	128
Longest remote-column-name	128
Longest DataJoiner index-name	18
Longest remote index-name	128
Longest remote password	32

How DataJoiner Enforces Limits on Column Names and Index Names

NAME fields in SYSCAT.COLUMNS and SYSCAT.INDEXES for remote columns and indexes are restricted to 18 characters in length. If a name is longer than 18 characters, DataJoiner truncates the name to 18 characters. If the name is no longer unique after truncation, DataJoiner replaces the last character with 0. If this still does not make the name unique, DataJoiner changes the last character to 1. This process is repeated with numbers 0 through 9 and, if necessary, it is repeated again in the next to last column and so on, until a unique name is generated. For example, the name ABCDEFGHIJKLMNOPQRSTUVWXYZ has been specified for a remote column. The names ABCDEFGHIJKLMNOPQR, and ABCDEFGHIJKLMNOPQ0 already exist. The new name is over 18 characters so it is truncated to ABCDEFGHIJKLMNOPQR. Since this name already exists, DataJoiner changes the truncated name to ABCDEFGHIJKLMNOPQ0. This name exists too, so DataJoiner changes the new name to ABCDEFGHIJKLMNOPQ1. This name does not already exist, so DataJoiner now accepts it as a new name. For more information, see "SYSCAT.COLUMNS" on page 176 and "SYSCAT.INDEXES" on page 179.

Ensuring the Case of Case-Sensitive Values

In certain DataJoiner SQL statements, you sometimes need to specify identifiers and passwords that are case-sensitive at the data source. To ensure that their case is correct when they're passed to the data source, follow these guidelines:

- Specify them in the required case and enclose them in double quotes.
- If you're specifying a user ID, set the fold_id server option to 'n' ("No, don't change case") for the data source. If you're specifying a password, set the fold_pw server option to 'n' for the data source.

  There is an alternative for user IDs and passwords. If a data source requires a user ID to be in lowercase, you can specify it in any case and set the fold_id server option to 'l' ("Send this ID to the data source in lowercase"). If the data source requires the ID to be in uppercase, you can specify it in any case and set fold_id to 'u' ("Send this ID to the data source in upper case"). In the same way, if a data source requires a password to be in lowercase or uppercase, you can meet this requirement by setting the fold_pw server option to 'l' or 'u'.

  For information about setting server options, see "Server Options" on page 11. For more information about the fold_id and fold_pw server options, see "Summary of Server Options and Their Settings" on page 17.

- If you enclose a case-sensitive identifier or password in double quotes at an operating system's command prompt, you need to ensure that the system parses the double quotes correctly. To do this:
  - On a UNIX system, enclose the statement in single quotes.
  - On a Windows NT system, precede each quote with a backward slash.

For example, names of Sybase schemas, tables, and views are case-sensitive. Suppose you want to create a nickname, NICK1, for a Sybase view, `myschema.myview`, that resides in a server called SYBASE1. (In this example, the only case-sensitive parameters are `myschema` and `myview`.)

If you're entering the SQL for creating the nickname from a UNIX command prompt, you would type:

```
db2 'create nickname nick1 for sybase1."myschema"."myview"'
```

From an NT command prompt, you would type:

```
db2 create nickname nick1 for sybase1.\"myschema\".\"myview\"
```

If you enter the SQL from the DB2 interactive mode command prompt, or if you specify it in an application program, you don't need the single quotes or the slashes. For example, from the DB2 command prompt on either a UNIX or NT system, you would type:

```
create nickname nick1 for sybase1."myschema"."myview"
```

Finding Node Names

For the "node-name" parameter of a CREATE SERVER MAPPING or ALTER SERVER MAPPING statement, you must specify the node where the server specified in the statements resides. When the statement is run, the node you specify is cataloged as a value of NODE in SYSCAT.SERVERS. To determine what a server's node is, consult the following list.

Classic Connect and CrossAccess (AIX)
The server name specified in the configuration file parameter TASK INFO ENTRY.

DB2 Family (AIX and Windows NT)
The node name specified in the `DB2 node` directory. To view this directory, issue the **db2 list node directory** command.

Generic (AIX)
The server name specified in the `odbc.ini` file.

Generic (Windows NT)
The server name specified in the `odbc.ini` file. To access this name, select **ControlPanel**, then the **ODBC** icon, and finally **SystemDSN**. If you don't have an **ODBC** icon, proceed as follows:

1. From the Command Prompt window, type:

   ```
   regedt32
   ```

   Several windows open, including the Registry Editor and HKEY_LOCAL_MACHINE on Local Machine windows.
2. From the HKEY_LOCAL_MACHINE on Local Machine window:

a. Open the **HKEY_LOCAL_MACHINE** folder by double-clicking on it.
b. Open the **SOFTWARE** folder by double-clicking on it.
c. Open the **ODBC** folder by double-clicking on it.
d. Open the **ODBC.INI** folder by double clicking on it.
e. Under the **ODBC.INI** folder, double-click on the name that represents the `odbc.ini` file. The nodes in that file are listed on the right side of the HKEY_LOCAL_MACHINE on Local Machine window.

Informix (AIX)
The server name specified in the Informix `sqlhosts` file.

Informix (Windows NT)
The server name as defined by Informix SetNet 32.

Microsoft SQL Server (AIX)
For data sources accessed through the dblib protocol: The server name specified in the Sybase `interfaces` file.

For data sources accessed through ODBC: The server name specified in the `odbc.ini` file.

Microsoft SQL Server (Windows NT)
The server name specified in the `odbc.ini` file. To find out how to access this file, see the guidelines in this list under "Generic (Windows NT)".

Oracle (AIX and Windows NT)
The server name specified in the Oracle `tnsnames.ora` file. To access this name on the Windows NT platform, specify the **View Configuration Information** option of the Oracle SQL Net Easy Configuration tool.

RDB (Windows NT)
The server name specified in the `odbc.ini` file. To find out how to access this file, see the guidelines in this list under "Generic (Windows NT)".

SQL Anywhere (Windows NT)
The server name specified in the `odbc.ini` file. To find out how to access this file, see the guidelines in this list under "Generic (Windows NT)".

Sybase (AIX and Windows NT)
The server name specified in the Sybase `interfaces` file. To access this name on the Windows NT platform, select **Sybase DSEDIT**; then select **InterfaceDrivers**.

For more information, see the *DataJoiner Planning, Installation, and Configuration Guide* for your platform.

Chapter 4. DataJoiner SQL Statements

This chapter contains the complete syntax diagrams and descriptions for new SQL statements or statements with substantial modifications or enhancements specific to DataJoiner. Only the DataJoiner-specific parameters are described. For a complete description of the syntax for the DB2 SQL statements supported by DataJoiner, see the *DATABASE 2 SQL Reference for common servers*.

DataJoiner supports the SQL dialect of DB2. It supports the DB2 SQL DML, translating it to vendor dialects as necessary. DataJoiner also makes it appear to users as if DB2 SQL function exists in a data source even when that function is not actually supported by the data source. For information about this capability, referred to as *compensation*, see page 8.

In pass-through mode, SQL that is unique to a data source can be passed to the data source without any processing by DataJoiner. This feature allows existing vendor-specific SQL applications to work with DataJoiner. For more information, see "Pass-Through Sessions for Querying Data Sources in Their Own SQL" on page 8.

This chapter describes the SQL statements that are unique to DataJoiner or that contain clauses that are unique to DataJoiner. Table 4 lists these statements and the pages on which they're described.

Table 4. DataJoiner SQL Statements

DataJoiner SQL Statement	**Page**
ALLOCATE CURSOR	55
ALTER NICKNAME	56
ALTER SERVER MAPPING	59
ALTER SERVER OPTION	62
ALTER TABLE	65
ALTER USER MAPPING	69
CALL	71
COMMENT ON	74
CREATE ALIAS	77
CREATE FUNCTION	79
CREATE FUNCTION MAPPING	80
CREATE INDEX	84
CREATE NICKNAME	86
CREATE REVERSE TYPE MAPPING	89
CREATE SERVER MAPPING	96
CREATE SERVER OPTION	100
CREATE STORED PROCEDURE NICKNAME	103

Table 4. DataJoiner SQL Statements (continued)

DataJoiner SQL Statement	**Page**
CREATE TABLE	107
CREATE TYPE MAPPING	112
CREATE USER MAPPING	118
DESCRIBE CURSOR	120
DROP	121
GRANT PASSTHRU	126
REVOKE PASSTHRU	128
SET PASSTHRU	129
SET PASSTHRU RESET	131
SET SERVER OPTION	132

ALLOCATE CURSOR

Use the ALLOCATE CURSOR statement to enable an application to retrieve results returned from a stored procedure. This statement declares a cursor and associates this cursor with the last stored procedure specified by a CALL statement.

Invocation

You must embed this statement in an application program. It cannot be issued dynamically.

Authorization

None required.

Syntax

```
►►──ALLOCATE──cursor-name──CURSOR──FOR──PROCEDURE──procedure-name──────────────────────►◄
```

Description

cursor-name
: Is the name of the cursor. The name cannot match any other declared or allocated cursor within the source program.

PROCEDURE *procedure-name*
: Specifies the stored procedure that you want to associate the cursor with.

Notes

- A cursor associated with stored procedures does not require an OPEN against that cursor. For more information, see “Considerations and Restrictions” on page 35.
- An SQLCODE +467 on the CLOSE SQL statement indicates that another result set exists.

Examples

Declare cursor CUR1 and associate it with stored procedure S_INVENTORY:

```
EXEC SQL ALLOCATE CUR1 CURSOR FOR PROCEDURE S_INVENTORY
```

ALTER NICKNAME

Use the ALTER NICKNAME statement to change the local specification of the name and data type of a column in a table identified by a nickname. You can use it also to indicate that a VARCHAR or VARCHAR2 column in an Oracle table doesn't contain trailing blanks. Be aware that the statement doesn't change the remote specification of the data type that it references.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold one of the following permissions on the DataJoiner database:

- SYSADM authority
- DBADM authority
- CREATETAB privilege

The authorization ID must also have authorization at the data source to access both the data source system catalog and the table.

Syntax

```
►►──ALTER NICKNAME──table-nickname──SET COLUMN──old-local-column-name────────────────►

►──┬─LOCAL NAME──new-local-column-name──────────────────┬───────────────────────────►◄
   ├─LOCAL TYPE──data-type──────────────────────────────┤
   ├─REMOTE TYPE──remote-data-type──────────────────────┤
   └─LOCAL TYPE──data-type──REMOTE TYPE──remote-data-type─┘
```

Description

table-nickname
: References the nickname for a table that contains the column referenced by *old-local-column-name. table-nickname* must already exist as a valid DataJoiner nickname.

SET COLUMN *old-local-column-name*
: Specifies the current local name of a column in the table referenced by *table-nickname*. The name must be a valid DB2 identifier. One of the following statements is true of the column:

  - You want to change the column's local name.
  - You want to change the local specification of the column's data type.

- The column belongs to an Oracle table; the column’s data type is VARCHAR; and you want to indicate that the column doesn’t contain trailing blanks. You can also change the local specification of the maximum length of the column’s values.

LOCAL NAME *new-local-column-name*
Specifies the local name that is to replace the name specified in *old-local-column-name.* The new local name must be a valid DB2 identifier.

LOCAL TYPE *data-type*
Specifies a new data type in the local definition of the column referenced by *old-local-column-name.* Any standard DB2 data type to which the column can be converted is supported (though no conversion actually takes place). These types are listed in *DATABASE 2 SQL Reference* (CREATE TABLE SQL syntax).

REMOTE TYPE *remote-data-type*
Indicates that the column referenced by *old-local-column-name* doesn’t contain trailing blanks. The only valid value for *remote-data-type* is VARCHARNTB.

Notes

When you change the local specification of a column’s data type, DataJoiner invalidates any statistics (HIGH2KEY, LOW2KEY, and so on) gathered for that column.

Examples

Example 1: Change the local name of a table column from COL1 to NEWCOL. The table’s nickname is NICKNAME1.

```
ALTER NICKNAME NICKNAME1
   SET COLUMN COL1
   LOCAL NAME NEWCOL
```

Example 2: Change the local specification of the data type of a table column to DECIMAL (10, 5). The table’s nickname is NICKNAME1 and the column’s name is COL1.

```
ALTER NICKNAME NICKNAME1
   SET COLUMN COL1
   LOCAL TYPE DECIMAL(10,5)
```

Example 3: Indicate that in an Oracle table, a column with the data type of VARCHAR doesn’t have trailing blanks. The table’s nickname is NICKNAME2 and the column’s name is COL1.

```
ALTER NICKNAME NICKNAME2
   SET COLUMN COL1
   REMOTE TYPE VARCHARNTB
```

Example 4: Indicate that in an Oracle table, a column with the data type of VARCHAR2 doesn't have trailing blanks. In addition, change the local specification of the maximum length of the column's values to 20. The table's nickname is NICKNAME3 and the column's name is COL1.

```
ALTER NICKNAME NICKNAME3
   SET COLUMN COL1
   LOCAL TYPE VARCHAR (20)
   REMOTE TYPE VARCHARNTB
```

ALTER SERVER MAPPING

Use the ALTER SERVER MAPPING statement to change the characteristics of a mapping between a local server name (for example, SYBASE1) and the server database at the data source.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold either SYSADM or DBADM authority on the DataJoiner database. The authorization ID must also have authorization at the data source to access the data source system catalog.

Syntax

```
►►──ALTER SERVER MAPPING FROM──server-name──SET─────────────────────────────────►

     ┌─────────────────────────────────────┐
     ▼                                     │
►────┬─NODE──"node-name"──────────────┬────┴────────────────────────────────────►◄
     ├─DATABASE──"remote-database-name"─┤
     ├─TYPE──server-type──────────────┤
     ├─VERSION──server-version────────┤
     ├─PROTOCOL──"protocol-name"──────┤
     ├─CPU RATIO──value───────────────┤
     ├─IO RATIO──value────────────────┤
     └─COMM RATE──value───────────────┘
```

Description

FROM *server-name*
: Specifies the name by which the specified data source database is known locally (for example, the server-name used in a CREATE NICKNAME or SET PASSTHRU statement). *server-name* is a long identifier.

NODE *"node-name"*
: Identifies the node on which the specified data source resides. This field is protocol-dependent and its value varies. For example, a Sybase value corresponds to an entry in the interfaces file. An Oracle value corresponds to an entry in the tnsnames.ora file. DB2 values correspond to entries in a node directory. To find out how to determine *"node-name"* for your data sources, see “Finding Node Names” on page 51.

| *node-name* is a long identifier with a maximum of 70 characters and must be
| enclosed in double-quotes.

DATABASE *"remote-database-name"*
Identifies the specific database on the data source to be accessed when referring to the *server-name*. This field is protocol-dependent and its value varies. This field is not required for Oracle data sources because Oracle instances contain only one database. For Sybase and DB2, this value corresponds to a specific database within an instance or, for DB2 for OS/390, the database LOCATION value.

The valid values for this field match the allowed values for the DBNAME column in the catalog view SYSCAT.SERVERS. See "SYSCAT.SERVERS" on page 187 for details.

DataJoiner defines a unique index on the NODE and DBNAME columns in SYSCAT.SERVERS. This unique index prevents you from specifying an identical combination of values for NODE and NODENAME with two different values in the SERVER column. *remote-database-name* is a long identifier and must be enclosed in double-quotes.

TYPE *server-type*
Identifies the type of data source.

The valid values for this field match the allowed values for the TYPE column in the catalog view SYSCAT.SERVERS. See "SYSCAT.SERVERS" on page 187 for details. Delimited identifiers are converted to uppercase automatically.

server-type can have up to 30 characters.

Examples of valid values:

```
TYPE ORACLE
TYPE "informix"
TYPE DB2/MVS
TYPE "sybase"
```

Examples of invalid values:

```
TYPE NEWDATASOURCE
TYPE unknowndb
```

VERSION *server-version*
Identifies the version of the data source. Valid values for *server-version* are composed of digits and, optionally, one or two decimal points. Do not enclose these values in quotes or place a decimal point at the end of them. For example:

```
| VERSION 7
| VERSION 8.0.3
```

The valid values for this field match the allowed values for the VERSION column in the catalog view SYSCAT.SERVERS. For details, see "SYSCAT.SERVERS" on page 187 .

| *server-version* can have up to 18 characters.

PROTOCOL *″protocol-name″*
: Identifies the data access protocol used by the data source.

  The set of valid values for this field matches the allowed values for the SERVER_PROTOCOL column in the catalog view SYSCAT.SERVERS. Examples are “drda” and “db2ra”. See “SYSCAT.SERVERS” on page 187 for details.

  ″protocol-name″ is case-sensitive and cannot exceed 30 characters.

CPU RATIO *value*
: Represents how much faster or slower the data source CPU is compared to the local CPU. Valid values are unsigned floating-point constants, unsigned decimal constants, or unsigned integer constants. For example, if the remote CPU is twice as fast as the local CPU, enter the value 0.5.

IO RATIO *value*
: Represents how much faster or slower the data source workstation I/O is compared to the local workstation. Valid values for *value* are unsigned floating-point constants, unsigned decimal constants, or unsigned integer constants. For example, if the remote workstation I/O is twice as fast as the local workstation I/O, enter the value 0.5.

COMM RATE *value*
: Represents the communication rate between the data source workstation and the local workstation. *value* represents the number of bytes per second; use unsigned integer constants. The default is 2 megabytes per second.

Notes

Identifiers without delimiters are converted to uppercase in SYSCAT.SERVERS. The following examples show how you can use quotes to pass case-sensitive characters to specific columns in SYSCAT.SERVERS.

Examples

Example 1: Map the local server SYBASE1 to a different database, TEST_DB2.

```
ALTER SERVER MAPPING FROM SYBASE1 SET DATABASE "test_db2"
```

Example 2: The local server SYBASE1 is mapped to a Sybase database that predates version 10. Map SYBASE1 to a Sybase 10 database called TEST_DB2.

```
ALTER SERVER MAPPING FROM SYBASE1
   SET DATABASE "test_db2"
   VERSION 10.0
```

Example 3: Update the server characteristics for DB2MVS1 by setting both CPU ratio and I/O ratio to 3.0. This ratio means that the local CPU runs three times as fast as the remote CPU and the local I/O is three times faster than the remote I/O.

```
ALTER SERVER MAPPING FROM DB2MVS1
   SET CPU RATIO 3.0
   IO RATIO 3.0
```

ALTER SERVER OPTION

Use the ALTER SERVER OPTION statement to change, for an indefinite period, a server option setting that was defined by the CREATE SERVER OPTION statement. For an introduction to the ALTER SERVER OPTION statement, see “Server Options” on page 11.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold either SYSADM or DBADM authority.

Syntax

```
►►──ALTER SERVER OPTION──option-name──FOR──┤ Remote Server Clause ├──────────────►

►─SETTING──option-value─────────────────────────────────────────────────────────►◄
```

Remote Server Clause

```
├──┬─SERVER──server-name──────────────────────────────────────────────┬──────────┤
   ├─SERVER TYPE──server-type──VERSION──server-version──┤ Protocol ├──┤
   ├─SERVER TYPE──server-type──VERSION──server-version────────────────┤
   └─SERVER TYPE──server-type─────────────────────────────────────────┘
```

Protocol

```
├──PROTOCOL──"server-protocol" ──────────────────────────────────────────────────┤
```

Description

option-name
: Is the name of the server option. Valid *option-name* values are listed after the OPTION column definition for the catalog view SYSCAT.SERVER_OPTIONS (see “SYSCAT.SERVER_OPTIONS” on page 195). *option-name* is an identifier; it cannot exceed 30 characters.

SERVER *server-name*
: Identifies the server to which the option specified in *option-name* applies. This field must match an entry in the SERVER column of the SYSCAT.SERVERS catalog view (see “SYSCAT.SERVERS” on page 187). *server-name* is a long identifier.

SERVER TYPE *server-type*
Identifies the type of server to which the option specified in *option-name* applies. Example values include DB2/MVS, ORACLE, and informix. Valid values for *server-type* match those listed for the SERVER_TYPE column of SYSCAT.SERVERS (see “SYSCAT.SERVERS” on page 187). Values for *server-type* cannot exceed 30 characters. Delimited identifiers are converted to uppercase.

VERSION *server-version*
Identifies the version of the data source. Valid values for *server-version* are composed of digits and, optionally, one or two decimal points. Do not enclose these values in quotes or place a decimal point at the end of them. For example:

```
| VERSION 7
| VERSION 8.0.3
```

The valid values for this field match the allowed values for the VERSION column in the catalog view SYSCAT.SERVERS. For details, see “SYSCAT.SERVERS” on page 187 .

| *server-version* can have up to 18 characters.

PROTOCOL *"server-protocol"*
| Identifies the server protocol to which the option specified in *option-name* applies.
| Field values must match one of the allowed entries for the SERVER_PROTOCOL
| column of the SYSCAT.SERVERS catalog view (see “SYSCAT.SERVERS” on
| page 187). *"server-protocol"* is case-sensitive and cannot exceed 30 characters.

SETTING *option-value*
Identifies the option setting. Valid values for *option-value* depend on the value entered for *option-name*. See the examples below. *option-value* can be an integer constant, floating-point constant, decimal constant, or character literal. If it’s a character literal, it can contain up to 254 characters, and it must be enclosed in single quotes. For descriptions of valid settings, see “Summary of Server Options and Their Settings” on page 17.

Notes

- Server options can be entered in uppercase or lowercase (case does not matter).
- If you set an option to one value for a server type, and set this same option to another value for an instance of the type, the second value overrides the first for the instance. For example, suppose that you set remote_query_caching to ’y’ for server type sybase and use this option’s default (’n’) for a Sybase server named SIBYL. As a result, query results won’t be cached for SIBYL, but they’ll be cached for other Sybase servers whenever doing so enhances performance.
- For an overview of the SQL for setting server options, see “Server Options” on page 11 .

Examples

Example 1: Discontinue two-phase commit on server NOMATCH.

```
ALTER SERVER OPTION TWO_PHASE_COMMIT
   FOR SERVER NOMATCH
   SETTING 'n'
```

Example 2: Begin folding user IDs to uppercase when sending them to DB2 for OS/390 data sources.

```
ALTER SERVER OPTION FOLD_ID
   FOR SERVER TYPE DB2/MVS
   SETTING 'y'
```

Example 3: Remote query caching is not enabled for Informix servers. Enable it for Informix server INFOMAX, so that it's available to all applications that use the server.

```
ALTER SERVER OPTION REMOTE_QUERY_CACHING
   FOR SERVER INFOMAX
   SETTING 'y'
```

Example 4: Push down optimization is not enabled for Sybase data sources. Make it available now.

```
ALTER SERVER OPTION PUSHDOWN
   FOR SERVER TYPE SYBASE
   SETTING 'y'
```

ALTER TABLE

Use the ALTER TABLE statement to alter data source tables that were created with the DataJoiner CREATE TABLE statement. (To alter tables in the DataJoiner database, use the DB2 for CS ALTER TABLE statement.)

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold one of the following permissions on the DataJoiner database:

- SYSADM authority
- DBADM authority
- ALTER privilege

The authorization ID must also have authorization at the data source to alter data source tables.

Syntax

```
►►──ALTER TABLE──table-name──────────────────────────────────────────────►

   ┌──────────────────────────────────────────────────┐
   ▼     (1)       ┌─COLUMN──┐                          │
►──────ADD────────┬┴─────────┴─┤ Column Definition ├──┬┴──────────────►◄
                  └─┤ Primary Key Constraint ├───────┘
```

Notes:

1. ADD is optional for unnamed PRIMARY KEY constraints.

Column Definition

```
├──column-name──┤ Data Type ├──────────────────────────────────────────┤
```

Data Type

```
├──┬─┬─INTEGER─┬──────────────────────────────────────────────────────────┬──────────┤
   │ └─INT─────┘                                                          │
   ├─SMALLINT─────────────────────────────────────────────────────────────┤
   ├─┬─DOUBLE───────────┬─────────────────────────────────────────────────┤
   │ ├─DOUBLE PRECISION─┤                                                 │
   │ └─FLOAT────────────┘                                                 │
   ├──┬─DECIMAL─┬──┬──────────────────────────────────────┬───────────────┤
   │  ├─DEC─────┤  └─(──integer──┬─────────────┬──)──────┘                │
   │  ├─NUMERIC─┤                └─,──integer──┘                          │
   │  └─NUM─────┘                                                         │
   ├──┬──┬─CHARACTER─┬──┬──────────────────┬──┬──┬─────────────────────┬──┤
   │  │  └─CHAR──────┘  └─(──integer──)────┘  │  │                 (1) │  │
   │  ├──┬─VARCHAR────────────────(──integer──)─┤  └─FOR BIT DATA──────┘  │
   │  │  ├─CHARACTER──VARYING──┤              │                          │
   │  │  └─CHAR──VARYING───────┘              │                          │
   │  └─LONG VARCHAR──────────────────────────┘                          │
   │                                                                      │
   ├──┬─BLOB───┬──(──integer──┬─────┬──)──────────────────────────────────┤
   │  ├─CLOB───┤              ├─K───┤                                     │
   │  └─DBCLOB─┘              ├─M───┤                                     │
   │                          └─G───┘                                     │
   ├─GRAPHIC──┬────────────────────┬──────────────────────────────────────┤
   │          └─(──integer──)──────┘                                      │
   ├─VARGRAPHIC──(──integer──)────────────────────────────────────────────┤
   ├─LONG VARGRAPHIC──────────────────────────────────────────────────────┤
   ├─DATE─────────────────────────────────────────────────────────────────┤
   ├─TIME─────────────────────────────────────────────────────────────────┤
   ├─TIMESTAMP────────────────────────────────────────────────────────────┤
   └─distinct-type-name───────────────────────────────────────────────────┘
```

Notes:

1. You can specify FOR BIT DATA in random order with the other column constraints that follow.

Primary Key Constraint

```
                                                       ┌─,───────────┐
                                                       ▼             │
├──┬────────────────────────────────┬──PRIMARY KEY──(────column-name─┴──)──────────┤
   └─CONSTRAINT──constraint-name────┘
```

Description

table-name
: Is the name of the table that you're altering, with or without a qualifier. If you specify a qualifier, it must be the same one that qualifies the table's nickname.

ADD
: Adds a column or primary key to the table. If the table has any rows, every value in

the newly-added column is the column’s default value. The new column is the table’s “last” column; that is, if the table initially has `n` columns, the added column is `n+1`.

column-name
Is the name of the new column, without a qualifier. The name cannot be the same as that of any other column in the table.

Data Type
The data type of the column specified in the *column-name* parameter. This data type is defined to the DataJoiner database. It’s valid for the table that you’re altering only if there’s a reverse mapping between it and a data type at the data source where the table resides. To find out what reverse mappings exist between data source types and types defined to the DataJoiner database, query the SYSCAT.REVTYPEMAPPINGS catalog view (documented on page 184). This view shows both default and user-defined mappings. For lists of default mappings only, see “Appendix B. Default Reverse Type Mappings” on page 155. If you don’t find a mapping that you need, you can create it with the CREATE REVERSE TYPE MAPPING statement (page 89).

For an explanation of the keywords and parameters in the Data Type clause, see *DATABASE 2 SQL Reference for common servers*.

Primary Key Constraint
For an explanation of this clause, see *DATABASE 2 SQL Reference for common servers*. Be aware that you can include columns in a primary key only if the columns have already been defined as not nullable.

Notes

- Some data sources do not support DDL in two-phase commit mode. If you’re altering a table for such a data source, and this data source is configured for two-phase commit transactions, you need to reconfigure it so that it can commit the ALTER TABLE statement in one phase. To do this for a single unit of work, follow either of these procedures:
  - Use the SET SERVER OPTION statement to set the two_phase_commit server option to ’n’. Place the statement immediately after the CONNECT statement.
  - Place any CREATE REVERSE TYPE MAPPING statements that you code and the ALTER TABLE statement immediately after the CONNECT statement.

  To reconfigure a data source for one-phase commits indefinitely, run an ALTER SERVER OPTION statement in which the two_phase_commit option for the data source is set to ’N’.

  For information about the SET SERVER OPTION and ALTER SERVER OPTION statements, see “Using Server Options to Configure Data Sources” on page 11.
- After you alter the table by running the DataJoiner ALTER TABLE statement, you can, if you want, alter it further; for example, add referential constraints to it. To do this, open a pass-through session with the data source where the table resides, and use the data source’s native ALTER TABLE statement to make the changes that you want.

Examples

Add a column C10 to table T1 at an SQL Anywhere data source. Designate an existing column, C9, as a primary key without a name. Assume that:

- T1 was created with the DataJoiner CREATE TABLE statement.
- C9 is not nullable.

```
ALTER TABLE T1
  ADD COLUMN C10 CHAR(200)
  ADD PRIMARY KEY (C9)
```

ALTER USER MAPPING

Use the ALTER USER MAPPING statement to change the characteristics of a mapping between an authorization ID, password, or connect option defined locally to DataJoiner, and the corresponding ID, password, or option used at a specific data source.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold either SYSADM or DBADM authority on the DataJoiner database.

Syntax

```
                             ┌─USER──────────┐
►►──ALTER USER MAPPING FROM──┴─local-authid──┴──TO SERVER──server-name──────────────►

        ┌─────────────────────────────────┐
        ▼                                 │
►─SET──────┬─AUTHID──remote-authid───────┬┴────────────────────────────────────────►◄
           ├─PASSWORD──remote-password───┤
           └─CONNECTOPT──'string'────────┘
```

Description

local-authid or USER
: *local-authid* identifies the authorization ID under which a specific user connects to the DataJoiner database.

The valid values for *local-authid* match the allowed values for the AUTHID column in the catalog view SYSCAT.REMOTEUSERS. Identifiers without delimiters are converted to uppercase. See “SYSCAT.REMOTEUSERS” on page 183 for details. *local-authid* is a short identifier with a maximum of 8 characters.

USER denotes the special register USER, which specifies the authorization ID under which you run the ALTER USER MAPPING statement. If you specify USER, this ID maps automatically to the ID specified by *remote-authid*.

TO SERVER *server-name*
: Identifies the server that can be connected to under the authorization ID denoted by *remote-authid*. This field must match an entry in the SERVER column of the SYSCAT.SERVERS catalog view (see “SYSCAT.SERVERS” on page 187).

The unique index for SYSCAT.REMOTEUSERS includes the AUTHID and SERVER columns. You cannot specify two rows in this table with identical values for both of these columns.

server-name is a long identifier.

AUTHID *remote-authid*
: Identifies an ID that's required for connecting to the specified server and that corresponds to *local-authid*. *remote-authid* is an identifier that cannot exceed 30 characters.

PASSWORD *remote-password*
: Identifies the password for the *remote-authid* on the specified server. *remote-password* is an identifier that cannot exceed 32 characters.

CONNECTOPT *'string'*
: Identifies the connect option for the specified server. The value of *'string'* is protocol-dependent (for example, for drda protocol, the value is an accounting string). *'string'* is a character string; its maximum length is 256 characters.

Notes

- Identifiers without delimiters are converted to uppercase in SYSCAT.REMOTEUSERS. For guidelines on preserving lowercase, see "Ensuring the Case of Case-Sensitive Values" on page 50.
- If *remote-password* isn't specified, the server option for validating passwords (password) must be set to `'n'` for the server denoted by *server-name.* For more information about this option, see "SYSCAT.SERVER_OPTIONS" on page 195.

Examples

Example 1: Alter a user mapping.

```
ALTER USER MAPPING FROM "Kleewein"
   TO SERVER ORACLE1
   SET AUTHID "jimk"
```

Example 2: Alter a user mapping with a specified password. Use the special register USER.

```
ALTER USER MAPPING FROM USER
   TO SERVER DB2MVS1
   SET AUTHID "jimk"
   PASSWORD "jimmvspw"
```

CALL

Use the CALL statement to invoke a stored procedure. This statement uses the stored procedure's name, its nickname, or a host variable, to identify the stored procedure to DataJoiner.

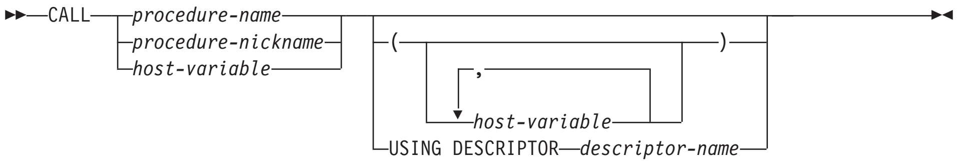
Invocation

You must embed this statement in an application program. It cannot be issued dynamically.

Authorization

The authorization ID under which you run this statement must hold the EXECUTE privilege on the package associated with the stored procedure.

Syntax



Description

CALL *procedure-name* or *procedure-nickname* or *host-variable*
Identifies the stored procedure to execute. The name can be specified directly, via a stored procedure nickname, or indirectly within a host variable.

(*host-variable,...***)**
Each specification of *host-variable* is a parameter of CALL. The *n*th parameter of the statement corresponds to the *n*th parameter of the stored procedure.

USING DESCRIPTOR *descriptor-name*
Identifies an SQL descriptor area (SQLDA) that contains a valid description of host variables. The *n*th SQLVAR element corresponds to the *n*th parameter of the stored procedure.

Notes

- Stored procedures can be invoked at DataJoiner or at data sources.
- DataJoiner supports result sets at data sources that support stored procedures that return result sets. Applications can retrieve result sets by using a cursor. The ALLOCATE CURSOR statement associates the cursor with the stored procedure. The DESCRIBE CURSOR statement returns a description of a result set. The FETCH statement returns a single row of a result set, and the CLOSE statement closes the cursor. Here's an example:

```
CALL S_SELECT;
ALLOCATE C1 CURSOR FOR PROCEDURE S_SELECT;
DESCRIBE CURSOR C1 INTO :*PGM_SQLDA;
FETCH C1 USING DESCRIPTOR :*PGM_SQLDA;
CLOSE C1;
```

Important: Restrictions exist on invoking stored procedures that return result sets, See “Considerations and Restrictions” on page 35.

- You can qualify a stored procedure name if you’re using the SDK provided by DataJoiner or the SDK provided by UDB Version 5. The following statements contain an example of a qualified stored procedure name:

  ```
  CALL J15USER1.S_SELECT;
  ALLOCATE C1 CURSOR FOR PROCEDURE J15USER1.S_SELECT
  ```
- If you’re using an SDK other than one provided by DataJoiner or UDB Version 5, and if this SDK supports the CALL statement, then:
  - In the CALL statement, you must qualify a stored procedure name through the use of a host variable. For example:

    ```
    STRCPY(PROCNAME,"J15USER3.S_SELECT")
    CALL :PROCNAME
    ```
  - In the ALLOCATE CURSOR statement, the stored procedure name cannot be qualified.
- The stored procedure search sequence is:
  1. Search for the procedure at the application location
  2. Search for the procedure at the database engine location
  3. End the search; pass the stored procedure directly to the database engine for processing
- SQLCODE +466 on the CALL SQL statement indicates that results exist.
- If you are specifying LOB data from an SQLDA, double the number of allocated SQLVAR entries.
- Do not use the CALL statement with existing DB2 DARI procedures.

Examples

Example 1 (fragment): Invoke stored procedure SIMPLE.

```
CALL SIMPLE;
```

Example 2 (fragment that illustrates the use of an input host variable): Invoke stored procedure SIMP_IN.

```
QUANTITY=2;
CALL SIMP_IN(:QUANTITY);
```

Example 3 (fragment that illustrates the use of input and output host variables): Invoke stored procedure SIMP_INOUT.

```
QUANTITY=2;
MEMSET(FRUIT,'\0',6);
CALL SIMP_INOUT(:QUANTITY,:FRUIT);
```

For more examples, see “Notes” on page 71 and “Appendix D. Sample Program Fragment for Invoking a Stored Procedure” on page 173.

COMMENT ON

Use the COMMENT ON statement to add or replace comments in catalog descriptions of objects.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The privileges held by the authorization ID under which you run this statement must vary according to the type of object that you want to comment on. For a list of objects mapped to privileges, see *DATABASE 2 SQL Reference for common servers*.

Syntax

```
►►──COMMENT ON─────────────────────────────────────────────────────────────►

►──┬─┤ Objects ├──IS──string-constant─────────────────────────┬────────────►◄
   │                             ┌─,───────────────────────┐  │
   │                             ▼                         │  │
   └──┬─table-name─┬──(─────column-name──IS──string-constant─┴──)──┘
      └─view-name──┘
```

Objects

```
|──┬─ALIAS──alias-name──────────────────────────────────────────┬──────────────────|
   ├─COLUMN──┬─table-name.column-name─┬─────────────────────────┤
   │         └─view-name.column-name──┘                         │
   ├─CONSTRAINT──table-name.constraint-name─────────────────────┤
   │              (1)                                           │
   ├─DISTINCT TYPE──────distinct-type-name──────────────────────┤
   │         (2)                                                │
   ├─FUNCTION──────function-name──┬─────────────────────────┬───┤
   │                              └─(─┬───────────────────┬─)─┘ │
   │                                  │  ┌─,─────────┐    │     │
   │                                  │  ▼           │    │     │
   │                                  └───data-type──┴────┘     │
   ├─SPECIFIC FUNCTION──specific-name───────────────────────────┤
   ├─FUNCTION MAPPING──function-mapping-name────────────────────┤
   ├─INDEX──index-name──────────────────────────────────────────┤
   ├─PACKAGE──package-name──────────────────────────────────────┤
   ├─SERVER MAPPING──server-name────────────────────────────────┤
   ├─SERVER OPTION──option-name──FOR──| Remote Server Clause |──┤
   ├─STORED PROCEDURE NICKNAME──stored-procedure-nickname───────┤
   ├─TABLE──┬─table-name─┬──────────────────────────────────────┤
   │        └─view-name──┘                                      │
   ├─TABLESPACE──tablespace-name────────────────────────────────┤
   ├─TRIGGER──trigger-name──────────────────────────────────────┤
   └─TYPE MAPPING──type-mapping-name────────────────────────────┘
```

Remote Server Clause

```
|──┬─SERVER──server-name──────────────────────────────────────────────┬────────────|
   ├─SERVER TYPE──server-type──VERSION──server-version──| Protocol |──┤
   ├─SERVER TYPE──server-type──VERSION──server-version────────────────┤
   └─SERVER TYPE──server-type─────────────────────────────────────────┘
```

Protocol

```
|──PROTOCOL──"server-protocol"──────────────────────────────────────────────────────|
```

Notes:

1. The keyword DATA can be used as a synonym for DISTINCT
2. The keyword ROUTINE can be used as a synonym for FUNCTION

Description

FUNCTION MAPPING *function-mapping-name*
Indicates that a comment will be added or replaced for a function mapping name. The *function-mapping-name* must identify a distinct function mapping in the catalog. The comment replaces the value in the REMARKS column of the SYSCAT.SERVER_FUNCTIONS catalog view for the row describing the function mapping.

SERVER MAPPING *server-name*
: Identifies the server mapping being commented on. *server-name* is a long identifier.

SERVER OPTION *option-name*
: Identifies the server option being commented on. See Table 2 on page 17 for a complete list of valid options.

STORED PROCEDURE NICKNAME *stored-procedure-nickname*
: Indicates that a comment will be added or replaced for a stored procedure nickname. The *stored-procedure-nickname* must identify a distinct stored procedure nickname in the catalog. The comment replaces the value in the REMARKS column of the SYSCAT.PROCEDURES catalog view for the row describing the stored procedure nickname.

TYPE MAPPING *type-mapping-name*
: Indicates that a comment will be added or replaced for a type mapping name. The *type-mapping-name* must identify a distinct type mapping in the catalog. The comment replaces the value in the REMARKS column of the SYSCAT.SERVER_DATATYPES catalog view for the row describing the type mapping.

Examples

Supply a comment on a stored procedure.

```
COMMENT ON STORED PROCEDURE NICKNAME seb_proc
IS 'Returns sum of all publication orders'
```

CREATE ALIAS

Use the CREATE ALIAS statement to create an alternate name for a table or view in the DataJoiner database, or an alternate nickname for a remote table or view.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

If the authorization ID under which you run this statement matches the name of the schema of the alias that you're creating, that ID carries sufficient authorization to run the statement. If the ID differs from the name of the schema, you need either SYSADM or DBADM authority.

Syntax

```
►►──CREATE──┬─ALIAS───┬──alias-name──FOR──┬─table-name──┬──────────────────────────►◄
            └─SYNONYM─┘                    ├─view-name───┤
                                           ├─alias-name2─┤
                                           └─nickname────┘
```

Description

table-name
: Is either the fully-qualified name of a table in the DataJoiner database, or the name only. If the name's qualifier is a schema name that's the same as the authorization ID under which you run this statement, then specify the name only. Otherwise, if the qualifier differs from the authorization ID, then specify the fully-qualified name.

  The qualifier must not be SYSIBM.

view-name
: Is either the fully-qualified name of a view in the DataJoiner database, or the name only. If the name's qualifier is a schema name that's the same as the authorization ID under which you run this statement, then specify the name only. Otherwise, if the qualifier differs from the authorization ID, then specify the fully-qualified name.

  The qualifier must not be SYSIBM.

nickname
: Is either the fully-qualified nickname for a data source table or view, or the nickname only. If the nickname's qualifier is a schema name that's the same as the authorization ID under which you run this statement, then specify the nickname only. Otherwise, if the qualifier differs from the authorization ID, then specify the fully-qualified nickname.

  The qualifier must not be SYSIBM.

Examples

Designate A1 as an alias for a table referenced by the nickname FUZZYBEAR.

```
CREATE ALIAS A1 FOR fuzzybear
```

CREATE FUNCTION

In DataJoiner, use the CREATE FUNCTION statement to define a function template that you can map to a function at a data source. The template consists only of a function name and input and output parameters. It does not include executable code. For more information, see “Enabling DataJoiner to Access UDFs and New Built-In Functions at Data Sources” on page 38.

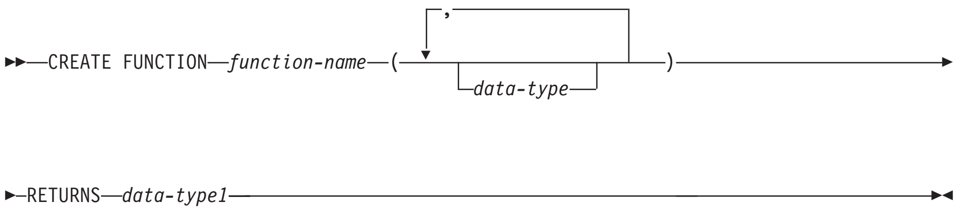
Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

If the authorization ID under which you run this statement matches the name of the schema of the function specification that you’re creating, that ID carries sufficient authorization to run the statement. If the ID differs from the name of the schema, you need either SYSADM or DBADM authority.

Syntax



Notes

- When a data source function maps to a DataJoiner function template, the function can be invoked if the access plan chooses to evaluate it at the data source. If, instead, the access plan chooses to evaluate the function template, a runtime error can result, because the template has no executable code.
- DataJoiner supports the DB2 for CS CREATE FUNCTION statement.

Examples

Define to DataJoiner a function template that you can map to a function called CENTRE.

```
   CREATE FUNCTION CENTRE (FLOAT, FLOAT) RETURNS FLOAT
```

CREATE FUNCTION MAPPING

Use the CREATE FUNCTION MAPPING statement to create a mapping between a local (DataJoiner) function or function template (that is, a definition without executable code) and a function at a data source. The argument signatures of the local function or template and the data source function must correspond. That is:

- The local function or template must have the same number of parameters as the data source function.
- Mappings must exist between the data types of the parameters of the local function or template, and the data types of the parameters of the data source function. You can find out if such mappings exist by querying the SYSCAT.SERVER_DATATYPES catalog view.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold either SYSADM authority or DBADM authority on the DataJoiner database.

Syntax

```
►►──CREATE FUNCTION MAPPING──┬──────────────────────────┬──────────────────────────────►
                             └─function-mapping-name────┘

►─FROM──┤ Local Function Clause ├──TO──┤ Remote Server Clause ├────────────────────────►

►─┤ Remote Function Clause ├──┤ Function Statistics Clause ├──┬────────────┬──────────►◄
                                                             └─WITH INFIX─┘
```

Local Function Clause

```
                                 ┌─,─────────────┐
                                 ▼               │
├──┬─local-function-name──(──┬─────────────┬──┴──)──┬────────────────────────────┤
   │                         └─data-type───┘        │
   ├─FUNCTION ID──function-ID───────────────────────┤
   └─SPECIFIC──local-specific-name──────────────────┘
```

Remote Server Clause

```
├──┬─SERVER──server-name─────────────────────────────────────┬──────────┤
   ├─SERVER TYPE──server-type──VERSION──server-version──┤ Protocol ├─┤
   ├─SERVER TYPE──server-type──VERSION──server-version───────────┤
   └─SERVER TYPE──server-type────────────────────────────────────┘
```

Protocol

```
├──PROTOCOL──"server-protocol" ──────────────────────────────────────────┤
```

Remote Function Clause

```
├──FUNCTION──remote-function-name────────────────────────────────────────┤
```

Function Statistics Clause

```
├──┬───────────────────────┬──┬─────────────────────────┬────────────────►
   └─IOS_PER_INVOC──value──┘  └─INSTS_PER_INVOC──value──┘

►──┬─────────────────────────┬──┬─────────────────────────┬──────────────►
   └─INSTS_PER_INVOC──value──┘  └─IOS_PER_ARGBYTE──value──┘

►──┬───────────────────────────┬──┬──────────────────────────┬───────────►
   └─INSTS_PER_ARGBYTE──value──┘  └─PERCET_ARGBYTES──value───┘

►──┬───────────────────────┬──┬─────────────────────┬────────────────────┤
   └─INITIAL_INSTS──value──┘  └─INITIAL_IOS──value──┘
```

Description

function-mapping-name
Is a long identifier that names the function mapping. The name must not identify a function mapping that is already described in SYSCAT.SERVER.FUNCTIONS. If it does, you receive an error message (SQL0601N).

If *function-mapping-name* is not specified, a unique name is generated by the system.

local-function-name
Identifies a local system function or user-defined function. If a schema name is not specified, the current user ID is assumed. If a schema name is specified, the *local-function-name* must be either qualified by SYSIBM or unique within the DataJoiner database. If the name meets neither of these conditions, you receive an error message (SQL0476N).

FUNCTION ID *function-ID*
Identifies the unique function ID assigned to each system-defined or user-defined function.

SPECIFIC *local-specific-name*
Identifies the specific name of a local system function or user-defined function. *local-specific-name* must be used if the function does not have a unique *local-function-name* within the DataJoiner database.

SERVER *server-name*
Identifies the server on which the remote function exists. This field must match an entry in the SERVER column of the SYSCAT.SERVERS table. *server-name* is a long identifier.

SERVER TYPE *server-type*
Identifies the type of server on which the function exists. Valid values for *server-type* must match those listed for the SERVER_TYPE column of SYSCAT.SERVERS_TYPE. Values for *server-type* cannot exceed 30 characters. Delimited identifiers will be converted to uppercase.

VERSION *server-version*
| Identifies the version of the server on which the function resides. Values for
| *server-version* are composed of digits and, optionally, one or two decimal points.
| Do not enclose these values in quotes or place a decimal point at the end of them.
| For example:

```
VERSION 7
VERSION 8.0.3
```

| *server-version* can have up to 18 characters.

| If the server is built-in, specifying this value assures that the mapping that you're
| creating applies to this and all future versions of the server type (and protocol, if
| present).

| The valid values for this field match the allowed values for the VERSION column in
| the catalog view SYSCAT.SERVERS. For details, see "SYSCAT.SERVERS" on
| page 187.

PROTOCOL *"server-protocol"*
| Identifies the server protocol on which the protocol exists. Field values must match
| one of the allowed entries for the SERVER_PROTOCOL column of the
| SYSCAT.SERVERS.PROTOCOL. *"server-protocol"* is case-sensitive and cannot
| exceed 30 characters.

FUNCTION *remote-function-name*
Identifies the name of the function on the data source.

value
Identifies the value of the specified statistic.

WITH INFIX
| Specifies that the remote function be generated in infix format. For example, a local
| function "="(integer,integer) is mapped to the following remote infix function:
| integer=integer.

Notes

Functions in the SYSIBM schema do not have a specific name. To override the default function mapping for a function in the SYSIBM schema, specify *local-function-name* with qualifier SYSIBM and function name (such as LENGTH). This specification will affect all invocations of this specific SYSIBM function.

Examples

Example 1: Map a local function template to a UDF that all Oracle data sources can access. The template is called CENTRE and belongs to a schema called MATH. The Oracle UDF is called MIDPOINT and belongs to a schema called SMITH.

```
CREATE FUNCTION MAPPING MY_ORACLE_FUN1
   FROM MATH.CENTRE
   TO SERVER TYPE ORACLE
   FUNCTION SMITH.MIDPOINT
```

Example 2: Map a local function template called CENTRE to a UDF, also called CENTRE, that's used at an Oracle data source called ORACLE1.

```
CREATE FUNCTION MAPPING MY_ORACLE_FUN1
   FROM CENTRE
   TO SERVER ORACLE1
   FUNCTION CENTRE
```

CREATE INDEX

Use the CREATE INDEX statement to supply the DataJoiner catalog with a specification of an index for a data source table that has been defined to DataJoiner, but that has no index, or that has an index that DataJoiner doesn't recognize. CREATE INDEX doesn't create an actual index. The specification that it supplies is used by the DataJoiner optimizer to improve access to the table.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold one of the following permissions:

- SYSADM authority on the DataJoiner database
- DBADM authority on the DataJoiner database
- CONTROL privilege on the nickname or table
- INDEX privilege on the nickname or table

Syntax

```
►►──CREATE──┬────────┬──INDEX──index-name──────────────────────────────►
            └─UNIQUE─┘

                        ┌─,──────────────────────┐
                        ▼           ┌─ASC──┐     │
►──ON──┬─nickname───┬──(────column-name──┼──────┼─────┴──)──────────────►◄
       └─table-name─┘                    └─DESC─┘
```

Description

UNIQUE
Specifies that the table on which the index is defined cannot contain two or more rows with the same value of the index key. This constraint is enforced when rows of the table are updated or new rows are inserted.

When UNIQUE is used, nulls are treated as values. For example, if the key is a single column that can contain nulls, that column can contain no more than one null.

INDEX *index-name*
Names the index. The name, including the implicit or explicit qualifier, must not identify an index described in the catalog. The qualifier cannot be SYSIBM.

ON *nickname* or *table-name*
Specifies the nickname or actual name of the table on which the index is defined. The table must reside at a data source; it cannot be a catalog table.

column-name
Represents a column that is part of the index key. You can set *column-name* to a value in the COLNAME column of the SYSCAT.COLUMNS catalog view. This value is in the local definition of the table that you're specifying; the value needn't be the actual column name used at the data source. You can specify up to 16 column names. Each must be unqualified; none can be repeated.

The sum of the lengths of the specified columns must not exceed 255. To find out the length of a column, consult the LENGTH column of the SYSCAT.COLUMN catalog view.

ASC
Signifies that the index entries are in ascending order by column. This setting is the default.

DESC
Signifies that the index entries are in descending order by column.

Notes

- Because a nickname does not actually contain data, CREATE INDEX on a nickname creates only a description of the index.
- The index name that you use with a nickname is not required to match the remote index name to which it corresponds.

Examples

Create a local definition of an existing index for a remote table that is referenced by the nickname EMPLOYEE. The name of the index is JOB_BY_DPT.

```
CREATE INDEX JOB_BY_DPT ON EMPLOYEE (WORKDEPT, JOB)
```

CREATE NICKNAME

| Use the CREATE NICKNAME statement to create a nickname for a remote table or
| view. To create a nickname for a stored procedure, see “CREATE STORED
| PROCEDURE NICKNAME” on page 103.

Before using nicknames, review “Considerations and Restrictions” on page 24 to be sure that you understand the restrictions on their usage.

Invocation

| You can embed this statement in an application program or issue it dynamically.

Authorization

| The authorization ID under which you run this statement must hold one of the following
| permissions on the DataJoiner database:

| • SYSADM authority
| • DBADM authority
| • CREATETAB privilege

| The authorization ID must also have authorization at the data source to access both the
| data source system catalog and the table or view.

Syntax

```
►►──CREATE NICKNAME──nickname──FOR──remote-object-name──────────────────────────────►◄
```

Description

nickname
: Represents an identifier with or without a qualifier. If no qualifier is supplied, the authorization ID of the statement is used as the default value. The qualifier must not be SYSIBM.

remote-object-name
: Represents a two- or three-part name.

- If you’re creating a nickname for a table or view at a data source that supports schemas, *remote-object-name* is:

  data-source-name.remote-authorization-name.remote-table-name

  —where:

  data-source-name
  : Is a value in the SERVER column of the SYSCAT.SERVERS view.

  remote-authorization-name
  : Is the schema to which the table or view belongs.

remote-table-name
Is the unique name or an alias of the table or view.

- If the data source doesn't support schemas, *remote-object-name* is:

  data-source-name.remote-table-name

| If *remote-authorization-name* or *remote-table-name* is case-sensitive at the data
| source, enclose it in delimiters. For example:

```
sybase1."myschema"."myview"
```

| Be sure to put the period that separates these names outside the delimiters.
| Otherwise, the period is considered part of the delimited string and does not act as
| a separator. For example, A."B"."C" is the correct form, but A."B.C" is not. Delimited
| identifiers must be used for table and view names that are case-sensitive at the
| data source.

Notes

- If you're creating a nickname for a table or view stored in a single-schema RDB database, *remote-object-name* is:

  data-source-name.remote-table-name

  If you're creating a nickname for a table or view stored in a multi-schema RDB database, *remote-object-name* is:

  data-source-name.remote-authorization-name.remote-table-name
- The remote table must be an existing table or view on a remote server.
- An unqualified nickname cannot have the same fully-qualified name as an existing table, view, or nickname.
- Because data types might be incompatible between data sources, DataJoiner makes minor adjustments to store remote catalog data locally as needed. See "Data Type Mappings" on page 26 for details.
- NAME fields in SYSCAT.COLUMNS and SYSCAT.INDEXES for remote columns and indexes are restricted to 18 characters in length. If a name is longer than 18 characters, DataJoiner truncates the name to 18 characters. See "How DataJoiner Enforces Limits on Column Names and Index Names" on page 50 for information about how DataJoiner enforces this limit when truncation causes the name to no longer be unique.
- For more information, see "Nicknames" on page 23.

Examples

Example 1: Create a nickname for a user-defined table, DEPT, owned by PERSON on a data source with server name DB2MVS1.

```
CREATE NICKNAME DEPT FOR DB2MVS1.PERSON.DEPT
```

Example 2: Create a nickname for system object table, SYSOBJECTS, on data source SYBASE1.

```
   CREATE NICKNAME USERID.A1 FOR SYBASE1."dbo"."sysobjects"
```

Example 3: Select all records from the remote table DEPT. (You must use a nickname for a remote table. You *cannot* reference a remote table directly as shown in the following example except in a pass-through session.)

`SELECT * FROM DB2MVS1.PERSON.DEPT`	Invalid
`SELECT * FROM DEPT`	Valid after nickname DEPT is created

CREATE REVERSE TYPE MAPPING

Use this statement to create a mapping between:

- A data type that's local to DataJoiner; and
- A corresponding data type that you want for a column in a table that you plan to create from DataJoiner

When you create the table, the corresponding data type will be defined for the column.

A data type mapping that determines what type is to be defined at a data source for a table column is called a *reverse type mapping.*

You need to create a reverse type mapping only if an existing one doesn't meet your requirements. To find out what default reverse type mappings exist, you might check "Appendix B. Default Reverse Type Mappings" on page 155. For listings of all reverse type mappings—default and non-default—query the SYSCAT.REVTYPEMAPPINGS view (described in "SYSCAT.REVTYPEMAPPINGS" on page 184).

For more information about reverse type mappings, see "Data Type Mappings" on page 26 .

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold either SYSADM or DBADM authority on the DataJoiner database.

Syntax

```
►►──CREATE REVERSE TYPE MAPPING──┬─────────────────────┬──FROM──┤ Remote Server ├──────────►
                                 └─type-mapping-name───┘

►──┬──────────┬──TYPE──┤ Remote Type ├──TO──┤ DataJoiner Schema and Type ├──────────────►◄
   └─DISTINCT─┘
```

Remote Server

```
├──┬─SERVER──server-name──────────────────────────────────────────────┬──────────────────┤
   ├─SERVER TYPE──server-type──VERSION──server-version──┤ Protocol ├──┤
   ├─SERVER TYPE──server-type──VERSION──server-version────────────────┤
   └─SERVER TYPE──server-type─────────────────────────────────────────┘
```

Protocol

```
├──PROTOCOL──"server-protocol"──────────────────────────────────────────┤
```

Remote Type

```
├──┬──────────────────────┬──data-source-data-type──────────────────────►
   └─data-source-schema.──┘

►──┬──────────────────────────────────────────┬─────────────────────────┤
   └─(──p──┬──────┬──)──┬────────────────┬────┘
           └─,s───┘     └─FOR BIT DATA───┘
```

DataJoiner Schema and Type

```
├──┬───────────────────────┬──┤ DataJoiner Type ├──┬────────────┬───────┤
   └─datajoiner-schema.────┘                       └─NOT NULL───┘
```

DataJoiner Type

```
├──┬─┬─INTEGER─┬──────────────────────────────────────────────────────────────┬──┤
   │ └─INT─────┘                                                              │
   ├─SMALLINT─────────────────────────────────────────────────────────────────┤
   ├─┬─DOUBLE────────────┬────────────────────────────────────────────────────┤
   │ ├─DOUBLE PRECISION──┤                                                    │
   │ └─FLOAT─────────────┘                                                    │
   ├──┬─DECIMAL─┬──┬─────────────────────────────────────────────────────┬────┤
   │  ├─DEC─────┤  └─(──┬─p───────┬──┬──────────┬──)──┬────────┬─────────┘    │
   │  ├─NUMERIC─┤       └─[p..p]──┘  ├─,s───────┤     ├─p=s────┤              │
   │  └─NUM─────┘                    └─,[s..s]──┘     ├─p>s────┤              │
   │                                                  ├─p<s────┤              │
   │                                                  ├─p>=s───┤              │
   │                                                  ├─p<=s───┤              │
   │                                                  └─p<>s───┘              │
   ├──┬──┬─CHARACTER─┬──┬──────────────────────────┬──┬──┬──────────────────┬─┤
   │  │  └─CHAR──────┘  ├─(──integer──)────────────┤  │  └─FOR BIT DATA─────┘ │
   │  │                 └─[integer..integer]───────┘  │                       │
   │  ├──┬─VARCHAR──────────────┬──┬─(──integer──)──────────┬─┤               │
   │  │  ├─CHARACTER──VARYING───┤  └─[integer..integer]─────┘ │               │
   │  │  └─CHAR──VARYING────────┘                             │               │
   │  └─LONG VARCHAR──────────────────────────────────────────┘               │
   ├──┬─BLOB────┬──(──integer──┬─────┬──)─────────────────────────────────────┤
   │  ├─CLOB────┤              ├─K───┤                                        │
   │  └─DBCLOB──┘              ├─M───┤                                        │
   │                           └─G───┘                                        │
   ├─GRAPHIC──┬─────────────────────────┬─────────────────────────────────────┤
   │          ├─(──integer──)───────────┤                                     │
   │          └─[integer..integer]──────┘                                     │
   ├─VARGRAPHIC──┬─(──integer──)─────────┬────────────────────────────────────┤
   │             └─[integer..integer]────┘                                    │
   ├─LONG VARGRAPHIC──────────────────────────────────────────────────────────┤
   ├─DATE─────────────────────────────────────────────────────────────────────┤
   ├─TIME─────────────────────────────────────────────────────────────────────┤
   ├─TIMESTAMP────────────────────────────────────────────────────────────────┤
   └─distinct-type-name───────────────────────────────────────────────────────┘
```

Description

type-mapping-name
: Name of the mapping that you want to create. The name cannot be the same as that of any other mapping that's described in the SYSCAT.REVTYPEMAPPINGS catalog view. And it can't be more than 18 characters long. If you don't specify a name, DataJoiner will supply one.

SERVER *server-name*
: Name of the data source to which the data type that you're mapping *from* is defined.

SERVER TYPE *server-type*
: Type of data source to which the data type that you're mapping *from* is defined.

Valid values for *server-type* are the same as those in the SERVER_TYPE column of SYSCAT.SERVERS (see “SYSCAT.SERVERS” on page 187). These values cannot exceed 30 characters. Delimited identifiers will be converted to uppercase.

VERSION *server-version*
Identifies the version of the data source. Valid values for *server-version* are composed of digits and, optionally, one or two decimal points. Do not enclose these values in quotes or place a decimal point at the end of them. For example:

```
VERSION 7
VERSION 8.0.3
```

The valid values for this field match the allowed values for the VERSION column in the catalog view SYSCAT.SERVERS. For details, see “SYSCAT.SERVERS” on page 187. *server-version* can have up to 18 characters.

If the data source data type is built-in, specifying a value for *server-version* ensures that the mapping that you’re defining will apply to this and all future versions of the specified server type (and protocol, if present).

PROTOCOL *″server-protocol″*
Identifies the protocol used in accessing *data-source-data-type*. Valid values for *″server-protocol″* are the same as those in the SERVER_PROTOCOL column of SYSCAT.SERVERS (described in “SYSCAT.SERVERS” on page 187). *″server-protocol″* is case-sensitive and cannot exceed 30 characters.

DISTINCT
Indicates that the data type denoted by **Remote Type** is user-defined.

data-source-schema.
Schema of the data type that you’re mapping from.

data-source-data-type
Data type that you’re mapping from.

p For information about this parameter, see page 93.

s For information about this parameter, see page 93.

FOR BIT DATA
Indicates whether the type denoted by *data-source-data-type* is for bit data. DataJoiner will determine this attribute if it is not specified on a character data type.

datajoiner-schema.
Schema of the data type that you’re mapping to. This type is defined to the DataJoiner database. If this type is built-in, the schema is SYSIBM. Otherwise, if the type is user-defined, the schema is either SYSFUN or a schema that a user has created.

DataJoiner Type
The data type that you’re mapping to. It must be defined to the DataJoiner database. If it’s built-in, do not specify the long form; for example, specify CHAR instead of CHARACTER. For an explanation of this clause, see the section on CREATE TABLE in the *DB2 SQL Reference for common servers.*

If you specify an incorrect data type, or a value that isn't valid for *integer*, you'll receive an error message (for example, SQLCODE -204).

p For a decimal data type, *p* specifies the maximum number of digits that a value can have. For other character data types, *p* specifies the maximum number of characters that a value can have. If you don't specify *p* and the data type requires it, DataJoiner will determine the best match.

For Informix DATETIME data types, *p* specifies the first precision. In this instance, use the field qualifier values listed in Table 5.

Table 5. Field Qualifier Values for Informix DATETIME Data Types

Precision	**Qualifier**
year	0
month	2
day	4
hour	6
minute	8
second	10
fraction(1)	11
fraction(2)	12
fraction(3)	13
fraction(4)	14
fraction(5)	15

[p..p]
For a decimal data type, *[p..p]* specifies the minimum and maximum number of digits that a value can have. For other character data types, *[p..p]* specifies the minimum and maximum number of characters allowed for an entire value. In all cases, the maximum must equal or exceed the minimum; and both numbers must be valid with respect to the data type.

For Informix DATETIME data types, *[p..p]* specifies an inclusive range of the first precision. Use the field qualifier values listed in Table 5.

s For a decimal data type, *s* specifies the maximum number of digits that are allowed to the right of the decimal point. The number you specify must be valid with respect to the data type. If you don't specify a number and the data type requires one, DataJoiner will determine the best match.

For Informix DATETIME data types, *s* specifies the second precision value. For example, in Informix, DATETIME year to month is used to specify the month, which should be specified with a value of 2 as in Table 5.

[s..s]
For a decimal data type, *[s..s]* specifies the minimum and maximum number of digits allowed to the right of the decimal point. The maximum must equal or exceed the minimum, and both numbers must be valid with respect to the data type.

For Informix datetime types, *[s..s]* specifies an inclusive range of the first precision. Use the field qualifier values listed in Table 5 on page 93.

p_operand_s
For a decimal data type, *p_operand_s* specifies a comparison between the maximum number of digits that a value can have and the maximum number of digits allowed to the right of the decimal point. For example, the operand = indicates that the numbers are the same, and the operand > indicates that the first number is greater than the second. Specify this attribute only if you need to enforce this level of checking.

CHARACTER*(integer)* or **CHAR***(integer)*
For a fixed-length character string of length *integer*, where *integer* is any whole number from 1 through 254. The default is 1.

CHARACTER*[integer..integer]* or **CHAR***[integer..integer]*
For a fixed-length character string whose length lies within the range denoted by *[integer..integer].* The lower bound of the range can be as low as 1; the upper bound can be as high as 254.

VARCHAR*(integer)*, or **CHARACTER VARYING***(integer)*, or **CHAR VARYING***(integer)*
For varying-length character strings of maximum length *integer*, where *integer* is any whole number from 1 through 4000.

VARCHAR*[integer..integer],* or **CHARACTER VARYING***[integer..integer],* or **CHAR**

VARYING*[integer..integer]*
For varying-length character strings whose lengths lie within the range denoted by *[integer..integer].* The lower bound of the range can be as low as 1; the upper bound can be as high as 4000.

GRAPHIC*(integer)*
For a fixed-length graphic string of length *integer*, where *integer* is any whole number from 1 through 127. The default is 1.

GRAPHIC*[integer..integer]*
For a fixed-length graphic string whose lengths lies within the range denoted by *[integer..integer].* The lower bound of the range can be as low as 1; the upper bound can be as high as 127.

VARGRAPHIC*(integer)*
For varying-length graphic strings of maximum length *integer*, where *integer* is any whole number from 1 through 2000.

VARGRAPHIC*[integer..integer]*
For varying-length graphic strings whose lengths lie within the range denoted by *[integer..integer].* The lower bound of the range can be as low as 1; the upper bound can be as high as 2000.

Notes

- Together, the values for *datajoiner-schema* and **DataJoiner Type** make up the fully-qualified name of the data type that you’re mapping from.

- Together, the value for *data-source-schema* and the value for *data-source-data-type* make up the fully-qualified name of the data type that you're mapping to.
- If *data-source-data-type* is case-sensitive at the data source, put double quotes around the values that you assign to *data-source-schema* (before the period) and to *data-source-data-type.* For example:

  ```
  CREATE REVERSE TYPE MAPPING ADDRESS_BOOK
     FROM SYBASE
     DISTINCT TYPE "jones"."address"
     TO ADDRESS
  ```

  In addition, if you're submitting the CREATE TYPE MAPPING statement from a UNIX command prompt, put single quotes around the whole statement. For example:

  ```
  'CREATE REVERSE TYPE MAPPING ADDRESS_BOOK
     FROM SYBASE
     DISTINCT TYPE "jones"."address"
     TO ADDRESS'
  ```

  If you're submitting the statement from a Windows NT command prompt, precede each double quote with a backward slash. For example:

  ```
  CREATE REVERSE TYPE MAPPING ADDRESS_BOOK
     FROM SYBASE
     DISTINCT TYPE \"jones\".\"address\"
     TO ADDRESS
  ```

Examples

You plan to use DataJoiner's CREATE TABLE statement to create Oracle 8 tables that can contain dollar amounts. Accordingly, you create a distinct data type for dollar amounts at the Oracle 8 data sources, and a similar type in a schema called COSTS at the DataJoiner RDBMS. You call both types MONEY. Before you can create the tables, you need to create a reverse type mapping between the two data types.

```
CREATE REVERSE TYPE MAPPING MONEYMAP
   FROM SERVER TYPE ORACLE
   VERSION 8.0.3
   TYPE MONEY
   TO COSTS.MONEY
```

For another example, see page 111.

CREATE SERVER MAPPING

Use the CREATE SERVER MAPPING statement to define a data source to DataJoiner as a server. After you create this definition, you can use DataJoiner to perform operations such as:

- Creating nicknames by which DataJoiner can reference tables, views, and stored procedures stored at the data source
- Creating tables at the data source
- Producing result tables that combine information from this data source with information from other data sources

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold SYSADM or DBADM authority on the DataJoiner database.

Syntax

```
►►──CREATE SERVER MAPPING FROM──server-name──TO NODE──"node-name"──────────────────►

►──┬─────────────────────────────────────┬──TYPE──server-type──────────────────────►
   └─DATABASE──"remote-database-name"────┘

►─VERSION──server-version──PROTOCOL──"protocol-name"──────────────────────────────►

   ┌────────────────────────────────────────────┐
   ▼                                            │
►──┴───┬─CPU RATIO──value──────────────────────┬┴──────────────────────────────────►◄
       ├─IO RATIO──value───────────────────────┤
       ├─COMM RATE──value──────────────────────┤
       └─┬─AUTHID──name──PASSWORD──password──┬─┘
         └─AUTHID──name──────────────────────┘
```

Description

FROM *server-name*
: Identifies the name by which the specified data source database will be known locally. *server-name* is a long identifier.

TO NODE *″node-name″*
: Identifies the node on which the specified data source resides. This field is protocol dependent and its value varies. For example, DB2 values correspond to entries in a node directory. An Oracle value corresponds to an entry in the tnsnames.ora file. A Sybase value corresponds to an entry in the interfaces file. To find out how to determine *″node-name″* for your data sources, see “Finding Node Names” on page 51 . *″node-name″* is a long identifier with a maximum of 70 characters and must be enclosed in double-quotes.

DATABASE *″remote-database-name″*
: Identifies the specific database on the data source accessed when referring to *server-name*. This field is protocol dependent and its value varies. This field is not required for Oracle data sources because Oracle instances contain only one database. For Sybase and DB2, this value corresponds to a specific database within an instance or, if DB2 for OS/390, the database LOCATION value.

The valid values for this field match the allowed values for the DBNAME column in the catalog view SYSCAT.SERVERS. See “SYSCAT.SERVERS” on page 187 for details.

DataJoiner defines a unique index on the NODE and DBNAME columns in SYSCAT.SERVERS. This unique index prevents you from specifying an identical combination of values for NODE and DATABASE with two different values in the SERVER column. *remote-database-name* is a long identifier and must be enclosed in double-quotes.

TYPE *server-type*
: Identifies the type of data source. Example values include ORACLE, INFORMIX, and DB2/MVS.

The valid values for this field match the allowed values for the TYPE column in the catalog view SYSCAT.SERVERS. See “SYSCAT.SERVERS” on page 187 for details. Delimited identifiers are converted to uppercase automatically.

server-type is an identifier with a maximum of 30 characters.

VERSION *server-version*
: | Identifies the version of the data source. Valid values for *server-version* are composed of digits and, optionally, one or two decimal points. Do not enclose these values in quotes or place a decimal point at the end of them. For example:

```
| VERSION 7
| VERSION 8.0.3
```

| The valid values for this field match the allowed values for the VERSION column in the catalog view SYSCAT.SERVERS. For details, see “SYSCAT.SERVERS” on page 187.

server-version can have up to 18 characters.

PROTOCOL *"protocol-name"*
Identifies the data access protocol used by the data source. The set of valid values for this field matches the allowed values for the SERVER_PROTOCOL column in the catalog view SYSCAT.SERVERS. Examples are “drda” and “db2ra”. See “SYSCAT.SERVERS” on page 187 for details.

"protocol-name" is case-sensitive and cannot exceed 30 characters.

CPU RATIO *value*
Represents how much faster or slower the data source CPU is compared to the local CPU. Valid values are unsigned floating-point constants, unsigned decimal constants, or unsigned integer constants. For example, if the remote CPU is twice as fast as the local CPU, enter the value 0.5. If this value is not specified, see CPU_RATIO in “SYSCAT.SERVERS” on page 187 for the default value.

IO RATIO *value*
Represents how much faster or slower the data source workstation I/O is compared to the local workstation. Valid values for *value* are unsigned floating-point constants, unsigned decimal constants, or unsigned integer constants. For example, if the remote workstation I/O is twice as fast as the local workstation I/O, enter the value 0.5. If this value is not specified, see IO_RATIO in “SYSCAT.SERVERS” on page 187 for the default value.

COMM RATE *value*
Represents the communication rate between the data source workstation and the local workstation. *value* represents the number of bytes per second; use unsigned integer constants.

AUTHID *name*
Specifies the authorization name under which any necessary actions, such as binding packages that DataJoiner requires, are performed at the data source when the CREATE SERVER MAPPING statement is processed. The name must hold the authority (BINDADD or its equivalent) that the necessary actions require.

Type *name* in the case required by the data source; if any letters in *name* must be in lowercase, enclose *name* in quotation marks. If you don’t specify an authorization name, the one under which you connected to DataJoiner will be used.

PASSWORD *password*
Specifies the password associated with the authorization name represented by *name.* Type *password* in the case required by the data source; if any letters in *password* must be in lowercase, enclose *password* in quotation marks.

If you specify *name* but not *password*, you’re indicating that there’s no need to validate the password.

Notes

Identifiers without delimiters are converted to uppercase in SYSCAT.SERVERS. The following examples show how you can use quotes to pass case-sensitive characters to specific columns in SYSCAT.SERVERS.

Examples

Example 1: Create a server mapping with a specified database.

```
CREATE SERVER MAPPING FROM SYBASE1
   TO NODE "Sybase1"
   DATABASE "test_db"
   TYPE SYBASE
   VERSION 10.0
   PROTOCOL "ctlib"
```

Example 2: Create a server mapping with a specified CPU ratio.

```
CREATE SERVER MAPPING FROM ORACLE1
   TO NODE "mvpdb0"
   TYPE ORACLE
   VERSION 7.0
   PROTOCOL "sqlnet"
   CPU RATIO 2.0
```

CREATE SERVER OPTION

Use the CREATE SERVER OPTION statement to set configuration options that persist for multiple connections to specific servers, or to servers of a specific type. With these options, you can optimize performance and control certain interactions between DataJoiner and data sources. The catalog view SYSCAT.SERVER_OPTIONS contains the server option settings. For an introduction to the CREATE SERVER OPTION statement, see “Server Options” on page 11.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold SYSADM or DBADM authority on the DataJoiner database.

Syntax

```
►►──CREATE SERVER OPTION──option-name──FOR──┤ Remote Server Clause ├──────────────►

►─SETTING──option-value──────────────────────────────────────────────────────────►◄
```

Remote Server Clause

```
├──┬─SERVER──server-name─────────────────────────────────────────────────┬──────────┤
   ├─SERVER TYPE──server-type──VERSION──server-version──┤ Protocol ├──┤
   ├─SERVER TYPE──server-type──VERSION──server-version────────────────┤
   └─SERVER TYPE──server-type─────────────────────────────────────────┘
```

Protocol

```
├──PROTOCOL──"server-protocol" ──────────────────────────────────────────────────────┤
```

Description

option-name
: Is the name of the server option. Valid *option-name* values are listed in Table 2 on page 17. *option-name* is an identifier; it cannot exceed 30 characters.

SERVER *server-name*
: Identifies the server to which *option-name* applies. This field must match an entry in the SERVER column of the SYSCAT.SERVERS catalog view (see “SYSCAT.SERVERS” on page 187). *server-name* is a long identifier.

SERVER TYPE *server-type*

Identifies the type of server to which *option-name* applies. Example values include DB2/MVS, ORACLE, and informix. Valid values for *server-type* match those listed for the SERVER_TYPE column of SYSCAT.SERVERS_OPTIONS catalog view (see Table 43 on page 196). Values for *server-type* values cannot exceed 30 characters. Delimited identifiers will be converted to uppercase.

VERSION *server-version*

Identifies the version of the server to which *option-name* applies. Valid values for *server-version* are composed of digits and, optionally, one or two decimal points. Do not enclose these values in quotes or place a decimal point at the end of them. For example:

```
VERSION 7
VERSION 8.0.3
```

The valid values for this field match the allowed values for the VERSION column in the catalog view SYSCAT.SERVERS. For details, see “SYSCAT.SERVERS” on page 187 .

server-version can have up to 18 characters.

PROTOCOL *"server-protocol"*

Identifies the server protocol to which *option-name* applies. Field values must match one of the allowed entries for the SERVER_PROTOCOL column of the SYSCAT.SERVERS_OPTIONS catalog view (see Table 43 on page 196). *"server-protocol"* is case-sensitive and cannot exceed 30 characters.

SETTING *option-value*

Identifies the option setting. Valid values for *option-value* depend on the value entered for *option-name*. See the examples below. *option-value* can be an integer constant, floating-point constant, decimal constant, or character literal. If it’s a character literal, it can contain up to 254 characters and must be enclosed in single quotes. For descriptions of valid *option-value*s, see “Summary of Server Options and Their Settings” on page 17.

Notes

- Server options can be in uppercase or lowercase (case does not matter).
- If you set an option to one value for a server type, and set this same option to another value for an instance of the type, the second value overrides the first one for the instance. For example, suppose that you set two_phase_commit to ’y’ for server type ORACLE and to ’n’ for an Oracle server named DELPHI. As a result, DELPHI won’t operate in two-phase commit mode, but all other Oracle servers will.
- There is a unique index on the catalog columns (OPTION, SERVER, SERVER_TYPE, SERVER_VERSION, SERVER_PROTOCOL); therefore, an existing server option cannot be duplicated. Use the ALTER SERVER OPTION statement to change server options.
- For an overview of the SQL for setting server options, see “Server Options” on page 11 . For additional information about the CREATE SERVER OPTION statement, see the *DataJoiner Administration Supplement.*

Examples

Example 1: Enable two-phase commit for server WRANGLER.

```
CREATE SERVER OPTION TWO_PHASE_COMMIT
   FOR SERVER WRANGLER
   SETTING 'y'
```

Example 2: Indicate that the collating sequence at all Oracle data sources is case-insensitive.

```
CREATE SERVER OPTION COLSEQ
   FOR SERVER TYPE ORACLE
   SETTING 'i'
```

Example 3: Send all Oracle passwords to, and validate them at, Oracle data sources.

```
CREATE SERVER OPTION PASSWORD
   FOR SERVER TYPE ORACLE
   SETTING 'y'
```

Example 4: Make push down optimization available for all Sybase data sources.

```
CREATE SERVER OPTION PUSHDOWN
   FOR SERVER TYPE SYBASE
   SETTING 'y'
```

CREATE STORED PROCEDURE NICKNAME

Use the CREATE STORED PROCEDURE NICKNAME statement to create a nickname for a remote stored procedure. This statement is actually an extension to the CREATE NICKNAME statement. CREATE STORED PROCEDURE NICKNAME creates entries in the system catalog; these entries can be seen in the catalog views SYSCAT.PROCEDURES and SYSCAT.PROCPARMS.

Before using nicknames, review “Considerations and Restrictions” on page 24 to be sure you understand the restrictions on their usage.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold either SYSADM or DBADM authority on the DataJoiner database. The authorization ID must also have authorization at the data source to access both the data source system catalog and the stored procedure that you’re creating a nickname for.

Syntax

```
►►──CREATE STORED PROCEDURE NICKNAME──stored-procedure-nickname──────────────────────►

►─FOR──remote-object-name────────────────────────────────────────────────────────────►

►─────────────────────────────────────────────────────────────────────────────────────►◄
   │                                                                    │
   │     ┌─,──────────────────────────────────────────────────────┐     │
   │     ▼                                                        │     │
   └─(───────┬─IN────┬───┬────────────────┬─┤ Remote Type ├───────┴──)──┘
             ├─OUT───┤   └─parameter-name─┘
             └─INOUT─┘
```

Remote Type

```
├──REMOTE TYPE──data-source-data-type──┬──────────┬──┤ Local Type ├──────────────────┤
                                       └─NOT NULL─┘
```

Local Type

```
├──┬──────────────────────────────────────────────────────────────┬──────────────────┤
   └─LOCAL TYPE──┤ DataJoiner Data Type ├──┬──────────┬───────────┘
                                           └─NOT NULL─┘
```

DataJoiner Data Type

```
|──┬──┬─INTEGER─┬────────────────────────────────────────────┬────────────|
   │  └─INT─────┘                                            │
   ├─SMALLINT────────────────────────────────────────────────┤
   ├──┬─DOUBLE───────────┬───────────────────────────────────┤
   │  ├─DOUBLE PRECISION─┤                                   │
   │  └─FLOAT────────────┘                                   │
   ├──┬─DECIMAL─┬──┬───────────────────────────────┬─────────┤
   │  ├─DEC─────┤  └─(─integer──┬────────────┬─)──┘         │
   │  ├─NUMERIC─┤               └─,─integer──┘               │
   │  └─NUM─────┘                                            │
   ├──┬──┬─CHARACTER─┬──┬──────────────┬──┬──┬──────────────┬┤
   │  │  └─CHAR──────┘  └─(─integer─)──┘  │  └─FOR BIT DATA─┘│
   │  ├──┬─VARCHAR───────────────┬─(─integer─)─┤             │
   │  │  ├─CHARACTER──VARYING────┤             │             │
   │  │  └─CHAR──VARYING─────────┘             │             │
   │  └─LONG VARCHAR───────────────────────────┘             │
   ├──┬─BLOB───┬──(─integer──┬─────┬──)──────────────────────┤
   │  ├─CLOB───┤             ├─K───┤                         │
   │  └─DBCLOB─┘             ├─M───┤                         │
   │                         └─G───┘                         │
   ├─GRAPHIC──┬───────────────┬──────────────────────────────┤
   │          └─(─integer─)───┘                              │
   ├─VARGRAPHIC──(─integer─)─────────────────────────────────┤
   ├─LONG VARGRAPHIC─────────────────────────────────────────┤
   ├─DATE────────────────────────────────────────────────────┤
   ├─TIME────────────────────────────────────────────────────┤
   ├─TIMESTAMP───────────────────────────────────────────────┤
   └─distinct-type-name──────────────────────────────────────┘
```

Description

stored-procedure-nickname
Represents a short identifier with or without a qualifier. If no qualifier is supplied, the authorization ID of the statement is used as the default value. A nickname cannot exceed 8 characters. The qualifier must not be SYSIBM.

remote-object-name
Represents a two- or three-part name.

- If you're creating a nickname for a table or view at a data source that supports schemas, *remote-object-name* is:

  data-source-name.remote-authorization-name.remote-table-name

  —where:

  data-source-name
  Is a value in the SERVER column of the SYSCAT.SERVERS view. This value can't have more than 18 characters.

  remote-authorization-name
  Is the schema to which the table or view belongs. This value can't have more than 128 characters.

remote-table-name
: Is the unique name or an alias of the table or view. This value can't have more than 128 characters.

- If the data source doesn't support schemas, *remote-object-name* is:

  data-source-name.remote-table-name

| If *remote-authorization-name* or *remote-table-name* is case-sensitive at the data source, enclose it in delimiters. For example:

```
sybase1."myschema"."myview"
```

| Be sure to put the period that separates these names outside the delimiters. Otherwise, the period is considered part of the delimited string and does not act as a separator. For example, A."B"."C" is the correct form, but A."B.C" is not. Delimited identifiers must be used for stored procedure names that are case-sensitive at the data source.

data-source-data-type
: The data type of the stored procedure as defined on the data source. This value cannot exceed 128 characters.

DataJoiner Data Type
: Providing one of the listed data types (shown in the syntax diagram on page "Syntax" on page 103) is optional. If no data type is provided, DataJoiner uses the catalog view SYSCAT.SERVER_DATATYPES to derive the DataJoiner data type. For the definitions for the listed data types, see the CREATE TABLE section in *DATABASE 2 SQL Reference for common servers*.

Notes

- A stored procedure nickname must not be the same as the name of a local stored procedure; otherwise, the local stored procedure is used instead of the procedure identified by the nickname.
- The parameters specified in the CREATE STORED PROCEDURE NICKNAME statement should describe only the input and output parameters of a stored procedure, not any result sets that the stored procedure returns.
- If you specify both a REMOTE TYPE and a LOCAL TYPE for an input or output parameter, no checking is done at CREATE STORED PROCEDURE NICKNAME run time to ensure that the data types are compatible.
- Remote types correspond to the remote schema and remote type name specified in the catalog view SYSCAT.SERVER_DATATYPES.
- Because data types might be incompatible between data sources, DataJoiner makes minor adjustments to store remote catalog data locally as needed. See "Data Type Mappings" on page 26 for details.

Examples

Example 1: Create a nickname for a stored procedure that resides at a Sybase data source (no parameters).

```
CREATE STORED PROCEDURE NICKNAME S_SIMPLE FOR SYBASE1."j15user1"."simple"
```

Example 2: Create a nickname for a stored procedure that resides at a Sybase data source (input parameter).

```
CREATE STORED PROCEDURE NICKNAME S_IN
   FOR SYBASE1."j15user1"."simp_in"
   (IN  C1
   REMOTE TYPE "dbo"."char"(15))
```

Note: The data type specification matches the remote schema and remote type name specified in SYSCAT.SERVER_DATATYPES.

Example 3: Create a nickname for a stored procedure that resides at a Sybase data source (input and output parameters).

```
CREATE STORED PROCEDURE NICKNAME S_INOUT
   FOR SYBASE1."j15user1"."simp_inout"
   (IN QUANTITY
   REMOTE TYPE "dbo"."int",
   OUT FRUIT
   REMOTE TYPE "dbo"."char"(5))
```

CREATE TABLE

Use this statement to create tables at the following types of data sources: DB2 Family, Generic, Informix, Microsoft SQL Server, Oracle, SQL Anywhere, and Sybase. With this statement, you can define a data source table's name, columns, and primary key (or keys). You can also define an option that's unique to the data source's native CREATE TABLE statement; for example, the IN DATABASE option in the DB2 for OS/390 CREATE TABLE statement. To define additional attributes—for example, foreign keys, check constraints, and table space options—use the data source's native CREATE TABLE statement in a DataJoiner pass-through session.

You cannot use this statement to create tables for the DataJoiner database. To create these tables, use the DB2 for CS CREATE TABLE statement.

For the combined syntax of the DataJoiner and DB2 for CS CREATE TABLE statements, see "Appendix C. Combined DataJoiner and DB2 for CS Syntax for CREATE TABLE" on page 167.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold one of the following permissions on the DataJoiner database:

- SYSADM authority
- DBADM authority
- CREATETAB privilege

The authorization ID must also have authority at the data source to create data source tables.

Syntax

```
                                 ┌─,──────────────────────────┐
                                 ▼                            │
►►──CREATE TABLE──table-name──(──┬─┤ Column Definition ├──────┬┴─)──────────►
                                 └─┤ Primary Key Constraint ├─┘

►──IN──server-name──┬─────────────────────────────────┬──────────────────►◄
                    └─REMOTE OPTION──'remote-option'──┘
```

Column Definition

```
                                   ┌──────────────────┐
                                   ▼                  │
├──column-name──┤ Data Type ├──────┬──────────────┬───┴─────────────────────────┤
                                   ├─NOT NULL─────┤
                                   └─PRIMARY KEY──┘
```

Data Type

```
├──┬──┬─INTEGER─┬─────────────────────────────────────────────────────────┬────┤
   │  └─INT─────┘                                                         │
   ├─SMALLINT─────────────────────────────────────────────────────────────┤
   ├──┬─DOUBLE───────────┬────────────────────────────────────────────────┤
   │  ├─DOUBLE PRECISION─┤                                                │
   │  └─FLOAT────────────┘                                                │
   ├──┬─DECIMAL─┬──┬────────────────────────────────┬─────────────────────┤
   │  ├─DEC─────┤  └─(──integer──┬──────────────┬─)─┘                     │
   │  ├─NUMERIC─┤                └─,──integer───┘                         │
   │  └─NUM─────┘                                                         │
   ├──┬──┬─CHARACTER─┬──┬──────────────────┬──┬──┬────────────────────┬───┤
   │  │  └─CHAR──────┘  └─(──integer──)────┘  │  │               (1)  │   │
   │  ├──┬─VARCHAR───────────────┬─(──integer──)─┤  └─FOR BIT DATA─────┘   │
   │  │  ├─CHARACTER──VARYING────┤               │                        │
   │  │  └─CHAR──VARYING─────────┘               │                        │
   │  └─LONG VARCHAR─────────────────────────────┘                        │
   │                                                                      │
   ├──┬─BLOB───┬──(──integer──┬─────┬──)──────────────────────────────────┤
   │  ├─CLOB───┤              ├─K───┤                                     │
   │  └─DBCLOB─┘              ├─M───┤                                     │
   │                          └─G───┘                                     │
   ├─GRAPHIC──┬───────────────────┬───────────────────────────────────────┤
   │          └─(──integer──)─────┘                                       │
   ├─VARGRAPHIC──(──integer──)────────────────────────────────────────────┤
   ├─LONG VARGRAPHIC──────────────────────────────────────────────────────┤
   ├─DATE─────────────────────────────────────────────────────────────────┤
   ├─TIME─────────────────────────────────────────────────────────────────┤
   ├─TIMESTAMP────────────────────────────────────────────────────────────┤
   └─distinct-type-name───────────────────────────────────────────────────┘
```

Notes:

1. You can specify FOR BIT DATA in random order with the other column constraints that follow.

Primary Key Constraint

```
                                                            ┌─,───────────────┐
                                                            ▼                 │
├──┬──────────────────────────────────┬──PRIMARY KEY──(─────column-name──────┴──)──┤
   └─CONSTRAINT──constraint-name──────┘
```

Description

table-name
: Name of the table that you're creating, with or without a qualifier. The name cannot be the same as any alias, view name, or other table name in the DataJoiner catalog.

  You can qualify the name with the name of a local schema (other than SYSIBM, SYSCAT, and SYSSTAT). If you do, this schema name will be used as the qualifier in the table's fully-qualified name. If you don't specify a qualifier, the qualifier in the fully-qualified name will be your authorization ID at the data source.

  Note on the table's nickname: When the table is created, DataJoiner automatically creates a nickname for it. If you specified a table name qualified by a local schema name, the fully-qualified nickname will be the same as the name and qualifier that you specified. If you specified a table name without a qualifier, the fully-qualified nickname will be the name that you specified, qualified by your local authorization ID.

column-name
: Name of a table column, without a qualifier. The name cannot be the same as that of any other column in the table.

Data Type
: The data type of the values of the column specified in the *column-name* parameter. This data type is defined to the DataJoiner database. It's valid for the table that you're creating only if there's a reverse mapping between it and a data type at the data source where the table resides. To find out what reverse mappings exist between data source types and types defined to the DataJoiner database, query the SYSCAT.REVTYPEMAPPINGS catalog view (documented on page 184). This view shows both default and user-defined mappings. For lists of default mappings only, see "Appendix B. Default Reverse Type Mappings" on page 155. If you don't find a mapping that you need, you can create it with the CREATE REVERSE TYPE MAPPING statement (described on page 89).

  For an explanation of the keywords and parameters in the Data Type clause, see *DATABASE 2 SQL Reference for common servers*.

NOT NULL
: The column specified in the *column-name* parameter cannot contain nulls. If you don't specify this option, the column will be marked nullable by default.

PRIMARY KEY
: Indicates that the column that you're defining is to be a primary key. Be aware that a column used as a primary key cannot be nullable.

  If you want the primary key to consist of a single column, you can specify the key in either the Column Definition clause or in the Primary Key Constraint clause. For example, if you're defining a column named C, and you want to designate it as a primary key, you can do so in either of the following ways:

- In the Column Definition clause, specify:

  PRIMARY KEY

- In the Primary Key Constraint clause, specify:

  PRIMARY KEY (C**)**

Primary Key Constraint
: For an explanation of this clause, see *DATABASE 2 SQL Reference for common servers*. Be aware that the columns in a primary key cannot be nullable.

IN *server-name*
: Name of the data source at which the table is to reside.

REMOTE OPTION *'remote-option'*
: Any option within the data source's native CREATE TABLE DDL that's specific to the data source. For example, the CREATE TABLE DDL for DB2 for OS/390 includes syntax for specifying the database in which a table is to be stored. If you're creating a DB2 for OS/390 table that you want stored in a database called TEST01, you could specify:

```
REMOTE OPTION 'IN DATABASE TEST01'
```

Notes

- Some data sources do not support DDL in two-phase commit mode. If you're creating a table for such a data source, and this data source is configured for two-phase commit transactions, you need to reconfigure it so that it can commit the CREATE TABLE statement in one phase. To do this for a single unit of work, follow either of these procedures:
  - Use the SET SERVER OPTION statement to set the two_phase_commit server option to 'n'. Place the statement immediately after the CONNECT statement.
  - Place any CREATE REVERSE TYPE MAPPING statements that you code and the CREATE TABLE statement immediately after the CONNECT statement.

  To reconfigure a data source for one-phase commits indefinitely, run an ALTER SERVER OPTION statement in which the two_phase_commit option for the data source is set to 'N'.

  For information about the SET SERVER OPTION and ALTER SERVER OPTION statements, see "Using Server Options to Configure Data Sources" on page 11.
- After the table is created, you can use pass-through to define attributes for it that the data-source supports; for example, data source-specific constraints and indexes.

Examples

Example 1: Create a table with four columns at a Sybase data source called SYBASE1. Assume that:

- The DataJoiner and data source authorization IDs under which your CREATE TABLE statement will be processed are HOMEUSR1 and SYUSR1, respectively.

- You want the table's name to be qualified by SYUSR1 by default.
- You want the first two columns to comprise a primary key called LARGO.
- Earlier, in a pass-through session, you used Sybase SQL to create a segment, called MY_SEGMENT, to hold the table.

```
CREATE TABLE T1
   (C1 INT NOT NULL,
   C2 CHAR(2) NOT NULL,
   C3 CHAR(3),
   C4 CHAR(2),
   CONSTRAINT LARGO
   PRIMARY KEY (C1, C2))
   IN SYBASE1
   REMOTE OPTION 'ON MY_SEGMENT'
```

After this statement is processed, the table's fully-qualified name will be SYUSR1.T1, and its fully-qualified nickname will be HOMEUSR1.T1.

Example 2: You want to create Oracle 8 tables with decimal values that have:

- A maximum precision of 30
- A scale that can vary from 1 to 4 digits

Because the scale can vary, the Oracle data type for these values—NUMBER(30,4)—must be mapped to by a DataJoiner-supported data type with floating decimal points; namely, DOUBLE.

In one unit of work, you create:

- This mapping.
- A table T2 in an Oracle 8 data source called DELPHI. In T2, column C1 is to be the primary key. Column C2 is for the decimal values.

```
CONNECT TO LOCAL_DB;
CREATE REVERSE TYPE MAPPING NEWMAP
   FROM SERVER TYPE ORACLE
   VERSION 8.0.3
   TYPE NUMBER(30,4)
   TO SYSIBM.DOUBLE;
CREATE TABLE T2
   (C1 CHARACTER NOT NULL PRIMARY KEY,
   C2 DOUBLE)
   IN DELPHI
```

CREATE TYPE MAPPING

Use this statement to create a mapping between these data types:

- A data type of a column of a data source table or view that you plan to define to DataJoiner
- A corresponding data type that's local to DataJoiner. This corresponding data type will be defined locally for the column when you define the table or view to DataJoiner.

A data type mapping that determines what type is to be locally defined for a column of a data source table or view is called a *forward type mapping.*

You need to create a forward type mapping only if an existing one doesn't meet your requirements. To find out what default forward type mappings exist, you might check "Appendix A. Default Forward Type Mappings" on page 135. For listings of all forward type mappings—default and non-default—query the SYSCAT.SERVER_DATATYPES view (described in "SYSCAT.SERVER_DATATYPES" on page 190).

For more information about forward type mappings, see "Data Type Mappings" on page 26 .

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold SYSADM or DBADM authority on the DataJoiner database.

Syntax

```
►►──CREATE TYPE MAPPING──┬───────────────────┬──FROM──┤ Local Type ├──TO──────────►
                         └─type-mapping-name─┘

►──┤ Remote Server ├──┬──────────┬──TYPE──┤ Remote Type Name ├─────────────────►◄
                      └─DISTINCT─┘
```

Local Type

```
├──┬─────────────────────┬──┤ DataJoiner Data Type ├──┬──────────┬──────────────┤
   └─datajoiner-schema.──┘                            └─NOT NULL─┘
```

DataJoiner Data Type

```
├──┬─┬─INTEGER─┬──────────────────────────────────────────────┬───────────────┤
   │ └─INT─────┘                                              │
   ├─SMALLINT─────────────────────────────────────────────────┤
   ├─┬─DOUBLE────────────┬────────────────────────────────────┤
   │ ├─DOUBLE PRECISION──┤                                    │
   │ └─FLOAT─────────────┘                                    │
   ├──┬─DECIMAL─┬──┬──────────────────────────────────┬───────┤
   │  ├─DEC─────┤  └─(──integer──┬──────────────┬──)──┘       │
   │  ├─NUMERIC─┤                └─,──integer───┘             │
   │  └─NUM─────┘                                             │
   ├──┬──┬─CHARACTER─┬──┬───────────────────┬──┬──┬──────────────────┬─┤
   │  │  └─CHAR──────┘  └─(──integer──)─────┘  │  └─FOR BIT DATA─────┘ │
   │  ├──┬─VARCHAR──────────────┬─(──integer──)─┤                      │
   │  │  ├─CHARACTER──VARYING───┤               │                      │
   │  │  └─CHAR──VARYING────────┘               │                      │
   │  └─LONG VARCHAR────────────────────────────┘                      │
   ├──┬─BLOB────┬──(──integer──┬─────┬──)─────────────────────────────┤
   │  ├─CLOB────┤              ├─K───┤                                │
   │  └─DBCLOB──┘              ├─M───┤                                │
   │                           └─G───┘                                │
   ├─GRAPHIC──┬───────────────────┬───────────────────────────────────┤
   │          └─(──integer──)─────┘                                   │
   ├─VARGRAPHIC──(──integer──)────────────────────────────────────────┤
   ├─LONG VARGRAPHIC──────────────────────────────────────────────────┤
   ├─DATE─────────────────────────────────────────────────────────────┤
   ├─TIME─────────────────────────────────────────────────────────────┤
   ├─TIMESTAMP────────────────────────────────────────────────────────┤
   └─distinct-type-name───────────────────────────────────────────────┘
```

Remote Server

```
├──┬─SERVER──server-name────────────────────────────────────────────┬──────────┤
   ├─SERVER TYPE──server-type──VERSION──server-version──┤ Protocol ├─┤
   ├─SERVER TYPE──server-type──VERSION──server-version───────────────┤
   └─SERVER TYPE──server-type─────────────────────────────────────────┘
```

Protocol

```
├──PROTOCOL──"server-protocol"──────────────────────────────────────────────┤
```

Remote Type Name

```
├──┬────────────────────────┬─data-source-data-type──────────────────────────►
   └─data-source-schema.────┘
```

```
►──┬──────────────────────────────────────────────────────────────────┬──────────────────────┤
   └─(──┬─p──────┬──┬─────────┬──)──┬────────┬──┬───────────────┬─────┘
        └─[p..p]─┘  ├─,s──────┤     ├─p=s────┤  └─FOR BIT DATA──┘
                    └─,[s..s]─┘     ├─p>s────┤
                                    ├─p<s────┤
                                    ├─p>=s───┤
                                    ├─p<=s───┤
                                    └─p<>s───┘
```

Description

type-mapping-name
: Identifies the data type mapping. The name must not identify a data type mapping that is already described in SYSCAT.SERVER_DATATYPES. *type-mapping-name* cannot exceed 18 characters. If *type-mapping-name* is not specified, DataJoiner generates a unique name.

datajoiner-schema.
: Is the name of the schema of a data type that's defined to the DataJoiner database and that you want to map to a data type at a data source. If the data type at the DataJoiner database is built-in, the schema is SYSIBM. Otherwise, if this data type is user-defined, the schema is either SYSFUN or a schema that a user created.

DataJoiner Data Type
: Identifies a data type that's defined to the DataJoiner database and that you want to map to a data type at a data source. For an explanation of this clause, see the discussion of the CREATE TABLE statement in *DATABASE 2 SQL Reference for common servers* .

If you specify an incorrect data type, or a value that isn't valid for *integer*, you'll receive an error message (for example, SQLCODE -204).

SERVER TYPE *server-type*
: Identifies the type of server to which *data-source-data-type* is defined. Valid values for *server-type* are the same as those in the SERVER_TYPE column of SYSCAT.SERVERS (see "SYSCAT.SERVERS" on page 187). These values cannot exceed 30 characters. Delimited identifiers are converted to uppercase.

VERSION *server-version*
: Identifies the version of the server to which *data-source-data-type* is defined. Valid values for *server-version* are composed of digits and, optionally, one or two decimal points. Do not enclose these values in quotes or place a decimal point at the end of them. For example:

```
VERSION 7
VERSION 8.0.3
```

The valid values for this field match the allowed values for the VERSION column in the catalog view SYSCAT.SERVERS. For details, see "SYSCAT.SERVERS" on page 187.

server-version can have up to 18 characters.

| If the data source data type is built-in, specifying a value for *server-version* ensures
| that the mapping that you're defining will apply to this and all future versions of the
| specified server type (and protocol, if present).

PROTOCOL *"server-protocol"*
| Identifies the protocol used in accessing *data-source-data-type*. Valid values for
| *server-protocol* are the same as those in the SERVER_PROTOCOL column of
| SYSCAT.SERVERS (see "SYSCAT.SERVERS" on page 187). These values are
| case-sensitive and cannot exceed 30 characters.

DISTINCT
Indicates that you're mapping the type denoted by **DataJoiner Data Type** to a user-defined data type. Be sure to specify this keyword when it applies; otherwise, DataJoiner might not generate the required casting operator for SQL statements at the data source.

data-source-schema.
Is the name of the schema of data type to which you're mapping the type denoted by **DataJoiner Data Type**.

data-source-data-type
Is the data type to which you're mapping the type denoted by **DataJoiner Data Type**. *data-source-data-type* can be either built-in or user-defined. If it's built-in, do not specify the long form; for example, specify CHAR instead of CHARACTER.

p For a decimal data type, *p* specifies the maximum number of digits that a value can have. For all other data types for character data, *p* specifies the maximum number of characters that a value can have. If you don't specify *p* and the data type requires it, DataJoiner will determine the best match.

For Informix DATETIME data types, *p* specifies the first precision. In this instance, use the field qualifier values listed in Table 5 on page 93.

Table 6. Field Qualifier Values for Informix DATETIME Data Types

Precision	Qualifier
year	0
month	2
day	4
hour	6
minute	8
second	10
fraction(1)	11
fraction(2)	12
fraction(3)	13
fraction(4)	14
fraction(5)	15

[p..p]
| For a decimal data type, *[p..p]* specifies the minimum and maximum number of

digits that a value can have. For all other data types for character data, *[p..p]* specifies the minimum and maximum number of characters that a value can have. In all cases, the maximum must equal or exceed the minimum; and both numbers must be valid with respect to the data type.

For Informix DATETIME data types, *[p..p]* specifies an inclusive range of the first precision. Use the field qualifier values listed in Table 5 on page 93.

s For a decimal data type, *s* specifies the allowable maximum number of digits to the right of the decimal point. The number you specify must be valid with respect to the data type. If you don't specify a number and the data type requires one, DataJoiner will determine the best match.

For Informix DATETIME data types, *s* specifies the second precision value. For example, in Informix, DATETIME year to month is used to specify the month, which should be specified with a value of 2 as in Table 5 on page 93.

[s..s]
For a decimal data type, *[s..s]* specifies the minimum and maximum number of digits allowed to the right of the decimal point. The maximum must equal or exceed the minimum, and both numbers must be valid with respect to the data type.

For Informix datetime types, *[s..s]* specifies an inclusive range of the first precision. Use the field qualifier values listed in Table 5 on page 93.

p_operand_s
For a decimal data type, *p_operand_s* specifies a comparison between the maximum number of digits allowed to the left of the decimal point and the maximum number of digits allowed to the right of the decimal point. For example, the operand = indicates that the numbers are the same, and the operand > indicates that the first number is greater than the second. Specify this attribute only if you need to enforce this level of checking.

FOR BIT DATA
Indicates whether *data-source-data-type* is for bit data. DataJoiner will determine this attribute if it is not specified on a character data type.

Notes

- Together, the values for *datajoiner-schema* and **DataJoiner Data Type** constitute the fully-qualified name of a data type that's defined to the DataJoiner database. In this name, the value for *datajoiner-schema* qualifies the value for **DataJoiner Data Type**.
- Together, the values for *data-source-schema* and *data-source-data-type* constitute the fully-qualified name of a data type that's defined to a data source. In this name, the value for *data-source-schema* qualifies the value for *data-source-data-type.*
- If *data-source-data-type* is case-sensitive at the data source, put double quotes around the values that you assign to *data-source-schema* (before the period) and to *data-source-data-type.* For example:

```
CREATE TYPE MAPPING DBO_AU_ID
   FROM SYSIBM.CHAR(11)
   TO SERVER TYPE MSSQLSERVER
   TYPE "dbo"."id"
```

| In addition, if you're submitting the CREATE TYPE MAPPING statement from a UNIX
| command prompt, put single quotes around the whole statement. For example:

```
| 'CREATE TYPE MAPPING DBO_AU_ID
|    FROM SYSIBM.CHAR(11)
|    TO SERVER TYPE MSSQLSERVER
|    TYPE "dbo"."id"'
```

| If you're submitting the statement from a Windows NT command prompt, precede
| each double quote with a backward slash. For example:

```
| CREATE TYPE MAPPING DBO_AU_ID
|    FROM SYSIBM.CHAR(11)
|    TO SERVER TYPE MSSQLSERVER
|    TYPE \"dbo\".\"id\"
```

Examples

Example 1: Map SYSIBM.DATE, which is defined to the DataJoiner database, to the Oracle data type DATE at all Oracle servers.

```
CREATE TYPE MAPPING MY_ORACLE_DATE
   FROM SYSIBM.DATE
   TO SERVER TYPE ORACLE1
   TYPE DATE
```

Example 2: Map SYSIBM.DATE, which is defined to the DataJoiner database, to the Oracle data type DATE at the Oracle server ORACLE1.

```
CREATE TYPE MAPPING MY_ORACLE_DATE
   FROM SYSIBM.DATE
   TO SERVER ORACLE1
   TYPE DATE
```

CREATE USER MAPPING

Use the CREATE USER MAPPING statement to create a mapping between the authorization ID under which a user accesses the DataJoiner database and the authorization ID under which this user accesses a data source.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold one of the following permissions on the DataJoiner database:

- SYSADM authority
- DBADM authority
- CREATETAB privilege

Syntax

```
►►──CREATE USER MAPPING FROM───┬─local-authid─┬──────────────────────────────────►
                               └─USER─────────┘

►─TO SERVER──server-name──AUTHID──remote-authid──────────────────────────────────►

►──┬──────────────────────────────────┬──┬──────────────────────────┬───────────►◄
   └─PASSWORD──remote-password────────┘  └─CONNECTOPT──'string'─────┘
```

Description

FROM *local-authid* or USER
: *local-authid* identifies the specific authorization ID under which a user connects to the DataJoiner database. The valid values for this field match the allowed values for the AUTHID column in the catalog view SYSCAT.REMOTEUSERS. Identifiers without delimiters are converted to uppercase. See "SYSCAT.REMOTEUSERS" on page 183 for details. *local-authid* is a short identifier with a maximum of 8 characters.

    USER denotes the special register USER, which specifies the authorization ID under which you run the CREATE USER MAPPING statement. If you specify USER, this ID maps automatically to the ID specified by *remote-authid*.

TO SERVER *server-name*
: Identifies the name of the server containing authid *remote-authid*. This field must match an entry in the SERVER column of the SYSCAT.SERVERS catalog view (see "SYSCAT.SERVERS" on page 187).

The unique index for SYSCAT.REMOTEUSERS includes the AUTHID and SERVER columns. You cannot specify two rows in this table with identical values for both of these columns.

server-name is a long identifier.

AUTHID *remote-authid*
Identifies the authorization ID on the specified server that corresponds to the *local-authid. remote-authid* is an identifier that cannot exceed 30 characters.

PASSWORD *remote-password*
Identifies the password for the *remote-authid* on the specified server. *remote-password* is an identifier that cannot exceed 32 characters.

CONNECTOPT *'string'*
Identifies the connect option for the specified server. This value is protocol-dependent. *'string'* is a character string and has a maximum length of 256 characters.

Notes

- Identifiers without delimiters are converted to uppercase in SYSCAT.REMOTEUSERS. The following examples show how you can use quotes to pass case-sensitive characters for each keyword, such as, AUTHID, SERVER, and PASSWORD.
- If *remote-password* isn't specified, the server option for validating passwords (password) must be set to 'N' for the server denoted by *server-name.* For more information about this option, see "SYSCAT.SERVER_OPTIONS" on page 195.

Examples

Example 1: Create a user mapping.

```
CREATE USER MAPPING FROM BENHAM
   TO SERVER ORACLE1
   AUTHID "shawnb"
```

Example 2: Create a user mapping with a specified password. Use the special register USER.

```
CREATE USER MAPPING FROM USER
   TO SERVER DB2MVS1
   AUTHID "sysadm"
   PASSWORD "sysadmpw"
```

DESCRIBE CURSOR

Use the DESCRIBE CURSOR SQL statement to return the description of a result set associated with a cursor. The cursor may be associated with a SELECT or a CALL SQL statement.

Invocation

You must embed this statement in an application program. It cannot be issued dynamically.

Authorization

None required.

Syntax

```
►►──DESCRIBE CURSOR──cursor-name──INTO──descriptor-name──────────────────────────────►◄
```

Description

cursor-name
: Is the name of a cursor that has been allocated or declared within this application program. If the cursor was closed by a COMMIT or a ROLLBACK operation, or if no cursor was allocated or declared, the system returns SQLCODE -504 (The cursor <name> is not defined).

INTO *descriptor-name*
: Identifies an SQL descriptor area (SQLDA). The information returned in the SQLDA describes the result set associated with the cursor.

The SQLDA that is returned is similar to the SQLDA that is returned on a DESCRIBE of a select-list. The differences are:

- The first 5 bytes of the SQLDAID field are set to SQLRS.
- Bytes 6 to 8 of the SQLDAID field are reserved.

Data from multi-row or non-uniform result sets will be retrieved with the cursor that was declared through the ALLOCATE CURSOR SQL statement. Use the FETCH statement to cause the cursor to point to the rows of the result sets, so that the rows can be returned to the calling application. Then use the CLOSE statement to close the cursor.

Examples

Describe a cursor in a stored procedure (application logic fragment).

```
EXEC SQL DESCRIBE CURSOR CUR1 INTO :OUTPUT_SQLDA
```

DROP

Use this statement to delete:

- An object from the DataJoiner database. When you delete the object, any objects that are directly or indirectly dependent on it are also deleted. In addition, its description is deleted from the catalog, and any packages that reference it are invalidated.
- A data source table that was created with the DataJoiner CREATE TABLE statement. When you delete the table, its local specification—its nickname, the names of its columns, and so on—are deleted from the catalog.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold one of the following permissions:

- SYSADM authority
- DBADM authority
- CONTROL privilege on the object you want to drop
- CONTROL privilege on a table whose index you want to drop

Syntax

```
                     (1)
►►──DROP───┬─ALIAS──────────alias-name─────────────────────────────────────────┬──────────►◄
           │       (2)                                                          │
           ├─DISTINCT───────TYPE──distinct-type-name────────────────────────────┤
           ├─EVENT MONITOR──event-monitor-name──────────────────────────────────┤
           ├─FUNCTION──function-name──┬─────────────────────────────────┬───────┤
           │                          └─(──┬──────────────────────┬──)──┘       │
           │                               │  ┌─,──────────────┐  │           │
           │                               │  ▼                │  │           │
           │                               └─────data-type─────┴──┘           │
           │                        (3)                                         │
           ├─SPECIFIC FUNCTION─────────specific-name────────────────────────────┤
           ├─FUNCTION MAPPING──function-mapping-name────────────────────────────┤
           ├─INDEX──index-name──────────────────────────────────────────────────┤
           ├─NICKNAME──nickname─────────────────────────────────────────────────┤
           │        (4)                                                         │
           ├─PACKAGE──────────package-name──────────────────────────────────────┤
           ├─SERVER MAPPING──server-name────────────────────────────────────────┤
           ├─SERVER OPTION──option-name──FOR──┤ Remote Server Clause ├──────────┤
           ├─STORED PROCEDURE NICKNAME──stored-procedure-nickname───────────────┤
           ├─TABLE──┬─DataJoiner-table-name──┬──────────────────────────────────┤
           │        └─data-source-table-name─┘                                  │
           ├─TABLESPACE──tablespace-name────────────────────────────────────────┤
           ├─TRIGGER──trigger-name──────────────────────────────────────────────┤
           ├─TYPE MAPPING──type-mapping-name────────────────────────────────────┤
           ├─USER MAPPING FROM──┬─local-authid─┬──TO SERVER──server-name────────┤
           │                    └─USER─────────┘                                │
           └─VIEW──view-name────────────────────────────────────────────────────┘
```

Remote Server Clause

```
├──┬─SERVER──server-name──────────────────────────────────────────────────┬──────────────┤
   ├─SERVER TYPE──server-type──VERSION──server-version──┤ Protocol ├──────┤
   ├─SERVER TYPE──server-type──VERSION──server-version────────────────────┤
   └─SERVER TYPE──server-type─────────────────────────────────────────────┘
```

Protocol

```
├──PROTOCOL──"server-protocol" ────────────────────────────────────────────────────────────┤
```

Notes:

1. SYNONYM can be used as a synonym for ALIAS.
2. DATA can be used as a synonym for DISTINCT.
3. ROUTINE can be used as a synonym for FUNCTION.
4. PROGRAM can be used as a synonym for PACKAGE.

Description

FUNCTION MAPPING *function-mapping-name*
Identifies the function mapping to be dropped. *function-mapping-name* must be a user-created function mapping that exists in the catalog view SYSCAT.SERVER_FUNCTIONS. System function mappings generated by DataJoiner cannot be dropped. *function-mapping-name* is a long identifier.

Dropping a local function causes all function mappings dependent on this specific function to be dropped.

INDEX *index-name*
Identifies the index to be dropped. The index specified must be described in the index catalog. When *index-name* represents index information for a nickname, the index information is dropped in DataJoiner. The actual index at the data source is not affected.

NICKNAME *nickname*
Identifies the nickname to be dropped. The nickname specified must be described in the catalog. DataJoiner deletes the nickname from the database, along with information about the column, views, and indexes associated with the nickname. The table at the data source, for which the nickname was defined, is not affected.

PACKAGE *package-name*
Identifies the package to be dropped. The package specified must be described in the catalog.

SERVER MAPPING *server-name*
Identifies the server mapping to be dropped. The mapping between a local (DataJoiner) server name and a server database at a data source is dropped. All nicknames dependent upon that server are dropped. Also, all type mapping, server mapping, user mapping, and server option catalog entries are similarly dropped. All plans dependent on the dropped objects are invalidated. *server-name* is a long identifier.

SERVER OPTION *option-name* **FOR**
Identifies the server option. Valid values for *option-name* match those listed for the column OPTION in the catalog view SYSCAT.SERVER_OPTIONS (see “SYSCAT.SERVER_OPTIONS” on page 195). Values for *option-name* cannot exceed 30 characters.

SERVER *server-name*
Identifies the server to which *option-name* applies. This parameter identifies the server that will no longer have the specified server option. This field must match an entry in the SERVER column of the SYSCAT.SERVERS catalog view. *server-name* is a long identifier.

SERVER TYPE *server-type*
Identifies the type of server to which *option-name* applies. This parameter identifies the server type that will no longer have the specified server option. Valid values for *server-type* match those listed for the SERVER_TYPE column of

SYSCAT.SERVER_OPTIONS (see “SYSCAT.SERVER_OPTIONS” on page 195). Values for *server-type* cannot exceed 30 characters. Delimited identifiers will be converted to uppercase.

VERSION *server-version*
| Identifies the version of the server to which *option-name* applies. Valid values for *server-version* are composed of digits and, optionally, one or two decimal points. Do not enclose these values in quotes or place a decimal point at the end of them. For example:

```
VERSION 7
VERSION 8.0.3
```

| The valid values for this field match the allowed values for the VERSION column in the catalog view SYSCAT.SERVERS. For details, see “SYSCAT.SERVERS” on page 187.

| *server-version* can have up to 18 characters.

PROTOCOL *"server-protocol"*
Identifies the server protocol to which *option-name* applies. *"server-protocol"* is case-sensitive and cannot exceed 30 characters.

STORED PROCEDURE NICKNAME *stored-procedure-nickname*
Represents a short identifier with or without a qualifier. If no qualifier is supplied, the authorization ID of the statement is used as the default value. A nickname cannot exceed 8 characters. The qualifier must not be SYSIBM.

TABLE *DataJoiner-table-name* or *data-source-table-name*
| Identifies the table to be dropped. The table must be described in the catalog and cannot be a catalog table. All indexes, views, primary keys, and foreign keys defined on the table are dropped, and all foreign keys that reference the table are also dropped.

| *DataJoiner-table-name* is a table in the DataJoiner database. *data-source-table-name* is a data source table that was created with DataJoiner’s CREATE TABLE statement.

TYPE MAPPING *type-mapping-name*
| Identifies the data type mapping to be dropped. *type-mapping-name* must exist in the catalog view SYSCAT.SERVER_DATATYPES. System type mappings generated by DataJoiner cannot be dropped. *type-mapping-name* is a long identifier.

| Dropping a local distinct type causes all functions (and also all function mappings dependent on the function) that use that data type to be dropped. Also, dropping a local distinct type causes all type mappings dependent on that local data type to be dropped.

| **USER MAPPING FROM** *local-authid* or USER **TO SERVER** *server-name*
| Identifies the mapping to be dropped between a local DataJoiner authorization ID and the authorization ID on the specified data source server. The variables *local-authid* or USER identify the local authorization ID. *local-authid* is a specific ID.

| Its values must be a maximum of 8 characters and are automatically changed to
| uppercase. USER refers to the special register USER, which stands for the
| authorization ID under which you run the DROP statement. Therefore, if you're
| dropping a mapping between this ID and a remote one, you can specify USER
| rather than the ID itself.

| *server-name* is the name of the server that is accessible under an authorization ID
| to which *local-authid* or USER maps.

VIEW *view-name*
Identifies the view to be dropped. The view specified must be described in the catalog and cannot be a catalog view. The definition of the view is deleted from the catalog. The definition of any view that is directly or indirectly dependent on that view is also deleted.

Notes

When dropping objects, be aware of object dependencies. Ensure that you want all associated objects to be dropped.

Examples

Example 1: Drop table CORPDATA.TDEPT.

```
DROP TABLE CORPDATA.TDEPT
```

Example 2: Drop server mapping DJTOSYBASE1.

```
DROP SERVER MAPPING DJTOSYBASE1
```

Example 3: Drop the user mapping SHAWN from the server ORACLE1.

```
DROP USER MAPPING FROM SHAWN TO SERVER ORACLE1
```

Example 4: Drop the view VDEPT.

```
DROP VIEW VDEPT
```

Example 5: Drop the nickname A1.

```
DROP NICKNAME A1
```

Example 6: Drop the two-phase commit server option for a particular server.

```
DROP SERVER OPTION TWO_PHASE_COMMIT FOR SERVER SYBASE1
```

Example 7: Drop the two-phase commit server option for a specific server type and version.

```
DROP SERVER OPTION TWO_PHASE_COMMIT
   FOR SERVER TYPE INFORMIX
   VERSION 4.1
```

GRANT PASSTHRU

Use the GRANT PASSTHRU statement to grant a user or group the authority to query a specified server in pass-through mode.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold either DBADM or SYSADM authority on the DataJoiner database.

Syntax

```
►►──GRANT PASSTHRU ON SERVER──server-name──TO─────────────────────────────►

    ┌─,────────────────────────────┐
    ▼                              │
►────┬─┬───────┬─authorization-name─┴──────────────────────────────────►◄
     │ ├─USER──┤                 │
     │ └─GROUP─┘                 │
     └─PUBLIC────────────────────┘
```

Description

SERVER *server-name*
: Identifies the server on which you're granting pass-through authority. This field must match an entry in the SERVER column of the SYSCAT.SERVERS catalog view (see "SYSCAT.SERVERS" on page 187). *server-name* is a long identifier.

Examples

Example 1: Give users whose user IDs are SMITH and JONES the authority to pass through to server SERVALL.

```
GRANT PASSTHRU ON SERVER SERVALL
   TO USER SMITH,
   USER JONES
```

Example 2: Grant PASSTHRU authority on server EASTWING to a group whose ID is D024. There is a user whose ID is also D024.

```
GRANT PASSTHRU ON SERVER EASTWING TO GROUP D024
```

The GROUP keyword must be specified; otherwise, an error will occur because D024 is a user’s ID as well as the specified group’s ID. Any member of group D024 will be allowed to pass through to EASTWING. Therefore, if user D024 belongs to the group, this user will be able to pass through to EASTWING.

REVOKE PASSTHRU

Use the REVOKE PASSTHRU statement to revoke a user's or group's authority to query a specified server in pass-through mode.

Invocation

You can embed this statement in an application program or issue it dynamically.

Authorization

The authorization ID under which you run this statement must hold either DBADM or SYSADM authority on the DataJoiner database.

Syntax

```
►►──REVOKE PASSTHRU ON SERVER──server-name──FROM──────────────────────────────►

   ┌─,────────────────────────────────┐
   ▼                                  │
►──┬─┬─────────┬──authorization-name──┴─┬──────────────────────────────────►◄
   │ ├─USER────┤                        │
   │ └─GROUP───┘                        │
   └─PUBLIC─────────────────────────────┘
```

Description

SERVER *server-name*
: Identifies the server for which you're revoking pass-through authority. This field must match an entry in the SERVER column of the SYSCAT.SERVERS catalog view (see "SYSCAT.SERVERS" on page 187). *server-name* is a long identifier.

Examples

Example 1: Revoke USER6's authority to pass through to server MOUNTAIN.

```
REVOKE PASSTHRU ON SERVER MOUNTAIN FROM USER USER6
```

Example 2: Revoke group D024's authority to pass through to server EASTWING.

```
REVOKE PASSTHRU ON SERVER EASTWING FROM GROUP D024
```

The members of group D024 will be no longer be able to use their group ID to pass through to EASTWING. But if any members have authority to pass through to EASTWING under their user IDs, they will retain this authority.

SET PASSTHRU

| Use the SET PASSTHRU statement open a session for submitting a data source's
| native SQL directly to that data source.

Invocation

You issue this statement dynamically.

Authorization

| The authorization ID under which you run this statement must carry authorization to:

- | Pass through to the data source. For information about how this authority is granted, see "Pass-Through Sessions for Querying Data Sources in Their Own SQL" on page 8 and "GRANT PASSTHRU" on page 126.
- | Satisfy security measures at the data source.

Syntax

```
►►──SET PASSTHRU──data-source-name──────────────────────────────────────►◄
```

Description

data-source-name
Represents the data source name in the SERVER column of the SYSCAT.SERVERS catalog view.

Notes

For factors to consider and restrictions to observe when you use pass-through, see "Considerations and Restrictions" on page 10.

Examples

Example 1: Start a pass-through session to the data source BACKEND.

```
STRCPY (PASS_THRU,"SET PASSTHRU BACKEND");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU;
```

Example 2: Start a pass-through session with a PREPARE statement.

```
STRCPY (PASS_THRU,"SET PASSTHRU BACKEND");
EXEC SQL PREPARE STMT FROM :PASS_THRU;
EXEC SQL EXECUTE STMT;
```

Example 3: Start a pass-through session and perform updates at the data source.

```
EXEC SQL DECLARE CURSOR C1 FOR         native mode
   SELECT * FROM AS1T1;                alias for a table at BACKEND1
EXEC SQL OPEN CURSOR C1;
EXEC SQL FETCH C1 INTO :HV;
EXEC SQL UPDATE AS2T2                  native mode
   SET COL1='1' WHERE COL2='X';        alias to a table at BACKEND2
STRCPY (PASS_THRU,"SET PASSTHRU BACKEND2");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU;
EXEC SQL PREPARE STMT2                 pass-through mode
   FROM "CREATE UNIQUE
         CLUSTERED INDEX TABLE2_INDEX
         ON USER2.TABLE2               table is not an
         WITH IGNORE DUP KEY";         alias
EXEC SQL EXECUTE STMT2;
EXEC SQL UPDATE AS2T1                  native mode
   SET COL1='2' WHERE COL2='Y';        alias to a table at BACKEND2
strcpy (PASS_THRU_RESET,"SET PASSTHRU RESET");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU_RESET;
```

SET PASSTHRU RESET

| Use the SET PASSTHRU RESET statement to terminate one or more pass-through
| sessions.

Invocation

| You issue this statement dynamically.

Authorization

None required.

Syntax

```
►►──SET PASSTHRU RESET──────────────────────────────────────────────────►◄
```

Examples

Example 1: End a pass-through session.

```
   STRCPY (PASS_THRU_RESET,"SET PASSTHRU RESET");
   EXEC SQL EXECUTE IMMEDIATE :PASS_THRU_RESET;
```

| *Example 2:* Use the PREPARE and EXECUTE statements to end a pass-through
| session.

```
|  STRCPY (PASS_THRU_RESET,"SET PASSTHRU RESET");
|  EXEC SQL PREPARE STMT FROM :PASS_THRU_RESET;
|  EXEC SQL EXECUTE STMT;
```

SET SERVER OPTION

| Use the SET SERVER OPTION statement to set configuration options that persist while
| an application client is connected to a specific server. With these options, you can help
| to optimize performance while the connection is in effect. After the connection ends,
| options set by the CREATE SERVER OPTION or ALTER SERVER OPTION statement
| are reinstated. The catalog view SYSCAT.SERVER_OPTIONS contains the server
| option settings.

Invocation

| You can embed this statement in an application program or issue it dynamically.

Authorization

| The authorization ID under which you run this statement must hold one SYSADM or
| DBADM authority on the DataJoiner database.

Syntax

```
►►──SET SERVER OPTION──option-name──TO──option-value──FOR──SERVER──server-name──────────►◄
```

Description

option-name
: Is the name of the server option. Valid *option-name* values are listed after the OPTION column definition for the catalog view SYSCAT.SERVER_OPTIONS (see "SYSCAT.SERVER_OPTIONS" on page 195). *option-name* is an identifier; it cannot exceed 30 characters.

TO *option-value*
: Identifies the option setting. Valid values for *option-value* depend on the value of *option-name*. See the examples in this section. *option-value* can be an integer constant, floating-point constant, decimal constant, or character literal. If it's a character literal, it can contain up to 254 characters, and it must be enclosed in single quotes. For descriptions of valid *option-value*s, see "Summary of Server Options and Their Settings" on page 17.

SERVER *server-name*
: Identifies the server to which *option-name* applies. This field must match an entry in the SERVER column of the SYSCAT.SERVERS catalog view (see "SYSCAT.SERVERS" on page 187). *server-name* is a long identifier.

Notes

- Server options can be entered in uppercase or lowercase (case does not matter).
- You must specify the SET SERVER OPTION statement immediately after the CONNECT statement.

| • For an overview of the SQL for setting server options, see “Server Options” on
| page 11 . For additional information about the SET SERVER OPTION statement, see
| the *DataJoiner Administration Supplement.*

Examples

Example 1: Server SYBASE1 is defined to DataJoiner database DJDB. During a specific session with a database on SYBASE1, have the database operate in two-phase commit mode.

```
CONNECT TO DJDB
   SET SERVER OPTION TWO_PHASE_COMMIT
   TO 'y'
   FOR SERVER SYBASE1
```

Example 2: Server RATCHIT is defined to DataJoiner database DJDB. A database on RATCHIT is configured for a case-insensitive collating sequence that differs from the one configured for DJDB. You want DataJoiner to recognize this difference during a specific session between an application and the database on RATCHIT.

```
CONNECT TO DJDB
   SET SERVER OPTION COLSEQ
   TO 'i'
   FOR SERVER RATCHIT
```

Example 3: Server ORACLE1 is defined to DataJoiner database DJDB. During a specific session between an application and the database on ORACLE1, have all passwords validated at that database.

```
CONNECT TO DJDB
   SET SERVER OPTION PASSWORD TO 'y'
   FOR SERVER ORACLE1
```

Appendix A. Default Forward Type Mappings

This appendix shows default forward mappings between DB2 for CS data types defined to DataJoiner and data types defined to data sources. In these mappings, the types defined to the data sources point to the types defined to DataJoiner. The following list identifies these data sources. It also cites tables that show the default reverse type mappings between the data sources and DataJoiner.

- Classic Connect (Table 7)
- CrossAccess (Table 8 on page 136)
- DB2 for CS (Table 9 on page 137)
- DB2 for OS/390 (Table 10 on page 139)
- DB2 for OS/400 (Table 11 on page 141)
- DB2 for VM (SQL/DS) (Table 12 on page 142)
- Generic (Table 13 on page 143)
- Informix (Table 14 on page 145)
- Microsoft SQL Server (Table 15 on page 146)
- Oracle (Table 16 on page 149)
- RDB (Table 17 on page 150)
- SQL Anywhere (Table 18 on page 151)
- Sybase (Table 19 on page 152)

Default Type Mappings from Classic Connect Data Sources to DataJoiner

Table 7. Classic Connect Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
INTEGER	S	N	-	-	-	-	-	INTEGER	Y	4	-
SMALLINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
DECIMAL	S	N	1	31	0	31	-	DECIMAL	Y	-	-

Table 7. Classic Connect Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
DECIMAL	S	N	32	32	0	31	-	DOUBLE	Y	8	-
FLOAT	S	N	-	-	-	-	-	DOUBLE	Y	8	-
CHAR	S	N	1	254	-	-	-	CHARACTER	N	-	-
CHAR	S	N	255	4000	-	-	-	VARCHAR	N	-	-
VARCHAR	S	N	1	4000	-	-	-	VARCHAR	N	-	-
VARCHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
LONGVAR	S	N	-	-	-	-	-	LONG VARCHAR	N	32700	-
LONGVARCHAR	S	N	-	-	-	-	-	LONG VARCHAR	N	32700	-
DATE	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-

Default Type Mappings from CrossAccess Data Sources to DataJoiner

Table 8. CrossAccess Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
INTEGER	S	N	-	-	-	-	-	INTEGER	Y	4	-
SMALLINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
DECIMAL	S	N	1	31	0	31	-	DECIMAL	Y	-	-
DECIMAL	S	N	32	32	0	31	-	DOUBLE	Y	8	-
FLOAT	S	N	-	-	-	-	-	DOUBLE	Y	8	-
CHAR	S	N	1	254	-	-	-	CHARACTER	N	-	-

Table 8. CrossAccess Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
CHAR	S	N	255	4000	-	-	-	VARCHAR	N	-	-
VARCHAR	S	N	1	4000	-	-	-	VARCHAR	N	-	-
VARCHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
LONGVARCHAR	S	N	-	-	-	-	-	LONG VARCHAR	N	32700	-
LONGVAR	S	N	-	-	-	-	-	LONG VARCHAR	N	32700	-
DATE	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-

Default Type Mappings from DB2 for CS Data Sources to DataJoiner

Table 9. DB2 for CS Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
INTEGER	S	N	-	-	-	-	-	INTEGER	Y	4	-
SMALLINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
DECIMAL	S	N	1	31	0	31	-	DECIMAL	Y	-	-
DECIMAL	S	N	32	32	0	31	-	DOUBLE	Y	8	-
DOUBLE	S	N	-	-	-	-	-	DOUBLE	Y	8	-
FLOAT	S	N	-	-	-	-	-	DOUBLE	Y	8	-
CHAR	S	N	1	254	-	-	-	CHARACTER	N	-	-
CHAR	S	N	255	4000	-	-	-	VARCHAR	N	-	-
CHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-

Table 9. DB2 for CS Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
CHAR	S	Y	4001	32700	-	-	-	LONG VARCHAR	Y	-	-
VARCHAR	S	Y	1	4000	-	-	-	VARCHAR	Y	-	-
VARCHAR	S	Y	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
VARCHAR	S	N	1	4000	-	-	-	VARCHAR	N	-	-
VARCHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
LONGVARCHAR	S	N	-	-	-	-	-	LONG VARCHAR	N	32700	-
GRAPHIC	S	N	1	127	-	-	-	GRAPHIC	N	-	-
GRAPHIC	S	N	128	2000	-	-	-	VARGRAPHIC	Y	-	-
GRAPHIC	S	N	2001	16350	-	-	-	LONG VARGRAPHIC	Y	16350	-
VARGRAPH	S	N	1	2000	-	-	-	VARGRAPHIC	Y	-	-
VARGRAPH	S	N	2001	16350	-	-	-	LONG VARGRAPHIC	Y	16350	-
LONGVARG	S	N	-	-	-	-	-	LONG VARGRAPHIC	Y	16350	-
BLOB	S	N	-	-	-	-	-	BLOB	Y	2147483647	-
CLOB	S	N	-	-	-	-	-	CLOB	N	2147483647	-
DBCLOB	S	N	-	-	-	-	-	DBCLOB	Y	1073741823	-
CHAR	S	Y	1	254	-	-	-	CHARACTER	Y	-	-
CHAR	S	Y	255	4000	-	-	-	VARCHAR	Y	-	-
CHAR	S	Y	4001	32700	-	-	-	VARCHAR	Y	-	-
CHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
VARCHAR	S	Y	1	4000	-	-	-	VARCHAR	Y	-	-
VARCHAR	S	Y	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
VARCHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
LONGVAR	S	Y	-	-	-	-	-	LONG VARCHAR	Y	32700	-
LONGVARCHAR	S	Y	-	-	-	-	-	LONG VARCHAR	Y	32700	-
LONGVARCHAR	S	Y	4001	32700	-	-	-	VARCHAR	Y	32700	-
DATE	S	N	-	-	-	-	-	DATE	Y	4	-

Table 9. DB2 for CS Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
TIME	S	N	-	-	-	-	-	TIME	Y	3	-
TIMESTMP	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-
TIMESTAMP	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-

Default Type Mappings from DB2 for OS/390 Data Sources to DataJoiner

Table 10. DB2 for OS/390 Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
INTEGER	S	N	-	-	-	-	-	INTEGER	Y	4	-
SMALLINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
DECIMAL	S	N	1	31	0	31	-	DECIMAL	Y	-	-
DECIMAL	S	N	32	32	0	31	-	DOUBLE	Y	8	-
FLOAT	S	N	-	-	-	-	-	DOUBLE	Y	8	-
CHAR	S	N	1	254	-	-	-	CHARACTER	N	-	-
CHAR	S	N	255	4000	-	-	-	VARCHAR	N	-	-
CHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
VARCHAR	S	N	1	4000	-	-	-	VARCHAR	N	-	-
VARCHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
LONGVAR	S	N	-	-	-	-	-	LONG VARCHAR	N	32700	-
LONGVAR	S	N	4001	32700	-	-	-	VARCHAR	N	4000	-

Table 10. DB2 for OS/390 Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
CHAR	S	Y	1	254	-	-	-	CHARACTER	Y	-	-
CHAR	S	Y	255	4000	-	-	-	VARCHAR	Y	-	-
CHAR	S	Y	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
VARCHAR	S	Y	1	4000	-	-	-	VARCHAR	Y	-	-
VARCHAR	S	Y	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
LONGVAR	S	Y	-	-	-	-	-	LONG VARCHAR	Y	32700	-
GRAPHIC	S	N	1	127	-	-	-	GRAPHIC	N	-	-
GRAPHIC	S	N	128	2000	-	-	-	VARGRAPHIC	Y	-	-
GRAPHIC	S	N	2001	16350	-	-	-	LONG VARGRAPHIC	Y	16350	-
VARG	S	N	1	2000	-	-	-	VARGRAPHIC	Y	-	-
VARGRAPH	S	N	1	2000	-	-	-	VARGRAPHIC	Y	-	-
VARG	S	N	2001	16350	-	-	-	LONG VARGRAPHIC	Y	16350	-
VARGRAPH	S	N	2001	16350	-	-	-	LONG VARGRAPHIC	Y	16350	-
LONGVARG	S	N	-	-	-	-	-	LONG VARGRAPHIC	Y	16350	-
DATE	S	N	-	-	-	-	-	DATE	Y	4	-
TIME	S	N	-	-	-	-	-	TIME	Y	3	-
TIMESTMP	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-

Default Type Mappings from DB2 for OS/400 Data Sources to DataJoiner

Table 11. DB2 for OS/400 Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
INTEGER	S	N	-	-	-	-	-	INTEGER	Y	4	-
SMALLINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
DECIMAL	S	N	1	31	0	31	-	DECIMAL	Y	-	-
DECIMAL	S	N	32	32	0	31	-	DOUBLE	Y	8	-
FLOAT	S	N	-	-	-	-	-	DOUBLE	Y	8	-
CHAR	S	N	1	254	-	-	-	CHARACTER	N	-	-
CHAR	S	N	255	4000	-	-	-	VARCHAR	N	-	-
CHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
CHAR	S	Y	1	254	-	-	-	CHARACTER	Y	-	-
CHAR	S	Y	255	4000	-	-	-	VARCHAR	Y	-	-
VARCHAR	S	N	1	4000	-	-	-	VARCHAR	N	-	-
VARCHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
GRAPHIC	S	N	1	127	-	-	-	GRAPHIC	N	-	-
GRAPHIC	S	N	128	2000	-	-	-	VARGRAPHIC	Y	-	-
GRAPHIC	S	N	2001	16350	-	-	-	LONG VARGRAPHIC	Y	16300	-
VARG	S	N	1	2000	-	-	-	VARGRAPHIC	Y	-	-
VARG	S	N	2001	16350	-	-	-	LONG VARGRAPHIC	Y	16300	-
LONGVARG	S	N	-	-	-	-	-	LONG VARGRAPHIC	Y	32700	-
DATE	S	N	-	-	-	-	-	DATE	Y	4	-
TIME	S	N	-	-	-	-	-	TIME	Y	3	-
TIMESTMP	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-
NUMERIC	S	N	1	31	0	31	-	DECIMAL	Y	-	-

Default Type Mappings from DB2 for VM Data Sources to DataJoiner

Table 12. DB2 for VM (SQL/DS) Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
INTEGER	S	N	-	-	-	-	-	INTEGER	Y	4	-
DBAINT	S	N	-	-	-	-	-	INTEGER	Y	4	-
SMALLINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
DBAHW	S	N	-	-	-	-	-	SMALLINT	Y	2	-
DECIMAL	S	N	1	31	0	31	-	DECIMAL	Y	-	-
DECIMAL	S	N	32	32	0	31	-	DOUBLE	Y	8	-
FLOAT	S	N	-	-	-	-	-	DOUBLE	Y	8	-
CHAR	S	N	1	254	-	-	-	CHARACTER	N	-	-
CHAR	S	N	255	4000	-	-	-	VARCHAR	N	-	-
CHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
CHAR	S	Y	1	254	-	-	-	CHARACTER	Y	-	-
CHAR	S	Y	255	4000	-	-	-	VARCHAR	Y	-	-
CHAR	S	Y	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
VARCHAR	S	N	1	4000	-	-	-	VARCHAR	N	-	-
VARCHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
VARCHAR	S	Y	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
LONGVARCHAR	S	N	-	-	-	-	-	LONG VARCHAR	Y	32700	-
LONGVARCHAR	S	Y	-	-	-	-	-	LONG VARCHAR	N	32700	-
GRAPHIC	S	N	1	127	-	-	-	GRAPHIC	N	-	-
GRAPHIC	S	N	128	2000	-	-	-	VARGRAPHIC	Y	-	-
GRAPHIC	S	N	2001	16350	-	-	-	LONG VARGRAPHIC	Y	16350	-
VARGRAPH	S	N	1	2000	-	-	-	VARGRAPHIC	Y	-	-
VARGRAPH	S	N	2001	16350	-	-	-	LONG VARGRAPHIC	Y	16350	-
LONGVARG	S	N	-	-	-	-	-	LONG VARGRAPHIC	Y	16350	-

Table 12. DB2 for VM (SQL/DS) Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
DATE	S	N	-	-	-	-	-	DATE	Y	4	-
TIME	S	N	-	-	-	-	-	TIME	Y	3	-
TIMESTMP	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-
LNGVCHAR	S	N	-	-	-	-	-	LONG VARCHAR	N	32700	-
LNGVCHAR	S	Y	-	-	-	-	-	LONG VARCHAR	Y	32700	-

Default Type Mappings from Generic Data Sources to DataJoiner

Table 13. Generic Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
SQL_INTEGER	S	N	-	-	-	-	-	INTEGER	Y	4	-
SQL_SMALLINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
SQL_DECIMAL	S	N	1	31	0	31	-	DECIMAL	Y	-	-
SQL_DECIMAL	S	N	32	32	0	31	-	DOUBLE	Y	8	-
SQL_NUMERIC	S	N	1	31	0	31	-	DECIMAL	Y	-	-
SQL_NUMERIC	S	N	32	32	0	31	-	DOUBLE	Y	8	-
SQL_FLOAT	S	N	-	-	-	-	-	DOUBLE	Y	8	-
SQL_DOUBLE	S	N	-	-	-	-	-	DOUBLE	Y	8	-
SQL_REAL	S	N	-	-	-	-	-	DOUBLE	Y	8	-
SQL_CHAR	S	N	1	254	-	-	-	CHARACTER	N	-	-

Table 13. Generic Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
SQL_CHAR	S	N	255	4000	-	-	-	VARCHAR	N	-	-
SQL_CHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
SQL_BINARY	S	N	1	254	-	-	-	CHARACTER	Y	-	-
SQL_BINARY	S	N	255	4000	-	-	-	VARCHAR	Y	-	-
SQL_BINARY	S	N	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
SQL_VARCHAR	S	N	1	4000	-	-	-	VARCHAR	N	-	-
SQL_VARCHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
SQL_VARBINARY	S	N	1	4000	-	-	-	VARCHAR	Y	-	-
SQL_VARBINARY	S	N	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
SQL_LONGVARCHAR	S	N	-	-	-	-	-	CLOB	N	2147483647	-
SQL_LONGVARBINARY	S	N	-	-	-	-	-	BLOB	Y	2147483647	-
SQL_DATE	S	N	-	-	-	-	-	DATE	Y	4	-
SQL_TIME	S	N	-	-	-	-	-	TIME	Y	3	-
SQL_TIMESTAMP	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-
SQL_BIT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
SQL_TINYINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
SQL_BIGINT	S	N	-	-	-	-	-	DECIMAL	Y	-	-

Default Type Mappings from Informix Data Sources to DataJoiner

Table 14. Informix Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
SMALLINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
INTEGER	S	N	-	-	-	-	-	INTEGER	Y	4	-
SERIAL	S	N	-	-	-	-	-	INTEGER	Y	4	-
DECIMAL	S	N	1	31	0	31	-	DECIMAL	Y	-	-
DECIMAL	S	N	32	32	-255	255	-	DOUBLE	Y	8	-
DECIMAL	S	N	1	31	255	255	-	DOUBLE	Y	8	-
MONEY	S	N	1	31	0	31	-	DECIMAL	Y	-	-
MONEY	S	N	32	32	-	-	-	DOUBLE	Y	8	-
FLOAT	S	N	-	-	-	-	-	DOUBLE	Y	8	-
SMALLFLOAT	S	N	-	-	-	-	-	DOUBLE	Y	8	-
INTERVAL	S	N	-	-	-	-	-	CHARACTER	N	-	-
BYTE	S	N	-	-	-	-	-	BLOB	Y	2147483647	-
CHAR	S	N	1	254	-	-	-	CHARACTER	N	-	-
CHAR	S	N	255	4000	-	-	-	VARCHAR	N	-	-
CHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
VARCHAR	S	N	1	4000	-	-	-	VARCHAR	N	-	-
VARCHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
NCHAR	S	N	1	254	-	-	-	CHARACTER	N	-	-
NCHAR	S	N	255	4000	-	-	-	VARCHAR	N	-	-
NCHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
NVARCHAR	S	N	1	4000	-	-	-	VARCHAR	N	-	-
NVARCHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
TEXT	S	N	-	-	-	-	-	CLOB	N	2147483647	-
DATE	S	N	-	-	-	-	-	DATE	Y	4	-
DATETIME	S	N	0	4	0	4	-	DATE	Y	4	-
DATETIME	S	N	6	10	6	10	-	TIME	Y	3	-

Table 14. Informix Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
DATETIME	S	N	0	4	6	15	-	TIMESTAMP	Y	10	-
DATETIME	S	N	6	10	11	15	-	TIMESTAMP	Y	10	-

Default Type Mappings from Microsoft SQL Server Data Sources to DataJoiner

Table 15. Microsoft SQL Server Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
int	S	N	-	-	-	-	-	INTEGER	Y	4	-
intn	S	N	-	-	-	-	-	INTEGER	Y	4	-
smallint	S	N	-	-	-	-	-	SMALLINT	Y	2	-
tinyint	S	N	-	-	-	-	-	SMALLINT	Y	2	-
bit	S	N	-	-	-	-	-	SMALLINT	Y	2	-
float	S	N	-	-	-	-	-	DOUBLE	Y	8	-
floatn	S	N	-	-	-	-	-	DOUBLE	Y	8	-
real	S	N	-	-	-	-	-	DOUBLE	Y	8	-
money	S	N	-	-	-	-	-	DECIMAL	Y	19	4
moneyn	S	N	-	-	-	-	-	DECIMAL	Y	19	4
smallmoney	S	N	-	-	-	-	-	DECIMAL	Y	10	4
smallmoneyn	S	N	-	-	-	-	-	DECIMAL	Y	10	4
decimal	S	N	1	31	0	31	-	DECIMAL	Y	-	-

Table 15. *Microsoft SQL Server Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown) (continued)*

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
decimal	S	N	32	38	0	38	-	DOUBLE	Y	8	-
decimaln	S	N	1	31	0	31	-	DECIMAL	Y	-	-
decimaln	S	N	32	38	0	38	-	DOUBLE	Y	8	-
numeric	S	N	1	31	0	31	-	DECIMAL	Y	-	-
numeric	S	N	32	38	0	38	-	DOUBLE	Y	8	-
numericn	S	N	1	31	0	31	-	DECIMAL	Y	-	-
numericn	S	N	32	38	0	38	-	DOUBLE	Y	8	-
char	S	N	1	254	-	-	-	CHARACTER	N	-	-
sysname	S	N	1	254	-	-	-	CHARACTER	N	-	-
char	S	N	255	4000	-	-	-	VARCHAR	N	-	-
char	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
varchar	S	N	1	4000	-	-	-	VARCHAR	N	-	-
varchar	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
text	S	N	-	-	-	-	-	CLOB	N	2147483647	-
nchar	S	N	1	254	-	-	-	CHARACTER	N	-	-
nchar	S	N	255	4000	-	-	-	VARCHAR	N	-	-
nchar	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
nvarchar	S	N	1	4000	-	-	-	VARCHAR	N	-	-
nvarchar	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
binary	S	N	1	254	-	-	-	CHARACTER	Y	-	-
binary	S	N	255	4000	-	-	-	VARCHAR	Y	-	-
binary	S	N	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
varbinary	S	N	1	4000	-	-	-	VARCHAR	Y	-	-
varbinary	S	N	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
image	S	N	-	-	-	-	-	BLOB	Y	2147483647	-
datetime	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-
datetimn	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-
smalldatetime	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-

Table 15. *Microsoft SQL Server Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown) (continued)*

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
timestamp	S	N	-	-	-	-	-	VARCHAR	Y	8	-
sysname	S	N	-	-	-	-	-	VARCHAR	N	30	-
SQL_INTEGER	S	N	-	-	-	-	-	INTEGER	Y	4	-
SQL_SMALLINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
SQL_DECIMAL	S	N	1	31	0	31	-	DECIMAL	Y	8	-
SQL_DECIMAL	S	N	32	38	0	38	-	DOUBLE	Y	8	-
SQL_NUMERIC	S	N	1	31	0	31	-	DECIMAL	Y	-	-
SQL_NUMERIC	S	N	32	32	0	31	-	DOUBLE	Y	8	-
SQL_FLOAT	S	N	-	-	-	-	-	DOUBLE	Y	8	-
SQL_DOUBLE	S	N	-	-	-	-	-	DOUBLE	Y	8	-
SQL_REAL	S	N	-	-	-	-	-	DOUBLE	Y	8	-
SQL_CHAR	S	N	1	254	-	-	-	CHARACTER	N	-	-
SQL_CHAR	S	N	255	4000	-	-	-	VARCHAR	N	-	-
SQL_CHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
SQL_BINARY	S	N	1	254	-	-	-	CHARACTER	Y	-	-
SQL_BINARY	S	N	255	4000	-	-	-	VARCHAR	Y	-	-
SQL_BINARY	S	N	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
SQL_VARCHAR	S	N	1	4000	-	-	-	VARCHAR	N	-	-
SQL_VARBINARY	S	N	1	4000	-	-	-	VARCHAR	Y	-	-
SQL_VARBINARY	S	N	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
SQL_LONGVARCHAR	S	N	-	-	-	-	-	CLOB	N	2147483647	-
SQL_LONGVARBINARY	S	N	-	-	-	-	-	BLOB	Y	2147483647	-
SQL_DATE	S	N	-	-	-	-	-	DATE	Y	4	-
SQL_TIME	S	N	-	-	-	-	-	TIME	Y	3	-
SQL_TIMESTAMP	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-
SQL_BIT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
SQL_TINYINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
SQL_BIGINT	S	N	-	-	-	-	-	DECIMAL	Y	-	-

Default Type Mappings from Oracle Data Sources to DataJoiner

Table 16. Oracle Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
CHAR	S	N	1	254	-	-	-	CHARACTER	N	-	-
CHAR	S	N	255	4000	-	-	-	VARCHAR	N	-	-
| CHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
VARCHAR2	S	N	1	4000	-	-	-	VARCHAR	N	-	-
VARCHAR2	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
| LONG	S	N	-	-	-	-	-	CLOB	N	2147483647	-
NUMBER	S	N	1	38	-84	127	-	DOUBLE	Y	8	-
NUMBER	S	N	1	31	0	31	>=	DECIMAL	Y	-	-
NUMBER	S	N	1	4	0	0	-	SMALLINT	Y	2	-
NUMBER	S	N	5	9	0	0	-	INTEGER	Y	4	-
FLOAT	S	N	-	-	-	-	-	DOUBLE	Y	8	-
RAW	S	N	1	254	-	-	-	CHARACTER	Y	-	-
RAW	S	N	255	4000	-	-	-	VARCHAR	Y	-	-
| RAW	S	N	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
| LONG RAW	S	N	-	-	-	-	-	BLOB	Y	2147483647	-
DATE	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-
ROWID	S	N	-	-	-	-	-	CHARACTER	N	18	-
| MLSLABEL	S	N	-	-	-	-	-	VARCHAR	N	255	-

Default Type Mappings from RDB Data Sources to DataJoiner

Table 17. RDB Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
SQL_INTEGER	S	N	-	-	-	-	-	INTEGER	Y	4	-
SQL_BIGINT	S	N	-	-	-	-	-	DECIMAL	Y	-	-
SQL_SMALLINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
SQL_TINYINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
SQL_DECIMAL	S	N	1	31	0	31	-	DECIMAL	Y	-	-
SQL_DECIMAL	S	N	32	32	0	31	-	DOUBLE	Y	8	-
SQL_NUMERIC	S	N	1	31	0	31	-	DECIMAL	Y	-	-
SQL_NUMERIC	S	N	32	32	0	31	-	DOUBLE	Y	8	-
SQL_FLOAT	S	N	-	-	-	-	-	DOUBLE	Y	8	-
SQL_DOUBLE	S	N	-	-	-	-	-	DOUBLE	Y	8	-
SQL_REAL	S	N	-	-	-	-	-	DOUBLE	Y	8	-
SQL_CHAR	S	N	1	254	-	-	-	CHARACTER	N	-	-
SQL_CHAR	S	N	255	4000	-	-	-	VARCHAR	N	-	-
SQL_CHAR	S	N	4001	32700	-	-	-	CLOB	N	2147483647	-
SQL_CHAR	S	N	32701	65271	-	-	-	CLOB	N	2147483647	-
SQL_BINARY	S	N	1	254	-	-	-	CHARACTER	Y	-	-
SQL_BINARY	S	N	255	4000	-	-	-	VARCHAR	Y	-	-
SQL_BINARY	S	N	4001	32700	-	-	-	BLOB	Y	2147483647	-
SQL_VARCHAR	S	N	1	4000	-	-	-	VARCHAR	N	-	-
SQL_VARCHAR	S	N	4001	32700	-	-	-	CLOB	N	2147483647	-
SQL_VARCHAR	S	N	32701	65269	-	-	-	CLOB	N	2147483647	-
SQL_VARBINARY	S	N	1	4000	-	-	-	VARCHAR	Y	-	-
SQL_VARBINARY	S	N	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
SQL_LONGVARCHAR	S	N	-	-	-	-	-	CLOB	N	2147483647	-
SQL_LONGVARBINARY	S	N	-	-	-	-	-	BLOB	Y	2147483647	-
SQL_DATE	S	N	-	-	-	-	-	DATE	Y	4	-
SQL_TIME	S	N	-	-	-	-	-	TIME	Y	3	-

Table 17. RDB Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
SQL_TIMESTAMP	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-

Default Type Mappings from SQL Anywhere Data Sources to DataJoiner

Table 18. SQL Anywhere Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
SQL_INTEGER	S	N	-	-	-	-	-	INTEGER	Y	4	-
SQL_SMALLINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
SQL_DECIMAL	S	N	1	31	0	31	-	DECIMAL	Y	-	-
SQL_DECIMAL	S	N	32	32	0	31	-	DOUBLE	Y	8	-
SQL_NUMERIC	S	N	1	31	0	31	-	DECIMAL	Y	-	-
SQL_NUMERIC	S	N	32	32	0	31	-	DOUBLE	Y	8	-
SQL_FLOAT	S	N	-	-	-	-	-	DOUBLE	Y	84	-
SQL_DOUBLE	S	N	-	-	-	-	-	DOUBLE	Y	8	-
SQL_REAL	S	N	-	-	-	-	-	DOUBLE	Y	8	-
SQL_CHAR	S	N	1	254	-	-	-	CHARACTER	N	-	-
SQL_CHAR	S	N	255	4000	-	-	-	VARCHAR	N	-	-
SQL_BINARY	S	N	1	254	-	-	-	CHARACTER	Y	-	-
SQL_BINARY	S	N	255	4000	-	-	-	VARCHAR	Y	-	-
SQL_VARCHAR	S	N	1	4000	-	-	-	VARCHAR	N	-	-

Table 18. SQL Anywhere Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
SQL_VARBINARY	S	N	1	4000	-	-	-	VARCHAR	Y	-	-
SQL_LONGVARCHAR	S	N	-	-	-	-	-	CLOB	N	2147483647	-
SQL_LONGVARBINARY	S	N	-	-	-	-	-	BLOB	Y	2147483647	-
SQL_DATE	S	N	-	-	-	-	-	DATE	Y	4	-
SQL_TIME	S	N	-	-	-	-	-	TIME	Y	3	-
SQL_TIMESTAMP	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-
SQL_BIT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
SQL_TINYINT	S	N	-	-	-	-	-	SMALLINT	Y	2	-
SQL_BIGINT	S	N	-	-	-	-	-	DECIMAL	Y	-	-
SQL_CHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
SQL_BINARY	S	N	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
SQL_VARCHAR	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
SQL_VARBINARY	S	N	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-

Default Type Mappings from Sybase Data Sources to DataJoiner

Table 19. Sybase Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
int	S	N	-	-	-	-	-	INTEGER	Y	4	-
intn	S	N	-	-	-	-	-	INTEGER	Y	4	-

Table 19. Sybase Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
smallint	S	N	-	-	-	-	-	SMALLINT	Y	2	-
tinyint	S	N	-	-	-	-	-	SMALLINT	Y	2	-
bit	S	N	-	-	-	-	-	SMALLINT	Y	2	-
float	S	N	-	-	-	-	-	DOUBLE	Y	8	-
floatn	S	N	-	-	-	-	-	DOUBLE	Y	8	-
real	S	N	-	-	-	-	-	DOUBLE	Y	8	-
money	S	N	-	-	-	-	-	DECIMAL	Y	19	4
moneyn	S	N	-	-	-	-	-	DECIMAL	Y	19	4
smallmoney	S	N	-	-	-	-	-	DECIMAL	Y	10	4
smallmoneyn	S	N	-	-	-	-	-	DECIMAL	Y	10	4
decimal	S	N	1	31	0	31	-	DECIMAL	Y	-	-
decimal	S	N	32	32	-	-	-	DOUBLE	Y	8	-
decimaln	S	N	1	31	0	31	-	DECIMAL	Y	-	-
decimaln	S	N	32	32	-	-	-	DOUBLE	Y	8	-
numeric	S	N	1	31	0	31	-	DECIMAL	Y	-	-
numeric	S	N	32	32	-	-	-	DOUBLE	Y	8	-
numericn	S	N	1	31	0	31	-	DECIMAL	Y	-	-
numericn	S	N	32	32	-	-	-	DOUBLE	Y	8	-
char	S	N	1	254	-	-	-	CHARACTER	N	-	-
sysname	S	N	1	254	-	-	-	CHARACTER	N	-	-
char	S	N	255	4000	-	-	-	VARCHAR	N	-	-
char	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
varchar	S	N	1	4000	-	-	-	VARCHAR	N	-	-
varchar	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-
text	S	N	-	-	-	-	-	CLOB	N	2147483647	-
nchar	S	N	1	254	-	-	-	CHARACTER	N	-	-
nchar	S	N	255	4000	-	-	-	VARCHAR	N	-	-
nchar	S	N	4001	32700	-	-	-	LONG VARCHAR	N	32700	-

Table 19. *Sybase Default Data Type Mappings in SYSCAT.SERVER_DATATYPES (Not All Columns Shown) (continued)*

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_S_OPR_P	TYPENAME	BIT_DATA	LENGTH	SCALE
nvarchar	S	N	1	4000	-	-	-	VARCHAR	N	-	-
nvarchar	S	N	4001	32700	-	-	-	LONG VARCHAR	N	327400	-
binary	S	N	1	254	-	-	-	CHARACTER	Y	-	-
binary	S	N	255	4000	-	-	-	VARCHAR	Y	-	-
binary	S	N	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
varbinary	S	N	1	4000	-	-	-	VARCHAR	Y	-	-
varbinary	S	N	4001	32700	-	-	-	LONG VARCHAR	Y	32700	-
image	S	N	-	-	-	-	-	BLOB	Y	2147483647	-
datetime	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-
datetimn	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-
smalldatetime	S	N	-	-	-	-	-	TIMESTAMP	Y	10	-
timestamp	S	N	-	-	-	-	-	VARCHAR	Y	8	-
sysname	S	N	-	-	-	-	-	VARCHAR	N	30	-

Appendix B. Default Reverse Type Mappings

This appendix shows default reverse mappings between DB2 for CS data types defined to DataJoiner and data types defined to data sources. In these mappings, the types defined to DataJoiner point to the types at the data sources. The following list identifies these data sources. It also cites tables that show the default reverse type mappings between the data sources and DataJoiner.

- DB2 for CS (Table 20)
- DB2 for OS/390 (Table 21 on page 156)
- DB2 for OS/400 (Table 22 on page 157)
- DB2 for VM (SQL/DS) (Table 23 on page 159)
- Generic (Table 24 on page 159)
- Informix (Table 25 on page 160)
- Microsoft SQL Server (Table 26 on page 162)
- Oracle (Table 27 on page 163)
- SQL Anywhere (Table 28 on page 164)
- Sybase (Table 29 on page 165)

Default Type Mappings from DataJoiner to DB2 for CS Data Sources

Table 20. DB2 Common Server Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
INTEGER	S	N	-	-	INTEGER	Y	-	4	-	-	-
SMALLINT	S	N	-	-	SMALLINT	Y	-	2	-	-	-
DECIMAL	S	N	-	-	DECIMAL	Y	-	-	-	-	-
DOUBLE	S	N	-	-	DOUBLE	Y	-	8	-	-	-
FLOAT	S	N	-	-	DOUBLE	Y	-	8	-	-	-
CHAR	S	N	-	-	CHARACTER	N	-	-	-	-	-
VARCHAR	S	N	-	-	VARCHAR	N	-	-	-	-	-

Table 20. DB2 Common Server Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
GRAPHIC	S	N	-	-	GRAPHIC	N	-	-	-	-	-
VARGRAPH	S	N	-	-	VARGRAPHIC	Y	-	-	-	-	-
LONGVARCH	S	N	-	-	LONG VARCHAR	N	-	32700	-	-	-
LONGVARG	S	N	-	-	LONG VARGRAPHIC	Y	-	16350	-	-	-
BLOB	S	N	-	-	BLOB	Y	-	2147483647	-	-	-
CLOB	S	N	-	-	CLOB	N	-	2147483647	-	-	-
DBCLOB	S	N	-	-	DBCLOB	Y	-	1073741823	-	-	-
DATE	S	N	-	-	DATE	Y	-	4	-	-	-
TIME	S	N	-	-	TIME	Y	-	3	-	-	-
TIMESTAMP	S	N	-	-	TIMESTAMP	Y	-	10	-	-	-
CHAR	S	Y	-	-	CHARACTER	Y	-	-	-	-	-
VARCHAR	S	Y	-	-	VARCHAR	Y	-	-	-	-	-
LONGVARCH	S	Y	-	-	LONG VARCHAR	Y	-	32700	-	-	-

Default Type Mappings from DataJoiner to DB2 for OS/390 Data Sources

Table 21. DB2 for OS/390 Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
INTEGER	S	N	-	-	INTEGER	Y	-	4	-	-	-
SMALLINT	S	N	-	-	SMALLINT	Y	-	2	-	-	-
DECIMAL	S	N	-	-	DECIMAL	Y	-	-	-	-	-

Table 21. DB2 for OS/390 Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
FLOAT	S	N	-	-	DOUBLE	Y	-	8	-	-	-
CHAR	S	N	-	-	CHARACTER	N	-	-	-	-	-
VARCHAR	S	N	-	-	VARCHAR	N	-	-	-	-	-
CHAR	S	Y	-	-	CHARACTER	Y	-	-	-	-	-
VARCHAR	S	Y	-	-	VARCHAR	Y	-	-	-	-	-
LONGVAR	S	Y	-	-	LONG VARCHAR	Y	-	32700	-	-	-
GRAPHIC	S	N	-	-	GRAPHIC	N	-	-	-	-	-
VARGRAPH	S	N	-	-	VARGRAPHIC	Y	-	-	-	-	-
DATE	S	N	-	-	DATE	Y	-	4	-	-	-
TIME	S	N	-	-	TIME	Y	-	3	-	-	-
TIMESTAMP	S	N	-	-	TIMESTAMP	Y	-	10	-	-	-
LONGVARCH	S	N	-	-	CLOB	N	-	2147483647	-	-	-
LONGVARCH	S	N	-	-	LONG VARCHAR	N	-	32700	-	-	-
FLOAT	S	N	-	-	DOUBLE	Y	-	8	-	-	-
LONGVAR	S	N	-	-	BLOB	Y	-	2147483647	-	-	-

Default Type Mappings from DataJoiner to OS/400 Data Sources

Table 22. DB2 for OS/400 Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
INTEGER	S	N	-	-	INTEGER	Y	-	4	-	-	-

Table 22. DB2 for OS/400 Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
SMALLINT	S	N	-	-	SMALLINT	Y	-	2	-	-	-
DECIMAL	S	N	-	-	DECIMAL	Y	-	-	-	-	-
FLOAT	S	N	-	-	DOUBLE	Y	-	8	-	-	-
CHAR	S	N	-	-	CHARACTER	N	-	-	-	-	-
VARCHAR	S	N	-	-	VARCHAR	N	-	-	-	-	-
GRAPHIC	S	N	-	-	GRAPHIC	N	-	-	-	-	-
VARG	S	N	-	-	VARGRAPHIC	Y	-	-	-	-	-
DATE	S	N	-	-	DATE	Y	-	4	-	-	-
TIME	S	N	-	-	TIME	Y	-	3	-	-	-
TIMESTMP	S	N	-	-	TIMESTAMP	Y	-	10	-	-	-
NUMERIC	S	N	-	-	DECIMAL	Y	-	-	-	-	-
CHAR	S	Y	-	-	CHARACTER	Y	-	-	-	-	-
VARCHAR	S	Y	-	-	VARCHAR	Y	-	-	-	-	-
LONGVARCH	S	Y	-	-	LONG VARCHAR	Y	-	32700	-	-	-
BLOB	S	N	-	-	BLOB	Y	-	2147483647	-	-	-
CLOB	S	N	-	-	CLOB	N	-	2147483647	-	-	-
DBCLOB	S	N	-	-	DBCLOB	Y	-	1073741823	-	-	-
LONGVARCH	S	N	-	-	LONG VARCHAR	N	-	32700	-	-	-

Default Type Mappings from DataJoiner to DB2 for VM Data Sources

Table 23. DB2 for VM (SQL/DS) Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
INTEGER	S	N	-	-	INTEGER	Y	-	4	-	-	-
SMALLINT	S	N	-	-	SMALLINT	Y	-	2	-	-	-
DECIMAL	S	N	-	-	DECIMAL	Y	-	-	-	-	-
DECIMAL	S	N	31	-	DOUBLE	Y	-	8	-	-	-
FLOAT	S	N	-	-	DOUBLE	Y	-	8	-	-	-
CHAR	S	N	-	-	CHARACTER	N	-	-	-	-	-
VARCHAR	S	N	-	-	VARCHAR	N	-	-	-	-	-
LONGVARCHAR	S	N	-	-	LONG VARCHAR	N	-	32700	-	-	-
GRAPHIC	S	N	-	-	GRAPHIC	N	-	-	-	-	-
LONGVARG	S	N	-	-	VARGRAPHIC	Y	-	-	-	-	-
DATE	S	N	-	-	DATE	Y	-	4	-	-	-
TIME	S	N	-	-	TIME	Y	-	3	-	-	-
TIMESTMP	S	N	-	-	TIMESTAMP	Y	-	10	-	-	-

Default Type Mappings from DataJoiner to Generic Data Sources

Table 24. Generic Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
SQL_INTEGER	S	N	-	-	INTEGER	Y	-	4	-	-	-

Table 24. Generic Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
SQL_SMALLINT	S	N	-	-	SMALLINT	Y	-	2	-	-	-
SQL_DECIMAL	S	N	-	-	DECIMAL	Y	-	-	-	-	-
SQL_NUMERIC	S	N	-	-	DECIMAL	Y	-	-	-	-	-
SQL_FLOAT	S	N	-	-	DOUBLE	Y	-	8	-	-	-
SQL_DOUBLE	S	N	-	-	DOUBLE	Y	-	8	-	-	-
SQL_CHAR	S	N	-	-	CHARACTER	N	-	-	-	-	-
SQL_VARCHAR	S	N	-	-	VARCHAR	N	-	-	-	-	-
SQL_BINARY	S	N	-	-	CHARACTER	Y	-	-	-	-	-
SQL_VARBINARY	S	N	-	-	VARCHAR	Y	-	-	-	-	-
SQL_LONGVARCHAR	S	N	-	-	LONG VARCHAR	N	-	32700	-	-	-
SQL_LONGVARCHAR	S	N	-	-	CLOB	N	-	2147483647	-	-	-
SQL_LONGVARBINARY	S	N	-	-	BLOB	Y	-	2147483647	-	-	-
SQL_DATE	S	N	-	-	DATE	Y	-	4	-	-	-
SQL_TIME	S	N	-	-	TIME	Y	-	3	-	-	-
SQL_TIMESTAMP	S	N	-	-	TIMESTAMP	Y	-	10	-	-	-

Default Type Mappings from DataJoiner to Informix Data Sources

Table 25. Informix Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
SMALLINT	S	N	-	-	SMALLINT	Y	-	2	-	-	-

Table 25. Informix Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown) (continued)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
INTEGER	S	N	-	-	INTEGER	Y	-	4	-	-	-
DECIMAL	S	N	-	-	DECIMAL	Y	-	-	-	-	-
DECIMAL	S	N	31	-	DOUBLE	Y	-	8	-	-	-
FLOAT	S	N	-	-	DOUBLE	Y	-	8	-	-	-
CHAR	S	N	-	-	CHARACTER	N	-	-	-	-	-
BYTE	S	N	-	-	BLOB	Y	-	2147483647	-	-	-
VARCHAR	S	N	-	-	VARCHAR	N	1	254	-	-	-
TEXT	S	N	-	-	VARCHAR	N	255	4000	-	-	-
TEXT	S	N	-	-	LONG VARCHAR	N	-	32700	-	-	-
TEXT	S	N	-	-	CLOB	N	-	2147483647	-	-	-
DATE	S	N	-	-	DATE	Y	-	4	-	-	-
DATETIME	S	N	-	-	TIME	Y	-	3	-	-	-
DATETIME	S	N	-	-	TIMESTAMP	Y	-	10	-	-	-
BYTE	S	N	-	-	CHARACTER	Y	-	-	-	-	-
BYTE	S	N	-	-	VARCHAR	Y	-	-	-	-	-
BYTE	S	N	-	-	LONG VARCHAR	Y	-	32700	-	-	-

Default Type Mappings from DataJoiner to Microsoft SQL Server Data Sources

Table 26. Microsoft SQL Server Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
int	S	N	-	-	INTEGER	Y	-	4	-	-	-
smallint	S	N	-	-	SMALLINT	Y	-	2	-	-	-
float	S	N	-	-	DOUBLE	Y	-	8	-	-	-
decimal	S	N	-	-	DECIMAL	Y	-	-	-	-	-
numeric	S	N	-	-	DECIMAL	Y	-	-	-	-	-
char	S	N	-	-	CHARACTER	N	-	-	-	-	-
char	S	N	-	-	VARCHAR	N	1	254	-	-	-
text	S	N	-	-	VARCHAR	N	255	4000	-	-	-
text	S	N	-	-	LONG VARCHAR	N	-	32700	-	-	-
text	S	N	-	-	CLOB	N	-	2147483647	-	-	-
binary	S	N	-	-	CHARACTER	Y	-	-	-	-	-
varbinary	S	N	-	-	VARCHAR	Y	1	254	-	-	-
image	S	N	-	-	VARCHAR	Y	255	4000	-	-	-
image	S	N	-	-	LONG VARCHAR	Y	-	32700	-	-	-
image	S	N	-	-	BLOB	Y	-	2147483647	-	-	-
datetime	S	N	-	-	TIMESTAMP	Y	-	10	-	-	-
datetime	S	N	-	-	TIME	Y	-	3	-	-	-
datetime	S	N	-	-	DATE	Y	-	4	-	-	-

Default Type Mappings from DataJoiner to Oracle Data Sources

Table 27. Oracle Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
CHAR	S	N	-	-	CHARACTER	N	1	254	-	-	-
VARCHAR2	S	N	-	-	VARCHAR	N	1	2000	-	-	-
LONG	S	N	-	-	VARCHAR	N	2001	4000	-	-	-
LONG	S	N	-	-	CLOB	N	-	2147483647	-	-	-
LONG	S	N	-	-	LONG VARCHAR	N	-	32700	-	-	-
FLOAT	S	N	8	-	DOUBLE	Y	-	8	-	-	-
FLOAT	S	N	8	-	DOUBLE	Y	-	8	-	-	-
NUMBER	S	N	-	-	DECIMAL	Y	-	-	-	-	-
NUMBER	S	N	-	-	DECIMAL	Y	-	-	-	-	-
NUMBER	S	N	5	0	SMALLINT	Y	-	2	-	-	-
NUMBER	S	N	10	0	INTEGER	Y	-	4	-	-	-
RAW	S	N	-	-	CHARACTER	Y	-	-	-	-	-
RAW	S	N	-	-	VARCHAR	Y	1	255	-	-	-
LONG RAW	S	N	-	-	VARCHAR	Y	256	4000	-	-	-
LONG RAW	S	N	-	-	LONG VARCHAR	Y	-	32700	-	-	-
LONG RAW	S	N	-	-	BLOB	Y	-	2147483647	-	-	-
DATE	S	N	-	-	TIMESTAMP	Y	-	10	-	-	-
DATE	S	N	-	-	DATE	Y	-	4	-	-	-
DATE	S	N	-	-	TIME	Y	-	3	-	-	-

Default Type Mappings from DataJoiner to SQL Anywhere Data Sources

Table 28. SQL Anywhere Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
INTEGER	S	N	-	-	INTEGER	Y	-	4	-	-	-
SMALLINT	S	N	-	-	SMALLINT	Y	-	2	-	-	-
DECIMAL	S	N	-	-	DECIMAL	Y	-	-	-	-	-
NUMERIC	S	N	-	-	DECIMAL	Y	-	-	-	-	-
FLOAT	S	N	-	-	DOUBLE	Y	-	8	-	-	-
DOUBLE	S	N	-	-	DOUBLE	Y	-	8	-	-	-
CHAR	S	N	-	-	CHARACTER	N	-	-	-	-	-
VARCHAR	S	N	-	-	VARCHAR	N	-	-	-	-	-
BINARY	S	N	-	-	CHARACTER	Y	1	254	-	-	-
BINARY	S	N	-	-	VARCHAR	Y	-	-	-	-	-
LONG BINARY	S	N	-	-	LONG VARCHAR	Y	-	32700	-	-	-
LONG VARCHAR	S	N	-	-	LONG VARCHAR	N	-	32700	-	-	-
LONG VARCHAR	S	N	-	-	CLOB	N	-	2147483647	-	-	-
LONG BINARY	S	N	-	-	BLOB	Y	-	2147483647	-	-	-
DATE	S	N	-	-	DATE	Y	-	4	-	-	-
TIME	S	N	-	-	TIME	Y	-	3	-	-	-
TIMESTAMP	S	N	-	-	TIMESTAMP	Y	-	10	-	-	-

Default Type Mappings from DataJoiner to Sybase Data Sources

Table 29. Sybase Default Data Type Mappings in SYSCAT.REVTYPEMAPPINGS (Not All Columns Shown)

REMOTE_TYPENAME	REMOTE_META_TYPE	REMOTE_BIT_DATA	REMOTE_LENGTH	REMOTE_SCALE	TYPENAME	BIT_DATA	LOCAL_LOWER_LEN	LOCAL_UPPER_LEN	LOCAL_LOWER_SCALE	LOCAL_UPPER_SCALE	LOCAL_S_OPR_P
int	S	N	-	-	INTEGER	Y	-	4	-	-	-
smallint	S	N	-	-	SMALLINT	Y	-	2	-	-	-
float	S	N	-	-	DOUBLE	Y	-	8	-	-	-
decimal	S	N	-	-	DECIMAL	Y	-	-	-	-	-
numeric	S	N	-	-	DECIMAL	Y	-	-	-	-	-
char	S	N	-	-	CHARACTER	N	-	-	-	-	-
char	S	N	-	-	VARCHAR	N	1	254	-	-	-
text	S	N	-	-	VARCHAR	N	255	4000	-	-	-
text	S	N	-	-	CLOB	N	-	2147483647	-	-	-
text	S	N	-	-	LONG VARCHAR	N	-	32700	-	-	-
binary	S	N	-	-	CHARACTER	Y	-	-	-	-	-
binary	S	N	-	-	VARCHAR	Y	1	254	-	-	-
binary	S	N	-	-	VARCHAR	Y	255	4000	-	-	-
image	S	N	-	-	LONG VARCHAR	Y	-	32700	-	-	-
image	S	N	-	-	BLOB	Y	-	2147483647	-	-	-
datetime	S	N	-	-	TIMESTAMP	Y	-	10	-	-	-
datetime	S	N	-	-	TIME	Y	-	3	-	-	-
datetime	S	N	-	-	DATE	Y	-	4	-	-	-

Appendix C. Combined DataJoiner and DB2 for CS Syntax for CREATE TABLE

You use the DB2 for CS CREATE TABLE statement to create DataJoiner tables, and the DataJoiner CREATE TABLE statement to create data source tables from DataJoiner. The DataJoiner statement contains elements of the DB2 for CS statement, plus elements of its own. This appendix shows both statements in combination.

For more information about the DB2 for CS CREATE TABLE statement (for example, descriptions of keywords and parameters), see *DATABASE 2 SQL Reference for common servers*. For more information about the DataJoiner CREATE TABLE statement, see “CREATE TABLE” on page 107.

In the following diagram:

- Syntax used in the DB2 for CS CREATE TABLE statement is marked “DB2”.
- Syntax used in the DataJoiner CREATE TABLE statement is marked “DataJoiner”.

```
                              (1)
►►──CREATE TABLE──table-name──────────────────────────────────────────────►

      ┌─,─────────────────────────────────────────────────┐
      ▼                                                   │
►──(──┬─┤ Column Definition (DataJoiner and DB2) ├────────┬┴──)───────────►
      ├─┤ Primary Key Constraint (DataJoiner and DB2) ├───┤
      ├─┤ Referential Constraint (DB2) ├──────────────────┤
      └─┤ Check Constraint (DB2) ├────────────────────────┘

    ┌─DATA CAPTURE NONE────┐     (2)
►───┴─DATA CAPTURE CHANGES─┴──────────────────────────────────────────────►

                (3)
►───┬─IN──server-name─────┬──────────────────────────────────┬─┬─────────►◄
    │                     │                           (4)    │ │
    │                     └─REMOTE OPTION──'remote-option'───┘ │
    │                 (5)                                      │
    └─IN──tablespace-name───┤ Tablespace Options (DB2) ├───────┘
```

Notes:

1. DB2 and DataJoiner
2. DB2
3. Required in the DataJoiner CREATE TABLE statement
4. DataJoiner
5. DB2

Column Definition (DataJoiner)

```
                                  ┌─────────────┐
                                  ▼             │
├──column-name──┤ Data Type ├──────┬─────────────┬┴──────────────────────────┤
                                  ├─NOT NULL────┤
                                  └─PRIMARY KEY─┘
```

Column Definition (DB2)

```
├──column-name──┤ Data Type ├──────────────────────────────────────────────────►

  ┌──────────────────────────────────────────────────────────────────────┐
  ▼                                                                      │
►───┬──────────────────────────────────────────────────────────────────┬──┴──┤
    ├─NOT NULL─────────────────────────────────────────────────────────┤
    ├─┤ Default Clause ├───────────────────────────────────────────────┤
    │                            (1)                                   │
    ├─┤ Lob Options Clause ├───────────────────────────────────────────┤
    │                                                         (3)      │
    └─┬─────────────────────────────────┬─┬─PRIMARY KEY─────────────┬─┘
      │          (2)                    │ ├─┤ References Clause ├───┤
      └─CONSTRAINT─────constraint-name──┘ └─CHECK──(──check-condition──)─┘
```

Notes:

1. This clause applies only to large object types (BLOB, CLOB, and DBCLOB) and distinct types based on large object types.
2. For compatibility with DB2/6000, you can omit the CONSTRAINT keyword if your Column Definition Clause includes a References Clause.
3. DataJoiner and DB2

Data Type for DataJoiner and DB2

```
├──┬─┬─INTEGER─┬──────────────────────────────────────────────────────┬──┤
   │ └─INT─────┘                                                      │
   ├─SMALLINT─────────────────────────────────────────────────────────┤
   ├─┬─DOUBLE───────────┬─────────────────────────────────────────────┤
   │ ├─DOUBLE PRECISION─┤                                             │
   │ └─FLOAT────────────┘                                             │
   ├──┬─DECIMAL─┬──┬──────────────────────────────────┬───────────────┤
   │  ├─DEC─────┤  └─(──integer──┬────────────┬──)──┘                │
   │  ├─NUMERIC─┤                └─,──integer─┘                       │
   │  └─NUM─────┘                                                     │
   ├──┬──┬─CHARACTER─┬──┬──────────────┬──┬──┬────────────────────┬───┤
   │  │  └─CHAR──────┘  └─(──integer──)─┘  │  │                (1) │   │
   │  ├──┬─VARCHAR─────────────┬─(──integer──)─┤  └─FOR BIT DATA───────┘   │
   │  │  ├─CHARACTER──VARYING─┤                │                          │
   │  │  └─CHAR──VARYING──────┘                │                          │
   │  └─LONG VARCHAR───────────────────────────┘                          │
   │                                                                  │
   ├──┬─BLOB───┬───(──integer──┬───┬──)───────────────────────────────┤
   │  ├─CLOB───┤               ├─K─┤                                  │
   │  └─DBCLOB─┘               ├─M─┤                                  │
   │                           └─G─┘                                  │
   ├─GRAPHIC──┬──────────────┬────────────────────────────────────────┤
   │          └─(──integer──)┘                                        │
   ├─VARGRAPHIC──(──integer──)────────────────────────────────────────┤
   ├─LONG VARGRAPHIC──────────────────────────────────────────────────┤
   ├─DATE─────────────────────────────────────────────────────────────┤
   ├─TIME─────────────────────────────────────────────────────────────┤
   ├─TIMESTAMP────────────────────────────────────────────────────────┤
   └─distinct-type-name───────────────────────────────────────────────┘
```

Notes:

1. You can specify FOR BIT DATA in random order with the other column constraints that follow.

Default Clause (DB2)

```
   ┌─WITH─┐
├──┴──────┴──┬─────────┬──┬─constant──────────────────────────────────────────┬──┤
             └─DEFAULT─┘  ├─datetime-special-register─────────────────────────┤
                          ├─USER──────────────────────────────────────────────┤
                          ├─NULL──────────────────────────────────────────────┤
                          └─cost-function──(──┬─constant──────────────────┬─)─┘
                                              ├─datetime-special-register─┤
                                              └─PRIMARY KEY───────────────┘
```

Lob Options Clause (DB2)

```
   ┌─LOGGED─────────┐ ┌─NOT COMPACT─┐
├──┼────────────────┼─┼─────────────┼──────────────────────────────────────────┤
   └──NOT LOGGED────┘ └──COMPACT────┘
```

References Clause (DB2)

```
├──REFERENCES──table-name──┬─────────────────────────────┬──┤ Rule Clause (DB2) ├────┤
                           │          ┌─,───────────┐    │
                           │          ↓             │    │
                           └─(────────column-name───┴─)──┘
```

Rule Clause (DB2)

```
   ┌─DATA CAPTURE NONE─────────┐
├──┴─ON DELETE──┬─RESTRICT─┬───┴─────────────────────────────────────────────────┤
                ├─CASCADE──┤
                └─SET NULL─┘
```

Primary Key Constraint (DataJoiner and DB2)

```
                                                        ┌─,───────────┐
                                                        ↓             │
├──┬─────────────────────────────────┬──PRIMARY KEY──(──column-name───┴──)────────┤
   └─CONSTRAINT──constraint-name─────┘
```

Referential Constraint (DB2)

```
                                                        ┌─,──────────────────┐
                                                        ↓                (1) │
├──┬─────────────────────────────────┬──FOREIGN KEY──(──column-name──────────┴──)────►
   └─CONSTRAINT──constraint-name─────┘

►──┤ References Clause ├──────────────────────────────────────────────────────────────┤
```

Notes:

1. For compatibility with DB2 /6000, you can omit the CONSTRAINT keyword and specify constraint-name after the FOREIGN KEY keyword.

Check Constraint (DB2)

```
├──┬─────────────────────────────────┬──CHECK──(──check-condition──)──────────────┤
   └─CONSTRAINT──constraint-name─────┘
```

Tablespace Options (DB2)

```
├──┬──────────────────────────────────┬──┬───────────────────────────────┬──────────────────┤
   │                             (1)  │  └─LONG IN──tablespace-name──┘
   └─INDEX IN──tablespace-name────────┘
```

Notes:

1. You can specify which table space will contain a table's index only when the table is created.

Appendix D. Sample Program Fragment for Invoking a Stored Procedure

This appendix contains an example of SQL statements that invoke a Sybase stored procedure. The statements are CONNECT, ALLOCATE CURSOR, CALL, DESCRIBE CURSOR, FETCH, and CLOSE.

The following list itemizes the key events specified in this fragment. The number next to each item also appears to the right of the code for that item.

1 Connect to the database.

2 Associate a cursor with this stored procedure's nickname.

3 Invoke the stored procedure.

4 If SQLCODE +466 is returned, it means that a result set exists.

5 Describe the result set.

6 Retrieve a row of data from the stored procedure.

7 Close the cursor.

8 If SQLCODE +467 is returned, it means that another result set exists.

```
printf( "Dynamic Insert Program\n" );
EXEC SQL CONNECT TO DJV2 IN SHARE MODE;  1
printf( "CONNECT :  SQLCODE = %ld\n", SQLCODE );
if ( SQLCODE != 0 )
   exit(1);

EXEC SQL ALLOCATE c1 CURSOR for PROCEDURE s_two1;  2
printf( "ALLOCATE c1\n" );

EXEC SQL CALL S_TWO1;  3
printf("CALL s_two %ld\n",SQLCODE);

if (SQLCODE == +466)
   more_results = TRUE;  4
else
   more_results = FALSE;

 while (more_results == TRUE)
   {
   EXEC SQL DESCRIBE CURSOR c1 INTO :*pgm_sqlda;  5
   printf("DESCRIBE CURSOR %ld\n",SQLCODE);
   if (SQLCODE >= 0)
     for (i=0; isqld; i++)
       {
       length = pgm_sqlda->sqlvar[i].sqllen;
       stg_ptr = (char *)malloc(length);
       memset(stg_ptr,'\0',length);
       pgm_sqlda->sqlvar[i].sqldata = stg_ptr;
       length = 2;
       stg_ptr = (char *)malloc(length);
     memset(stg_ptr,'\0',length);
     pgm_sqlda->sqlvar[i].sqlind = (short *)stg_ptr;
     }

 while ((SQLCODE >= 0)  & (SQLCODE != 100))
  {
  EXEC SQL FETCH c1 USING DESCRIPTOR :*pgm_sqlda;  6
  printf("FETCH c1 %ld\n",SQLCODE);
  if (SQLCODE == 0)
    {
    if (pgm_sqlda->sqld == 2)
      {
      memcpy(fruit,pgm_sqlda->sqlvar[0].sqldata,
             pgm_sqlda->sqlvar[0]sqllen);
      quantity = (long *)(pgm_sqlda->sqlvar[1].sqldata);
      printf("Fruit is %s     quantity is %ld\n",fruit,*quantity);
      }
    else
      {
      quantity = (long *)(pgm_sqlda->sqlvar[0].sqldata);
      printf("Quantity is %ld\n",*quantity);
      }
    }
  }

EXEC SQL CLOSE c1;  7
printf("CLOSE c1 %ld\n",SQLCODE);
if (SQLCODE == +467)
  more_results = TRUE;  8
else
  more_results = FALSE;
}
```

Appendix E. System Catalog Views

DataJoiner maintains a set of system catalog views for each database created. All system catalog views have one of two qualifiers: SYSCAT or SYSSTAT. Several of these views are identical to the catalog views maintained by DB2 for its databases. The DataJoiner catalog views that contain values specific to DataJoiner and the new catalog views for DataJoiner are described in this appendix. See the *DATABASE 2 SQL Reference* for information on catalog views that are identical in DataJoiner and DB2. The views that have been modified for use by DataJoiner include:

View	**Page**
SYSCAT.COLUMNS	176
SYSCAT.INDEXES	179
SYSCAT.TABLES	196

The DataJoiner views (not provided with DB2) contain information that is used in communicating with data sources. The DataJoiner views are:

View	**Page**
SYSCAT.PASSTHRU_AUTH	181
SYSCAT.PROCEDURES	181
SYSCAT.PROCPARMS	182
SYSCAT.REMOTEUSERS	183
SYSCAT.REVTYPEMAPPINGS	184
SYSCAT.SERVERS	187
SYSSTAT.SERVERS	190
SYSCAT.SERVER_DATATYPES	190
SYSCAT.SERVER_FUNCTIONS	193
SYSSTAT.SERVER_FUNCTIONS	194
SYSCAT.SERVER_OPTIONS	195

All the system catalog views are created when you run the CREATE DATABASE statement. You cannot explicitly create or drop the catalog views. The system catalog views are updated during normal operation as a result of SQL data definition statements, environment routines, and certain utilities.

Data in the system catalog views is available through normal SQL query facilities.

SYSCAT.COLUMNS

The SYSCAT.COLUMNS catalog view is filled in by DataJoiner when you create a nickname. DataJoiner inserts one row in the SYSCAT.COLUMNS catalog view for each column that is defined for a nickname, table, or view. All catalog views have entries in SYSCAT.COLUMNS. Table 30 describes the columns in this view.

Table 30. Columns in SYSCAT.COLUMNS Catalog View

Name	Data Type	Nullable?	Content
TABSCHEMA	CHAR(8)	No	Qualifier of the nickname, table, or view.
TABNAME	VARCHAR(18)	No	Nickname, table, or view, that contains the column.
COLNAME	VARCHAR(18)	No	Column name. You can specify a different name from the name used at the data source.
COLNO	SMALLINT	No	Numerical place of column in table or view.
TYPESCHEMA	CHAR(8)	No	Data type qualifier, if the data type of the column is distinct. If not distinct, TYPESCHEMA contains the value SYSIBM and TYPENAME contains the data type of the column (in long form; for example, CHARACTER).
TYPENAME	VARCHAR(18)	No	Data type of the column (LONGVAR for LONG VARCHAR type, VARGRAPH for VARGRAPHIC type, LONGVARG for LONG VARGRAPHIC type, and TIMESTMP for TIMESTAMP type).
LENGTH	INTEGER	No	For decimal data types, LENGTH is the maximum precision that a value can have. For character data types, LENGTH is the maximum number of characters that a value can have.
SCALE	SMALLINT	No	Scale for DECIMAL fields; 0 if not DECIMAL.
DEFAULT	VARCHAR(254)	Yes	Default value for the column of a table expressed as a constant, special register, or cast-function appropriate for the data type of the column. Can also be the keyword NULL. (See note 1.) Values can be converted from what was specified as a default value. For example, date and time constraints are presented in ISO format and cast-function names are delimited. (See note 2.) A null value is assumed if a DEFAULT clause was not specified or the column is a view column DataJoiner does not retrieve the default value for columns of a nickname.
NULLS (See note 3.)	CHAR(1)	No	Valid values are: Y Nullable N Not nullable

Table 30. Columns in SYSCAT.COLUMNS Catalog View (continued)

Name	Data Type	Nullable?	Content
CODEPAGE	SMALLINT	No	0 if the column is defined with the FOR BIT DATA option. Contains an SBCS code page ID if COLTYPE=CHAR, CLOB, VARCHAR, or LONG VARCHAR, and if FOR BIT DATA is not set. Otherwise, contents are unpredictable.
LOGGED	CHAR(1)	No	Applies only to columns whose type is LOB or distinct based on LOB (blank otherwise). Y Column is logged. N Column is not logged.
COMPACT	CHAR(1)	No	Applies only to columns whose type is LOB or distinct based on LOB (blank otherwise). Y Column is compacted in storage. N Column is not compacted.
COLCARD	INTEGER	No	Number of distinct values in the column; −1 if statistics are not gathered.
HIGH2KEY	VARCHAR(33)	No	Second highest value of the column. This field is empty if statistics are not gathered. The data in columns with noncharacter data types is printable. (See note 4.)
LOW2KEY	VARCHAR(33)	No	Second lowest value of the column. This field is empty if statistics are not gathered. The data in columns with noncharacter data types is printable. (See note 4.)
AVGCOLLEN	INTEGER	No	Average column length.
KEYSEQ	SMALLINT	Yes	Ordinality of the column within the primary key of its table. This field is null or 0 if the column is not part of the primary key.
NQUANTILES	SMALLINT	No	Number of quantile values recorded in SYSCAT.SYSCOLDIST for this column; -1 if no statistics.
NMOSTFREQ	SMALLINT	No	Number of most-frequent values recorded in SYSCAT.SYSCOLDIST for this column; -1 if no statistics.
REMOTE_COLNAME	VARCHAR(128)	Yes	Name of the column as defined on the data source; Null when TABNAME is not a nickname. This field is case-sensitive.
REMOTE_TYPESCHEMA	VARCHAR(128)	Yes	Qualified name of the data type name as defined on the data source. Null when TABNAME is not a nickname or if no qualified name is provided.

Table 30. Columns in SYSCAT.COLUMNS Catalog View (continued)

Name	Data Type	Nullable?	Content
REMOTE_TYPENAME	VARCHAR(128)	Yes	Data type of column as defined on the data source. It is updatable only on Oracle with types of VARCHAR or VARCHAR2 to VARCHARNTB, which indicates that the remote column contains varying-length character strings with no trailing blanks. This field applies only to nicknames. It is null when TABNAME is not a nickname.
REMOTE_SOURCENAME	VARCHAR(128)	Yes	Name of the source type for distinct type. Note that this field contains only system built-in type names. Null when TABNAME is not a nickname or REMOTE_TYPE is not a distinct type.
REMOTE_LENGTH	INTEGER	Yes	Applies to columns of tables or views at data sources. For columns with decimal data types, LENGTH is the maximum precision that a value can have. For columns with character data types, LENGTH is the maximum number of characters that a value can have.
REMOTE_SCALE	SMALLINT	Yes	Scale for remote DECIMAL fields. 0 if REMOTE_TYPE is not DECIMAL. Null when TABNAME is not a nickname.
REMARKS	VARCHAR(254)	Yes	User’s comments.

Notes on Table 30 on page 176:

1. Value D (indicating not null with a default) is no longer used. Instead, use of WITH DEFAULT is indicated by a non-null value in the DEFAULT column.
2. Previously, case-function names were not delimited and can still appear this way in the DEFAULT column. Also, some view columns included default values which will still appear in the DEFAULT column.
3. Some data sources support declaration of columns as both WITH DEFAULT and NULLABLE. If a column is declared at a data source as both WITH DEFAULT and NULLABLE, the NULLS column of SYSIBM.SYSCOLUMNS contains a value of ’Y’, meaning NULLABLE. This value is expected and does not inhibit any SQL.

   When views are created on either Oracle or Sybase, the DEFAULT attribute is not carried forward from the base table column to the view column (for example, if you select from Oracle’s SYS.ALL_TAB_COLUMNS, the DEFAULT_LENGTH column always contains ’0’ for views). Because DataJoiner has no mechanism to determine whether an Oracle or Sybase view column is WITH DEFAULT, it assumes that any NOT NULL column of an Oracle or Sybase view is WITH DEFAULT and sets the NULLS column accordingly. This setting does not inhibit any SQL but defers detection of missing column errors (errors identified by an SQLCODE of -407) from bind/prep time until run time.
4. DataJoiner does not acquire information for the HIGH2KEY and LOW2KEY columns when nicknames are created. To get this information, run the RUNSTATS utility on the nickname.

SYSCAT.INDEXES

DataJoiner inserts one row in the SYSCAT.INDEXES catalog view for each index that is defined for a nickname or table.

The base table that underlies SYSCAT.INDEXES is populated when:

- The CREATE NICKNAME statement is run to create a nickname for a table that has an index, and information about the index is directly accessible from the data source where the table is stored.
- The CREATE INDEX statement is run to create a local definition of an index of a table that is stored at a data source.

The base table that underlies SYSCAT.INDEXES is updated when:

- The RUNSTATS utility is run against a table with an index for which entries already exist in SYSCAT.INDEXES.
- An index is dropped.

Table 31 describes the columns in SYSCAT.INDEXES.

Table 31. Columns in SYSCAT.INDEXES Catalog View

Name	Data Type	Nullable?	Content
INDSCHEMA	CHAR(8)	No	Qualifier for the index.
INDNAME	VARCHAR(18)	No	Name of the index.
DEFINER	CHAR(8)	No	User that created the index.
TABSCHEMA	CHAR(8)	No	Qualifier for the name of the table on which the index is defined.
TABNAME	VARCHAR(18)	No	Nickname or the name of the table on which the index is defined.
COLNAMES	VARCHAR(320)	No	List of column names, each preceded by + or − to indicate ascending or descending order. This column corresponds to the COLNAME column in the SYSCAT.COLUMNS catalog view.
UNIQUERULE	CHAR(1)	No	Valid values are: D Duplicates are allowed. U Only unique entries are allowed. P Primary index.
COLCOUNT	SMALLINT	No	Number of columns in the key.
IID	SMALLINT	No	Internal index ID.
NLEAF	INTEGER	No	Number of leaf pages; −1 if statistics are not gathered.
NLEVELS	SMALLINT	No	Number of index levels; −1 if statistics are not gathered.

Table 31. Columns in SYSCAT.INDEXES Catalog View (continued)

Name	Data Type	Nullable?	Content
FIRSTKEYCARD	INTEGER	No	Number of distinct first key values; −1 if statistics are not gathered.
FULLKEYCARD	INTEGER	Yes	Number of distinct full key values; −1 if statistics are not gathered.
CLUSTERRATIO	SMALLINT	No	Degree of data clustering with the index; −1 if statistics are not gathered or if detailed index statistics are gathered (in which case, CLUSTERFACTOR is used instead).
CLUSTERFACTOR	DOUBLE	No	Finer measurement of the degree of data clustering with the index; −1 if statistics are not gathered or if the index is defined for a nickname.
USER_DEFINED	SMALLINT	No	1 if this index was defined by a user and has not been dropped; otherwise 0.
SYSTEM_REQUIRED	SMALLINT	No	Number of primary key constraints that are supported by this index.
CREATE_TIME	TIMESTAMP	No	Time when the index was created.
STATS_TIME	TIMESTAMP	Yes	Last time when any change was made to recorded statistics for this index. Null if no statistics are available.
PAGE_FETCH_PAIRS	VARCHAR(254)	No	A list of pairs of integers, represented in character form. Each pair represents the number of pages in a hypothetical buffer and the number of page fetches required to scan the table with this index using that hypothetical buffer. This is a zero-length string if no data is available.
REMOTE_INDSCHEMA	VARCHAR(128)	Yes	Authorization ID on the data source. This field is case-sensitive. It is null when any of the following statements are true: • The index is on a local table. • The index is on a nickname rather than on a table represented by the nickname. • The index was created by a user over a nickname.
REMOTE_INDNAME	VARCHAR(128)	Yes	Name of the index as defined on the data source. This field is case-sensitive. It is null when: • The index is on a local table. • The index is on a nickname rather than on a table represented by the nickname.
REMARKS	VARCHAR(254)	Yes	User’s comments.

SYSCAT.PASSTHRU_AUTH

This catalog view contains information about authorizations to query data sources in pass-through sessions. A constraint on the base table requires that the values in SERVER correspond to the values in the SERVER column of SYSCAT.SERVERS. None of the fields in SYSCAT.PASSTHRU_AUTH are nullable.

Table 32 describes the columns in SYSCAT.PASSTHRU_AUTH.

Table 32. Columns in SYSCAT.PASSTHRU_AUTH Catalog View

Name	Data Type	Content
GRANTOR	CHAR(8)	Authorization ID of the user who granted the privilege.
GRANTEE	CHAR(8)	Authorization ID of the user or group who holds the privilege.
GRANTEETYPE	CHAR(1)	A letter that specifies the type of grantee: U Grantee is an individual user. G Grantee is a group.
SERVER	VARCHAR(18)	Name of the data source that the user or group is being granted authorization to.

SYSCAT.PROCEDURES

This catalog view contains information about stored procedures.

A unique index is defined on the PROC_NICKNAME, PROC_SCHEMA, and SERVER columns. A constraint on the base table requires that the values in SERVER correspond to the values in the SERVER column of SYSCAT.SERVERS.

Table 33 describes the columns in SYSCAT.PROCEDURES.

Table 33. Columns in SYSCAT.PROCEDURES Catalog View

Name	Data Type	Nullable?	Content
PROC_NICKNAME	CHAR(8)	No	Nickname for the stored procedure specified in the CALL statement.
PROCSCHEMA	CHAR(8)	No	Qualifier of the stored procedure nickname.
SERVER	VARCHAR(18)	No	Name of the data source on which the stored procedure resides. The value of this field must correspond to a value in the SERVER column of SYSCAT.SERVERS.
REMOTE_SCHEMA	VARCHAR(128)	Yes	Owner of the stored procedure at the data source. If the data source does not support qualified stored procedure names, this column can be null.
REMOTE_PROC	VARCHAR(128)	No	Name of the stored procedure at the data source.

Table 33. Columns in SYSCAT.PROCEDURES Catalog View (continued)

Name	Data Type	Nullable?	Content
REMARKS	VARCHAR(254)	Yes	User-supplied comment.

SYSCAT.PROCPARMS

This catalog view contains information about stored procedure parameters.

A unique index is defined on the PROC_NICKNAME, PROC_SCHEMA, SERVER, and POSITION columns. A constraint on the base table requires that the values in the PROC_NICKNAME correspond to the values in the PROC_NICKNAME column of SYSCAT.PROCEDURES.

Table 34 describes the columns in SYSCAT.PROCPARMS.

Table 34. Columns in SYSCAT.PROCPARMS Catalog View

Name	Data Type	Nullable?	Content
PROC_NICKNAME	CHAR(8)	No	Nickname for the stored procedure. The value of this field corresponds to the NICKNAME column in SYSCAT.PROCEDURES.
PROCSCHEMA	CHAR(8)	No	Qualifier of the stored procedure nickname.
SERVER	VARCHAR(18)	No	Name of the data source on which the stored procedure resides.
POSITION	INTEGER	No	Position in the parameter list.
NAME	VARCHAR(128)	Yes	The parameter name at the data source.
TYPE	CHAR(5)	No	Whether this parameter is used for: • IN • OUT • INOUT
CODEPAGE	SMALLINT	Yes	DataJoiner SBCS codepage if applicable. This field will be null for numeric fields.
DBCS_CODEPAGE	SMALLINT	Yes	DataJoiner DBCS if applicable. This field will be null for numeric fields.
DATATYPE	CHAR(8)	No	DataJoiner data type of the parameter.
NULLS	CHAR(1)	No	Values valid locally to DataJoiner: Y Nullable; N Not nullable

Table 34. Columns in SYSCAT.PROCPARMS Catalog View (continued)

Name	Data Type	Nullable?	Content
LENGTH	SMALLINT	No	DataJoiner parameter length. For decimal fields, this value indicates the precision.
SCALE	SMALLINT	No	DataJoiner scale for decimal fields; 0 for nondecimal fields.
REMOTE_CODEPAGE	SMALLINT	Yes	Data source codepage if applicable. This field will be null for numeric fields. It is reserved for future use.
REMOTE_DATATYPE	VARCHAR(128)	No	Data source parameter type.
REMOTE_NULLS	CHAR(1)	No	Values valid at the data source: Y Nullable N Not nullable
REMOTE_LENGTH	SMALLINT	No	Data source parameter length. For decimal fields, this value indicates the precision.
REMOTE_SCALE	SMALLINT	No	Data source scale for decimal fields; 0 for nondecimal fields.

SYSCAT.REMOTEUSERS

This view contains information about the authorization IDs and passwords that are used to access remote data sources. The unique index for the source table includes the AUTHID and SERVER columns. You cannot specify two rows with identical values for both of these columns. None of the fields in SYSCAT.REMOTEUSERS are nullable.

Table 35 describes the columns in SYSCAT.REMOTEUSERS.

Table 35. Columns in SYSCAT.REMOTEUSERS Catalog View

Name	Data Type	Content
AUTHID	CHAR(8)	DataJoiner (local) authorization ID. This field is uppercase.
SERVER	VARCHAR(18)	Name of the server at which the remote_authid is defined. This field must match an entry in the SERVER column of the SYSSERVERS table. This field is uppercase.
REMOTE_AUTHID	VARCHAR(30)	Authorization ID used at the data source. This field is case-sensitive.
REMOTE_PW	VARCHAR(32)	Password of the remote_authid used at the data source, stored in an encrypted form. SELECT statements performed on SYSREMOTEUSERS do not return a value for this column (they return an asterisk (*)). If this password is not available when you create the entry, insert an empty string (''), because this column cannot be null. This field is case-sensitive.
CONNECTOPT	VARCHAR(256)	The valid values of this field are listed in Table 36 on page 184.

The content of CONNECTOPT is protocol-dependent. The use by protocol is summarized in Table 36.

Table 36. Protocol Options in the CONNECTOPT Column of SYSCAT.REMOTEUSERS

Data Access Module	Use
drda	Accounting string (see the *DDCS Version 2.3 User's Guide* for more information).
drdaIP	Accounting string (see the *DDCS Version 2.3 User's Guide* for more information).
db2ra	Not Used (NU)
informix	NU
sqlnet	NU
dblib	NU
ctlib	NU

SYSCAT.REVTYPEMAPPINGS

This catalog view shows reverse data type mappings; that is, mappings from (1) data types locally defined to DataJoiner to (2) data source data types. A mapping might associate one user-defined type with another, or a user-defined type with a built-in type, or one built-in type with another. Most of the mappings between built-in types are defaults; for a list, see "Appendix B. Default Reverse Type Mappings" on page 155. All other mappings were created with the CREATE REVERSE TYPE MAPPING STATEMENT (page 89).

DataJoiner uses these mappings to determine what data source data types to define for columns of tables created with DataJoiner's CREATE TABLE statement. For more information, see "Data Type Mappings" on page 26.

Table 37 describes the columns in SYSCAT.REVTYPEMAPPINGS.

Table 37. Columns in SYSCAT.REVTYPEMAPPINGS Catalog View

Name	Data Type	Nullable?	Content
TYPE_MAPPING	VARCHAR(18)	No	Name of the mapping. (If a mapping is created with the CREATE REVERSE TYPE MAPPING statement, and no name is specified in the statement, DataJoiner supplies the name. DataJoiner also supplies the name for all default mappings.)
TYPESCHEMA	CHAR(8)	Yes	Schema to which the local type belongs. If TYPESCHEMA is null, assume that this type is built-in.

Table 37. Columns in SYSCAT.REVTYPEMAPPINGS Catalog View (continued)

Name	Data Type	Nullable?	Content
TYPENAME	VARCHAR(18)	No	The local data type. TYPESCHEMA and TYPENAME make up the type's fully-qualified name. This is the name by which DataJoiner references the remote type. The name is stored in the DataJoiner catalog when a nickname for the table is created.
DEFINER	CHAR(8)	No	Authorization ID under which the mapping was created. If the value of DEFINER is sysibm, it means that the local type is built-in.
LOCAL_LOWER_LEN	INTEGER	Yes	If the local type is decimal, LOCAL_LOWER_LEN is the minimum number of digits that values of this type can have. If the type is a character type other than decimal, LOCAL_LOWER_LEN is the minimum number of characters that values of the type can have.
LOCAL_UPPER_LEN	INTEGER	Yes	Maximum precision. If the local type is decimal, LOCAL_UPPER_LEN is the maximum number of digits that values of this type can have. If the type is a character type other than decimal, LOCAL_UPPER_LEN is the maximum number of characters that values of the type can have. Together, LOCAL_LOWER_LEN and LOCAL_UPPER_LEN indicate the full allowable precision for a value.
LOCAL_LOWER_SCALE	SMALLINT	Yes	For local decimal data types, LOCAL_LOWER_SCALE is the minimum number of digits allowed to the right of the decimal point. If LOCAL_LOWER_SCALE is null, assume that users do not need to set this minimum.
LOCAL_UPPER_SCALE	SMALLINT	Yes	For local decimal data types, LOCAL_UPPER_SCALE is the maximum number of digits allowed to the right of the decimal point. Together, LOCAL_LOWER_SCALE and LOCAL_UPPER_SCALE indicate the full allowable scale.
LOCAL_S_OPR_P	CHAR(2)	Yes	Relationship between local scale and remote precision. Basic comparison operators (=, <, >, <=, >=, <>) can be used. A null indicates that no specific relationship is required.
LOCAL_BIT_DATA	CHAR(1)	Yes	Applies only to types for character strings. Indicates whether the local type is for bit data: Y Yes, this type is for bit data. N No, this type is not for bit data.

Table 37. Columns in SYSCAT.REVTYPEMAPPINGS Catalog View (continued)

Name	**Data Type**	**Nullable?**	**Content**
SERVER	VARCHAR(18)	Yes	Name of the data source that supports the remote type.
SERVER_TYPE	VARCHAR(30)	Yes	Type of the data source that supports the remote data type; for example, Oracle or Sybase.
SERVER_VERSION	VARCHAR(18)	Yes	Version of the type specified in SERVER_TYPE; for example, if SERVER_TYPE is Informix, SERVER_VERSION might be 7.1 or 7.2.
SERVER_PROTOCOL	VARCHAR(30)	Yes	The protocol used in accessing the remote type.
SERVER_TYPESCHEMA	VARCHAR(128)	Yes	Schema to which the remote type belongs.
REMOTE_TYPENAME	VARCHAR(128)	No	The remote data type. Together, SERVER_TYPESCHEMA and REMOTE_TYPENAME make up this type's fully-qualified name.
REMOTE_META_TYPE	CHAR(1)	Yes	Indicates whether the remote type is built-in or user-defined: S The type is built-in. T The type is user-defined.
REMOTE_LENGTH	INTEGER	Yes	If the remote type is decimal, REMOTE_LENGTH is the maximum number of digits that values of this type can have. If the remote type is a character type other than decimal, REMOTE_LENGTH is the maximum number of characters that values of this type can have. If REMOTE_LENGTH is null, assume that users do not need to determine the length.
REMOTE_SCALE	SMALLINT	Yes	For remote decimal types, REMOTE_SCALE is the maximum number of digits allowed to the right of the decimal point. If REMOTE_SCALE is null, assume that users do not need to determine the scale.
REMOTE_BIT_DATA	CHAR(1)	Yes	Applies only to types for character strings. Indicates whether the remote type is for bit data: Y Yes, this type is for bit data. N No, this type is not for bit data.
CREATE_TIME	TIMESTAMP	No	The time at which this mapping was created.
REMARKS	VARCHAR(254)	Yes	User-supplied comment.

SYSCAT.SERVERS

When you use the CREATE SERVER MAPPING SQL statement, one or more rows in the SYSCAT.SERVERS catalog view are added for each data source accessed by DataJoiner. You do not need entries in SERVERS for tables that are stored in the same DataJoiner instance that contains this view. Table 38 describes the columns in SYSCAT.SERVERS.

Table 38. Columns in SYSCAT.SERVERS Catalog View

Name	Data Type	Nullable?	Content
SERVER	VARCHAR(18)	No	Name of the data source as it is known to DataJoiner. This name is defined by the person who adds a row to SYSSERVERS and does not need to be the same name used outside of DataJoiner. The entry in this field identifies a server node and database. This field is the primary key for this table. The server name can consist of the single-byte uppercase letters (A-Z), the Arabic numerals (0-9), the underscore character (_), and the three special characters #, @, and $. The three special characters provide compatibility with host database products; however, they should be used with care in an NLS environment, because they are not included in the NLS host (EBCDIC) invariant character set.
NODE	VARCHAR(70)	No	Node at which the data source resides. For guidelines on finding out what values to assign to NODE, see “Finding Node Names” on page 51. This field is case-sensitive.
DBNAME	VARCHAR(18)	No	Name of the database on the data source. This field is required when a database manager supports multiple databases for each instance and has a catalog for each database or requires a connect for each database. This field is case-sensitive.

Table 38. Columns in SYSCAT.SERVERS Catalog View *(continued)*

Name	Data Type	Nullable?	Content
SERVER_TYPE	VARCHAR(30)	No	The type of data source. The valid values for this field are: CLASSIC CROSSACCESS DATAJOINER DB2/6000 DB2/CS DB2/HP DB2/MVS DB2/NT DB2/PE DB2/SUN DB2/VM DB2/VSE DB2/2 DB2/400 GENERIC INFORMIX MSSQLSERVER ORACLE RDB SQLANYWHERE SQL/DS SYBASE XACCESS These character constants define the set of data sources supported by DataJoiner. This field is uppercase.
SERVER_VERSION	VARCHAR(18)	No	The version of the data source specified in the TYPE field. The valid values for this field are defined by the set of data sources supported by DataJoiner. For example, the VERSION can contain values such as: 7.3.2 10.0

Table 38. Columns in SYSCAT.SERVERS Catalog View (continued)

Name	Data Type	Nullable?	Content
SERVER_PROTOCOL	VARCHAR(30)	No	Data access protocol used. The protocol must match the one used to configure the data source as described in the *Planning, Installation, and Configuration Guide* for your platform. The default protocols in DataJoiner are: drda: Used with DRDA application servers such as DB2 for OS/390 (accessed via SNA) drdaIP: Used with DRDA application servers such as DB2 for OS/390 (accessed via TCP/IP) db2ra: Used with DB2 for CS and DB2 for PE djxclassic: Used with Classic Connect ctlib: Used with Sybase System 10 and above dblib: Used with Sybase and Microsoft SQL Server djxmssql: Used with Microsoft SQL Server informix: Used with Informix informix7: Used with Informix 7 net8: Used with Oracle 8.0.3 or higher sqlnet: Used with Oracle 7 You can add other protocols to your system by following procedures described in the *Planning, Installation, and Configuration Guide* for your platform. This field is case-sensitive.
CODESET	VARCHAR(10)	No	Reserved for future use.
CPU_RATIO	FLOAT	Yes	CPU ratio. This ratio represents how much faster or slower the remote CPU is compared to the local CPU. For example, if the remote CPU is twice as fast as the local CPU, enter the value 0.5. If the local CPU is twice as fast as the remote CPU, enter the value 2. If no value is provided, 1.0 is used by the database manager.

Table 38. Columns in SYSCAT.SERVERS Catalog View (continued)

Name	Data Type	Nullable?	Content
IO_RATIO	FLOAT	Yes	I/O ratio. This ratio represents the local I/O device rate, compared to the remote I/O device. For example, if the remote I/O device is twice as fast as the local I/O device, enter the value 0.5. If the local I/O device is twice as fast as the remote I/O device, enter the value 2. When no value is provided, 1.0 is used by the database manager.
COMM_RATE	INTEGER	Yes	Communication rate between the local and remote systems. Enter the rate in the unit of bytes per second. If no value is provided, the database manager uses a default of 2MB per second.
REMARKS	VARCHAR(254)	Yes	User's comments.

SYSSTAT.SERVERS

This is a partly-updatable catalog view on SYSIBM.SYSSERVERS. It contains the statistics for a server. Table 39 describes its columns.

Table 39. Columns in SYSSTAT.SERVERS Catalog View

Name	Data Type	Nullable?	Updatable?	Content
SERVER	VARCHAR(18)	No	No	Name of the server.
COMM_RATE	INTEGER	Yes	Yes	Communication rate to the data source.
IO_RATIO	FLOAT	Yes	Yes	I/O ratio for the data source.
CPU_RATIO	FLOAT	Yes	Yes	CPU ratio for the data source.

SYSCAT.SERVER_DATATYPES

This catalog view shows forward data type mappings; that is, mappings from data source data types to data types defined locally to DataJoiner. A mapping might associate one user-defined type with another, or a user-defined type with a built-in type, or one built-in type with another. Most of the mappings between built-in types are defaults; for a list, see "Appendix A. Default Forward Type Mappings" on page 135. All other mappings were created with the CREATE TYPE MAPPING STATEMENT (page 112).

DataJoiner uses these mappings to determine what data types to define locally for columns of a data source table or view that was created with the data source's own SQL. For more information, see "Data Type Mappings" on page 26.

Table 40 on page 191 describes the columns in SYSCAT.SERVER_DATATYPES.

Table 40. Columns in SYSCAT.SERVER_DATATYPES Catalog View

Name	Data Type	Nullable?	Content
TYPE_MAPPING	VARCHAR(18)	No	Name of the mapping. (If a mapping is created with the CREATE TYPE MAPPING statement, and no name is specified in the statement, DataJoiner supplies the name. DataJoiner also supplies the name for all default mappings.)
TYPESCHEMA	CHAR(8)	Yes	Schema to which the local type belongs. If TYPESCHEMA is null, assume that this type is built-in.
TYPENAME	VARCHAR(18)	No	The local data type. TYPESCHEMA and TYPENAME make up the type's fully-qualified name. This is the name by which DataJoiner references the remote type. The name is stored in the DataJoiner catalog when a nickname for the table is created.
DEFINER	CHAR(8)	No	Authorization ID under which the mapping was created. If the value of DEFINER is sysibm, it means that the local type is built-in.
LENGTH	INTEGER	Yes	If the local type is decimal, LENGTH is the maximum number of digits that values of this type can have. If the local type is a character type other than decimal, LENGTH is the maximum number of characters that values of this type can have. If LENGTH is null, DataJoiner determines what this maximum should be.
SCALE	SMALLINT	Yes	For local decimal types, SCALE is the maximum number of digits allowed to the right of the decimal point. If SCALE is null, DataJoiner determines what this maximum should be.
BIT_DATA	CHAR(1)	Yes	Applies only to types for character strings. Indicates whether a local type is for bit data: Y Yes, this type is for bit data. N No, this type is not for bit data.
SERVER	VARCHAR(18)	Yes	Name of the data source that supports the remote type.
SERVER_TYPE	VARCHAR(30)	Yes	Type of the data source that supports the remote data type; for example, Oracle or Sybase.
SERVER_VERSION	VARCHAR(18)	Yes	Version of the type specified in SERVER_TYPE; for example, if SERVER_TYPE is Informix, SERVER_VERSION might be 7.1 or 7.2.
SERVER_PROTOCOL	VARCHAR(30)	Yes	The protocol used in accessing the remote type.
REMOTE_TYPESCHEMA	VARCHAR(128)	Yes	Schema to which the remote type belongs.

Table 40. Columns in SYSCAT.SERVER_DATATYPES Catalog View (continued)

Name	Data Type	Nullable?	Content
REMOTE_TYPENAME	VARCHAR(128)	No	The remote data type. Together, REMOTE_TYPESCHEMA and REMOTE_TYPENAME make up this type's fully-qualified name.
REMOTE_META_TYPE	CHAR(1)	Yes	Indicates whether the remote type is built-in or user-defined: S The type is built-in. T The type is user-defined.
REMOTE_LOWER_LEN	INTEGER	Yes	If the remote type is decimal, REMOTE_LOWER_LEN is the minimum number of digits that values of this type can have. If the type is a character type other than decimal, REMOTE_LOWER_LEN is the minimum number of characters that values of the type can have.
REMOTE_UPPER_LEN	INTEGER	Yes	Maximum precision. If the remote type is decimal, REMOTE_UPPER_LEN is the maximum number of digits that values of this type can have. If the type is a character type other than decimal, REMOTE_UPPER_LEN is the maximum number of characters that values of the type can have. Together, REMOTE_LOWER_LEN and REMOTE_UPPER_LEN indicate the full allowable precision for a value.
REMOTE_LOWER_SCALE	SMALLINT	Yes	For remote decimal data types, REMOTE_LOWER_SCALE is the minimum number of digits allowed to the right of the decimal point. If REMOTE_LOWER_SCALE is null, assume that users do not need to set this minimum.
REMOTE_UPPER_SCALE	SMALLINT	Yes	For remote decimal data types, REMOTE_UPPER_SCALE is the maximum number of digits allowed to the right of the decimal point Together, REMOTE_LOWER_SCALE and REMOTE_UPPER_SCALE indicate the full allowable scale.
REMOTE_S_OPR_P	CHAR(2)	Yes	Relationship between remote scale and remote precision. Basic comparison operators (=, <, >, <=, >=, <>) can be used. A null indicates that no specific relationship is required.

Table 40. Columns in SYSCAT.SERVER_DATATYPES Catalog View (continued)

Name	Data Type	Nullable?	Content
REMOTE_BIT_DATA	CHAR(1)	Yes	Applies only to types for character strings. Indicates whether the remote type is for bit data: Y Yes, this type is for bit data. N No, this type is not for bit data.
CREATE_TIME	TIMESTAMP	No	The time at which this mapping was created.
REMARKS	VARCHAR(254)	Yes	User's comments.

SYSCAT.SERVER_FUNCTIONS

This catalog view shows mappings between functions defined to data sources and functions or function templates defined locally to DataJoiner. Mappings can associate:

- Data source built-in functions with local built-in functions
- Data source built-in or user-defined functions with local user-defined functions or local user-defined function templates

A unique index is defined on the columns FUNCSCHEMA, FUNCNAME, SPECIFIC NAME, SERVER, SERVER_TYPE, SERVER_VERSION, SERVER_PROTOCOL, and CREATE_TIME.

Table 41 describes the columns in SYSCAT.SERVER_FUNCTIONS.

Table 41. Columns in SYSCAT.SERVER_FUNCTIONS Catalog View

Name	Data Type	Nullable?	Content
FUNCTION_MAPPING	VARCHAR(18)	No	Name of the function mapping (can be system-generated).
FUNCSCHEMA	CHAR(8)	Yes	Schema to which the function belongs. If FUNCSCHEMA is null, assume that this function is built-in.
FUNCNAME	VARCHAR(18)	No	Name of a local function or function template. FUNCSCHEMA and FUNCNAME make up the fully-qualified name of a local function or function template.
SPECIFICNAME	VARCHAR(18)	Yes	Name of the local function instance.
DEFINER	CHAR(8)	No	Authorization ID under which this mapping was created. A value of SYSIBM indicates that this is a system built-in function mapping.
SERVER	VARCHAR(18)	Yes	Name of the data source to which the remote function in this mapping is defined.
SERVER_TYPE	VARCHAR(30)	Yes	Type of data source to which the remote function in this mapping is defined; for example, Oracle or SQL Anywhere.

Table 41. Columns in SYSCAT.SERVER_FUNCTIONS Catalog View (continued)

Name	Data Type	Nullable?	Content
SERVER_VERSION	VARCHAR(18)	Yes	Version of the data source type specified in the SERVER_TYPE column. For example, if the type is Oracle, the Version might be 8.0.3.
SERVER_PROTOCOL	VARCHAR(30)	Yes	Protocol used in accessing the data source type.
REMOTE_FUNCNAME	VARCHAR(1024)	No	Fully-qualified name of the remote function in this mapping.
USER_DEFINED	CHAR(1)	Yes	Reserved for future use.
IOS_PER_INVOC	DOUBLE	No	Estimated number of I/Os per invocation of the remote function; -1 if not known (0 is the default).
INSTS_PER_INVOC	DOUBLE	No	Estimated number of instructions per invocation of the remote function; -1 if not known (450 is the default).
IOS_PER_ARGBYTE	DOUBLE	No	Estimated number of I/Os per input argument byte of the remote function; -1 if not known (0 is the default).
INSTS_PER_ARGBYTE	DOUBLE	No	Estimated number of instructions per input argument byte of the remote function; -1 if not known (0 is the default).
PERCENT_ARGBYTES	SMALLINT	No	Estimated average percent of input argument bytes that the remote function will actually read; -1 if not known (100 is the default).
INITIAL_IOS	DOUBLE	No	Estimated number of I/Os performed the first and last time the function is invoked; -1 if not known (0 is the default).
INITIAL_INSTS	DOUBLE	No	Estimated number of instructions executed the first and last time the function is invoked; -1 if not known (0 is the default).
CREATE_TIME	TIMESTAMP	No	The time at which this mapping is created.
REMARKS	VARCHAR(254)	Yes	User's comments.

SYSSTAT.SERVER_FUNCTIONS

This is a partly-updatable catalog view. The DataJoiner optimizer uses statistics in this view in developing access plans. Table 42 describes this view's columns.

Table 42. Columns in SYSSTAT.SERVER_FUNCTIONS Catalog View

Name	Data Type	Nullable?	Updatable?	Content
FUNCTION_MAPPING	VARCHAR(18)	No	No	Name of the function mapping (may be system-generated).
SERVER	VARCHAR(18)	Yes	No	Name of the data source to which the remote function in the mapping is defined.

Table 42. Columns in SYSSTAT.SERVER_FUNCTIONS Catalog View (continued)

Name	Data Type	Nullable?	Updatable?	Content
SERVER_TYPE	VARCHAR(30)	Yes	No	Type of data source to which the remote function in the mapping is defined; for example, Oracle or SQL Anywhere.
SERVER_VERSION	VARCHAR(18)	Yes	No	Version of the data source type specified in the SERVER_TYPE column. For example, if the type is Oracle, the version might be 8.0.3.
SERVER_PROTOCOL	VARCHAR(30)	Yes	No	Protocol used in accessing the data source type.
REMOTE_FUNCNAME	VARCHAR(1024)	No	No	Fully-qualified name of the remote function in this mapping.
IOS_PER_INVOC	DOUBLE	No	Yes	Estimated number of I/Os per invocation of the remote function; -1 if not known (0 is the default).
INSTS_PER_INVOC	DOUBLE	No	Yes	Estimated number of instructions per invocation of the remote function; -1 if not known (450 is the default).
IOS_PER_ARGBYTE	DOUBLE	No	Yes	Estimated number of I/Os per input argument byte of the remote function; -1 if not known (0 is the default).
INSTS_PER_ARGBYTE	DOUBLE	No	Yes	Estimated number of instructions per input argument byte of the remote function; -1 if not known (0 is the default).
PERCENT_ARGBYTES	SMALLINT	No	Yes	Estimated average percent of input argument bytes that the remote function will actually read; -1 if not known (100 is the default).
INITIAL_IOS	DOUBLE	No	Yes	Estimated number of I/Os performed the first/last time the function is invoked; -1 if not known (0 is the default).
INITIAL_INSTS	DOUBLE	No	Yes	Estimated number of instructions executed the first/last time the function is invoked; -1 if not known (0 is the default).

SYSCAT.SERVER_OPTIONS

This catalog view shows options that DBAs can set to help DataJoiner optimize services for data sources.

A unique index is defined on the columns OPTION, SERVER, SERVER_ TYPE, SERVER_VERSION, and SERVER_PROTOCOL. A constraint on the base table requires values in the SERVER column to match values in the SERVER column of the SYSCAT.SERVERS catalog view.

Table 43 describes the columns in SYSCAT.SERVER_OPTIONS.

Table 43. Columns in SYSCAT.SERVER_OPTIONS Catalog View

Name	Data Type	Nullable?	Content
OPTION	VARCHAR(30)	No	Name of the server option. The valid values of this field are listed in Table 2 on page 17.
SETTING	VARCHAR(254)	No	Option setting. The valid values depend on OPTION, and are listed in Table 2 on page 17.
SERVER	VARCHAR(18)	Yes	Name of data source to which an option setting applies. This field is uppercase.
SERVER_TYPE	VARCHAR(30)	Yes	Type of data source to which an option setting applies. The valid values for this column are the same as those in SYSCAT.SERVERS.TYPE.
SERVER_VERSION	VARCHAR(18)	Yes	Version of the type of data source to which an option setting applies. The valid values for this column are the same as those in SYSCAT.SERVERS.VERSION.
SERVER_PROTOCOL	VARCHAR(30)	Yes	Protocol used to access the data source named in the SERVER column or the type of data source specified in the SERVER_TYPE column. The valid values for this field are the same as those in SYSCAT.SERVERS.PROTOCOL.
REMARKS	VARCHAR(254)	Yes	User's comments.

SYSCAT.TABLES

DataJoiner inserts one row in the SYSCAT.TABLES catalog view for each nickname, table, or view that is created. All catalog views (including SYSCAT.TABLES) have entries in the TABLES catalog view.

Table 44 describes the columns in SYSCAT.TABLES.

Table 44. Columns in SYSCAT.TABLES Catalog View

Name	Data Type	Nullable?	Content
TABSCHEMA	CHAR(8)	No	Nickname, table, or view qualifier.
TABNAME	VARCHAR(18)	No	Nickname, table, or view name. If the REMOTE_TABNAME or SERVER column is null, the value in the TABNAME column represents a local table or view. Otherwise, TABNAME represents a nickname for a table or view at the data source.
DEFINER	CHAR(8)	No	User who created the table or view.

Table 44. Columns in SYSCAT.TABLES Catalog View (continued)

Name	Data Type	Nullable?	Content
TYPE	CHAR(1)	No	The type of object: A Alias; T Table, nickname for a table, or nickname for a view; V View
STATUS	CHAR(1)	No	The type of object: N Normal table, view, nickname for a table or view, or alias; C Check pending on table or nickname for a table; X Inoperative view or inoperative nickname for a view
BASE_TABSCHEMA	CHAR(8)	Yes	If TYPE=A, identifies the table, nickname, view, or alias schema that is referenced by this alias; otherwise, it is null.
BASE_TABNAME	VARCHAR(18)	Yes	If TYPE=A, identifies the table, nickname, view, or alias name that is referenced by this alias; otherwise, it is null.
CREATE_TIME	TIMESTAMP	No	Timestamp indicating when the nickname, table, or view was first created.
STATS_TIME	TIMESTAMP	Yes	Timestamp indicating when any change was made to recorded statistics for this nickname or table.
COLCOUNT	SMALLINT	No	Number of columns in the nickname or table.
TABLEID	SMALLINT	No	Internal table identifier.
TBSPACEID	SMALLINT	No	Internal identifier of primary table space for this table.
CARD	INTEGER	No	Total number of rows in the nickname or table; −1 if statistics are not gathered or if the row describes a view or alias.
NPAGES	INTEGER	No	Total number of pages on which the rows of the nickname or table exist; −1 if statistics are not gathered or if the row describes a view or alias.
FPAGES	INTEGER	No	Total number of pages in the file; −1 if statistics are not gathered or if the row describes a view or alias.
OVERFLOW	INTEGER	No	Total number of overflow records in the table; −1 if statistics are not gathered or if the row describes a view or alias.

Table 44. Columns in SYSCAT.TABLES Catalog View (continued)

Name	Data Type	Nullable?	Content
TBSPACE	VARCHAR(18)	Yes	Name of primary table space for the table. If no other table space is specified, all parts of the table are stored in this table space. Null for aliases and views.
INDEX_TABLESPACE	VARCHAR(18)	Yes	Specifies the name of the table space that holds all indexes created on this table.
LONG_TABLESPACE	VARCHAR(18)	Yes	Specifies the name of the table space that holds all long data (LONG or LOB column types) for this table.
PARENTS	SMALLINT	Yes	Number of parent tables of this table (the number of referential constraints in which this table is a dependent).
CHILDREN	SMALLINT	Yes	Number of dependent tables of this table (the number of referential constraints in which this table is a parent).
SELFREFS	SMALLINT	Yes	Number of self-referencing referential constraints for this table (the number of referential constraints in which this table is both a parent and a dependent).
KEYCOLUMNS	SMALLINT	No	Number of columns in the primary key of the table.
KEYINDEXID	SMALLINT	Yes	Index ID of the primary index. This field is null or 0 if no primary key exists.
KEYUNIQUE	SMALLINT	No	Reserved for future use.
CHECKCOUNT	SMALLINT	No	Number of check constraints defined on this table.
DATACAPTURE	CHAR(1)	No	Valid values are: Y Table participates in data propagation (DPROPR). N Table does not participate in data propagation.
CONST_CHECKED	CHAR(32)	No	Byte 1 represents foreign key constraints. Byte 2 represents check constraints. Other bytes are reserved. Encodes constraint information on checking. Values are: Y Checked by system U Checked by user N Not checked (pending)
REMOTE_SERVER	VARCHAR(18)	Yes	Name of the data source. This column is used only with nicknames. If the TABNAME column represents a table or view rather than a nickname, this column is null. This field is case-sensitive.

Table 44. Columns in SYSCAT.TABLES Catalog View (continued)

Name	Data Type	Nullable?	Content
REMOTE_TABSCHEMA	VARCHAR(128)	Yes	Authorization ID on the data source. This column is used only with nicknames. If the TABNAME column represents a table, view, or two-part nickname, this column is null. This field is case-sensitive.
REMOTE_TABNAME	VARCHAR(128)	Yes	Name of the remote table as defined on the data source. This column is used only with nicknames. If the TABNAME column represents a table or view rather than a nickname, this column is null. This field is case-sensitive.
REMARKS	VARCHAR(254)	Yes	User's comments.

Appendix F. 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:

- 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

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” on page 204.

Appendix G. 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 45 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 45 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 45).

To understand how the DataJoiner books in Table 45 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 45 document the function and syntax that DataJoiner has *in addition to* the function and syntax that it shares with DB2 for CS.

Table 45 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 45. 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 45. 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 45. 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 45. 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 45. 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 45 on page 207.

To view online documentation, follow the instructions located in the README files on the CD-ROM. Most of the books in Table 45 on page 207 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 H. 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

Trademarks

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

ADSTAR	IIN
Advanced Peer-to-Peer Networking	IMS
AIX	IMS/ESA
APPN	Language Environment
AS/400	MVS
AT	MVS/ESA
CICS	MVS/XA
CICS/6000	NetView
Client Acces	Operating System/2
Current	Operating System/400
DATABASE 2	OS/2
DataGuide	OS/390
DataJoiner	OS/400
DataPropagator	RACF
DataRefresher	RETAIN
DB2	RISC System/6000
DFSMS	RS/6000
Distributed Relational Database Architecture	RT
DProp	SP
DRDA	SQL/DS
Extended Services for OS/2	SQL/400
HACMP/6000	System/390
IBM	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

Readers' Comments — We'd Like to Hear from You

DB2 DataJoiner®
Application Programming and SQL Reference Supplement
Version 2 Release 1 Modification 1

Publication No. SC26-9148-01

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
SC26-9148-01

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
Department BWE/H3
PO Box 49023
San Jose, CA 95161-9945

Fold and Tape **Please do not staple** Fold and Tape

SC26-9148-01

Cut or Fold
Along Line

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-9148-01

Spine information:

DB2 DataJoiner®

Application Programming and SQL Reference Supplement

Version 2
Release 1
Modification 1

SC26-9148-01