



IBM Systems - iSeries

Database

Distributed database programming

Version 5 Release 4





IBM Systems - iSeries

Database

Distributed database programming

Version 5 Release 4

Note

Before using this information and the product it supports, read the information in "Notices," on page 251.

Seventh Edition (February 2006)

This edition applies to version 5, release 4, modification 0 of IBM i5/OS (product number 5722-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

© Copyright International Business Machines Corporation 1998, 2006. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Distributed database programming	1
What's new for V5R4	1
Printable PDF	1
Introduction to distributed database programming.	2
Distributed relational database processing	2
Remote unit of work.	5
Distributed unit of work	6
Other distributed relational database terms and concepts	7
Distributed relational database architecture support	8
DRDA and CDRA support.	9
Application requester driver programs	10
Distributed relational database on i5/OS	11
Manage an iSeries distributed relational database	12
Example: Spiffy Corporation distributed relational database	13
Spiffy organization and system profile	13
Business processes of the Spiffy Corporation Automobile Service.	15
Distributed relational database administration for the Spiffy Corporation	15
Planning and design	16
Identify your needs and expectations for a distributed relational database	17
Data needs for distributed relational databases	17
Distributed relational database capabilities	17
Goals and directions for a distributed relational database	17
Design the application, network, and data for a distributed relational database	18
Tips: Design distributed relational database applications	19
Network considerations for a distributed relational database	19
Data considerations for a distributed relational database	20
Develop a management strategy for a distributed relational database	21
General operations for a distributed relational database	21
Security considerations for a distributed relational database	22
Account for a distributed relational database	23
Problem analysis for a distributed relational database	23
Backup and recovery for a distributed relational database	24
Initial setup	24
Work management on the iSeries server	25
Set up your work management environment for DRDA	25
APPC subsystems	26
TCP/IP subsystems.	28
User databases on independent auxiliary storage pools	28

Use the relational database directory	29
Work with the relational database directory.	30
Add an entry for SNA usage	30
Add an entry for TCP/IP usage	32
Specify a relational database alias name	32
Add an entry for an application requester driver	33
Use the WRKRDBDIRE command	33
The *LOCAL directory entry.	34
Directory entries for user databases on independent auxiliary storage pools	35
Example: Set up a relational database directory	35
Set up DRDA security	38
Set up the TCP/IP server for DRDA	39
Set up SQL packages for Interactive SQL	39
Set up DDM files	40
Load data into tables in a distributed relational database	41
Load new data into the tables of a distributed relational database	41
Load data into a table using SQL	41
Manipulate data in tables and files using the iSeries query management function	41
Enter data, update tables, and make inquiries using data file utility	42
Move data from one iSeries server to another	42
Create a user-written application program	43
Query a database using Interactive SQL	43
Query remote servers using DB2 Universal Database for iSeries query management function	44
Copy files to and from tape	45
Move data between iSeries servers using copy file commands	45
Transfer data over network using Network File commands	47
Move a table using server save and restore commands.	47
Move a database to an iSeries server from a non-iSeries server	48
Move data from another IBM server	48
Move data from a non-IBM server.	49
Security	50
Elements of distributed relational database security.	50
Elements of security in an APPC network	52
APPN configuration lists	53
Conversation level security	53
DRDA application server security in an APPC network	54
Elements of security in a TCP/IP network	57
Application requester security in a TCP/IP network	57
Application server security in a TCP/IP network	59

Connection security protocols for DDM and DRDA	60	Delete an SQL package using the Delete SQL Package (DLTSQLPKG) command	97
Secure Sockets Layer for DDM and DRDA	61	Modify package authorizations	98
Internet Protocol Security Architecture for DDM and DRDA	61	Use the SQL DROP PACKAGE statement	98
Considerations for certain passwords being sent as clear text.	62	Administration	99
Ports and port restrictions for DDM/DRDA	62	Monitor relational database activity	99
DRDA server access control exit programs	63	Work with jobs in a distributed relational database	99
Example: DRDA server access control exit program	65	Work with user jobs in a distributed relational database.	100
Object-related security for DRDA	67	Work with active jobs in a distributed relational database.	101
Authority to distributed relational database objects	68	Work with commitment definitions in a distributed relational database.	102
Programs that run under adopted authority for a distributed relational database	69	Track request information with the job log of a distributed relational database	103
Protection strategies in a distributed relational database	69	Locate distributed relational database jobs	104
Application development.	70	Operate remote iSeries servers.	106
Programming considerations for a distributed relational database application	71	Control DDM conversations	107
Naming of distributed relational database objects	72	Reclaim DDM resources.	109
System (*SYS) naming convention	72	Display objects used by programs	110
SQL (*SQL) naming convention.	72	Example: Display program reference.	111
Default collection name	72	Drop a collection from a distributed relational database	112
Connect to a distributed relational database	73	Job accounting in a distributed relational database	113
Remote unit of work	73	Manage the TCP/IP server	114
Distributed unit of work	75	DRDA TCP/IP server terminology	115
Implicit connection management for the default activation group	78	TCP/IP communication support concepts for DDM	115
Implicit connection management for nondefault activation groups	79	Establish a DRDA or DDM connection over TCP/IP.	115
Explicit connection management	80	DRDA and DDM listener program	116
SQL specific to distributed relational database and SQL CALL	82	Start TCP/IP Server (STRTCPSVR) CL command.	117
Distributed relational database statements	83	End TCP/IP Server (ENDTCPSVR) CL command.	117
SQL CALL statement (stored procedures)	83	Start DDM listener in iSeries Navigator	118
DB2 Universal Database for iSeries coexistence	84	DRDA/DDM server jobs	118
End DRDA units of work.	85	Subsystem descriptions and prestart job entries with DDM	118
Stored procedures, user-defined functions, and commitment control	85	DRDA and DDM prestart jobs.	118
Coded character set identifier	86	Configure the DDM server job subsystem	121
iSeries server support	87	Identify server jobs	123
Other DRDA data conversion	89	iSeries job names	123
DDM files and SQL	89	Display server jobs	123
Prepare distributed relational database programs	90	Display the history log	124
Precompile programs with SQL statements	91	Audit the relational database directory	125
Listing	91	Data availability and protection	126
Temporary source file member	92	Recovery support for a distributed relational database	126
SQL package creation	92	Data recovery after disk failures for distributed relational databases	127
Precompiler commands	92	Auxiliary storage pools	127
Compile an application program	93	Checksum protection in a distributed relational database.	128
Bind an application.	93	Mirrored protection for a distributed relational database.	128
Testing and debugging	94	Journal management for distributed relational databases	128
Program references	96	Index recovery	129
Work with SQL packages	97		
Use the Create SQL Package (CRTSQLPKG) command	97		
Manage an SQL package	97		

Design tables to reduce index rebuilding time	130	Connection failures specific to Interactive SQL	162
System-managed access-path protection	130	Not enough prestart jobs at server	163
Transaction recovery through commitment control	131	Application problems.	163
Save and restore processing for a distributed relational database.	135	Listings	163
Save and restore indexes in the distributed relational database environment	136	Precompiler listing	163
Save and restore security information in the distributed relational database environment.	136	CRTSQLPKG listing	165
Save and restore SQL packages in the distributed relational database environment.	136	SQLCODEs and SQLSTATEs	166
Save and restore relational database directories	137	Distributed relational database	
Network redundancy considerations for a distributed relational database.	139	SQLCODEs and SQLSTATEs	167
Data redundancy in your distributed relational database network	141	System and communications problems	170
Performance.	143	Get data to report a failure	172
Improve distributed relational database performance through the network	143	Print a job log	172
Improve distributed relational database performance through the server	144	Find job logs from TCP/IP server prestart jobs	172
Improve distributed relational database performance through the database	145	Print the product activity log	173
Decide DRDA data location	145	Job tracing	174
Factors that affect blocking for DRDA	145	Trace job	174
DB2 Universal Database for iSeries to DB2 Universal Database for iSeries blocking.	145	Start trace	174
DB2 Universal Database for iSeries to non-DB2 Universal Database for iSeries blocking	146	Communications trace	175
Non-DB2 Universal Database for iSeries to DB2 Universal Database for iSeries blocking	146	Standard communications trace	175
Summarization of DRDA blocking rules	147	TCP/IP communications trace.	177
Factors that affect the size of DRDA query blocks	148	TCP/IP communication trace formatting	178
Troubleshooting	148	Find first-failure data capture data	180
iSeries problem handling overview	149	Start a service job to diagnose application server problems	181
Isolate distributed relational database problems	149	Service jobs for APPC servers	181
DRDA incorrect output problems.	150	Create your own TPN and set QCNTRVC	181
Application does not complete in the expected time	151	Set QCNTRVC as a TPN on a DB2	
Work with distributed relational database users	154	Universal Database for iSeries application requester	182
Copy screen	154	Create your own TPN for debugging a DB2 Universal Database for iSeries application server job.	182
Messages	155	Set QCNTRVC as a TPN on a DB2	
Message types	157	Universal Database for VM application requester	182
Distributed relational database messages	158	Set QCNTRVC as a TPN on a DB2	
Handle program start request failures for APPC	160	Universal Database for z/OS application requester	182
Handle connection request failures for TCP/IP	161	Set QCNTRVC as a TPN on a DB2	
Server is not started or the port ID is not valid	161	Universal Database for Linux, UNIX and Windows application requester	183
DRDA connect authorization failure	161	Service jobs for TCP/IP servers	183
Server not available	162	QRWOPTIONS data area	184
		Example: CL command to create the data area	185
		Examples: Application programming	186
		Example: Program definitions	187
		Example: RPG program	191
		Example: COBOL program	200
		Example: C program using embedded SQL	208
		Example: Java program	213
		Example: Program output	218
		User FAQs	219
		Connect to a distributed relational database	219
		iSeries system value QCCSID	219
		CCSID conversion considerations for DB2	
		Universal Database for z/OS and DB2 Universal Database server for VM database managers	221

Why am I getting a SQL5048N message when I attempt to connect from DB2 Universal Database for Linux, UNIX and Windows?	221	RWQY P0—Result Set Pseudo-Open.	232
Do iSeries files have to be journaled?	221	RWQY AR—Array Result Set Processed	232
When will query data be blocked for better performance?	222	RWQY DA—Array Result Set SQLDA	232
How do you interpret an SQLCODE and the associated tokens reported in a DBM SQL0969N error message?	223	RWQY DO—Debug Options	232
How can the host variable type in WHERE clauses affect performance?	223	RWQY L1 and RWEX L1—Saved in Outbound LOB Table Trace Point.	232
Can I use a library list for resolving unqualified table and view names?	224	RWQY L2 and RWEX L2—Built in Datastream from LOB Table Trace Point	233
Can a user of DB2 Universal Database for Linux, UNIX and Windows specify that the NLSS sort sequence table of the DRDA job on the iSeries server be used instead of the usual EBCDIC sequence?	224	RWQY L0 and RWEX L0—Saved in Inbound LOB Table Trace Point	233
Why are no rows returned when I perform a query?	225	RWAC RQ—Access RDB Request Trace Point	233
What level of DB2 Universal Database for Linux, UNIX, Windows is required to interact with DB2 Universal Database for iSeries?	225	RWAC cb—Access RDB Control Block Trace Points	233
How can I get scrollable cursor support enabled from DB2 Universal Database for Linux, Unix and Windows Version 8 to the iSeries?	225	RWSY FN: SYNCxxx [TYPE:x] -- Source TCP SYNC/RESYNC Trace Point.	234
Other tips for interoperating in unlike environments	226	RWSY xx: yyyyyyy... -- Target TCP SYNC/RESYNC Trace Point	234
Interpret trace job and FFDC data	228	RW_ff_m—Application Requester Driver Control Block Trace Point	234
Interpret data entries for the RW component of trace job	229	First-failure data capture	235
Example: Analyze the RW trace data	230	A first-failure data capture dump.	235
Description of RW trace points	231	FFDC dump output description	238
RWff RC—Receive Data Stream Trace Point	231	DDM error codes	242
RWff SN—Send Data Stream Trace Point	231	Command check codes	242
RWQY S1—Partial Send Data Stream Trace Point 1	232	Conversational protocol error code descriptions	243
RWQY S2—Partial Send Data Stream Trace Point 2	232	DDM syntax error code descriptions.	244
RWQY BP—Successful Fetch Trace Point	232	Related information for distributed database programming	246
RWQY NB—Unsuccessful Fetch Trace Point	232	iSeries server information	246
		Distributed relational database library	247
		Other IBM distributed relational database platform libraries	248
		Architecture books	249
		IBM Redbooks	250
		Code license and disclaimer information	250
		Appendix. Notices	251
		Programming Interface Information	253
		Trademarks	253
		Terms and conditions.	254

Distributed database programming

Distributed database programming describes the distributed relational database management portion of the i5/OS™ licensed program. Distributed relational database management provides applications with access to data that is external to the applications and typically located across a network of computers.

This information is intended primarily for application programmers responsible for the development, administration, and support of a distributed relational database on one or more iSeries™ servers.

Application programmers who are not familiar with the iSeries database can also get a view of the total range of database support provided by the i5/Operating System (i5/OS). Application programmers might use this information to see the server context in which distributed relational database applications run.

Before using this information, you should be familiar with general programming concepts and terminology, and have a general understanding of the iSeries server and the i5/OS operating system.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 250.



What’s new for V5R4

This topic highlights the changes made to this topic collection for V5R4.

- | In V5R4, Distributed Relational Database Architecture™ (DRDA®) support has been extended to include the following functions. Prior to V5R4, this support was disabled over DRDA:
 - | • Distributed transaction processing (XA/JTA). (See Distributed transactions for more information.)
 - | • Server support for profile tokens. (See Security-related APIs for more information.)
 - | • DB2® Multisystem over TCP/IP. (See DB2 Multisystem for more information.)
- | The following functions are new to iSeries in V5R4. They are also supported over DRDA:
 - | • Increased maximum limit of SQL statements to 2 MB.
 - | • SQL descriptor area. (See SQL descriptor areas and SQLDA (SQL descriptor area) for more information.)
- | In V5R4, three example programs are updated or added in this topic collection:
 - | • “Example: C program using embedded SQL” on page 208
 - | • “Example: Java program” on page 213
 - | • “Example: Program definitions” on page 187

How to see what’s new or changed

To help you see where technical changes have been made, this information uses:

- The  image to mark where new or changed information begins.
- The  image to mark where new or changed information ends.

To find other information about what’s new or changed this release, see the Memo to users.

Printable PDF

Use this to view and print a PDF of this information.


To view or download the PDF version of this document, select Distributed database programming (about 2964 KB).

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF in your browser (right-click the link above).
2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

Downloading Adobe Reader

- | You need Adobe Reader installed on your system to view or print these PDFs. You can download a free copy from the Adobe Web site (www.adobe.com/products/acrobat/readstep.html) .

Introduction to distributed database programming

Distributed relational database support on the iSeries server consists of an implementation of IBM[®] Distributed Relational Database Architecture (DRDA) and integration of other SQL clients by use of Application Requester Driver (ARD) programs.

i5/OS and the DB2 Universal Database[™] for iSeries Query Manager and SQL Development Kit combine to provide this support.

This topic describes distributed relational database and how it is used on the iSeries server.

In addition, an example distributed relational database called Spiffy Corporation is described. This fictional company uses the iSeries server in a distributed relational database application program. This sample of the Spiffy Corporation forms the background for all examples used in this topic.

Distributed relational database processing

A relational database is a set of data stored in one or more tables in a computer.

A table is a two-dimensional arrangement of data consisting of horizontal rows and vertical columns as shown in the following table. Each row contains a sequence of values, one for each column of the table. A column has a name and contains a particular data type (for example, character, decimal, or integer).

Table 1. A typical relational table

Item	Name	Supplier	Quantity
78476	Baseball	ACME	650
78477	Football	Imperial	228
78478	Basketball	ACME	105
78479	Soccer ball	ACME	307

Tables can be defined and accessed in several ways on the server. One way to describe and access tables on the server is to use a language like Structured Query Language (SQL). SQL is the standard IBM database language and provides the necessary consistency to enable distributed data processing across different servers.

Another way to describe and access tables on the server is to describe physical and logical files using data description specifications (DDS) and access tables using file interfaces (for example, read and write high-level language statements).

SQL uses different terminology from that used on the iSeries server. For most SQL objects there is a corresponding server object on the iSeries server. The following table shows the relationship between SQL relational database terms and iSeries server terms.

Table 2. Relationship of SQL terms to system terms

SQL term	System term
<p>relational database. A database that can be perceived as a set of tables and can be manipulated in accordance with the relational model of data. There are three types of relational databases a user can access from an iSeries server, as listed under the system term column. For more information, see the Relational database topic.</p>	<p>system relational database, or system database. All the database objects that exist on disk attached to the iSeries server that are not stored on independent auxiliary storage pools.</p> <p>user relational database, or user database. All the database objects that exist in a single independent auxiliary storage pool group along with those database objects that are not stored on independent auxiliary storage pools.</p> <p>Notes:</p> <ul style="list-style-type: none"> • As of V5R2, an iSeries server can be host to multiple relational databases if independent auxiliary storage pools are configured on the server. There will always be one system relational database, and there can be one or more user relational databases. Each user database includes all the objects in the system database. • The user should be aware, however, that from a commitment control point of view, the system database is treated as a separate database, even when from an SQL point of view, it is viewed as being included within a user database. For more information, see the Transactions and commitment control topic.
<p>schema. Consists of a library, a journal, a journal receiver, an SQL catalog, and an optional data dictionary. A schema groups related objects and allows you to find the objects by name.</p> <p>Note: A schema is also commonly referred to as a collection.</p>	<p>remote relational database, or remote database. A database that resides on an iSeries or another server that can be accessed remotely.</p>
<p>table. A set of columns and rows.</p> <p>row. The horizontal part of a table containing a serial set of columns.</p> <p>column. The vertical part of a table of one data type.</p> <p>view. A subset of columns and rows of one or more tables.</p> <p>index. A collection of data in the columns of a table, logically arranged in ascending or descending order.</p> <p>package. An object that contains control structures for SQL statements to be used by an application server.</p>	<p>library. Groups related objects and allows you to find the objects by name.</p> <p>physical file. A set of records.</p> <p>record. A set of fields.</p> <p>field. One or more bytes of related information of one data type.</p> <p>logical file. A subset of fields, records or both of up to 32 physical files. A type of logical file.</p> <p>SQL package. Has the same meaning as the SQL term.</p>

Table 2. Relationship of SQL terms to system terms (continued)

SQL term	System term
catalog. A set of tables and views that contain information about tables, packages, views, indexes, and constraints. The catalog views in QSYS2 contain information about all tables, packages, views, indexes, and constraints on the iSeries server. Additionally, an SQL schema will contain a set of these views that only contains information about tables, packages, views, indexes, and constraints only in the schema.	No similar object. However, the Display File Description (DSPFD) command and the Display File Field Description (DSPFFD) command provide some of the same information that querying an SQL catalog provides.

A distributed relational database exists when the application programs that use the data and the data itself are located on different machines, or when the programs use data that is located on multiple databases on the same server. In the latter case, the database is distributed in the sense that DRDA protocols are used to access one or more of the databases within the single server. The connection to a database in such an environment will be one of two types: local or DRDA. There will be, at most, only one local database connection at one time. One simple form of a distributed relational database is shown in the following figure where the application program runs on one machine, and the data is located on a remote server.

When using a distributed relational database, the system on which the application program is run is called the application requester (AR), and the system on which the remote data resides is called the application server (AS). The term client is often used interchangeably with AR, and server with AS.

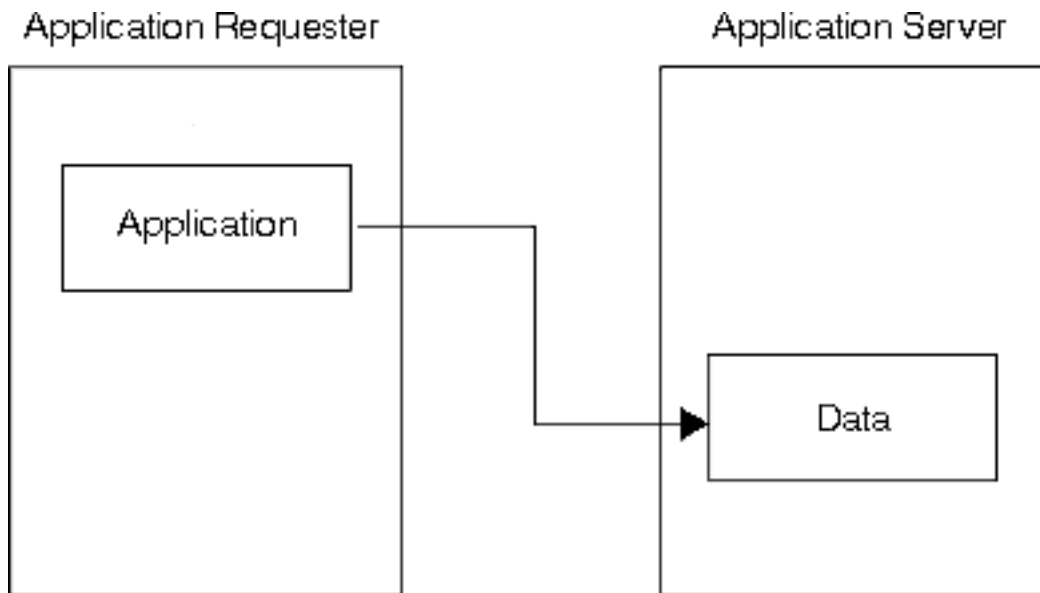


Figure 1. A distributed relational database

A unit of work is one or more database requests and the associated processing that make up a completed piece of work as shown in the following figure. A simple example is taking a part from stock in an inventory control application program. An inventory program can tentatively remove an item from a shop inventory account table and then add that item to a parts reorder table at the same location. The term transaction is another expression used to describe the unit of work concept.

In the preceding example, the unit of work is not complete until the part is both removed from the shop inventory account table and added to a reorder table. When the requests are complete, the application program can commit the unit of work. This means that any database changes associated with the unit of work are made permanent.

With unit of work support, the application program can also roll back changes to a unit of work. If a unit of work is rolled back, the changes made since the last commit or rollback operation are not applied. Thus, the application program treats the set of requests to a database as a unit.

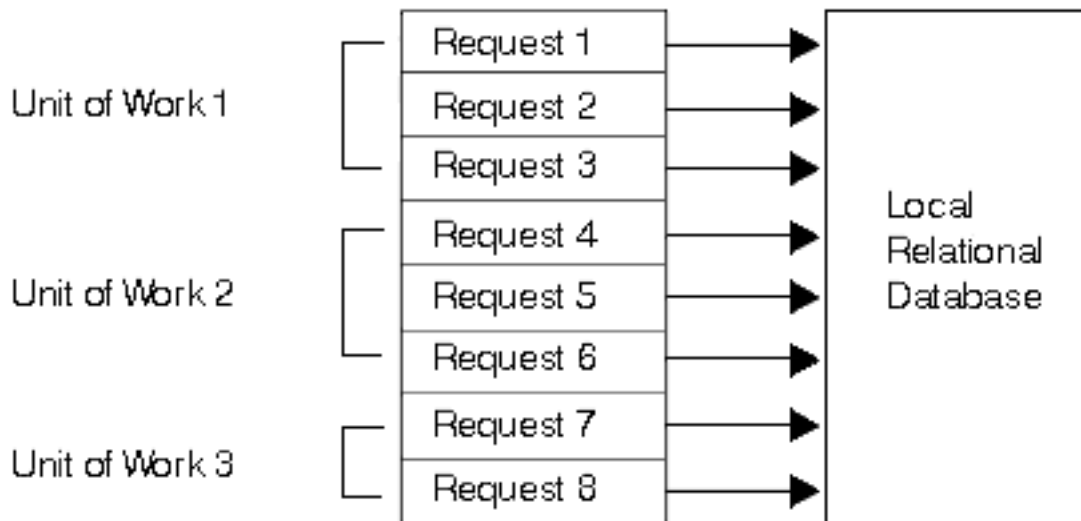


Figure 2. Unit of work in a local relational database

Related concepts

Relational database

Troubleshoot transactions and commitment control

XA transaction support for commitment control

Remote unit of work

Remote unit of work (RUW) is a form of distributed relational database processing in which an application program can access data on a remote database within a unit of work. A remote unit of work can include more than one relational database request, but all requests must be made to the same remote database. All requests to a relational database must be completed (either committed or rolled back) before requests can be sent to another relational database.

This is shown in the following figure.

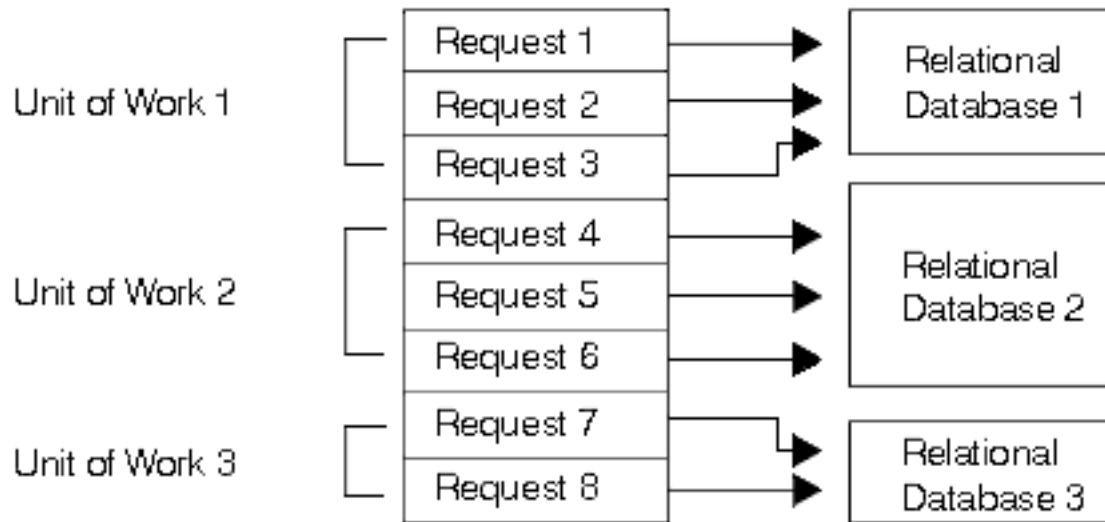


Figure 3. Remote unit of work in a distributed relational database

Remote unit of work is application-directed distribution because the application program must connect to the correct relational database system before issuing the requests. However, the application program only needs to know the name of the remote database to make the correct connection.

Remote unit of work support enables an application program to read or update data at more than one location. However, all the data that the program accesses within a unit of work must be managed by the same relational database management system. For example, the shop inventory application program must commit its inventory and accounts receivable unit of work before it can read or update tables that are in another location.

In remote unit of work processing, each computer has an associated relational database management system and an associated application requester program that help process distributed relational data requests. This allows you or your application program to request remote relational data in much the same way as you request local relational data.

Distributed unit of work

Distributed unit of work (DUW) enables a user or application program to read or update data at multiple locations within a unit of work. Within one unit of work, an application running on one system can direct SQL requests to multiple remote database management systems using the SQL supported by those systems.

For example, the shop inventory program can perform updates to the inventory table on one system and the accounts receivable table on another system within one unit of work. The following figure illustrates this idea.



Figure 4. Distributed unit of work in a distributed relational database

The target of the requests is controlled by the user or application with SQL statements such as CONNECT TO and SET CONNECTION. Each SQL statement must refer to data at a single location.

When the application is ready to commit the work, it initiates the commit; commitment coordination is performed by a synchronization-point manager or a transaction manager.

DUW allows update access or read-only access to multiple database management systems in one unit of work.

Whether an application can update a given database management system in a unit of work is dependent on the level of DRDA (if DRDA is used to access the remote relational database) and the order in which the connections and updates are made.

Other distributed relational database terms and concepts

This discussion provides an overview of additional distributed relational database concepts.

On IBM systems, some distributed relational database support is provided by the DB2 Universal Database for Linux[®], UNIX[®] and Windows[®] and DataPropagator[™] for iSeries products. In addition, you can use some of these concepts when writing iSeries application programs.

DB2 Universal Database for iSeries supports both the remote unit of work and distributed unit of work with Advanced Program-to-Program Communications (APPC) and TCP/IP communications, starting in OS/400[®] V5R1. A degree of processing sophistication beyond the distributed unit of work is a distributed request. This type of distributed relational database access enables a user or application program to issue a single SQL statement that can read or update data at multiple locations.

Tables in a distributed relational database do not have to differ from one another. Some tables can be exact or partial copies of one another. Extracts, snapshots, and replication are terms that describe types of copies using distributed processing.

Extracts are user-requested copies of tables. The copies are extracted from one database and loaded into another specified by the user. The unloading and loading process might be repeated periodically to obtain updated data. Extracts are most useful for one-time or infrequent occurrences, such as read-only copies of data that rarely changes.

Snapshots are read-only copies of tables that are automatically made by a server. The server refreshes these copies from the source table on a periodic basis specified by the user—perhaps daily, weekly, or monthly. Snapshots are most useful for locations that seek an automatic process for receiving updated information on a periodic basis.

Data replication means the server automatically updates copies of a table. It is similar to snapshots because copies of a table are stored at multiple locations. Data replication is most effective for situations that require high reliability and quick data retrieval with few updates.

Tables can also be split across computer servers in the network. Such a table is called a distributed table. Distributed tables are split either horizontally by rows or vertically by columns to provide easier local reference and storage. The columns of a vertically distributed table reside at various locations, as do the rows of a horizontally distributed table. At any location, the user still sees the table as if it were kept in a single location. Distributing tables is most effective when the request to access and update certain portions of the table come from the same location as those portions of the table.

Related concepts

“Distributed relational database on i5/OS” on page 11

DB2 Universal Database for i5/OS provides all the database management functions for i5/OS.

Distributed relational database support on the system is an integral part of the operating system, just as is support for communications, work management, security functions and other functions.

Distributed relational database architecture support

DRDA support for distributed relational database processing is used by IBM relational database products. DRDA support defines protocols for communication between an application program and a remote relational database.

Distributed Relational Database Architecture (DRDA) support provides distributed relational database management in both IBM and non-IBM environments. In IBM environments, relational data is managed with the following programs:

- DB2 Universal Database for iSeries
- DB2 Universal Database for z/OS®
- DB2 Universal Database for VSE/VM
- DB2 Universal Database for AIX®
- DB2 Universal Database for Linux
- DB2 Universal Database for HP-UX
- DB2 Universal Database for Sun Solaris
- DB2 Universal Database for Windows

DRDA support provides the structure for access to database information for relational database managers operating in **like** and **unlike** environments. For example, access to relational data between two or more DB2 UDB for iSeries is distribution in a like environment. Access to relational data between DB2 UDB for iSeries and another type of system or a client different from the one embedded in i5/OS is distribution in an unlike environment. One specific example of this is access to relational data between DB2 UDB for iSeries and IBM DB2 Universal Driver for SQLJ and JDBC.

SQL is the standard IBM database language. It provides the necessary consistency to enable distributed data processing across like and unlike operating environments. Within DRDA support, SQL allows users to define, retrieve, and manipulate data across environments that support a DRDA implementation.

| The Distributed Relational Database Architecture is an extension of the distributed data management
| (DDM) architecture. However, DRDA and DDM methods for accessing data are different. DRDA is an
| extension of SQL whereas DDM is an extension of native I/O.

| Using distributed relational database processing, an application can connect to a remote server using the
| relational database directory on the local system. The relational database directory provides the necessary
| links between a relational database name and the communications path to that database. An application
| running under the distributed relational database only has to identify the database name and run the
| SQL statements needed for processing.

Related reference

IBM DB2 Universal Driver for SQLJ and JDBC 1.0

DRDA and CDRA support

One of the interesting possibilities in a distributed relational database is that the database might not only span different types of computers, but those computers might be in different countries or regions. The same servers, such as iSeries servers, can encode data differently depending on the language used on the server.

Different types of servers encode data differently. For instance, an S/390[®], an iSeries server, and a PS/2[®] system encode numeric data in their own unique formats. In addition, an S/390 and an iSeries server use the EBCDIC encoding scheme to encode character data, while a PS/2 system uses an ASCII encoding scheme.

For numeric data, these differences do not matter. Unlike systems that provide DRDA support automatically convert any differences between the way a number is represented in one computer system to the way it is represented in another. For example, if an iSeries application program reads numeric data from a DB2 Universal Database for iSeries database, DB2 UDB for iSeries sends the numeric data in S/390 format and the i5/OS database management system converts it to iSeries numeric format.

However, the handling of character data is more complex, but this too can be handled within a distributed relational database.

Character conversion with CDRA

Not only can there be differences in encoding schemes, such as Extended Binary Coded Decimal Interchange Code (EBCDIC) versus American Standard Code for Information Interchange (ASCII), but there can also be differences related to language.

For instance, systems configured for different languages can assign different characters to the same code, or different codes to the same character. For example, a system configured for U.S. English can assign the same code to the character } that a system configured for the Danish language assigns to å. But those two systems can assign different codes to the same character such as \$.

If data is to be shared across different servers, character data needs to be seen by users and applications the same way. In other words, a PS/2 user in New York and an iSeries server user in Copenhagen both need to see a \$ as a \$, even though \$ might be encoded differently in each server. Furthermore, the user in Copenhagen needs to see a }, if that is the character that was stored at New York, even though the code might be the same as a Danish å. In order for this to happen, the \$ must be converted to the proper character encoding for a PS/2 system (that is, U.S. English character set, ASCII), and converted back to Danish encoding when it goes from New York to Copenhagen (that is, Danish character set, EBCDIC). This sort of character conversion is provided for by iSeries server as well as the other IBM distributed relational database managers. This conversion is done in a coherent way in accordance with the Character Data Representation Architecture (CDRA).

CDRA specifies the way to identify the attributes of character data so that the data can be understood across servers, even if the servers use different character sets and encoding schemes. For conversion to happen across servers, each server must understand the attributes of the character data it is receiving from the other server. CDRA specifies that these attributes be identified through a coded character set identifier (CCSID). All character data in DB2 Universal Database for z/OS, DB2 Universal Database for VM, and the i5/OS database management systems have a CCSID, which indicates a specific combination of encoding scheme, character set, and code page. All character data in an Extended Services[®] environment has a code page only (but the other database managers treat that code page identification as a CCSID). A code page is a specific set of assignments between characters and internal codes.

For example, CCSID 37 means encoding scheme 4352 (EBCDIC), character set 697 (Latin, single-byte characters), and code page 37 (USA/Canada country extended code page). CCSID 5026 means encoding scheme 4865 (extended EBCDIC), character set 1172 with code page 290 (single-byte character set for Katakana/ Kanji), and character set 370 with code page 300 (double-byte character set for Katakana/Kanji).

DRDA-enabled systems include mechanisms to convert character data between a wide range of CCSID-to-CCSID pairs and CCSID-to-code page pairs. Character conversion for many CCSIDs and code pages is already built into these products. For more information on CCSIDs supported by iSeries, see the i5/OS globalization topic. For a description of the use of CCSIDs on the iSeries server, see coded character set identifier (CCSID).

Related concepts

i5/OS globalization

Related reference

“Coded character set identifier” on page 86

Support for the national language of any country requires the proper handling of a minimum set of characters.

Application requester driver programs

An application requester driver (ARD) program is a type of exit program that enables SQL applications to access data managed by a database management system other than DB2 Universal Database for iSeries.

An iSeries client calls the ARD program during the following operations:

- The package creation step of SQL precompiling, performed using the Create Structured Query Language Package (CRTSQLPKG) command or CRTSQLxxx commands, when the relational database (RDB) parameter matches the RDB name corresponding to the ARD program.
- Processing of SQL statements when the current connection is to an RDB name corresponding to the ARD program.

These calls allow the ARD program to pass the SQL statements and information about the statements to a remote relational database and return results back to the application requester (AR). The AR then returns the results to the application or the user. Access to relational databases accessed by ARD programs appear like access to DRDA application servers in the unlike environment.

The ARD program is registered in the system by use of the Add Relational Database Directory Entry (ADDRDBDIRE) command. One of the parameters that is specified is the library in which the program is located. For a system configured with independent auxiliary storage pools, the ARD program must reside in a library in the system database (a library that is part of the system ASP or a configured basic ASP).

Related reference

Add Relational Database Directory Entry (ADDRDBDIRE) command

Create Structured Query Language Package (CRTSQLPKG) command

Related information

Application programming interfaces

Distributed relational database on i5/OS

DB2 Universal Database for i5/OS provides all the database management functions for i5/OS. Distributed relational database support on the system is an integral part of the operating system, just as is support for communications, work management, security functions and other functions.

The i5/OS operating system can be part of a distributed relational database network with other servers that support a Distributed Relational Database Architecture (DRDA) implementation. The i5/OS operating system can be an application requester (AR) or an application server (AS) in either like or unlike environments. Distributed relational database implementation on the iSeries system supports remote unit of work (RUW) and distributed unit of work (DUW). RUW allows you to submit multiple requests to a single database within a single unit of work, and DUW allows requests to multiple databases to be included within a single unit of work.

For example, using DUW support you can decrement the inventory count of a part on one server and increment the inventory count of a part on another server within a unit of work, and then commit changes to these remote databases at the conclusion of a single unit of work using a two-phase commit process. DB2 Universal Database for iSeries does not support distributed requests, so you can only access one database with each SQL statement. The level of support provided in an application program depends on the level of support available on the application server (AS) and the order in which connections and updates are made. See *Connect to a distributed relational database* for more information.

In addition to DRDA access, application requester driver (ARD) programs can be used to access databases that do not support DRDA. Connections to relational databases accessed through ARD programs are treated like connections to unlike servers. Such connections can coexist with connections to DRDA application servers, connections to the local relational database, and connections which access other ARD programs.

On the iSeries server, the distribution functions of snapshots and replication are not automatically performed by the server. You can install and configure the DataPropagator for iSeries, V8.1 product on iSeries servers to perform these functions. Also, you can use these functions in user-written application programs. More information about how you can organize these functions in a distributed relational database is discussed in *Data availability and protection for a distributed relational database*.

On the iSeries server, the distributed request function is not directly supported. However, the DataJoiner[®] product can perform distributed queries, joining tables from a variety of data sources. DataJoiner works synergistically with DataGuide, a comprehensive information catalog in the IBM Information Warehouse family of products. DataGuide provides a graphical user interface to complete information listings about a company's data resources.

The i5/OS licensed program includes runtime support for SQL. You do not need the DB2 UDB for iSeries Query Manager and SQL Development Kit licensed program installed on a DB2 UDB for iSeries application requester or application server to process distributed relational database requests or to create an SQL collection on an iSeries server. However, you do need the DB2 Universal Database for iSeries Query Manager and SQL Development Kit program to precompile programs with SQL statements, run Interactive SQL, or run DB2 Universal Database for iSeries Query Manager.

- | Communications support for the DRDA implementation on the iSeries is provided under either TCP/IP
- | or the IBM Systems Network Architecture (SNA) through the Advanced Program-to-Program
- | Communication (APPC) protocol, with or without Advanced Peer-to-Peer Networking[®] (APPN).

Related concepts

“Other distributed relational database terms and concepts” on page 7

This discussion provides an overview of additional distributed relational database concepts.

“Connect to a distributed relational database” on page 73

What makes a distributed relational database application *distributed* is its ability to connect to a relational database on another server.

“Data availability and protection” on page 126

In a distributed relational database environment, data availability not only involves protecting data on an individual server in the network, but also ensuring that users have access to the data across the network.

APPC, APPN and HPR

Communications Configuration PDF

Configure TCP/IP

OptiConnect

Manage an iSeries distributed relational database

Managing a distributed relational database on the iSeries server requires broad knowledge of the resources and tools within the i5/OS licensed program.

This topic collection provides an overview of the various functions available with the operating system that can help you administer a distributed relational database on the iSeries server. This topic explains distributed relational database functions and tasks in a network of iSeries servers (a *like* environment). Differences between iSeries distributed relational database functions in a like and unlike environment are presented only in a general discussion in this topic.

A properly implemented distributed relational database makes it easy to access a database on a remote server, process a database file without knowing where it resides, and move parts of a database to another server without requiring changes to the application programs.

To effectively implement your distributed relational database, you should be familiar with the requirements in the following key areas:

- The Planning and design topic discusses some important considerations when planning for and designing a distributed database.
- The Security topic provides information about the security considerations for an iSeries distributed relational database, including communications and DRDA access to remote relational databases.
- The Configuring topic provides information about ways to enter data into a distributed database, along with a discussion of subsystems and relational database directories on the iSeries server.
- The Manage a distributed relational database topic discusses ways that you can administer the distributed relational database work being done across a network.
- The Data availability and protection topic discusses tools and techniques to protect programs and data on an iSeries server and reduce recovery time in the event of a problem. It also provides information about alternatives that ensure your network users have access to the relational databases and tables across the network when it is needed.
- The Performance topic discusses ways to improve on the design of your network, the system, and your database.

If you want more information about another IBM system that supports DRDA, see the information provided with that system or the books listed in Distributed relational database library and Other IBM distributed relational database platform libraries in Related information.

Related concepts

“Administration” on page 99

As an administrator for a distributed relational database, you are responsible for work being done on several servers.

“Data availability and protection” on page 126

In a distributed relational database environment, data availability not only involves protecting data on an individual server in the network, but also ensuring that users have access to the data across the network.

“Initial setup” on page 24

The runtime support for an iSeries distributed relational database is provided by the operating system. Therefore, when the operating system is installed, distributed relational database support is installed.

“Performance” on page 143

No matter what kind of application programs you are running on a server, performance can always be a concern. For a distributed relational database, network, server, and application performance are all crucial.

“Planning and design” on page 16

The first requirement for the successful operation of a distributed relational database is thorough planning. You must consider the needs and goals of your enterprise when making the decision to use a distributed relational database.

“Security” on page 50

The iSeries server has security elements built into the operating system to limit access to the data resources of an application server. Security options range from simple physical security to full password security coupled with authorization to commands and data objects.

“User FAQs” on page 219

This topic collection concentrates on describing iSeries support for distributed relational databases in a network of iSeries servers (a *like* environment).

Related reference

“Distributed relational database library” on page 247

The books in this topic provide background and general support information for IBM Distributed Relational Database Architecture (DRDA) implementations.

“Other IBM distributed relational database platform libraries” on page 248

This topic describes other IBM distributed relational database platform libraries.

“Related information for distributed database programming” on page 246

Listed here are the product manuals and IBM Redbooks™ (in PDF format), Web sites and information center topics that relate to the distributed database programming topic. You can view or print any of the PDFs.

Example: Spiffy Corporation distributed relational database

The Spiffy Corporation is used in several IBM manuals to describe distributed relational database support. In this topic collection, this fictional company has been changed somewhat to illustrate iSeries server support for DRDA in an iSeries server network.

Examples used throughout this topic collection illustrate particular functions, connections, and processes. These might not correspond exactly to the examples used in other distributed relational database publications but an attempt has been made to make them look familiar.

Though the Spiffy Corporation is a fictional enterprise, the business practices described here are modeled after those in use in several companies of similar construction. However, this example does not attempt to describe all that can be done using a distributed relational database, even by this example company.

Spiffy organization and system profile

Spiffy Corporation is a fictional national product distributor that sells and services automobiles, among other products, to retail customers through a network of regional offices and local dealerships.

Given the high competitiveness of today’s automobile industry, the success of an operation like the Spiffy Corporation depends on high-quality servicing and timely delivery of spare parts to the customer. To meet this competition, Spiffy has established a vast service network incorporated within its dealership organization.

The dealership organization is headed by a central vehicle distributor located in Chicago, Illinois. There are several regional distribution centers across North America. Two of these are located in Minneapolis,

Minnesota and Kansas City, Missouri. These centers minimize the distribution costs of vehicles and spare parts by setting up regional inventories. The Minneapolis regional center serves approximately 15 dealerships while the Kansas City center serves as many as 30 dealerships.

The figure here illustrates a system organization chart for Spiffy Corporation.

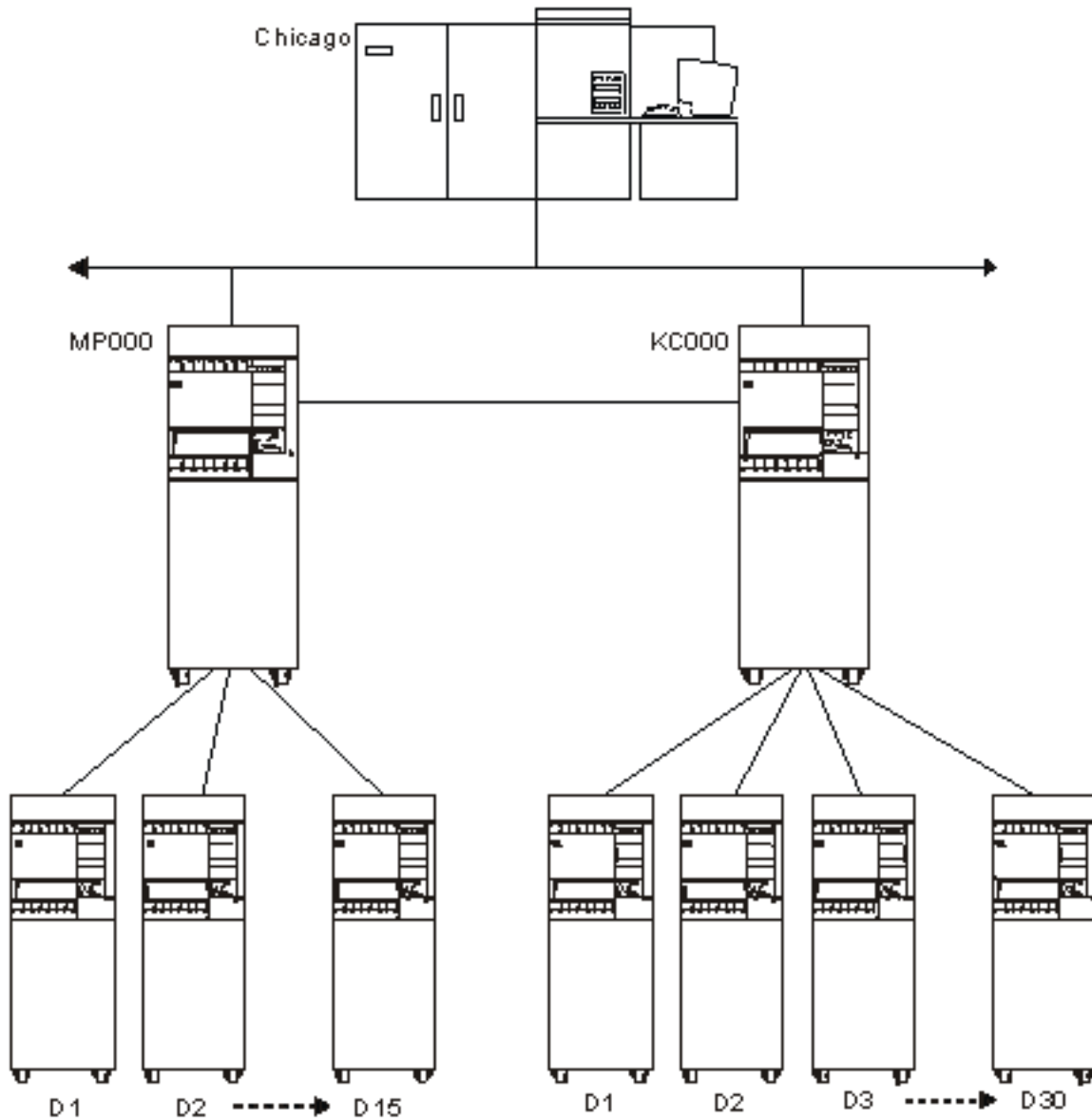


Figure 5. The Spiffy Corporation system organization

Spiffy is in the process of building up a nationwide integrated telecommunications network. For the automobile division they are setting up a network of iSeries servers for the regional distributions centers and the dealerships. These are connected to an S/390 at the central vehicle distributor. This network is considered a vital business asset for maintaining the competitive edge.

The central distributor runs DB2 Universal Database for z/OS on its S/390 with relevant decision support software. This system is used because of the large amounts of data that must be handled at any one time

in a variety of application programs. The central vehicle distributor system is not dedicated to automobile division data processing. It must handle work and processes for the corporation that do not yet operate in a distributed database environment. The regional centers are running iSeries systems. They use APPC/APPN with SNADS and 5250 Display Station Pass-through using an SDLC protocol.

All of the dealerships use iSeries servers that vary in size. These systems are connected to the regional office using SDLC protocol. The largest dealerships have a part time programmer and a system operator to tend to the data processing functioning of the enterprise. Most of the installations do not employ anyone with programming expertise and some of the smaller locations do not employ anyone with more than a very general knowledge of computers.

Business processes of the Spiffy Corporation Automobile Service

The Spiffy Corporation automobile division has business practices that are automated in this distributed relational database environment.

To keep the examples from becoming more complicated than necessary, consider just those functions in the company that pertain to vehicle servicing.

Dealerships can have a list of from 2000 to 20 000 customers. This translates to 5 service orders per day for a small dealership and up to 50 per day for a large dealership. These service orders include scheduled maintenance, warranty repairs, regular repairs, and parts ordering.

The dealers stock only frequently needed spare parts and maintain their own inventory databases. Both regional centers provide parts when requested. Dealer inventories are also stocked on a periodic basis by a forecast-model-controlled batch process.

Distributed relational database administration for the Spiffy Corporation

Each dealership manages its data processing resources and procedures as a stand-alone enterprise. Spiffy Corporation requires that each dealership have one or more iSeries servers and that those servers must be available to the network at certain times. However, the size of the server and the number of business processes that are automated on it are determined by each dealership's needs and the resources available to it.

The Spiffy Corporation requires all dealerships to be active in the inventory distributed relational database. Because the corporation operates its own dealerships, it has a full complement of dealership software that might or might not access the distributed relational database environment. The Spiffy dealerships use the full set of software tools. Most of the private franchises also use them because they are tailored specifically to the Spiffy Corporation way of doing business.

The regional distribution centers manage the inventory for their region. They also function as the database administrator for all distributed database resources used in the region. The responsibilities involved vary depending on the level of data processing competency at each dealership. The regional center is always the first contact for help for any dealership in the region.

The Minneapolis regional distribution center has a staff of iSeries programmers with a wide range of experience and knowledge about the servers and the network. The dealership load is about one half that of other regional centers to allow this center to focus on network-wide iSeries support functions. These functions include application program development, program maintenance, and problem handling.

Listed here are the database responsibilities for each level of activity in the network:

Dealerships

- Perform basic operation and administration of the server
- Enroll local users

Regional distribution centers

- Set up data processing for new dealerships
- Disperse database resources for discontinued dealerships
- Enroll network users in region
- Maintain inventory for region
- Develop service plans for dealerships
- Operate help desk for dealerships

Other activities

In addition to the regional distribution center activities, the Minneapolis iSeries server competency center does the following activities:

- Develop applications for iSeries servers
- Operate help desk for regional centers
- Tune database performance
- Resolve database problems

Examples used throughout this topic are associated with one or more of these activities. Many examples show the process of obtaining a part from inventory in order to schedule customer service or repairs. Others show distributed relational database administration tasks used to set up, secure, monitor, and resolve problems for servers in the Spiffy Corporation distributed relational database network.

Planning and design

The first requirement for the successful operation of a distributed relational database is thorough planning. You must consider the needs and goals of your enterprise when making the decision to use a distributed relational database.

How you code an application program, where it resides in relation to the data, and the network design that connects application programs to data are all important design considerations.

Database design in a distributed relational database is more critical than when dealing with just one iSeries relational database. With more than one iSeries server to consider, you must develop a consistent management strategy across the network. The following operations require particular attention when forming your strategy:

- General operations
- Networking protocol
- System security
- Accounting
- Problem analysis
- Backup and recovery processes

To prepare for a distributed relational database, you must understand both the needs of the business and relational database technology.

Related concepts

“Improve distributed relational database performance through the database” on page 145

Distributed relational database performance is affected by the overall design of the database. The location of distributed data, the level of commitment control you use, and the design of your SQL indexes all affect performance.

Identify your needs and expectations for a distributed relational database

Consider these items when analyzing your needs and expectations of a distributed relational database.

Data needs for distributed relational databases

The first step in your analysis is to determine which factors affect your data and how they affect it.

Ask yourself the following questions:

- What locations are involved?
- What kind of transactions do you envision?
- What data is needed for each transaction?
- What dependencies do items of data have on each other, especially referential limitations? For example, will information in one table need to be checked against the information in another table? (If so, both tables must be kept at the same location.)
- Does the data currently exist? If so, where is it located? Who "owns" it (that is, who is responsible for maintaining the accuracy of the data)?
- What priority do you place on the availability of the needed data? Integrity of the data across locations? Protection of the data from unauthorized access?
- What access patterns do you envision for the data? For instance, will the data be read, updated, or both? How frequently? Will a typical access return a lot of data or a little data?
- What level of performance do you expect from each transaction? What response time is acceptable?

Distributed relational database capabilities

The second step in your analysis is to decide whether your data needs lend themselves to a distributed relational database solution.

Applications where most database processing is done locally and access to remote data is needed only occasionally are typically *good* candidates for a distributed relational database.

Applications with the following requirements are usually *poor* candidates for a distributed relational database:

- The data is kept at a central site *and* most of the work that a remote user needs to do is at the central site.
- Consistently high performance, especially consistently fast response time, is needed. It takes longer to move data across a network.
- Consistently high availability, especially twenty-four hour, seven-day-a-week availability, is needed. Networks involve more systems and more in-between components, such as communications lines and communications controllers, which increases the chance of breakdowns.
- A distributed relational database function that you need is not currently available or announced.

Goals and directions for a distributed relational database

The third step in your analysis is to assess your short-term and long-term goals.

SQL is the standard IBM database language. If your goals and directions include portability or remote data access on unlike systems, you should use distributed relational database on the iSeries server.

The distributed database function of distributed unit of work, as well as the additional data copying function provided by DataPropagator for iSeries, V8.1, broadens the range of activities you can perform on the iSeries server. However, if your distributed database application requires a function that is not currently available on the iSeries server, other options are available until the function is made available on the operating system. For example, you can do one of the following things:

- Provide the needed function yourself.

- Stage your plans for distributed relational database to allow for the new function to become available.
- Reassess your goals and requirements to see if you can satisfy them with a currently available or announced function. Some alternative solutions are listed in the following table. These alternatives can be used to supplement or replace available function.

Table 3. Alternative solutions to distributed relational database

Solution	Description	Advantages	Disadvantages
Distributed Data Management (DDM)	A function of the operating system that allows an application program or user on one system to use database files stored on a remote system. The system must be connected by a communications network, and the remote system must also use DDM.	<ul style="list-style-type: none"> • For simple read and update accesses, the performance is better than for SQL. • Existing applications do not need to be rewritten. • Can be used to access S/38, S/36, and CICS®. 	<ul style="list-style-type: none"> • SQL is more efficient for complex functions. • Might not be able to access other distributed relational database platforms. • Does not perform CCSID and numeric data conversions.
Intersystem Communications Function/Common Programming Interface (ICF/CPI Communications)	ICF is a function of the operating system that allows a program to communicate interactively with another program or system. CPI Communications is a call-level interface that provides a consistent application interface for applications that use program-to-program communications. These interfaces make use of SNA's logical unit (LU) 6.2 architecture to establish a conversation with a program on a remote system, to send and receive data, to exchange control information, to end a conversation, and to notify a partner program of errors.	<ul style="list-style-type: none"> • Allows you to customize your application to meet your needs. • Can provide better performance. 	Compared to distributed relational database and DDM, a more complicated program is needed to support communications and data conversion requirements.
Display station pass-through	A communications function that allows a user to sign on to one iSeries server from another iSeries server and use that server's programs and data.	<ul style="list-style-type: none"> • Applications and data on remote systems are accessible from local systems. • Allows for quick access when data is volatile and a large amount of data on one server is needed by users on several servers. 	Response time on screen updates is slower than locally attached devices.

A distributed relational database usually evolves from simple to complex as business needs change and new products are made available. Remember to consider this when analyzing your needs and expectations.

Design the application, network, and data for a distributed relational database

Designing a distributed relational database involves making choices about applications, network considerations, and data considerations.

Tips: Design distributed relational database applications

Distributed relational database applications have different requirements from applications developed solely for use on a local database.

To properly plan for these differences, design your applications with the following considerations in mind:

- Take advantage of the distributed unit of work (DUW) function where appropriate.

Note: Before Version 5 Release 1 of OS/400, two-phase commit support was not available with TCP/IP on the iSeries server.

- Code programs using common interfaces.
- Consider dividing a complex application into smaller parts and placing each piece of the application in the location best suited to process it. One good way to distribute processing in an application is to make use of the SQL CALL statement to run a stored procedure at a remote location where the data to be processed resides. The stored procedure is not limited to SQL operations when it runs on a DB2 Universal Database for iSeries application server; it can use integrated database input/output or perform other types of processing.
- Investigate how the initial database applications will be prepared, tested, and used.
- Take advantage, when possible, of SQL set-processing capabilities. This will minimize communication with the application servers. For example, update multiple rows with one SQL statement whenever you can.
- Be aware that database updates within a unit of work must be done at a single site if the remote unit of work (RUW) connection method is used when the programs are prepared, or if the other nodes in the distributed application do not support DUW.
- Keep in mind that the DUW connection method restricts you from directing a single statement to more than one relational database.
- Performance is affected by the choice of connection management methods. Use of the RUW connection management method might be preferable if you do not have the need to switch back and forth among different remote relational databases. This is because more overhead is associated with the two-phase commit protocols used with DUW connection management.

However, if you have to switch frequently among multiple remote database management systems, use DUW connection management. When running with DUW connection management, communication conversations to one database management system do not have to be ended when you switch the connection to another database management system. In the like environment, this is not as big a factor as in the unlike environment, since conversations in the like environment can be kept active by use of the default DDMCNV(*KEEP) job definition attribute. Even in the like environment, however, a performance advantage can be gained by using DUW to avoid the cost of closing cursors and sending the communication flow to establish a new connection.

- The connection management method determines the semantics of the CONNECT statement. With the RUW connection management method, the CONNECT statement ends any existing connections before establishing a new connection to the relational database. With the DUW connection management method, the CONNECT statement does not end existing connections.

Network considerations for a distributed relational database

The design of a network directly affects the performance of a distributed relational database.

To properly design a distributed relational database that works well with a particular network, do the following things:

- Because the line speed can be very important to application performance, provide sufficient capacity at the appropriate places in the network to achieve efficient performance to the main distributed relational database applications.
- Evaluate the available communication hardware and software and, if necessary, your ability to upgrade.

- For Advanced Program-to-Program Communication (APPC) connections, consider the session limits and conversation limits specified when the network is defined.
- Identify the hardware, software, and communication equipment needed (for both test and production environments), and the best configuration of the equipment for a distributed relational database network.
- Consider the skills that are necessary to support TCP/IP as opposed to those that are necessary to support APPC.
- Take into consideration the initial service level agreements with end user groups (such as what response time to expect for a given distributed relational database application), and strategies for monitoring and tuning the actual service provided.
- Understand that you cannot use an APPC-protected DUW conversation to connect to a database from an application requester (AR) which has been set to an auxiliary storage pool (ASP) group for the current thread.
- Develop a naming strategy for database objects in the distributed relational database and for each location in the distributed relational database. A location is a specific relational database management system in an interconnected network of relational database management systems that participate in distributed relational database. A location in this sense can also be a user database in a system configured with independent ASP groups. Consider the following items when developing this strategy:
 - The fully qualified name of an object in a distributed database has three (rather than two) parts, and the highest-level qualifier identifies the location of the object.
 - Each location in a distributed relational database should be given a unique identification; each object in the database should also have a unique identification. Duplicate identifications can cause serious problems. For example, duplicate locations and object names might cause an application to connect to an unintended remote database, and once connected, access an unintended object. Pay particular attention to naming when networks are coupled.
 - Each location in a user database should also be given a unique identification. If a user database on two different servers were to be named PAYROLL, there would be a naming conflict if an application needed to access them both from the same server. Note that when an independent ASP device is configured, the user has an option to specify an RDB name for that device that is different from the name of the ASP device itself. It is the RDB name associated with the primary device in an ASP group by which that user database is known.

Related concepts

Communications Management PDF

Data considerations for a distributed relational database

The placement of data in respect to the applications that need it is an important consideration when designing a distributed relational database.

When making such placement decisions, consider the following items:

- The level of performance needed from the applications
- Requirements for the security, currency, consistency, and availability of the data across locations
- The amount of data needed and the predicted patterns of data access
- If the distributed relational database functions needed are available
- The skills needed to support the server and the skills that are actually available
- Who "owns" the data (that is, who is responsible for maintaining the accuracy of the data)
- Management strategy for cross-system security, accounting, monitoring and tuning, problem handling, data backup and recovery, and change control
- Distributed database design decisions, such as where to locate data in the network and whether to maintain single or multiple copies of the data

Develop a management strategy for a distributed relational database

When you are managing a distributed relational database, keep these strategies in mind.

General operations for a distributed relational database

To plan for the general operation of a distributed relational database, consider both performance and availability.

The following design considerations can help you improve both the performance and availability of a distributed relational database:

- If an application involves transactions that run frequently or that send or receive a lot of data, you should try to keep it in the same location as the data.
- For data that needs to be shared by applications in different locations, put the data in the location with the most activity.
- If the applications in one location need the data as much as the applications in another location, consider keeping copies of the data at both locations. When keeping copies at multiple locations, ask yourself the following questions about your management strategy:
 - Will users be allowed to make updates to the copies?
 - How and when will the copies be refreshed with current data?
 - Will all copies have to be backed up or will backing up one copy be sufficient?
 - How will general administration activities be performed consistently for all copies?
 - When is it permissible to delete one of the copies?
- Consider whether the distributed databases will be administered from a central location or from each database location.

You can also improve performance by doing the following things:

- If data and applications must be kept at different locations, do the following things to keep the performance within acceptable limits:
 - Keep data traffic across the network as low as possible by only retrieving the data columns that will be used by the application; that is, avoid using * in place of a list of column names as part of a SELECT statement.
 - Discourage programmers from coding statements that send large amounts of data to or receive large amounts of data from a remote location; that is, encourage the use of the WHERE clause of the SELECT statement to limit the number of rows of data.
 - Use referential integrity, triggers, and stored procedures (an SQL CALL statement after a CONNECT to a remote relational database management system); this improves performance by distributing processing to the application server (AS), which can substantially reduce line traffic.
 - Use read-only queries where appropriate by specifying the FOR FETCH ONLY clause.
 - Be aware of rules for blocking of queries. For example, in iSeries-to-iSeries queries, blocking of read-only data is done only for COMMIT(*NONE), or for COMMIT(*CHG) and COMMIT(*CS) when ALWBLK(*ALLREAD) is specified.
 - Keep the number of accesses to remote data low by using local data in place of remote data whenever possible.
 - Use SQL set operations to process multiple rows at the application requester with a single SQL request.
 - Try to avoid dropping of connections by using DDMCNV(*KEEP) when running with remote unit of work (RUW) connection management, or by running with distributed unit of work (DUW) connection management.
- Provide sufficient network capacity by doing the following things:
 - Increase the capacity of the network by installing high-speed, high-bandwidth lines or by adding lines at appropriate points in the network.

- Reduce the contention or improve the contention balance on certain processors. For example, move existing applications from a host server to a departmental server or group some distributed relational database work into batch.
- Encourage good table design. At the distributed relational database locations, encourage appropriate use of primary keys, table indexes, and normalization techniques.
- Ensure data types of host variables used in WHERE clauses are consistent with the data types of the associated key column data types. For example, a floating-point host variable has been known to disqualify the use of an index built over a column of a different data type.

You can also improve availability by doing the following things:

- In general, try to limit the amount of data traffic across the network.
- If data and applications must be kept at different locations, do the following things to keep the availability within acceptable limits:
 - Establish alternate network routes.
 - Consider the effect of time zone differences on availability:
 - Will qualified people be available to bring up the server?
 - Will off-hours batch work interfere with processing?
 - Ensure good backup and recovery features.
 - Ensure people are skilled in backup and recovery.

Security considerations for a distributed relational database

Part of planning for a distributed relational database involves the decisions you must make about securing distributed data.

These decisions include:

- What systems should be made accessible to users in other locations and which users in other locations should have access to those systems.
- How tightly controlled access to those systems should be. For example, should a user password be required when a conversation is started by a remote user?
- Is it required that passwords flow over the wire in encrypted form?
- Is it required that a user profile under which a client job runs be mapped to a different user identification or password based on the name of the relational database to which you are connecting?
- What data should be made accessible to users in other locations and which users in other locations should have access to that data.
- What actions those users should be allowed to take on the data.
- Whether authorization to data should be centrally controlled or locally controlled.
- If special precautions should be taken because multiple systems are being linked. For example, should name translation be used?

When making the previous decisions, consider the following items when choosing locations:

- Physical protection. For example, a location might offer a room with restricted access.
- Level of system security. The level of system security often differs between locations. The security level of the distributed database is no greater than the lowest level of security used in the network.

All servers connected by Advanced Program-to-Program Communication (APPC) can do the following things:

- If both servers are iSeries servers, communicate passwords in encrypted form.
- Verify that when one server receives a request to communicate with another server in the network, the requesting server is actually "who it says it is" and that it is authorized to communicate with the receiving server.

All servers can do the following things:

- Pass a user’s identification and password from the local server to the remote server for verification before any remote data access is allowed.
- Grant and revoke privileges to access and manipulate SQL objects such as tables and views.

The iSeries server includes security audit functions that allow you to track unauthorized attempts to access data, as well as to track other events pertinent to security. The server also provides a function that can prevent all distributed database access from remote servers.

- Security-related costs. When considering the cost of security, consider both the cost of buying security-related products and the price of your information staff’s time to perform the following activities:
 - Maintain server identification of remote-data-accessing users at both local and remote servers.
 - Coordinate auditing functions between sites.

Related concepts

“Security” on page 50

The iSeries server has security elements built into the operating system to limit access to the data resources of an application server. Security options range from simple physical security to full password security coupled with authorization to commands and data objects.

Account for a distributed relational database

You need to be able to account and charge for the use of distributed data.

Consider the following items:

- Accounting for the use of distributed data involves the use of resources in one or more remote servers, the use of resources on the local server, and the use of network resources that connect the servers.
- Accounting information is accumulated by each server independently. Network accounting information is accumulated independent of the data accumulated by the servers.
- The time zones of various servers might have to be taken into account when trying to correlate accounting information. Each server clock might not be synchronized with the remote server clock.
- Differences might exist between each server’s permitted accounting codes (numbers). For example, the iSeries server restricts accounting codes to a maximum of 15 characters.

The following functions are available to account for the use of distributed data:

- iSeries server job accounting journal. The iSeries server writes job accounting information into the job accounting journal for each distributed relational database application. The Display Journal (DSPJRN) command can be used to write the accumulated journal entries into a database file. Then, either a user-written program or query functions can be used to analyze the accounting data.
- NetView[®] accounting data. The NetView licensed program can be used to record accounting data about the use of network resources.

Related reference

Display Journal (DSPJRN) command

“Job accounting in a distributed relational database” on page 113

The job accounting function on the iSeries server gathers data so you can determine who is using the server and what server resources they are using. Typical job accounting provides details on the jobs running on a server and resources used, such as use of the processing unit, printer, display stations, and database and communications functions.

Problem analysis for a distributed relational database

Problem analysis needs to be managed in a distributed database environment. Problem analysis involves both identifying and resolving problems for applications that are processed across a network of servers.

Consider the following items:

- Distributed database processing problems manifest themselves in various ways. For example, an error return code might be passed to a distributed database application by the server that detects the problem. In addition, responses might be slow, wrong, or nonexistent.
- Tools are available to diagnose distributed database processing problems. For example, each distributed relational database product provides trace functions that can help diagnose distributed data processing problems.
- When server failures are detected by an iSeries server, the server logs information about program status immediately after the failure is detected.

Note: If a correction to an IBM program is required and if you have an S/390 with Network Distribution Manager (NDM) installed in the network, you can use the NDM and the Distributed System Node Executive products to receive and transmit updates and replacements to appropriate servers in the network.

Backup and recovery for a distributed relational database

In a single-server environment, backup and recovery take place locally. But in a distributed database, backup and recovery also affect remote locations.

The iSeries server allows individual tables, collections, or groups of collections to be backed up and recovered. Although backup and recovery can only be done locally, you might want to plan to have less critical data on a server that does not have adequate backup support. Backup and recovery procedures must be consistent with data that might exist on more than one application server. Because you have more than one server in the network, you might want to save such data to a second server so that it is always available to the network in some form. Strategies such as these need to be planned and laid out specifically before a database is distributed across the network.

Initial setup

The runtime support for an iSeries distributed relational database is provided by the operating system. Therefore, when the operating system is installed, distributed relational database support is installed.

However, some setup work might be required to make the application requesters and application servers ready to send and receive work, particularly in the Advanced Program-to-Program Communication (APPC) environment. One or more subsystems can be used to control interactive, batch, spooled, and communications jobs. All the application requesters (ARs) in the network must have their relational database directory set up with connection information. Finally, you might want to put data into the tables of the application servers throughout the network.

The relational database directory contains database names and values that are translated into communications network parameters. An AR must have an entry for each database in the network, including the local database and any user databases that are configured on independent auxiliary storage pools (IASPs, also known as independent disk pools). These local entries can be added automatically by the system, or manually. Each directory entry consists of a unique relational database name and corresponding communications path information. Information about the preferred password security for outbound connections can be specified. For access provided by ARD programs, the ARD program name must be added to the relational database directory entry.

There are a number of ways to enter data into a database. You can use an SQL application program, some other high-level language application program, or one of these methods:

- Interactive SQL
- i5/OS query management
- Data file utility (DFU)
- Copy File (CPYF) command

Connection and setup information for a distributed relational database network of unlike servers can be found in the *Distributed Relational Database Cross-Platform Connectivity* book, SG24-4311-02.

Related concepts

Independent auxiliary storage pool (IASP)

Independent disk pools

Related reference

Copy File (CPYF) command

Work management on the iSeries server

All of the work done on the iSeries server is submitted through the work management function. On an iSeries server, you can design specialized operating environments to handle different types of work to satisfy the requirements of your server.

However, when the operating system is installed, it includes a work management environment that supports interactive and batch processing, communications, and spool processing.

On the server, all user jobs operate in an environment called a subsystem, defined by a subsystem description, where the server coordinates processing and resources. Users can control a group of jobs with common characteristics independently of other jobs if the jobs are placed in the same subsystem. You can easily start and end subsystems as needed to support the work being done and to maintain the performance characteristics you want.

The basic types of jobs that run on the server are interactive, communications, batch, spooled, autostart, and prestart.

An interactive job starts when you sign on a workstation and ends when you sign off. An Advanced Program-to-Program Communication (APPC) batch job is a job started from a program start request from another system. A non-communications batch job is started from a job queue. Job queues are not used when starting a communications batch job. Spooling functions are available for both input and output. Autostart jobs perform repetitive work or one-time initialization work. Autostart jobs are associated with a particular subsystem, and each time the subsystem is started, the autostart jobs associated with it are started. Prestart jobs are jobs that start running before the remote program sends a program start request.

Related concepts

“Manage the TCP/IP server” on page 114

This topic collection describes how to manage the DRDA/DDM server jobs that communicate using sockets over TCP. It describes the subsystem in which the server runs, the objects that affect the server and how to manage those resources.

Set up your work management environment for DRDA

One subsystem, called a controlling subsystem, starts automatically when you load the server. Two controlling subsystem configurations are supplied by IBM.

The first configuration includes the following subsystems:

- QBASE, the controlling subsystem, supports interactive, batch, and communications jobs.
- QSPL supports processing of spooling readers and writers.
- QSYSWRK supports various server functions such as TCP/IP.
- QUSRWRK is the user work subsystem. It contains jobs that are started by servers to do work on behalf of a user.

QBASE automatically starts when the server is started. An automatically started job in QBASE starts QSPL.

The second controlling subsystem configuration supplied is more complex. This configuration includes the following subsystems:

- QCTL, the controlling subsystem, supports interactive jobs started at the console.
- QINTER supports interactive jobs started at other workstations.
- QCMN supports communications jobs.
- QBATCH supports batch jobs.
- QSPL supports processing of spooling readers and writers.
- QSYSWRK supports various server functions such as TCP/IP.
- QUSRWRK is the user work subsystem. It contains jobs that are started by servers to do work on behalf of a user.

If you change your configuration to use the QCTL controlling subsystem, it starts automatically when the system is started. An automatically started job in QCTL starts the other subsystems.

You can change your subsystem configuration from QBASE to QCTL by changing the system value QCTLSBSD (controlling subsystem) to QCTL on the Change System Value (CHGSYSVAL) command and starting the system again.

You can change the IBM-supplied subsystem descriptions or any user-created subsystem descriptions by using the Change Subsystem Description (CHGSBSD) command. You can use this command to change the storage pool size, storage pool activity level, and the maximum number of jobs for the subsystem description of an active subsystem.

Related concepts

Communications Management PDF

Manage work

Related reference

Change Subsystem Description (CHGSBSD) command

Change System Value (CHGSYSVAL) command

APPC subsystems

In a distributed relational database using an Systems Network Architecture (SNA) network, communications jobs and interactive jobs are the main types of work an administrator must plan to manage on each server.

Servers in the network start communications jobs to handle requests from an application requester (AR); an AR's communications requests to other servers normally originate from interactive or batch jobs on the local system.

Setting up an efficient work management environment for the distributed relational database network servers can enhance your overall network performance by allocating system resources to the specific needs of each application server (AS) and AR in the network.

When the i5/OS licensed program is first installed, QBASE is the default controlling subsystem. As the controlling subsystem, QBASE allocates system resources between the two subsystems QBASE and QSPL. Interactive jobs, communications jobs, batch jobs, and so on, allocate resources within the QBASE subsystem. Only spooled jobs are managed under a different subsystem, QSPL. This means you have less control of system resources for handling communications jobs versus interactive jobs than you would using the QCTL controlling subsystem.

Using the QCTL subsystem configuration, you have control of four additional subsystems for which the system has allocated storage pools and other system resources. Changing the QCTL subsystems, or creating your own subsystems gives you even more flexibility and control of your processing resources.

Different system requirements for some of the systems in the Spiffy Corporation distributed relational database network might require different work management environments for best network efficiency. The following discussions show how the distributed relational database administrator can plan a work management subsystem to meet the needs of each iSeries server in the Spiffy distributed relational database network.

In the Spiffy Corporation system organization, a small dealership might be satisfied with a QBASE level of control for the various jobs its users have on the server. For example, requests to a small dealership's relational database from the regional AR (to update dealer inventory levels for a shipment) are handled as communications jobs. Requests from a dealership user to the regional AS, to request a part not currently in stock locally, is handled as an interactive job on the dealership server. Both activities are relatively small jobs because the dealership is smaller and handles fewer service orders, parts sales and so on. The coordination of resources in the QBASE subsystem provides the level of control this enterprise requires for their interactive and communications needs.

A large dealership, on the other hand, probably manages its work through the QCTL subsystem, because of the different work loads associated with the different types of jobs.

The number of service orders booked each day can be high, requiring a query to the local relational database for parts or to the regional center AS for parts not in stock at the dealership. This type of activity starts interactive jobs on their system. The dealership also starts a number of interactive jobs that are not distributed relational database related jobs, such as enterprise personnel record keeping, marketing and sales planning and reporting, and so on. Requests to this dealership from the regional center for performance information or to update inventory or work plans are communications jobs that the dealership wants to manage in a separate environment. The large dealership can also receive a request from another dealership for a part that is out of stock at the regional center.

For a large dealership, the QCTL configuration with separate subsystem management for QINTER and QCMN provides more flexibility and control for managing its server work environment. In this example, interactive and communications jobs at the dealership server can be allocated more of the server resources than other types of jobs. Additionally, if communications jobs are typically fewer than interactive jobs for this system, resources can be targeted toward interactive jobs, by changing the subsystem descriptions for both QINTER and QCMN.

A work management environment tailored to a Spiffy Corporation regional center perspective is also important. In the Spiffy network, the regional center is an AR to each dealership when it updates the dealership inventory table with periodic parts shipment data, or updates the service plan table with new or updated service plans for specific repair jobs. Some of these jobs can be run as interactive jobs (on the regional system) in early morning or late afternoon when system usage is typically less, or run as batch jobs (on the regional server) after regular business hours. The administrator can tailor the QINTER and QBATCH subsystems to accommodate specific processing times and resource needs.

The regional center is also an AS for each dealership when a dealership needs to query the regional relational database for a part not in stock at the dealership, a service plan for a specific service job (such as rebuilding a steering rack), or for technical bulletins or recall notifications since the last update to the dealership relational database. These communications jobs can all be managed in QCMN.

However, a closer examination of some specific aspects of distributed relational database network use by the KC000 (Kansas City) regional center and the dealerships it serves suggests other alternatives to the distributed relational database administrator at Kansas City.

The KC000 server serves several very large dealerships that handle hundreds of service orders daily, and a few small dealerships that handle fewer than 20 service orders each day. The remaining medium-sized dealerships each handle about 100 service orders daily. One problem that presents itself to the distributed relational database administrator is how to fairly handle all the communications requests to the KC000

server from other systems. A large dealership could control QCMN resources with its requests so that response times and costs to other systems in the network are unsatisfactory.

The distributed relational database administrator can create additional communications subsystems so each class of dealerships (small, medium, or large) can request support from the AS and generally receive better response. By tailoring the subsystem attributes, prestart job entries, communications work entries, and routing entries for each subsystem description, the administrator controls how many jobs can be active on a subsystem and how jobs are processed in the subsystem.

The administrator can add a routing entry to change the class (and therefore the priority) of a DRDA/DDM job by specifying the class that controls the priority of the job and by specifying QCNTEDDM on the CMPVAL parameter, as in the following example:

```
ADDRTGE SBS(DQCMN) SEQNBR(280) CLS(QINTER) CMPVAL('QCNTEDDM' 37)
```

The administrator can also add a prestarted job for DRDA/DDM job by specifying QCNTEDDM as the prestarted job, as in the following example:

```
ADDPJE SBS(DQCMN) PGM(QCNTEDDM)
```

Related concepts

Communications Management PDF

Manage work

TCP/IP subsystems

By default, the DDM TCP/IP server prestart jobs used for DRDA TCP/IP connections run in the QUSRWRK subsystem. QUSRWRK is the user work subsystem. It contains jobs that are started by servers to do work on behalf of a user. The DRDA listener job that dispatches work to the prestart jobs runs in QSYSWRK.

User databases on independent auxiliary storage pools

The user might create additional relational databases on an iSeries server by configuring independent auxiliary storage pools (IASPs) on the server. Each independent auxiliary storage pool group is a relational database.

In this topic, independent auxiliary storage pool groups are called user databases. They consist of all the database objects that exist on the independent auxiliary storage pool group disks. Additionally, all database objects in the system relational database (called system database in this topic collection) of the iSeries server to which the independent auxiliary storage pool is varied on are logically included in a user relational database. However, from a commitment control perspective, the system database is treated differently.

There are a number of rules associated with the creation and use of user databases, besides those imposed by the commitment control considerations just mentioned. One example is that you cannot use an Advanced Program-to-Program Communication (APPC) protected distributed unit of work (DUW) conversation to connect to a database from an application requester (AR) which has been set to a user database (an auxiliary storage pool [ASP] group) for the current thread. Another example is that the name of any schema created in a user database must not already exist in that user database or in the associated system database. For more information about such restrictions, see the SQL reference topic.

There are certain DRDA-related objects that cannot be contained in user databases. DDM user exit programs must reside in libraries in the system database, as must any Application Requester Driver programs.

You should be aware that the process of varying on a user database causes the RDB directory to be unavailable for a period of time, which can cause attempts by a DRDA application requester or application server (AS) to make use of the directory to be delayed or to timeout. The exposure to having

directory operations timeout due to unavailability caused by varying on a database is much greater if multiple databases are varied on at the same time. As noted here, the first time a user database is varied on, an attempt is made by the server to add a directory entry for that database. If the directory is unavailable due to a concurrent vary on operation, the addition will fail, in which case the entry will have to be manually added.

Other considerations in the use of user databases concern configuration of entries in the RDB directory. One of the rules for naming user databases is that user RDB names cannot match the system name specified in the network attributes (as displayed by the Display Network Attributes (DSPNETA) command).

Local user database entries in the RDB directory are added automatically the first time that the associated databases are varied on. They are created using the *IP protocol type and with the remote location designated as LOOPBACK. LOOPBACK indicates that the database is on the same server as the directory. It is highly recommended that user databases that are intended to be switched among servers be configured to have a dedicated IP address associated with them. If the switchable database does not have a dedicated IP address, then whenever it is switched, manual updating of its directory entry on all the servers that reference that database must be done.

Related concepts

Manage application CRG IP addresses

Troubleshoot transactions and commitment control

“Use the relational database directory”

The i5/OS licensed program uses the relational database directory to define the relational database names that can be accessed by applications running on an iSeries server, to specify if the connection uses Systems Network Architecture (SNA) or IP, and to associate these relational database names with their corresponding network parameters.

Related reference

Display Network Attributes (DSPNETA) command

SQL reference

Use the relational database directory

The i5/OS licensed program uses the relational database directory to define the relational database names that can be accessed by applications running on an iSeries server, to specify if the connection uses Systems Network Architecture (SNA) or IP, and to associate these relational database names with their corresponding network parameters.

The relational database directory allows an application requester (AR) to accept a relational database name from the application and translate this name into the appropriate Internet Protocol (IP) address or host name and port, or the appropriate Systems Network Architecture network identifier and logical unit (LU) name values for communications processing. As of V5R2, the RDB directory is also used to specify the user's preferred outbound connection security mechanism. The relational database directory also allows associating an ARD program with a relational database name.

Each iSeries system in the distributed relational database network must have a relational database directory configured. There is only one relational database directory on a system. Each AR in the distributed relational database network must have an entry in its relational database directory for its local relational database and one for each remote and local user relational database the AR accesses. Any system in the distributed relational database network that acts only as an application server (AS) does not need to include the relational database names of other remote relational databases in its directory.

The relational database name assigned to the local relational database must be unique. That is, it should be different from any other relational database in the network. Names assigned to other relational databases in the directory identify remote relational databases, or local user databases. The names of remote RDBs must match the name an AS uses to identify its local system database or one of its user

databases, if configured. If the local system RDB name entry at an AS does not exist when it is needed, one will be created automatically in the directory. The name used will be the current system name displayed by the Display Network Attributes (DSPNETA) command.

Related reference

Display Network Attributes (DSPNETA) command

Work with the relational database directory

Use these instructions to work with the relational database directory.

Related reference

- Add Relational Database Directory Entry (ADDRDBDIRE) command
- Change Relational Database Directory Entry (CHGRDBDIRE) command
- Display Relational Database Directory Entry (DSRDBDIRE) command
- Remove Relational Database Directory Entry (RMVRDBDIRE) command
- Work with Relational Database Directory Entry (WRKRDBDIRE) command

Add an entry for SNA usage: The Add RDB Directory Entry (ADDRDBDIRE) display is shown here. You can use the prompts in this display or the Add Relational Database Directory Entry (ADDRDBDIRE) command to add an entry to the relational database directory.

```
Add RDB Directory Entry (ADDRDBDIRE)

Type choices, press Enter.

Relational database . . . . . MP311      Name
Relational database alias . . . *NONE
Remote location:
  Name or address . . . . . MP311      Name, *LOCAL, *ARDPGM
  Type . . . . . *SNA                *SNA, *IP
Text . . . . . 'Oak Street Dealership'
```

In this example, an entry is made to add a relational database named MP311 for a server with a remote location name of MP311 to the relational database directory on the local server. For SNA connections, the relational database alias field must be left with its *NONE default value. The remote location name does not have to be defined before a relational database directory entry using it is created. However, the remote location name must be defined before the relational database directory entry is used in an application. The relational database name (RDB) parameter and the remote location name (RMTLOCNAME) parameter are required for the Add Relational Database Directory Entry (ADDRDBDIRE) command. The second element of the RMTLOCNAME parameter defaults to *SNA. The descriptive text (TEXT) parameter is optional. As shown in this example, it is a good idea to make the relational database name the same as the server name or location name specified for this server in your network configuration. This can help you identify a database name and correlate it to a particular server in your distributed relational database network, especially if your network is complex.

To see the other optional parameters on this command, press F10 on the Add RDB Directory Entry (ADDRDBDIRE) display. These optional parameters are shown here.

Add RDB Directory Entry (ADDRDBDIRE)

Type choices, press Enter.

```
Relational database . . . . . MP311
Relational database alias . . . *NONE
Remote location
  Name or address . . . . . MP311
  Type . . . . . *SNA          *SNA, *IP
Text . . . . . 'Oak Street Dealership'

Device:
  APPC device description . . . *LOC          Name, *LOC
Local location . . . . . *LOC          Name, *LOC, *NETATR
Remote network identifier . . . *LOC          Name, *LOC, *NETATR, *NONE
Mode . . . . . *NETATR          Name, *NETATR
Transaction program . . . . . *DRDA       Character value, *DRDA
```

The server provides default *SNA values for the additional Add Relational Database Directory Entry (ADDRDBDIRE) command parameters:

- Device (DEV)
- Local location (LCLLOCNAME)
- Remote network identifier (RMTNETID)
- Mode (MODE)
- Transaction program (TNSPGM)

Notes:

1. For SNA connections, the relational database alias field must be left with its *NONE default value.
2. The transaction program name parameter in the iSeries server is TNSPGM. In SNA, it is TPN.
3. If you use the defaults with Advanced Program-to-Program Communication (APPC), the server determines the device, the local location, and the remote network identifier that will be used. The mode name defined in the network attributes is used and the transaction program name for Distributed Relational Database Architecture (DRDA) support is used. If you use the defaults with Advanced Peer-to-Peer Networking (APPN), the server ignores the device (DEV) parameter, and uses the local location name, remote network identifier, and mode name defined in the network attributes.

You can change any of these default values on the Add Relational Database Directory Entry (ADDRDBDIRE) command. For example, you might have to change the TNSPGM parameter to communicate with an DB2 Universal Database for VM server. By default for DB2 Universal Database for VM support, the TNSPGM is the name of the DB2 Universal Database for VM database to which you want to connect. The default TNSPGM parameter value for DRDA (*DRDA) is X'07F6C4C2'. QCNTEDDM and DB2DRDA also map to X'07F6C4C2'. For more information about transaction program name, see the related topics.

Related tasks

“Set QCNTRVC as a TPN on a DB2 Universal Database for VM application requester” on page 182
Change the UCOMDIR NAMES file to specify QCNTRVC in the TPN tag.

Related reference

Add Relational Database Directory Entry (ADDRDBDIRE) command

“Set QCNTRVC as a TPN on a DB2 Universal Database for iSeries application requester” on page 182

Specify the QCNTRVC on the TNSPGM parameter of the Add Relational Database Directory Entry (ADDRDBDIRE) or Change Relational Database Directory Entry (CHGRDBDIRE) command.

“Set QCNTSRVC as a TPN on a DB2 Universal Database for z/OS application requester” on page 182 Update the SYSIBM.LOCATIONS table to specify QCNTSRVC in the TPN column for the row that contains the RDB-NAME of the DB2 Universal Database for iSeries application server.

“Set QCNTSRVC as a TPN on a DB2 Universal Database for Linux, UNIX and Windows application requester” on page 183

If you are working with DB2 Universal Database for Linux, UNIX and Windows and would like instructions on how to set up the TPN on this family of products, there is a Web page to help you.

Add an entry for TCP/IP usage: The Add RDB Directory Entry (ADDRDBDIRE) display shown here demonstrates how the panel changes if you enter *IP as the second element of the RMTLOCNAME parameter, and what typical entries look like for an RDB that uses TCP/IP. Although usage of the relational database alias field is enabled for connections that use TCP/IP, this first TCP/IP example does not specify an alias.

```
Add RDB Directory Entry (ADDRDBDIRE)

Type choices, press Enter.

Relational database . . . . . > MP311
Relational database alias . . . > *NONE
Remote location:
  Name or address . . . . . > MP311.spiffy.com

Type . . . . . > *IP          *SNA, *IP
Text . . . . . > 'Oak Street Dealership'

Port number or service program      *DRDA
Remote authentication method:
  Preferred method . . . . . > *ENCRYPTED *USRID, *USRIDPWD...
  Allow lower authentication . . > *ALWLOWER *ALWLOWER, *NOALWLOWER
```

Specify a relational database alias name: The next example shows the addition of a directory entry that specifies an RDB alias name. This allows networks that have relational databases of the same name to uniquely identify each in a Distributed Relational Database Architecture (DRDA) environment. When an entry using an alias has been added to the RDB directory, the entry is identified by its alias name. To display or delete the entry, you must specify the alias name.

The following display has RDBALS specified as the relational database alias name.

```
Type choices, press Enter.

Relational database . . . . . > TEST Character value
Relational database alias . . . RDBALS
Remote location:
  Name or address . . . . . MP311.spiffy.com

Type . . . . . *IP          *SNA, *IP
Text . . . . . 'Oak Street Dealership'
```

When you add an entry for an alias using WRKRDBDIRE and option 1, you should first put the real RDB name in the **Entry** field, and press Enter. Then, after filling in the other fields including the alias name in the **Relational database alias** field, you will see the alias name replace the real RDB name in the **Entry** field of the list of RDB entries. Note that you must change **Type** for the remote location name from *SNA to *IP.

When removing a relational database entry with the Remove Relational Database Directory Entry (RMVRDBDIRE) command, the alias name, rather than the real relational database name, is used to specify which entry to remove.

If you identify a remote database by an alias, you cannot also refer to it by its real name in the same directory.

Note that instead of specifying MP311.spiffy.com for the RMTLOCNAME, you could have specified the IP address (for example, '9.5.25.176'). For IP connections to another iSeries server, leave the PORT parameter value set at the default, *DRDA, unless you need to use port 447. For example, you might have port 447 configured for transmission using IP Security Architecture (IPSec). For connections to an IBM Universal Database (UDB) server on some other platform, for example, you might need to set the port to a number such as 50000. Refer to the product documentation for the server you are using. If you have a valid service name defined for a DRDA port at a certain location, you can also use that instead of a number. However, on iSeries, *DRDA is preferred to the use of the 'drda' service name.

Add an entry for an application requester driver: To specify communication information and an application requester driver (ARD) program on the Add Relational Database Directory Entry (ADDRDBDIRE) command prompt, press F9 and page down. When the ARD program will not use the communication information specified on the ADDRDBDIRE command (which is normally the case), use the special value *ARDPGM on the RMTLOCNAME parameter. The ARD program must reside in a library in the system database (ASP numbers 1-32).

Related reference

Add Relational Database Directory Entry (ADDRDBDIRE) command

Use the WRKRDBDIRE command: The Work with RDB Directory Entries display provides you with options to add, change, display, or remove a relational database directory entry.

```
Work with RDB Directory Entries

Position to . . . . .

Type options, press Enter.
1=Add 2=Change 4=Remove 5=Display details 6=Print details

Option  Relational      Remote
        Database       Location Text
---
         KC000          KC000  Kansas City region database
         MP000          *LOCAL Minneapolis region database
         MP101          MP101  Dealer database MP101
         MP102          MP102  Dealer database MP102
         MP211          MP211  Dealer database MP211
         MP215          MP215  Dealer database MP215
4_      MP311          MP311  Dealer database MP311
```

As shown on the display, option 4 can be used to remove an entry from the relational database directory on the local server. If you remove an entry, you receive another display that allows you to confirm the remove request for the specified entry or select a different relational database directory entry. If you use the Remove Relational Database Directory Entry (RMVRDBDIRE) command, you have the option of specifying a specific relational database name, generic names, all directory entries, or just the remote entries.

You have the option on the Work with Relational Database Directory Entries display to display the details of an entry. Output from the Work with Relational Database Entries display is to a display. However, if you use the Display Relational Database Directory Entry (DSPRDBDIRE) command, you can send the output to a printer or an output file. The relational database directory is not an iSeries object, so using an output file provides a means of backup for the relational database directory. For more information about using the (DSPRDBDIRE) command with an output file for backing up the relational database directory, see Save and restore relational database directories.

You have the option on the Work with RDB Directory Entries display to change an entry in the relational database directory. You can also use the Change Relational Database Directory Entry (CHGRDBDIRE) command to make changes to an entry in the directory. You can change any of the optional command parameters and the remote location name of the server. You cannot change a relational database name for a directory entry. To change the name of a relational database in the directory, remove the entry for the relational database and add an entry for the new database name.

Note: If the remote location was changed in the relational database directory entry, then the remote journal has to be removed using the Remove Remote Journal (RMVRMTJRN) command or the QjoRemoveRemoteJournal API and readded using the Add Remote Journal (ADDRMTJRN) command or the QjoAddRemoteJournal API. If the remote location type, or authentication, or something else was changed, then remote journaling just needs to be ended using the Change Remote Journal (CHGRMTJRN) command or the QjoChangeJournalState API and restarted by also using the Change Remote Journal (CHGRMTJRN) command or the QjoChangeJournalState API. To get your change used for distributed files, you need to delete and re-create your node group, and then re-create the file.

Related tasks

“Save and restore relational database directories” on page 137

The relational database directory is not an iSeries object. Instead it is made up of files that are opened by the server at IPL time.

Related reference

Add Remote Journal (ADDRMTJRN) command

Change Relational Database Directory Entry (CHGRDBDIRE) command

Change Remote Journal (CHGRMTJRN) command

Display Relational Database Directory Entry (DSPRDBDIRE) command

Remove Remote Journal (RMVRMTJRN) command

Remove Relational Database Directory Entry (RMVRDBDIRE) command

The *LOCAL directory entry: The directory entry containing *LOCAL is unique in that there is only one such entry in the directory, and it specifies the name of the local system database. The associated RDB name can be used in the SQL statement CONNECT TO x (where x is the local server name) to connect to the local database. The effect of CONNECT TO x (where x is the local server name) is equivalent to using the SQL statement CONNECT RESET.

If you want to make a DRDA connection to the local server database, such as for program testing, there are two special RDB names that can be used for that purpose: ME and MYSELF. An example usage would be a programmer adding a directory entry with an RDB name of ME, with type of *IP, and with Remote Location name of LOOPBACK. He could then, in a program, do an SQL CONNECT TO ME and establish a sockets DRDA connection to the local system. However, general use of these RDB names is discouraged and they are documented only to warn that unexpected behavior can result from their use in some situations.

However, if you must change the name of the local RDB entry, the procedure includes doing the remove and add operation. But there are special considerations for removing the local entry, because that entry contains some system-wide DRDA attribute information. If you try to remove the entry, you will get message CPA3E01 (Removing or changing *LOCAL directory entry might cause loss of configuration data (C G)), and you will be given the opportunity to cancel (C) the operation or continue (G). The message text goes on to tell you that the entry is used to store configuration data entered with the Change DDM TCP/IP Attributes (CHGDDMTCPA) command. If the *LOCAL entry is removed, configuration data might be destroyed, and the default configuration values will be in effect. If the default values are not satisfactory, configuration data will have to be re-entered with the CHGDDMTCPA command. Before removing the entry, you might want to record the values specified in the CHGDDMTCPA command so that they can be restored after the *LOCAL entry is deleted and added with the correct local RDB name.

Related reference

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Directory entries for user databases on independent auxiliary storage pools: For a server with only one database (that is, without independent auxiliary storage pools (IASPs) configured), the *LOCAL entry refers to the single local database. For servers with multiple databases (one system database and one or more user databases), the *LOCAL entry refers to the system database.

The local user databases are represented by entries similar to remote *IP entries. The main difference is the Remote Location field. In cases where the database cannot be switched to a different server, this field will normally contain the word LOOPBACK. LOOPBACK represents the IP address of the host server. If the database can be switched, it is recommended that the user configure the server in such a way that a specific IP address is associated with the database regardless of the server to which it is attached. For an explanation on how dedicated IP address configuration is done, see the Manage application CRG IP addresses topic. In that case, the IP address would be used in the Remote Location field.

If LOOPBACK is used for a switchable database, then whenever it is switched from the local server, the user will have to manually change the directory entry to replace LOOPBACK with the IP address of the new server to which it is attached, and then change it back to LOOPBACK when the database is switched back.

Related reference

Manage application CRG IP addresses

Example: Set up a relational database directory

The Spiffy Corporation network provides an example to illustrate how the relational database directory is used on servers in a distributed relational database network and show how each is set up. The example assumes the use of Advanced Program-to-Program Communication (APPC) for communications, as opposed to TCP/IP, which would be simpler to set up.

However, some elements of the example are protocol-independent. The RDB directory entries needed for APPC use are also needed in a TCP/IP network, but the parameters differ. Host names or IP addresses and port identifications would replace logical unit (LU) names, device descriptions, modes, TPNs, and so forth.

A simple relationship to consider is the one between two regional offices as shown in the following figure:



Figure 6. Relational database directory set up for two servers

The relational database directory for each regional office must contain an entry for the local relational database and an entry for the remote relational database because each server is both an application requester (AR) and an application server (AS). The commands to create the relational database directory for the MP000 server are:

```
ADDRDBDIRE    RDB(MP000) RMTLOCNAME(*LOCAL) TEXT('Minneapolis region database')
ADDRDBDIRE    RDB(KC000) RMTLOCNAME(KC000) TEXT('Kansas City region database')
```

In the preceding example, the MP000 server identifies itself as the local relational database by specifying *LOCAL for the RMTLOCNAME parameter. There is only one relational database on an iSeries server. You can simplify identification of your network relational databases by making the relational database names in the RDB directory the same as the server name. The entry for the local location can have the same name as the local server name, and the entry for the remote location name can have the same name as the remote server name.

Note: The server name is specified on the SYSNAME parameter of the Change Network Attributes (CHGNETA) command. The local server is identified on the LCLLOCNAME parameter of the CHGNETA command during communications configuration. Remote locations using SNA (APPC) are identified with the RMTCPNAME parameter on the Create Controller Description (APPC) (CRTCTLAPPC) command during communications configuration. Using the same names for server names, network locations, and database names can help avoid confusion, particularly in complex networks.

The corresponding entries for the KC000 server relational database directory are:

```
ADDRDBDIRE    RDB(KC000) RMTLOCNAME(*LOCAL) TEXT('Kansas City region database')
ADDRDBDIRE    RDB(MP000) RMTLOCNAME(MP000) TEXT('Minneapolis region database')
```

A more complex example to consider is that of a regional office to its dealerships. For example, to access relational databases in the network shown in the following figure, the relational database directory for MP000 server must be expanded to include an entry for each of its dealerships.

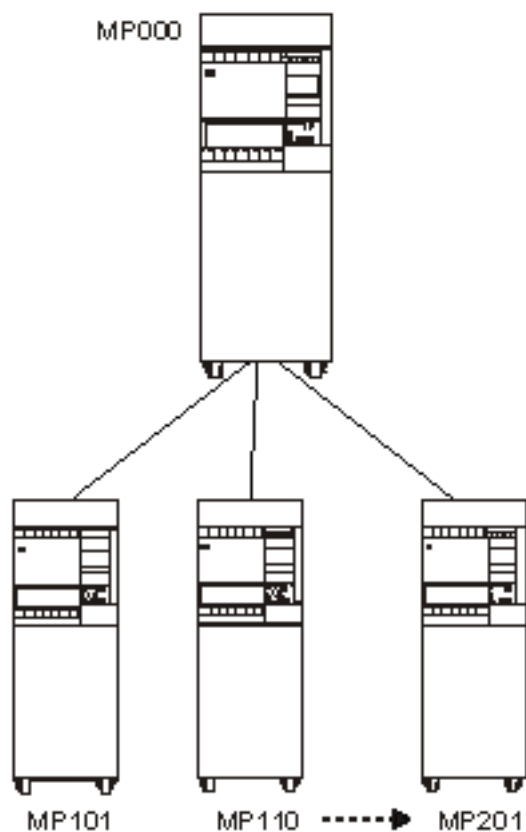


Figure 7. Relational database directory setup for multiple servers

A sample of the commands used to complete the MP000 relational database directory to include all its dealer databases is as follows:

```
PGM
ADDRDBDIRE   RDB(MP000) RMTLOCNAME(*LOCAL) +
TEXT('Minneapolis region database')
ADDRDBDIRE   RDB(KC000) RMTLOCNAME(KC000)
TEXT('Kansas City region database')
ADDRDBDIRE   RDB(MP101) RMTLOCNAME(MP101)
TEXT('Dealer database MP101')
ADDRDBDIRE   RDB(MP002) RMTLOCNAME(MP110)
TEXT('Dealer database MP110')
.
.
.
ADDRDBDIRE   RDB(MP215) RMTLOCNAME(MP201)
TEXT('Dealer database MP201')
ENDPGM
```

In the preceding example, each of the region dealerships is included in the Minneapolis relational database directory as a remote relational database.

Because each dealership can serve as an AR to MP000 and to other dealership application servers, each dealership must have a relational database directory that has an entry for itself as the local relational database and the regional office and all other dealers as remote relational databases. The database administrator has several options to create a relational database directory at each dealership server.

The most time-consuming and error-prone method is to create a relational database directory at each server by using the Add Relational Database Directory Entry (ADDRDBDIRE) command to create each directory entry on all servers that are part of the MP000 distributed relational database network.

A better alternative is to create a control language (CL) program like the one shown in the preceding example for the MP000. The distributed relational database administrator can copy this CL program for each of the dealership servers. To customize this program for each dealership, the database administrator changes the remote location name of the MP000 server to MP000, and changes the remote location name of the local dealership to *LOCAL. The distributed relational database administrator can distribute the customized CL program to each dealership to be run on that server to build its unique relational database directory.

A third method is to write a program that reads the relational database directory information sent to an output file as a result of using the Display Relational Database Directory Entry (DSPRDBDIRE) command. This program can be distributed to the dealerships, along with the output file containing the relational database directory entries for the MP000 server. Each server could read the MP000 output file to create a local relational database directory. The Change Relational Database Directory Entry (CHGRDBDIRE) command can then be used to customize the MP000 server directory for the local server.

Related tasks

“Save and restore relational database directories” on page 137

The relational database directory is not an iSeries object. Instead it is made up of files that are opened by the server at IPL time.

Related reference

Add Relational Database Directory Entry (ADDRDBDIRE) command

Change Relational Database Directory Entry (CHGRDBDIRE) command

Create Controller Description (APPC) (CRTCTLAPPC) command

Display Network Attributes (DSPNETA) command

Display Relational Database Directory Entry (DSPRDBDIRE) command

Set up DRDA security

Distributed Relational Database Architecture (DRDA) security is covered in the Security topic, but for the sake of completeness, it is mentioned here as a consideration before using DRDA, or in converting your network from the use of Advanced Program-to-Program Communication (APPC) to TCP/IP.

Security setup for TCP/IP is quite different from what is required for APPC. One thing to be aware of is the lack of the secure location concept that APPC has. Because a TCP/IP server cannot fully trust that a client server is who it says it is, the use of passwords on connection requests is more important. To make it easier to send passwords on connection requests, the use of server authorization lists associated with specific user profiles has been introduced with TCP/IP support. Entries in server authorization lists can be maintained by use of the xxxSVRAUTHE commands (where xxx represents ADD, CHG, and RMV) described in “Security” on page 50. An alternative to the use of server authorization entries is to use the USER/USING form of the SQL CONNECT statement to send passwords on connection requests.

Kerberos support provides another security option if you are using TCP/IP. Network authentication service supports Kerberos protocols and can be used to configure for Kerberos.

Setup at the server side includes deciding and specifying what level of security is required for inbound connection requests. For example, should unencrypted passwords be accepted? The default setting is that they are. The default setting can be changed by use of the Change DDM TCP/IP Attributes (CHGDDMTCPA) command.

Related concepts

Configure network authentication service

Control language

Related reference

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Set up the TCP/IP server for DRDA

If you own a Distributed Relational Database Architecture (DRDA) application server (AS) that will be using the TCP/IP protocol, you need to set up the DDM TCP/IP server.

Setting up the TCP/IP server can be as simple as insuring that it is started when it is needed, which can be done by running the following command if you want it to remain active at all times:

```
CHGDDMTCPA AUTOSTART(*YES)
```

But there are other parameters that you might want to adjust to tune the server for your environment. These include the initial number of prestart jobs to start, the maximum number of jobs, threshold when to start more, and so forth.

You might want to set up a common user profile for all clients to use when connecting, or set up a set of different user profiles with different levels of security for different classes of remote users. You can then use the Add Server Authentication Entry (ADDSVRAUTE) command at the application requester (AR) to map each user's profile name at the AR to what user profile they will run under at the AS.

Related concepts

“Application requester security in a TCP/IP network” on page 57

Different connectivity scenarios call for using different levels of authentication. Therefore, an administrator can set the lowest security authentication method required by the application requester (AR) when connecting to an application server (AS) by setting the preferred authentication method field in each RDB directory entry.

“Manage the TCP/IP server” on page 114

This topic collection describes how to manage the DRDA/DDM server jobs that communicate using sockets over TCP. It describes the subsystem in which the server runs, the objects that affect the server and how to manage those resources.

Related reference

Add Server Authentication Entry (ADDSVRAUTE) command

Set up SQL packages for Interactive SQL

This topic applies only to non-iSeries Application Servers (ASs).

If either of the following items is true, then you need to ensure that SQL packages are created at the servers:

- If you have the DB2 UDB Query Manager and SQL Development Kit and plan to use the Interactive SQL (STRSQL) function of that product
- If you plan to connect to non-iSeries DRDA servers that use TCP/IP from a pre-V5R1 iSeries client, or to ones that do not have two-phase commit capability

STRSQL does not require SQL packages for iSeries servers. Normally, SQL packages are created automatically at a non-iSeries application server (AS) for users of STRSQL. However, a problem can occur because the initial connection for STRSQL is to the local server, and that connection is protected by two-phase commit protocols. If a subsequent connection is made to a server that is only one-phase commit capable, or if TCP/IP is used from a pre-V5R1 iSeries client, then that connection is read-only. When an attempt is made to automatically create a package over such a connection, it fails because the creation of a package is considered an update, and cannot be done over a read-only connection.

The solution to this is to get rid of the connection to the local database before connecting to the remote AS. This can be done by doing a RELEASE ALL command followed by a COMMIT. Then the connection to the remote server can be made and because it is the first connection, updates can be made over it.

When you start Interactive SQL, you must specify a commitment control level of something other than *NONE. Also, the user ID that you use to connect with must have the proper authority to create an SQL package on the application server. If you receive an SQLSTATE of 42501 on the connection attempt, you might not have package creation authority.

Related reference

“Connection failures specific to Interactive SQL” on page 162

Sometimes when you are running a CONNECT statement from Interactive SQL, a general SQ30080 message is given.

Set up DDM files

The implementation of DRDA support on the iSeries server uses Distributed Data Management (DDM) conversations for communications. Because of this, you can use DDM in conjunction with distributed relational database processing.

You can use DDM to submit remote commands to an application server (AS), copy tables from one iSeries server to another, and process nondistributed relational database work on another server.

With distributed relational database, information the application requester (AR) needs to connect to a database is provided in the relational database directory. When you use DDM, you must create a separate DDM file for each file you want to work with on the application server (AS). The DDM file is used by the application on the application requester (AR) to identify a remote file on the application server (AS) and the communications path to the application server (AS).

As of V5R2, you can also create DDM files with a reference to an RDB directory entry. Some database administration tasks discussed in Manage a distributed relational database use DDM to access remote files. A DDM file is created using the Create Distributed Data Management File (CRTDDMF) command. You can create a DDM file before the file and communication path named in the file have been created. However, the file named in the DDM file and the communications information must be created before the DDM file is used by an application.

The following example shows one way a DDM file can be created:

```
CRTDDMF FILE (TEST/KC105TST) RMTLOCNAME(KC105)
      RMTFILE(SPIFFY/INVENT)
```

If the DDM file access in the example is to be over TCP/IP, you must specify *IP in the second element of the RMTLOCNAME parameter.

This command creates a DDM file named KC105TST and stores it in the TEST library on the application requester (AR). This DDM file uses the remote location KC105 to access a remote file named INVENT stored in the SPIFFY library on the target iSeries server.

You can use options on the Work with DDM Files display to change, delete, display or create DDM files.

Related concepts

“Administration” on page 99

As an administrator for a distributed relational database, you are responsible for work being done on several servers.

Distributed data management

“Operate remote iSeries servers” on page 106

As an administrator in a distributed relational database you might have to operate a remote iSeries server.

Related reference

Create Distributed Data Management File (CRTDDMF) command

Load data into tables in a distributed relational database

Applications in the distributed relational database environment operate on data stored in tables. In general, applications are used to query a table for information, to insert, update, or delete rows of a table or tables, or to create a new table. Other situations occur where data on one server must be moved to another server.

Load new data into the tables of a distributed relational database

You load data into a table by entering each data item into the table. On the iSeries server, you can use SQL, the DB2 Universal Database for iSeries Query Management function, or the data file utility portion of iSeries Application Development Tools to create applications that insert data into a table.

Load data into a table using SQL:

A simple method of loading data into a table is to use an SQL application and the SQL INSERT operation.

Consider a situation in which a Spiffy regional center needs to add inventory items to a dealership's inventory table on a periodic basis as regular inventory shipments are made from the regional center to the dealership.

```
INSERT INTO SPIFFY.INVENT
(PART, DESC, QTY, PRICE)
VALUES
('1234567', 'LUG NUT', 25, 1.15 )
```

The preceding statement inserts one row of data into a table called INVENT in an SQL collection named SPIFFY.

For each item on the regular shipment, an SQL INSERT statement places a row in the inventory table for the dealership. In the preceding example, if 15 different items were shipped to the dealership, the application at the regional office could include 15 SQL INSERT statements or a single SQL INSERT statement using host variables.

In this example, the regional center is using an SQL application to load data into a table at an application server (AS). Runtime support for SQL is provided in the i5/OS licensed program, so the AS does not need the IBM DB2 Query Manager and SQL Development Kit for iSeries licensed program. However, the IBM DB2 Query Manager and SQL Development Kit for iSeries licensed program is required to write the application.

Related concepts

SQL programming

Related reference

SQL reference

Manipulate data in tables and files using the iSeries query management function:

The i5/OS licensed program provides a DB2 Universal Database for iSeries query management function that allows you to manipulate data in tables and files. A query is created using an SQL query statement.

You can run the query through CL commands or through a query callable interface in your application program. Using the query management function, you can insert a row of data into a table for the inventory updates described in the "Load data into a table using SQL" as follows.

Create a source member INVLOAD in the source physical file INVLOAD and the SQL statement:

```
INSERT INTO SPIFFY/INVENT
  (PART, DESC, QTY, PRICE)
VALUES
  (&PARTVALUE, &DESCVALUE, &QTYVALUE, &PRICEVALUE)
```

Use a CL command to create a query management query object:

```
CRTQMORY QMORY(INVLOAD) SRCFILE(INVLOAD) SRCMBR(INVLOAD)
```

The following CL command places the INSERT SQL statement results into the INVENT table in the SPIFFY collection. Use of variables in the query (&PARTVALUE, &DESCVALUE, and so on) allows you to enter the desired values as part of the STRQMORY call, rather than requiring that you create the query management query again for each row.

```
STRQMORY QMORY(INVLOAD) RDB(KC000)
  SETVAR((PARTVALUE '1134567') (DESCVALUE '''Lug Nut''')
  (QTYVALUE 25) (PRICEVALUE 1.15))
```

The query management function is dynamic, which means its access paths are built at run time instead of when a program is compiled. For this reason the DB2 UDB for iSeries query management function is not as efficient for loading data into a table as an SQL application. However, you need the IBM DB2 Query Manager and SQL Development Kit for iSeries product to write an application; runtime support for SQL and query management is part of the i5/OS licensed program.

Related concepts

Query Management Programming PDF

Enter data, update tables, and make inquiries using data file utility:

The data file utility (DFU), which is part of the iSeries Applications Development Tools package available from IBM, is a program builder that helps you create programs to enter data, update tables, and make inquiries.

You do not need a programming language to use DFU. Your data entry, maintenance, or inquiry program is created when you respond to a series of displays. An advantage in using DFU is that its generic nature allows you to create a database update program to load data to a table faster than you could by using programming languages such as SQL. You can work with data on a remote server using DFU with DDM files, or by using display station pass-through to run DFU at the application server (AS).

Related concepts

ADTS/400: Data File Utility PDF

Move data from one iSeries server to another

A number of situations occur in enterprise operations that might require moving data from one iSeries server to another.

An example for a situation that might require moving data from an iSeries server to another server would be a new dealership opens in a region, and some clients from one or two other dealerships might be transferred to the new dealership as determined by client address. Perhaps a dealership closed or no longer represents Spiffy Corporation sales and service. That dealer's inventories and required service information must be allocated to either the regional office or other area dealerships. Perhaps a dealership has grown to the extent that it needs to upgrade its server, and the entire database must be moved to the new server.

Some alternatives for moving data from one iSeries server to another are:

- User-written application programs
- Interactive SQL (ISQL)
- DB2 Universal Database for iSeries Query Management functions
- Copy to and from tape devices

- Copy file commands with DDM
- The network file commands
- iSeries server save and restore commands

Create a user-written application program:

A program compiled with distributed unit of work (DUW) connection management can connect to a remote database and a local database and FETCH from one to INSERT into the other to move the data.

By using multirow FETCH and multirow INSERT, blocks of records can be processed at one time. Commitment control can be used to allow checkpoints to be performed at points during the movement of the data to avoid having to completely start the copy again in case of a failure.

Query a database using Interactive SQL:

Using the SQL SELECT statement and Interactive SQL, you can query a database on another iSeries server for data you need to create or update a table on the local server.

The SELECT statement allows you to specify the table name and columns containing the desired data, and selection criteria or filters that determine which rows of data are retrieved. If the SELECT statement is successful, the result is one or more rows of the specified table.

In addition to getting data from one table, SQL allows you to get information from columns contained in two or more tables in the same database by using a join operation. If the SELECT statement is successful, the result is one or more rows of the specified tables. The data values in the columns of the rows returned represent a composite of the data values contained in specified tables.

Using an Interactive SQL query, the results of a query can be placed in a database file on the local server. If a commitment control level is specified for the Interactive SQL process, it applies to the application server (AS); the database file on the local server is under a commitment control level of *NONE.

Interactive SQL allows you to do the following things:

- Create a new file for the results of a select.
- Replace an existing file.
- Create a new member in a file.
- Replace a member.
- Append the results to an existing member.

Consider the situation in which the KC105 dealership is transferring its entire stock of part number 1234567 to KC110. KC110 queries the KC105 database for the part they acquire from KC105. The result of this inventory query is returned to a database file that already exists on the KC110 server. This is the process you can use to complete this task:

Use the Start SQL (STRSQL) command to get the Interactive SQL display. Before you enter any SQL statement (other than a CONNECT) for the new database, specify that the results of this operation are sent to a database file on the local server by doing the following steps:

1. Select the Services option from the Enter SQL Statements display.
2. Select the Change Session Attributes option from the Services display.
3. Enter the Select Output Device option from the Session Attributes Display.
4. Type a 3 for a database file in the Output device field and press Enter. The following display is shown:

already exist in a collection (it can exist in a library). Also, query management uses SQL CREATE TABLE statements to provide data definition when a new table is created on the server as a result of the query. Tables created from a query management function follow the same guidelines and restrictions that apply to a table created using SQL.

However, the query management function does not allow you to specify a member when you want to add the results to a file or table. The results of a query function are placed in the first file member unless you use the Override with Database File (OVRDBF) command to specify a different member before starting the query management function.

Related concepts

Query Management Programming PDF

Related reference

Override with Database File (OVRDBF) command

Copy files to and from tape:

There are several different commands that you can use to copy files to and from tape.

You can copy a table or file to tape using the Copy to Tape (CPYTOTAP) command on the iSeries server.

Data on tape can be loaded on another server using the Copy from Tape (CPYFRMTAP) command. For more information about using the command, see the Storage solutions topic.

You can also use the Copy File (CPYF) command to load data on tape into DB2 Universal Database for iSeries. This is especially useful when loading data that was unloaded from DB2 Universal Database for z/OS, or DB2 Universal Database Server for VM (SQL/DS™). Nullable data can be unloaded from these servers in such a way that a single-byte flag can be associated with each nullable field. CPYF with the *NULLFLAGS option specified for the FMTOPT parameter can recognize the null flags and ignore the data in the adjacent field on the tape and make the field null in DB2 UDB for iSeries. Another useful FMTOPT parameter value for importing data from IBM mainframes is the *CVTFLOAT value. It allows floating point data stored on tape in S/390 format to be converted to the IEEE format used by DB2 UDB for iSeries.

Related concepts

Storage solutions

Related reference

Copy To Tape (CPYTOTAP) command

Copy From Tape (CPYFRMTAP) command

Copy File (CPYF) command

Move data between iSeries servers using copy file commands:

Another way to move data from one iSeries server to another is to copy the data using the copy file commands with DDM.

You can use the Copy File (CPYF), Copy Source File (CPYSRCF), and Copy From Query File (CPYFRMQRYF) commands to copy data between files on source and application servers (ASs). You can copy local relational database or device files from (or to) remote database files, and remote files can also be copied to remote files.

For example, if a dealership closes, the distributed relational database administrator can copy the client and inventory tables from the remote server to the local regional server. The administrator needs a properly authorized user profile on the application server (AS) to access and copy the tables and must create a DDM file on the application requester (AR) for each table or file that is copied. The following example shows the command the database administrator would use to copy a table called INVENT in a

collection called SPIFFY from a server with a remote location name of KC105 to a regional center server called KC000. A DDM file called INCOPY in a library called TEST on the application requester (AR) KC000 is used for the file access. These commands are run on the KC000 server:

```
CRTDDMF FILE(TEST/INCOPY) RMTFILE(SPIFFY/INVENT)
        RMTLOCNAME(KC105)
CPYF FROMFILE(TEST/INCOPY) TOFILE(TEST/INVENTDDM)
      MBROPT(*ADD)
```

In this example, the administrator runs the commands on the KC000 server. If the administrator is not on the KC000 server, then pass-through must be used to run these commands on the KC000 server. The Submit Remote Command (SBMRMTCMD) command cannot be used to run the above commands because the iSeries server cannot be an application requester (AR) and an application server (AS) for the same job.

Consider the following items when using this command with DDM:

- A DDM file can be specified on the FROMFILE and the TOFILE parameters for the Copy File (CPYF) command and Copy Source File (CPYSRCF) commands.

Note: For the Copy From Query File (CPYFRMQRYF) and Copy from Tape (CPYFRMTAP) commands, a DDM file name can be specified only on the TOFILE parameter; for the Copy to Tape (CPYTOTAP) command, a DDM file name can be specified only on the FROMFILE parameter.

- When a delete-capable file is copied to a non-delete capable file, you must specify COMPRESS(*YES), or an error message is sent and the job ends.
- If the remote file name on a DDM file specifies a member name, the member name specified for that file on the Copy File (CPYF) command must be the same as the member name on the remote file name on the DDM file. In addition, the Override with Database File (OVRDBF) command cannot specify a member name that is different from the member name on the remote file name on the DDM file.
- If a DDM file does not specify a member name and if the Override with Database File (OVRDBF) command specifies a member name for the file, the Copy File (CPYF) command uses the member name specified on the OVRDBF command.
- If the TOFILE parameter is a DDM file that refers to a file that does not exist, CPYF creates the file. Keep the following special considerations for remote files created with the Copy File (CPYF) command in mind:
 - The user profile for the target DDM job must be authorized to the Create Physical File (CRTPF) command on the application server (AS).
 - For an iSeries target, the TOFILE parameter has all the attributes of the FROMFILE parameter except those described in the File management topic.
- When using TCP/IP, the second element of the RMTLOCNAME parameter of the Create Distributed Data Management File (CRTDDMF) command must be *IP.

Related concepts

Distributed data management

Related reference

Copy File (CPYF) command

Copy Source File (CPYSRCF) command

Copy From Query File (CPYFRMQRYF) command

Copy from Tape (CPYFRMTAP) command

Copy To Tape (CPYTOTAP) command

Create Physical File (CRTPF) command

Create Distributed Data Management File (CRTDDMF) command

Override with Database File (OVRDBF) command

Submit Remote Command (SBMRMTCMD) command

Transfer data over network using Network File commands:

Data can be transferred over networks protocols that support Systems Network Architecture (SNA) distribution services (SNADS). In addition to APPC and APPN protocols used with distributed relational database processing, SNADS can be used with binary synchronous equivalence link (BSCCEL) and SNA Upline Facility (SNUF) protocols.

An iSeries server supported by SNADS can send data to another server with the Send Network File (SNDNETF) command and receive a network file from another server with the Receive Network File (RCVNETF) and Work with Network Files (WRKNETF) commands.

Related reference

Receive Network File (RCVNETF) command

Send Network File (SNDNETF) command

Work with Network File (WRKNETF) command

Move a table using server save and restore commands:

You can move a table from another iSeries server using the Save Object (SAVOBJ) and Restore Object (RSTOBJ) commands. The save commands save database files on tape or in a save file. The save file can be distributed to another server through communications.

The save and restore commands used to save and restore tables or files include:

- Save Library (SAVLIB) command saves one or more collections or libraries
- Save Object (SAVOBJ) command saves one or more objects (including database tables and views)
- Save Changed Object (SAVCHGOBJ) command saves any objects that have changed since either the last time the collection or library was saved or from a specified date
- Restore Library (RSTLIB) command restores a collection or library
- Restore Object (RSTOBJ) command restores one or more objects (including database tables and views)

For example, if two dealerships were merging, the save and restore commands could be used to save collections and tables for one relational database, which are then restored on the remaining server's relational database. To accomplish this an administrator would:

1. Use the Save Library (SAVLIB) command on server A to save a collection or use the Save Object (SAVOBJ) command on server A to save a table.
2. Specify whether the data is saved to a save file, which can be distributed using SNADS, or saved on tape.
3. Distribute the save file to server B or send the tape to server B.
4. Use the Restore Library (RSTLIB) command on server B to restore a collection or use the Restore Object (RSTOBJ) command on server B to restore a table.

A consideration when using the save and restore commands is the ownership and authorizations to the restored object. A valid user profile for the current object owner should exist on the server where the object is restored. If the current owner's profile does not exist on this server, the object is restored under the QDFTOWN default user profile. User authorizations to the object are limited by the default user profile parameters. A user with QSECOFR authority must either create the original owner's profile on this server and make changes to the restored object ownership, or specify new authorizations to this object for both local and remote users.

Related concepts

Backup and recovery

Related reference

Restore Library (RSTLIB) command

Restore Object (RSTOBJ) command

Save Changed Object (SAVCHGOBJ) command

Save Library (SAVLIB) command

Save Object (SAVOBJ) command

Move a database to an iSeries server from a non-iSeries server

You might need to move a file from another IBM server to an iSeries server or from a non-IBM server to the iSeries server. This topic collection lists alternatives for moving data to an iSeries server from a non-iSeries server. However, you must refer to manuals supplied with the other server or identified for the application for specific instructions on its use.

Move data from another IBM server:

There are a number of methods you can use to move data from another IBM server to an iSeries server.

You can use the methods listed here to move data:

- A high-level language program can be written to extract data from another server. A corresponding program for the server can be used to load data to the server.
- For servers supporting other Distributed Relational Database Architecture (DRDA) implementations, you can use SQL functions to move data. For example, with distributed unit of work, you can open a query against the source of the data and, in the same unit of work, insert the data into a table on the server. For best performance, blocking should be used in the query and a multirow insert should be done at the server.
- Data can be extracted from tables and files on the other server and sent to the iSeries server on tape or over communications lines.
 - From a DB2 Universal Database for z/OS database, a sample program called DSNTIAUL, supplied with the database manager, can be used to extract data from file or tables.
 - From a DB2 Universal Database Server for VM (SQL/DS) database, the Database Services Utility portion of the database manager can be used to extract data.
 - From both DB2 Universal Database for z/OS and DB2 Universal Database Server for VM databases, Data Extract (DXT™) can be used to extract data. However, DXT handling of null data is not compatible with the Copy File handling of null data described below. Therefore, DXT is not recommended for use in unloading relational data for migration to an iSeries server.
 - From IMS/DB hierarchical databases, DXT can be used to extract data.
- You can use standard tape management techniques to copy data to tape from DB2 Universal Database for z/OS or DB2 Universal Database Server for VM databases. The iSeries server uses the Copy from Tape (CPYFRMTAP) command to load data from tape. The Copy File (CPYF) command, however, provides special support for migrating data from IBM mainframe computers. CPYF can be used with tape data by the use of the Override with Tape File (OVRTAPF) command. The OVRTAPF command lets you specify special tape-specific parameters which might be necessary when you import data from a server other than the iSeries server.

The special CPYF support lets you import nullable data and floating point data. Nullable data can be unloaded from mainframes in such a way that a single-byte flag can be associated with each nullable field. With the *NULLFLAGS option specified for the FMTOPT parameter, the Copy File (CPYF) command can recognize the null flags and ignore the data in the adjacent field on the tape and make the field null in DB2 Universal Database for iSeries. The other useful FMTOPT parameter value for importing data from IBM mainframes is the *CVTFLOAT value. It allows floating point data stored on tape in S/390 format to be converted to the IEEE format used by DB2 UDB for iSeries.


For more information on using tape devices with the iSeries server, see the Storage solutions topic.

- Data sent over communications lines can be handled through SNADS support on the iSeries server. SNADS support transfers network files for BSCCEL and SNUF protocols in addition to the Advanced Program-to-Program Communication (APPC) or Advanced Peer-to-Peer Networking (APPN) protocols used for distributed relational database processing.

- From an MVS™ system, data can be sent to the iSeries server using TSO XMIT functions. The server uses the Work with Network Files (WRKNETF) or Receive Network File (RCVNETF) command to receive a network file.
- From a VM system, data can be sent to the server using SENDFILE functions. The server uses the Work with Network Files (WRKNETF) or Receive Network File (RCVNETF) command to receive a network file.
- From Microsoft® Windows, client data can be sent to the iSeries server using iSeries Access, a separately ordered IBM product.
- From a variety of workstation clients, you can use the DB2 Universal Database for Linux, UNIX and Windows IMPORT and EXPORT utilities to copy data to and from an iSeries server. The IMPORT can import data only into existing tables. See the Advanced Functions and Administration on DB2

Universal Database for iSeries, SG24-4249-03  redbook for examples of the IMPORT and EXPORT utilities. This redbook also provides information about what file types and data formats can be used with the IMPORT and EXPORT utilities.

- Data can also be sent over communications lines that do not support SNADS, such as asynchronous communications. File transfer support (FTS), a utility that is part of the i5/OS licensed program, can be used to send and receive data. For more information about working with communications and

communications files, see ICF Programming  .

Related concepts

Distributed data management

Storage solutions

Related reference

Copy From Tape (CPYFRMTAP) command

Copy File (CPYF) command

Override with Database File (OVRDBF) command

Receive Network File (RCVNETF) command

“Tips: Design distributed relational database applications” on page 19

Distributed relational database applications have different requirements from applications developed solely for use on a local database.

Work with Network File (WRKNETF) command

Move data from a non-IBM server:

You can copy files or tables to tape from the other server and load these files on an iSeries server.

Use the Copy From Import File (CPYFRMIMPF) command to do this.

Vendor independent communications functions are also supported through two separately licensed iSeries programs.

Peer-to-peer connectivity functions for both local and wide area networks are provided by the Transmission Control Protocol/Internet Protocol (TCP/IP). The File Transfer Protocol (FTP) function of the iSeries TCP/IP Connectivity Utilities/400 licensed program allows you to receive many types of files, depending on the capabilities of the remote server.

The OSI File Services/400 (OSIFS/400) licensed program provides file management and transfer services for open servers interconnection (OSI) networks. OSIFS/400, with the prerequisite licensed program OSI Communications Subsystem/400, connects the iSeries server to remote IBM or non-IBM servers that conform to OSI file transfer, access, and management (FTAM) standards.

OSIFS/400 provides either an interactive interface or an application programming interface (API) to copy or move files from a remote server to a local iSeries server. For more information, see the *OSI Communications Subsystem Programming and Concepts Guide*.

Related concepts

TCP/IP setup

Related reference

Copy From Import File (CPYFRMIMPF) command

Security

The iSeries server has security elements built into the operating system to limit access to the data resources of an application server. Security options range from simple physical security to full password security coupled with authorization to commands and data objects.

Users must be properly authorized to have access to the database whether it is local or remote. They must also have proper authorization to collections, tables, and other relational database objects necessary to run their application programs. This typically means that distributed database users must have valid user profiles for the databases they use throughout the network. Security planning must consider user and application program needs across the network.

A distributed relational database administrator is faced with two security issues to resolve:

- System to system protection
- Identification of users at remote sites

When two or more systems are set up to access each other's databases, it is important to make sure that the other side of the communications line is the intended location and not an intruder. For DRDA access to a remote relational database, the iSeries server use of Advanced Program-to-Program Communication (APPC) and Advanced Peer-to-Peer Networking (APPN) communications configuration capabilities provides options for you to do this network level security.

The second concern for the distributed relational database administrator is that data security is maintained by the system that stores the data. In a distributed relational database, the user has to be properly authorized to have access to the database (according to the security level of the system) whether the database is local or remote. Distributed relational database network users must be properly identified with a user ID on the application server (AS) for any jobs they run on the AS. Distributed Relational Database Architecture (DRDA) support using both APPC/APPN and TCP/IP communications protocols provides for the sending of user IDs and passwords along with connection requests.

This topic collection discusses security topics that are related to communications and DRDA access to remote relational databases. It discusses the significant differences between conversation-level security in an APPC network connection and the corresponding level of security for a TCP/IP connection initiated by a DRDA application. In remaining security discussions, the term *user* also includes remote users starting communications jobs.

Related reference

"Security considerations for a distributed relational database" on page 22

Part of planning for a distributed relational database involves the decisions you must make about securing distributed data.

Elements of distributed relational database security

A distributed relational database administrator needs to protect the resources of the application servers in the network without unnecessarily restricting access to data by application requesters (ARs) in the network.

An AR secures its objects and relational database to ensure only authorized users have access to distributed relational database programs. This is done using normal iSeries server object authorization to identify users and specify what each user (or group of users) is allowed to do with an object. Alternatively, authority to tables, views, and SQL packages can be granted or revoked using the SQL GRANT and REVOKE statements. Providing levels of authority to SQL objects on the AR helps ensure that only authorized users have access to an SQL application that accesses data on another system.

The level of system security in effect on the application server (AS) determines whether a request from an AR is accepted and whether the remote user is authorized to objects on the AS.

Some aspects of security planning for the iSeries server in a distributed relational database network include:

- Object-related security to control user access to particular resources such as confidential tables, programs, and packages
- Location security that verifies the identity of other systems in the network
- User-related security to verify the identity and rights of users on the local system and remote systems
- Physical security such as locked doors or secured buildings that surround the systems, modems, communication lines and terminals that can be configured in the line description and used in the route selection process

Location, user-related, and object-related security are only possible if the system security level is set at level 20 or above.

For Advanced Program-to-Program Communication (APPC) conversations, when the system is using level 10 security, an iSeries server connects to the network as a nonsecure system. The server does not validate the identity of a remote system during session establishment and does not require conversation security on incoming program start requests. For level 10, security information configured for the APPC remote location is ignored and is not used during session or conversation establishment. If a user profile does not exist on the server, one is created.

When the system is using security level 20 or above, an iSeries server connects to the network as a secure system. The iSeries system can then provide conversation-level security functions and, in the case of APPC, session level security as well.

Having system security set at the same level across the systems in your network makes the task of security administration easier. An AS controls whether the session and conversation can be established by specifying what is expected from the AR to establish a session. For example, if the security level on the AR is set at 10 and the security level on the AS is above 10, the appropriate information might not be sent and the session might not be established without changing security elements on one of the systems.

Passwords for DRDA access

The most common method of authorizing a remote user for database access is by flowing a user ID and password at connection time. One method an application programmer can use to do this is to code the USER/USING clause on an embedded SQL CONNECT statement. For example:

```
EXEC SQL CONNECT TO :locn USER :userid USING :pw
```

For Distributed Relational Database Architecture (DRDA) access to remote relational databases, once a conversation is established, you do not need to enter a password again. If you end a connection with either a RELEASE, DISCONNECT, or CONNECT statement when running with the remote unit of work (RUW) connection management method, your conversation with the first application server (AS) might or might not be dropped, depending on the kind of AS you are connected to and your application requester (AR) job attributes (for the specific rules, see Control DDM conversations). If the conversation to the first AS is not dropped, it remains unused while you are connected to the second AS. If you connect again to the first AS and the conversation is unused, the conversation becomes active again without you needing

to enter your user ID and password. On this second use of the conversation, your password is also not validated again.

Related concepts

Security

Related reference

APPC, APPN, and HPR

“Control DDM conversations” on page 107

The term *connection* in this topic collection refers to the concept of an SQL connection. An SQL connection lasts from the time an explicit or implicit SQL CONNECT is done until the logical SQL connection is terminated by such means as an SQL DISCONNECT, or a RELEASE followed by a COMMIT.

Elements of security in an APPC network

When Distributed Relational Database Architecture (DRDA) is used, the data resources of each server in the DRDA environment should be protected.

To protect data resources of each server in the DRDA environment, you can use three groups of security elements that are controlled by the following parameters:

- For system-related security or session, the *LOCPWD parameter* is used on each iSeries server to indicate the system validation password to be exchanged between the source and target systems when an Advanced Program-to-Program Communication (APPC) session is first established between them. Both systems must exchange the same password before the session is started. (On System/36™, this password is called the location password.) In an APPC network, the LOCPWD parameter on the Create Device Description (APPC) (CRTDEVAPPC) command specifies this password. Devices are created automatically using APPN, and the location password on the remote location list specifies a password that is used by the two locations to verify identities. Use the Create Configuration List (CRTCFGL) command to create a remote location list of type (*APPNRMT).
- For user-related or location security, the *SECURELOC parameter* is used on each iSeries server to indicate whether it (as a target server) accepts incoming access requests that have their security already verified by the source server or whether it requires a user ID and encrypted password. In an APPC network, the SECURELOC parameter on the Create Device Description (APPC) (CRTDEVAPPC) command specifies whether the local server allows the remote server to verify security. Devices are created automatically using APPN, and the secure-location on an APPN remote Configuration List is used to determine if the local server allows the remote server to verify user security information. The SECURELOC value can be specified differently for each remote location.

The SECURELOC parameter is used with the following security elements:

- The user ID sent by the source server, if allowed by this parameter
- The user ID and encrypted password, if allowed by this parameter
- The target server user profiles, including default user profiles

For more information, see the DDM source system security in an APPC network topic.

- For object-related security, the *DDMACC parameter* is used on the Change Network Attributes (CHGNETA) command to indicate whether the files on the iSeries server can be accessed at all by another server and, if so, at which level of security the incoming requests are to be checked. More information about this object-related parameter is provided in the topic DDM Network Attribute (DDMACC Parameter).
 - If *REJECT is specified on the DDMACC parameter, all DRDA requests received by the target iSeries server are rejected.
 - If *OBJAUT is specified on the DDMACC parameter, normal object-level security is used on the target server.
 - If the name of an optional, user-supplied user exit program (or access control program) is specified on the DDMACC parameter, an additional level of security is used. The user exit program can be

used to control whether a given user of a specific source server can use a specific command to access (in some manner) a specific file on the target server. (See the topic DDM server access control exit program for additional security for details.)

- When a file is created on the target server using DRDA, the library name specified contains the file. If no library name is specified on the DRDA request, the current library (*CURLIB) is used. The file authority defaults to allow only the user who created the file or the target server's security officer to access the file.

Most of the security controls for limiting remote file access are handled by the target server. Except for the user ID provided by the source server, all of these elements are specified and used on the target server. The source server, however, also limits access to target server files by controlling access to the DRDA file on the source server and by sending the user ID, when needed, to the target server.

Related concepts

DDM server access control exit program for additional security

"DRDA application server security in an APPC network" on page 54

When the target server is an iSeries server, several elements are used together to determine whether a request to access a remote file is allowed or not.

Related reference

Change Network Attributes (CHGNETA) command

Create Configuration List (CRTCFGL) command

Create Device Description (APPC) (CRTDEVAPPC) command

APPN configuration lists:

In an APPC network, location passwords are specified for those pairs of locations that are going to have end-to-end sessions between them.

Location passwords need not be specified for those locations that are intermediate nodes.

The remote location list is created with the Create Configuration List (CRTCFGL) command, and it contains a list of all remote locations, their location password, and whether the remote location is secure. There is one system-wide remote location configuration list on an iSeries server. A central site iSeries server can create location lists for remote iSeries servers by sending them a control language (CL) program.

Changes can be made to a remote configuration list using the Change Configuration List (CHGCFGL) command, however, they do not take effect until all devices for that location are all in a varied off state.

When the Display Configuration List (DSPCFGL) command is used, there is no indication that a password exists. The Change Configuration List (CHGCFGL) command indicates a password exists by placing *PASSWORD in the field if a password has been entered. There is no way to display the password. If you have problems setting up location security you might have to enter the password again on both systems to ensure that the passwords match.

Related reference

APPC, APPN, and HPR

Change Configuration List (CHGCFGL) command

Create Configuration List (CRTCFGL) command

Display Configuration List (DSPCFGL) command

Conversation level security:

Systems Network Architecture (SNA) logical unit (LU) 6.2 architecture identifies three conversation security designations that various types of systems in an SNA network can use to provide consistent conversation security across a network of unlike systems.

The SNA security levels are:

SECURITY(NONE)

No user ID or password is sent to establish communications.

SECURITY(SAME)

Sign the user on to the remote server with the same user ID as the local server.

SECURITY(PGM)

Both a user ID and a password are sent for communications.

SECURITY(PROGRAM_STRONG)

Both a user ID and a password are sent for communications only if the password will not be sent unencrypted, otherwise an error is reported. This is not supported by DRDA on i5/OS.

While the iSeries server supports all four SNA levels of conversation security, DRDA uses only the first three. The target controls the SNA conversation levels used for the conversation.

For the SECURITY(NONE) level, the target does not expect a user ID or password. The conversation is allowed using a default user profile on the target. Whether a default user profile can be used for the conversation depends on the value specified on the DFTUSR parameter of the Add Communications Entry (ADDCMNE) command or the Change Communications Entry (CHGCMNE) command for a given subsystem. A value of *NONE for the DFTUSR parameter means the application server (AS) does not allow a conversation using a default user profile on the target. SECURITY (NONE) is sent when no password or user ID is supplied and the target has SECURELOC(*NO) specified.

For the SECURITY(SAME) level, the remote server's SECURELOC value controls what security information is sent, assuming the remote server is an iSeries. If the SECURELOC value is *NONE, no user ID or password is sent, as if SECURITY(NONE) had been requested; see the previous paragraph for how SECURITY(NONE) is handled. If the SECURELOC value is *YES, the name of the user profile is extracted and sent along with an indication that the password has already been verified by the local server. If the SECURELOC value is *VFYENCPWD, the user profile and its associated password are sent to the remote server after the password has been encrypted to keep its value secret, so the user must have the same user profile name and password on both servers to use DRDA.

Note: SECURELOC(*VFYENCPWD) is the most secure of these three options because the most information is verified by the remote server; however, it requires that users maintain the same passwords on multiple servers, which can be a problem if users change one server but do not update their other servers at the same time.

For the SECURITY(PGM) level, the target expects both a user ID and password from the source for the conversation. The password is validated when the conversation is established and is ignored for any following uses of that conversation.

Related reference

Add Communications Entry (ADDCMNE) command

Change Communications Entry (CHGCMNE) command

DRDA application server security in an APPC network:

When the target server is an iSeries server, several elements are used together to determine whether a request to access a remote file is allowed or not.

User-related security elements

The user-related security elements include the SECURELOC parameter on the target server, the user ID sent by the source server (if allowed), the password for the user ID sent by the source server, and a user profile or default user profile on the target server.

Object-related security elements

The object-related security elements include the DDMACC parameter and, optionally, a user exit program supplied by the user to supplement normal object authority controls.

User-related elements of target security

A valid user profile must exist on the application server (AS) to process distributed relational database work. You can specify a default user profile for a subsystem that handles communications jobs on an iSeries server.

The name of the default user profile is specified on the DFTUSR parameter of the Add Communications Entry (ADDCMNE) command on the AS. The ADDCMNE command adds a communications entry to a subsystem description used for communications jobs.

If a default user profile is specified in a communications subsystem, whether the AS is a secure location or not determines if the default user profile is used for this request. The SECURELOC parameter on the Create Device Description (APPC) (CRTDEVAPPC) command, or the secure location designation on an APPN remote location list, specifies whether the AS is a secure location.

- If *YES is specified for SECURELOC or secure location on the AS, the AS considers the application requester (AR) a secure location. A user ID and an Already Verified indicator are expected from the AR with its request. If a user profile exists on the AS that matches the user ID sent by the requester, the request is allowed. If not, the request is rejected.
- If *NO is specified for the SECURELOC parameter on the AS, the AS does not consider the AR a secure location. Although the AR still sends a user ID, the AS does not use this for the request. Instead, a default user profile on the AS is used for the request, if one is available. If no default user profile exists on the AS, the request is rejected.
- If *VFYENCPWD is specified for SECURELOC on the AS, the AS considers the AR a secure location, but requires that the user ID and its password be sent (in encrypted form) to verify the identity of the current user. If the user profile exists on the AS that matches the user ID sent by the requester, and that requester has the same password on both systems, the request is allowed. Otherwise, the request is rejected.

The following table shows all of the possible combinations of the elements that control SNA SECURITY(PGM) on the iSeries server. A "Y" in any of the columns indicates that the element is present or the condition is met. An "M" in the PWD column indicates that the security manager retrieves the user's password and sends a protected (encrypted) password if password protection is active. If a protected password is not sent, no password is sent. A *protected password* is a character string that APPC substitutes for a user password when it starts a conversation. Protected passwords can be used only when the systems of both partners support password protection and when the password is created on a system that runs i5/OS or OS/400 Version 2 Release 2 or later.

Table 4. Remote access to a distributed relational database

Row	UID	PWD ¹	AVI	SEC(Y)	DFT	Valid	Access
1	Y	Y		Y	Y	Y	Use UID
2	Y	Y		Y	Y		Reject
3	Y	Y		Y		Y	Use UID
4	Y	Y		Y			Reject

Table 4. Remote access to a distributed relational database (continued)

Row	UID	PWD ¹	AVI	SEC(Y)	DFT	Valid	Access
5	Y	Y			Y	Y	Use UID
6	Y	Y			Y		Reject
7	Y	Y				Y	Use UID
8	Y	Y					Reject
9	Y		Y	Y	Y	Y	Use UID
10	Y		Y	Y	Y		Reject
11	Y		Y	Y		Y	Use UID
12	Y		Y	Y			Reject
13	Y	M ³			Y	Y	Use DFT or UID ²
14	Y	M ³			Y		Use DFT or UID ²
15	Y	M ³				Y	Reject or UID ²
16	Y	M ³					Reject or UID ²
17				Y	Y		Used DFT
18				Y			Reject
19					Y		Use DFT
20							Reject

Key:

UID User ID sent

PWD Password sent

AVI Already Verified Indicator set

SEC(Y) SECURELOC(YES) specified

DFT Default user ID specified in communication subsystem

Valid User ID and password are valid

Use UID

Connection made with supplied user ID

Use DFT

Connection made with default user ID

Reject Connection not made

1. If password protection is active, a protected password is sent.
2. Use UID when password protection is active.
3. If password protection is active, the password for the user is retrieved by the security manager, and a protected password is sent; otherwise, no password is sent.

To avoid having to use default user profiles, create a user profile on the AS for every AR user that needs access to the distributed relational database objects. If you decide to use a default user profile, however, make sure that users are not allowed on the system without proper authorization. For example, the following command specifies the default user parameter as DFTUSER(QUSER); this allows the system to accept job start requests without a user ID or password from a communications request. The communications job is signed on using the QUSER user profile.

```
ADDCMNE SBSDB(SAMPLE) DEV(*ALL) DFTUSER(QUSER)
```


Elements of security in a TCP/IP network

DDM and DRDA over native TCP/IP does not use i5/OS communications security services and concepts such as communications devices, modes, secure location attributes, and conversation security levels which are associated with Advanced Program-to-Program Communication (APPC). Therefore, security setup for TCP/IP is quite different.

Application requester security in a TCP/IP network:

Different connectivity scenarios call for using different levels of authentication. Therefore, an administrator can set the lowest security authentication method required by the application requester (AR) when connecting to an application server (AS) by setting the preferred authentication method field in each RDB directory entry.

The administrator might also allow the decision about authentication method to be negotiated with the server, by choosing to allow a lower security authentication method. In this case the preferred authentication method is still attempted, but if the AS cannot accept the preferred method, a lower method can be used, depending on the server security setting and other factors such as the availability of cryptographic support. For example, if two systems are in a physically unprotected environment, the administrator might choose to require Kerberos authentication without allowing lower security authentication methods.

On the application requester (client) side, you can use one of the two methods to send a password along with the user ID on DRDA TCP/IP connect requests. If you do not use either of these methods, the CONNECT command can send only a user ID.

The first way to send a password is to use the USER/USING form of the SQL CONNECT statement, as in the following example from the Interactive SQL environment:

```
CONNECT TO rdbname USER userid USING 'password'
```

In a program using embedded SQL, the values of the user ID and of the password can be contained in host variables in the USER/USING database.

In a program using CLI, the following example shows how the user ID and password are presented in host variables to the DRDA application requester (AR):

```
SQLConnect(hdbc,sysname,SQL_NTS, /*do SQL connect to the application server */  
          uid,SQL_NTS,pwd,SQL_NTS);
```

The second way to provide a password is to send a connect request over TCP/IP using a server authorization entry. A server authorization list is associated with every user profile on the system. By default, the list is empty; however, you can add entries by using the Add Server Authentication Entry (ADDSVRAUTE) command. When you attempt a DRDA connection over TCP/IP, the DB2 Universal Database for iSeries client (AR) checks the server authorization list for the user profile under which the client job is running. If it finds a match between the RDB name on the CONNECT statement and the SERVER name in an authorization entry (which must be in uppercase), the associated USRID parameter in the entry is used for the connection user ID. If a PASSWORD parameter is stored in the entry, that password is also sent on the connect request.

A server authorization entry can also be used to send a password over TCP/IP for a DDM file I/O operation. When you attempt a DDM connection over TCP/IP, DB2 UDB for iSeries checks the server authorization list for the user profile under which the client job is running. If it finds a match between either the RDB name (if RDB directory entries are used) or 'QDDMSERVER' and the SERVER name in an authorization entry, the associated USRID parameter in the entry is used for the connection user ID. If a PASSWORD parameter is stored in the entry, that password is also sent on the connect request.

To store a password using the Add Server Authentication Entry (ADDSVRAUTE) command, you must set the QRETSVRSEC system value to '1'. By default, the value is '0'. Type the following command to change this value:

```
CHGSYSVAL QRETSVRSEC VALUE('1')
```

The following example shows the syntax of the Add Server Authentication Entry (ADDSVRAUTE) command when using an RDB directory entry:

```
ADDSVRAUTE USRPRF(user-profile) SERVER(rdbname) USRID(userid) PASSWORD(password)
```

The USRPRF parameter specifies the user profile under which the application requester job runs. What you put into the SERVER parameter is normally the name of the RDB to which you want to connect. The exception is that if you are using DDM files which were not created to use the RDB directory, you should specify QDDMSERVER in the SERVER parameter. When you specify an RDB name, it must be in uppercase. The USRID parameter specifies the user profile under which the server job will run. The PASSWORD parameter specifies the password for the user profile.

If you omit the USRPRF parameter, it will default to the user profile under which the Add Server Authentication Entry (ADDSVRAUTE) command runs. If you omit the USRID parameter, it will default to the value of the USRPRF parameter. If you omit the PASSWORD parameter, or if you have the QRETSVRSEC value set to 0, no password will be stored in the entry and when a connect attempt is made using the entry, the security mechanism attempted will be user ID only.

You can use the Display Server Authentication Entries (DSPSVRAUTE) command to determine what authentication entries have been added to the server authentication list. The Retrieve Server Authentication Entries (QsyRetrieveServerEntries) (QSYRTVSE) API in a user-written program can also be used.

You can remove a server authorization entry by using the Remove Server Authentication Entry (RMVSVRAUTE) command. You can change a server authorization entry by using the Change Server Authentication Entry (CHGSVRAUTE) command

If a server authorization entry exists for a relational database (RDB), and the USER/USING form of the CONNECT statement is also used, the latter takes precedence.

Kerberos source configuration

Distributed Relational Database Architecture (DRDA) and distributed data management (DDM) can take advantage of Kerberos authentication if both systems are configured for Kerberos.

If a job's user profile has a valid ticket-granting ticket (TGT), the DRDA application requester (AR) uses this TGT to generate a service ticket and authenticate the user to the remote server. Having a valid TGT makes the need for a server authentication entry unnecessary, because no password is directly needed in that case. However, if the job's user profile does not have a valid TGT, the user ID and password can be retrieved from the server authentication entry to generate the necessary TGT and service ticket.

When using Kerberos, the remote location (RMTLOCNAME) in the RDB directory entry must be entered as the remote host name. IP addresses will not work for Kerberos authentication.

In cases where the Kerberos realm name differs from the DNS suffix name, it must be mapped to the correct realm. To do that, there must be an entry in the Kerberos configuration file (krb5.conf) to map each remote host name to its correct realm name. This host name entered must exactly match the remote location name (RMTLOCNAME). The remote location parameter displayed by the DSPRDBDIRE or DSPDDMF command must match the domain name in the krb5.conf file. The following figure shows an example of the DSPRDBDIRE display:

```

Display Relational Database Detail
Relational database . . . . . : RCHASXXX

Remote location:
Remote location . . . . . : rchasxxx.rchland.ibm.com
Type . . . . . : *IP
Port number or service name . . . : *DRDA
Remote authentication method . . . :
Preferred method . . . . . : *KERBEROS
Allow lower authentication . . . : *NOALLOWER
Text . . . . . :

Relational database type . . . . : *REMOTE

Press Enter to continue.
F3=Exit F12=Cancel

```

Here is a portion of the corresponding krb5.conf file contents showing the domain name matching the remote location name (Note: The Display File (DSPF) command is used to display the configuration file contents):

```

DSPF STMF('/QIBM/UserData/OS400/NetworkAuthentication/krb5.conf')

[domain_realm]
; Convert host names to realm names. Individual host names may be
; specified. Domain suffixes may be specified with a leading period
; and will apply to all host names ending in that suffix.
rchasxxx.rchland.ibm.com = REALM.RCHLAND.IBM.COM

```

Jobs using Kerberos must be restarted when configuration changes occur to the krb5.conf file.

Related concepts

Enterprise Identity Mapping

Related tasks

“Set up the TCP/IP server for DRDA” on page 39

If you own a Distributed Relational Database Architecture (DRDA) application server (AS) that will be using the TCP/IP protocol, you need to set up the DDM TCP/IP server.

Network authentication service

Related reference

- Add Server Authentication Entry (ADDSVRAUTE) command
- Display Server Authentication Entries (DSPSVRAUTE) command
- Retrieve Server Authentication Entries (QsyRetrieveServerEntries) (QSYRTVSE) API
- Remove Server Authentication Entry (RMVSVRAUTE) command
- Change Server Authentication Entry (CHGSVRAUTE) command
- Display File (DSPF) command

Application server security in a TCP/IP network:

The TCP/IP server has a default security of user ID with clear-text password. This means that, as the server is installed, inbound TCP/IP connection requests must have at least a clear-text password accompanying the user ID under which the server job is to run.

The security can either be changed with the Change DDM TCP/IP Attributes (CHGDDMTCPA) command or under the **Network** → **Servers** → **TCP/IP** → **DDM server properties** in iSeries Navigator. You must have *IOSYSCFG special authority to change this setting.

There are two settings that can be used for lower server security:

- PWDRQD(*NO)
Password is not required.
- PWDRQD(*VLDONLY)
Password is not required, but must be valid if sent.

The difference between *NO and *VLDONLY is that if a password is sent from a client system, it is ignored in the *NO option. In the *VLDONLY option, however, if a password is sent, the password is validated for the accompanying user ID, and access is denied if incorrect.

Encrypted password required or PWDRQD(*ENCRYPTED) and Kerberos or PWDRQD(*KERBEROS) can be used for higher security levels. If Kerberos is used, user profiles must be mapped to Kerberos principles using Enterprise Identity Mapping (EIM).

The following example shows the use of the Change DDM TCP/IP Attributes (CHGDDMTCPA) command to specify that an encrypted password must accompany the user ID. To set this option, enter:

```
CHGDDMTCPA PWDRQD(*ENCRYPTED)
```

Note: The DDM/DRDA TCP/IP server was enhanced in V4R4 to support a form of password encryption called password substitution. In V4R5, a more widely-used password encryption technique, referred to as the Diffie-Hellman public key algorithm was implemented. This is the DRDA standard algorithm and is used by the most recently released IBM DRDA application requesters. The older password substitute algorithm is used primarily for DDM file access from PC clients. In V5R1, a 'strong' password substitute algorithm was also supported. The client and server negotiate the security mechanism that will be used, and any of the three encryption methods will satisfy the requirement of PWDRQD(*ENCRYPTED), as does the use of Secure Sockets Layer (SSL) datastreams.

Related concepts

Enterprise Identity Mapping

Related reference

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Connection security protocols for DDM and DRDA:

Several connection security protocols are supported by the current DB2 UDB for iSeries implementation of distributed data management (DDM) or Distributed Relational Database Architecture (DRDA) over TCP/IP.

- User ID only
- User ID with clear-text password
- User ID with encrypted password
- Kerberos

With encrypted datastream support, the traditional communications trace support has little value. The Trace TCP/IP Application (TRCTCPAPP) command records outbound data streams at a point before encryption, and inbound data streams after decryption.

Related concepts

“Communications trace” on page 175

If you get a message in the CPF3Exx range or the CPF91xx range when using Distributed Relational Database Architecture (DRDA) to access a distributed relational database, you should run a communications trace.

Related reference

Trace TCP/IP Application (TRCTCPAPP) command

Secure Sockets Layer for DDM and DRDA:

DB2 Universal Database for iSeries Distributed Relational Database Architecture (DRDA) clients do not support Secure Sockets Layer (SSL).

However, similar function is available with Internet Protocol Security Architecture (IPSec).

The DDM TCP/IP server supports the SSL data encryption protocol. You can use this protocol to interoperate with clients such as iSeries Toolbox for Java™ and iSeries Access Family OLE DB Provider that support SSL for record-level access, and with any DDM file I/O clients provided by independent software vendors that might support SSL.

To use SSL with the iSeries DDM TCP/IP server, you must configure the client to connect to the well-known SSL port 448 on the server.

If you specify PWDRQD(*ENCRYPTED) on the Change DDM TCP/IP Attributes (CHGDDMTCPA) command on the server, you can use any valid password along with SSL. This is possible because the server recognizes that the whole data stream, including the password, is encrypted.

Related concepts

“Internet Protocol Security Architecture for DDM and DRDA”

Internet Protocol Security Architecture (IPSec) is a security protocol in the network layer that provides cryptographic security services. These services support confidential delivery of data over the Internet or intranets.

Secure Sockets Layer (SSL)

Related reference

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Required programs:

You will need to set up and install SSL support.

Related concepts

iSeries Access for Windows: Programming

iSeries server requirements:

For an iSeries server to communicate over SSL, it must be running V4R4 or later of the operating system, and have the following applications installed:

- TCP/IP Connectivity Utilities for i5, 5722-TC1 (Base TCP/IP support)
- IBM HTTP Server for i5/OS, 5722-DG1 (for access to Digital Certificate Manager)
- Digital Certificate Manager, 5722-SS1 - Boss Option 34

Internet Protocol Security Architecture for DDM and DRDA:

Internet Protocol Security Architecture (IPSec) is a security protocol in the network layer that provides cryptographic security services. These services support confidential delivery of data over the Internet or intranets.

On iSeries, IPsec, a component of the virtual private networking (VPN) support, allows all data between two IP addresses or port combinations to be encrypted, regardless of application (such as DRDA or DDM). You can configure the addresses and ports that are used for IPsec. IBM suggests using port 447 for IPsec for either DRDA access or DDM access.

Use of any valid password along with IPsec will not in general satisfy the requirement imposed by specifying PWDRQD(*ENCRYPTED) on the Change DDM TCP/IP Attributes (CHGDDMTCPA) command at the server, because the application (DRDA or DDM) will not be able to determine if IPsec is being used. Therefore, you should avoid using PWDRQD(*ENCRYPTED) with IPsec.

Related concepts

“Secure Sockets Layer for DDM and DRDA” on page 61

DB2 Universal Database for iSeries Distributed Relational Database Architecture (DRDA) clients do not support Secure Sockets Layer (SSL).

Virtual Private Networking (VPN)

“Considerations for certain passwords being sent as clear text”

Although iSeries supports the encryption of connection passwords, one of the connection security options you can specify in setting up an RDB directory entry is *USRIDPWD.

Related reference

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Considerations for certain passwords being sent as clear text:

Although iSeries supports the encryption of connection passwords, one of the connection security options you can specify in setting up an RDB directory entry is *USRIDPWD.

See the Add Relational Database Directory Entry command and the Change Relational Database Directory Entry command in Work with the relational database directory for more information.

If the server to which the connection is made allows the *USRIDPWD security option, the connection password can flow unencrypted. In V5R3, the SQL SET ENCRYPTION PASSWORD statement and the ENCRYPT function can also cause passwords to flow over the network unencrypted. Currently, there are two possible solutions for encrypting datastreams. One is to use IPsec. As the other possibility, if you are using an AR that supports SSL, you can use that protocol to encrypt data transmitted to and from an iSeries AS.

Related concepts

“Internet Protocol Security Architecture for DDM and DRDA” on page 61

Internet Protocol Security Architecture (IPsec) is a security protocol in the network layer that provides cryptographic security services. These services support confidential delivery of data over the Internet or intranets.

Related reference

“Work with the relational database directory” on page 30

Use these instructions to work with the relational database directory.

SET ENCRYPTION PASSWORD statement

Ports and port restrictions for DDM/DRDA:

With the advent of new choices for the security of distributed data management (DDM) communications, the iSeries server administrator can restrict certain communications modes by blocking the ports they use. This topic discusses some of these considerations.

The DDM or DRDA TCP/IP server listens on port 447 (the well-known DDM port) and 446 (the well-known DRDA port) as well as 448 (the well-known SSL port). The DB2 Universal Database for

iSeries implementation of DDM does not distinguish between the two ports 446 and 447, however, so both DDM and DRDA access can be done on either port.

Using the convention recommended for IPsec, the port usage for the DDM TCP/IP server follows:

- 446 for clear text data streams
- 447 for IPsec encrypted data streams (suggested)
- 448 for SSL encrypted data streams (required)

You can block usage of one or more ports at the server by using the Configure TCP/IP (CFGTCP) command. To do this, choose the Work with TCP/IP port restrictions option of that command. You can add a restriction so that only a specific user profile other than the one that QRWTLSTN runs under (normally QUSER) can use a certain port, such as 446. That effectively blocks 446. If 447 were configured for use only with IPsec, then blocking 446 would allow only encrypted data streams to be used for DDM and DRDA access over native TCP/IP. You could block both 447 and 448 to restrict usage only to SSL. It might be impractical to follow these examples for performance or other reasons (such as current limited availability of SSL-capable clients), but they are given to show the possible configurations.

Related reference

Configure TCP/IP (CFGTCP) command

DRDA server access control exit programs

A security feature of the Distributed Relational Database Architecture (DRDA) server, for use with both Advanced Program-to-Program Communication (APPC) and TCP/IP, extends the use of the DDMACC parameter of the Change Network Attributes (CHGNETA) command to DRDA.

The parameter previously applied only to DDM file I/O access. The DRDA usage of the function is limited to connection requests, however, and not to requests for data after the connection is made.

If you do not choose to use this security function, you normally do not need to do anything. The only exception is if you are currently using a DDM exit program that is coded to reject operations if an unknown function code is received, and you are also using DRDA to access data on that system. In this case, you must change your exit program so that a '1' is returned to allow DRDA access if the function code is 'SQLCNN'.

To use the exit program for blocking or filtering DRDA connections, you need to create a new DRDA exit program, or change an existing one.

Note: If your system is configured with multiple databases (ASP groups), the exit program must reside in a library in the system database (on an auxiliary storage pool in the range 1-32).

You can find general instructions for creating a DRDA exit program in the Distributed data management topic.

This security feature adds a DRDA function code to the list of request functions that can be input to the program in the input parameter structure. The function code, named 'SQLCNN' (SQL connect request), indicates that a DRDA connection request is being processed (see the FUNC parameter in Example: DRDA server access control exit program). The APP (application) input parameter is set to '*DRDA' instead of '*DDM' for DRDA connection request calls.

When you code exit programs for DRDA, the following fields in the parameter structure might be useful:

- The USER field allows the program to allow or deny DRDA access based on the user profile ID.
- The RDBNAME field contains the name of the RDB to which the user wants to connect. This can be the system database or a user database (ASP group). This field can be useful if you want to deny access to one or more databases in an environment where multiple databases are configured.

- The SRVNAME parameter in Example: DRDA server access control exit program might or might not be set by the caller of the exit program. If it is set, it indicates the name of the client system. If it is not set, it has the value *N. It will always be set when the DRDA Application Requester is an iSeries server.
- The TYPDEFN parameter gives additional information about the type of client that is connecting. For an IBM mainframe, TYPEDEFN will be QTDSQL370. For an iSeries server, it will be QTDSQL400. For an Intel® PC, it will be QTDSQLX86. For an RS/6000® client, it will be QTDSQLASC.
- The PRDID (product ID) parameter identifies the product that is attempting to connect, along with the product's release level. Here is a partial list of the first three characters of these codes (You should verify the non-IBM codes before you use them in an exit program):

QSQ IBM DB2 Universal Database for iSeries
DSN IBM DB2 Universal Database for z/OS
SQL IBM DB2 Universal Database for Linux, Unix and Windows (formerly called DDCS)
ARI IBM DB2 Universal Database for VSE & VM
GTW Oracle Corporation products
GVW Grandview DB/DC Systems products
XDB XDB Systems products
IFX Informix® Software products
RUM Wall Data Rumba for Database Access
SIG StarQuest products
STH FileTek products
JCC IBM DB2 Universal Driver for SQLJ and JDBC

The rest of the field is structured as vvrrm, where vv is version, rr is release, and m is modification level.

If the exit program returns a RTNCODE value of '0', and the connection request came from an iSeries client, then the message indicating the connection failure to the user will be SQ30060, 'User is not authorized to relational database'. In general, the response to a denial of access by the exit program is the DRDA RDBATHRM reply message, which indicates that the user is not authorized to the relational database. Note that different client platforms might report the error differently to the user.

Restrictions:

- If a function check occurs in the user exit program, the program returns the same reply message, and the connection attempt will fail. The exit program must not do any committable updates to DB2 UDB for iSeries, or unpredictable results might occur.
- You should not use exit programs to attempt to access a file that was opened in a prior call of the prestart server job.
- Prior to V5R2, a further restriction resulted when the prestart jobs used with the TCP/IP server were recycled for subsequent use. Some cleanup is done to prepare the job for its next use. Part of this processing involves using the Reclaim Activation Group (RCLACTGRP) command with the ACTGRP parameter with value of *ELIGIBLE. As a result, attempts to use any residual linkages in the prestart server job to activation groups destroyed by the RCLACTGRP can result in MCH3402 exceptions (where the program tried to refer to all or part of an object that no longer exists). One circumvention to this restriction is to set the MAXUSE value for the QRWTSRVR prestart jobs to 1 as follows: CHGPJE SBSD(QSYSWRK) PGM(QRWTSRVR) MAXUSE(1).

Related concepts

“Object-related security for DRDA” on page 67

If the iSeries server is an application server (AS), there are two object-related levels at which security can be enforced to control access to its relational database tables.

Distributed data management

Related reference

Reclaim Activation Group (RCLACTGRP) command

Change Network Attributes (CHGNETA) command

Example: DRDA server access control exit program

This exit program shows an example of a PL/I exit program that allows all DRDA operations and all DRDA connections except when the user ID is ALIEN.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 250.

```

/*****
/*
/* PROGRAM NAME: UEPALIEN
/*
/* FUNCTION:      USER EXIT PROGRAM THAT IS DESIGNED TO
/*                RETURN AN UNSUCCESSFUL RETURN CODE WHEN
/*                USERID 'ALIEN' ATTEMPTS A DRDA CONNECTION.
/*                IT ALLOWS ALL TYPES OF DDM OPERATIONS.
/*
/* EXECUTION:    CALLED WHEN ESTABLISHED AS THE USER EXIT
/*                PROGRAM.
/*
/* ALL PARAMETER VARIABLES ARE PASSED IN EXCEPT:
/*
/* RTNCODE -     USER EXIT RETURN CODE ON WHETHER FUNCTION IS
/*                ALLOWED: '1' INDICATES SUCCESS; '0' FAILURE.
/*
*****/
UEPALIEN: PROCEDURE (RTNCODE,CHARFLD);

DECLARE RTNCODE CHAR(1);          /* DECLARATION OF THE EXIT
/* PROGRAM RETURN CODE. IT
/* INFORMS REQUEST HANDLER
/* WHETHER REQUEST IS ALLOWED.
DECLARE
1 CHARFLD,                       /* DECLARATION OF THE CHAR
/* FIELD PASSED IN ON THE CALL.

2 USER CHAR(10),                 /* USER PROFILE OF DDM/DRDA USER
2 APP CHAR(10),                   /* APPLICATION NAME
2 FUNC CHAR(10),                  /* REQUESTED FUNCTION
2 OBJECT CHAR(10),                /* FILE NAME
2 DIRECT CHAR(10),                /* LIBRARY NAME
2 MEMBER CHAR(10),                /* MEMBER NAME
2 RESERVED CHAR(10),              /* RESERVED FIELD
2 LENGTH PIC '99999',             /* LENGTH OF USED SPACE IN REST
2 REST,                            /* REST OF SPACE = CHAR(2000)

3 LUNAME CHAR(10),                /* REMOTE LU NAME (IF SNA)
3 SRVNAME CHAR(10),               /* REMOTE SERVER NAME
3 TYPDEFN CHAR(9),                /* TYPE DEF NAME OF DRDA AR
3 PRDID,                           /* PRODUCT ID OF DRDA AR

5 PRODUCT CHAR(3),                /* PRODUCT CODE
5 VERSION CHAR(2),                /* VERSION ID
5 RELEASE CHAR(2),                /* RELEASE ID
5 MOD CHAR(1),                    /* MODIFICATION LEVEL
5 RDBNAME CHAR(18),               /* RDB NAME
5 REMAING CHAR(1965),             /* REMAINING VARIABLE SPACE

START:
IF (USER = 'ALIEN' &           /* IF USER IS 'ALIEN' AND
FUNC = 'SQLCNN') THEN          /* FUNCTION IS DRDA CONNECT
RTNCODE = '0';                 /* SET RETURN CODE TO UNSUCCESSFUL
ELSE                             /* IF ANY OTHER USER, OR DDM
RTNCODE = '1';                 /* SET RETURN CODE TO SUCCESSFUL

END UEPALIEN;

```

Figure 8. Example PL/I user exit program

Object-related security for DRDA

If the iSeries server is an application server (AS), there are two object-related levels at which security can be enforced to control access to its relational database tables.

The DDMACC parameter is used on the Change Network Attributes (CHGNETA) command to indicate whether the tables on this server can be accessed at all by another system and, if so, at which level of security the incoming DRDA requests are to be checked.

- If *REJECT is specified on the DDMACC parameter, all distributed relational database requests received by the AS are rejected. However, this system (as an application requester (AR)) can still use SQL requests to access tables on other systems that allow it. No remote system can access a database on any iSeries server that specifies *REJECT.

If *REJECT is specified while an SQL request is already in use, all *new* jobs from any system requesting access to this system's database are rejected and an error message is returned to those jobs; existing jobs are not affected.

- If *OBJAUT is specified on the DDMACC parameter, normal object-level security is used on the AS.

The DDMACC parameter is initially set to *OBJAUT. A value of *OBJAUT allows all remote requests, but they are controlled by the object authorizations on this AS. If the DDMACC value is *OBJAUT, the user profile used for the job must have appropriate object authorizations through private, public, group, or adopted authorities, or the profile must be on an authorization list for objects needed by the AR job. For each SQL object on the system, all users, no users, or only specific users (by user ID) can be authorized to access the object.

The user ID that must be authorized to objects is the user ID of the AS job. See the Elements of DDM Security in an APPC network topic for information about what user profile the AS job runs under.

In the case of a TCP/IP connection, the server job initially starts running under QUSER. After the user ID is validated, an exchange occurs so that the job then runs under the user profile specified on the connection request. The job inherits the attributes (for example, the library list) of that user profile.

When the value *OBJAUT is specified, it indicates that no further verification (beyond iSeries object level security) is needed.

- For DDM jobs, if the name of an optional, user-supplied user exit program (or access control program) is specified on the DDMACC parameter, an additional level of security is used. The user exit program can be used to control whether a user of a DDM client can use a specific command to access a specific file on the iSeries server.

For DRDA jobs, if the name of an optional, user-supplied user exit program (access control program) is specified on the DDMACC parameter, the system treats the entry as though *OBJAUT were specified, with one exception. The only effect that a user-written exit program can have on a DRDA job is to reject a connection request.

The DDMACC parameter, initially set to *OBJAUT, can be changed to one of the previously described values by using the Change Network Attributes (CHGNETA) command, and its current value can be displayed by the Display Network Attributes (DSPNETA) command. You can also get the value in a CL program by using the Retrieve Network Attributes (RTVNETA) command.

If the DDMACC parameter value is changed, although it takes effect immediately, it affects only *new* distributed relational database jobs started on this system (as the AS). Jobs running on this AS before the change was made continue to use the old value.

Related concepts

Communications Management PDF

"DRDA server access control exit programs" on page 63

A security feature of the Distributed Relational Database Architecture (DRDA) server, for use with both Advanced Program-to-Program Communication (APPC) and TCP/IP, extends the use of the DDMACC parameter of the Change Network Attributes (CHGNETA) command to DRDA.

Related reference

Change Network Attributes (CHGNETA) command
Display Network Attributes (DSPNETA) command
Retrieve Network Attributes (RTVNETA) command

Authority to distributed relational database objects

You can use either the SQL GRANT and REVOKE statements or the control language (CL) Grant Object Authority (GRTOBJAUT) and Revoke Object Authority (RVKOBJAUT) commands to grant and revoke a user's authority to relational database objects.

The SQL GRANT and REVOKE statements only operate on packages, tables, and views. In some cases, it is necessary to use GRTOBJAUT and RVKOBJAUT to authorize users to other objects, such as commands and programs.

The authority checked for SQL statements depends on whether the statement is static, dynamic, or being run interactively.

For Interactive SQL statements, authority is checked against the authority of the person processing the statement. Adopted authority is not used for Interactive SQL statements.

Users running a distributed relational database application need authority to run the SQL package on the application server (AS). The GRANT EXECUTE ON PACKAGE statement allows the owner of an SQL package, or any user with administrative privileges to it, to grant specified users the privilege to run the statements in an SQL package. You can use this statement to give all users authorized to the AS, or a list of one or more user profiles on the AS, the privilege to run statements in an SQL package.

Normally, users have processing privileges on a package if they are authorized to the distributed application program created using the CRTSQL xxx command. If the package is created using the Create Structured Query Language Package (CRTSQLPKG) command, you might have to grant processing privileges on the package to users. You can issue this statement in an SQL program or using Interactive SQL. A sample statement is as follows:

```
GRANT EXECUTE  
ON PACKAGE SPIFFY.PARTS1  
TO PUBLIC
```

The REVOKE EXECUTE ON PACKAGE statement allows the owner of an SQL package, or any user with administrative privileges to it, to remove the privilege to run statements in an SQL package from specified users. You can remove the EXECUTE privilege to all users authorized to the AS or to a list of one or more user profiles on the AS.

If you granted the same privilege to the same user more than once, revoking that privilege from that user nullifies all those grants. If you revoke an EXECUTE privilege on an SQL package you previously granted to a user, it nullifies any grant of the EXECUTE privilege on that SQL package, regardless of who granted it. A sample statement is as follows:

```
REVOKE EXECUTE  
ON PACKAGE SPIFFY.PARTS1  
FROM PUBLIC
```

You can also grant authority to an SQL package using the Grant Object Authority (GRTOBJAUT) command or revoke authority to an SQL package using the Revoke Object Authority (RVKOBJAUT) command.

Related concepts

Security for SQL objects

Related reference

Create Structured Query Language Package (CRTSQLPKG) command

Grant Object Authority (GRTOBJAUT) command

Revoke Object Authority (RVKOBJAUT) command

“Distributed relational database statements” on page 83

The statements included with the SQL language specifically support a distributed relational database.

Programs that run under adopted authority for a distributed relational database

A distributed relational database program can run under adopted authority, which means the user adopts the program owner’s authority to objects used by the program while running the program. When a program is created using the *SQL precompiler option for naming, the program runs under the program owner’s user profile.

An SQL package from an unlike system always adopts the package owner’s authority for all static SQL statements in the package. An SQL package created on an iSeries server using the CRTSQLxxx command with OPTION(*SQL) specified, also adopts the package owner’s authority for all static SQL statements in the package.

A distributed relational database administrator can check security exposure on application servers by using the Display Programs that Adopt (DSPPGMADP) command. The DSPPGMADP command displays the programs and SQL packages that use a specified user profile, as shown here. You can also send the results of the command to a printer or to an output file.

```
Display Programs That Adopt
User profile . . . . . : MPSUP

Object      Library  Type      Attribute  Text
INVENT     SPIFFY  *PGM                      Adopting program
CLIENT1    SPIFFY  *PGM                      Adopting program
TESTINV    TEST     *PGM      CLP         Test inventory pgm
INVENT1    SPIFFY  *SQLPKG                      SQL package
CLIENT1    SPIFFY  *SQLPKG                      SQL package
TESTINV    SPIFFY  *SQLPKG                      SQL package
```

```
Bottom
Press Enter to continue
```

```
F3=Exit  F12=Cancel  F17=Top  F18=Bottom
(C) COPYRIGHT IBM CORP. 1980, 1991.
```

Related reference

Display Programs that Adopt (DSPPGMADP) command

Protection strategies in a distributed relational database

Network security in an iSeries distributed relational database must be planned to protect critical data on any application server (AS) from unauthorized access. But because of the distributed nature of the relational database, security planning must ensure that availability of data in the network is not unnecessarily restricted.

One of the decisions that a distributed relational database administrator needs to make is the system security level in place for each system in the network. A system security level of 10 provides no security for application servers other than physical security at the system site. A system security level of 20 provides some protection to application servers because network security checking is done to ensure the

local and remote system are correctly identified. However, this level does not provide the object authorization necessary to protect critical database elements from unauthorized access. An iSeries server security level of 30 and above is the recommended choice for systems in a network that want to protect specific system objects.

The distributed relational database administrator must also consider how communications are established between application requesters (ARs) on the network and the application servers. Some questions that need to be resolved might include:

- Should a default user profile exist on an AS?

Maintaining many user profiles throughout a network can be difficult. However, creating a default user profile in a communications subsystem entry opens the AS to incoming communications requests if the AS is not a secure location. In some cases this might be an acceptable situation, in other cases a default user profile might reduce the system protection capabilities too far to satisfy security requirements.

For example, systems that serve many ARs need a high level of security. If their databases were lost or damaged, the entire network could be affected. Because it is possible to create user profiles or group profiles on an AS that identifies all potential users needing access, it is unnecessary for the database administrator to consider creating a default user profile for the communications subsystem or subsystems managing distributed relational database work.

In contrast, an iSeries server that rarely acts as an AS to other systems in the network and does not contain sensitive or critical data might use a default user profile for the communications subsystem managing distributed relational database work. This might prove particularly effective if the same application is used by all the other systems in the network to process work on this database.

Strictly speaking, the concept of a default user applies only to the use of APPC. However, a similar technique can be used with systems that are using TCP/IP. A single user ID can be established under which the server jobs can run. The Add Server Authentication Entry (ADDSVRAUTE) command can be used on all ARs to specify that user ID should be used for all users to connect with. The server authorization entries can have a password specified on them, or they can specify *NONE for the password, depending on the setting of the PWDRQD parameter on the Change DDM TCP/IP Attributes (CHGDDMTCPA) command at the AS. The default value of this attribute is that passwords are required.

- How should access to database objects be handled?

Authority to objects can be granted through private authority, group authority, public authority, adopted authority, and authorization lists. While a user profile (or default profile) has to exist on the AS for the communications request to be accepted, how the user is authorized to objects can affect performance.

Whenever possible, use group authority or authorization lists to grant access to a distributed relational database object. It takes less time and system resources to check these than to review all private authorities.

For TCP/IP connections, you do not need a private user ID for each user that can connect to an AS, because you can map user IDs.

Related reference

Add Server Authentication Entry (ADDSVRAUTE) command

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Application development

Programmers can write high-level language programs that use SQL statements for iSeries distributed application programs.

The main differences from programs written for local processing only are the ability to connect to remote databases and to create SQL packages. The CONNECT SQL statement can be used to explicitly connect an application requester to an application server, or the name of the relational database can be specified

when the program is created to allow an implicit connection to occur. Also, the SET CONNECTION, RELEASE, and DISCONNECT statements can be used to manage connections for applications that use distributed unit of work.

An SQL package is an iSeries object used only for distributed relational databases. It can be created as a result of the precompile process of SQL or can be created from a compiled program object. An SQL package resides on the application server. It contains SQL statements, host variable attributes, and access plans which the application server uses to process an application requester's request.

Because application programs can connect to many different servers, programmers might need to pay more attention to data conversion between servers. The iSeries server provides for conversion of various types of data, including coded character set identifier (CCSID) support for the management of character information.

You can create and maintain programs for a distributed relational database on the iSeries server using the SQL language the same way you use it for local-processing applications. You can embed static and dynamic Structured Query Language (SQL) statements with any one or more of the following high-level languages:

- iSeries PL/I
- ILE C/400[®]
- COBOL/400[®]
- ILE COBOL/400
- FORTRAN/400
- RPG/400[®]
- ILE RPG/400

The process of developing distributed applications is similar to that of developing SQL applications for local processing. The difference is that the application for distributed processing must specify the name of the relational database to which it connects. This might be done when you precompile the program or within the application.

The same SQL objects are used for both local and distributed applications, except that one object, the SQL package, is used exclusively for distributed relational database support. You create the program using the Create SQL program (CRTSQLxxx) command. The xxx in this command refers to the host language CI, CBL, CBLI, FTN, PLI, RPG, or RPGI. The SQL package might be a product of the precompile in this process. The Create Structured Query Language Package (CRTSQLPKG) command creates SQL packages for existing distributed SQL programs.

You must have the DB2 UDB Query Manager and SQL Development Kit licensed program installed to precompile programs with SQL statements. However, you can create SQL packages from existing distributed SQL programs with only the compiled program installed on your server. The DB2 UDB Query Manager and SQL Development Kit licensed program also allows you to use Interactive SQL to access a distributed relational database. This is helpful when you are debugging programs because it allows you to test SQL statements without having to precompile and compile a program.

Related information

Create Structured Query Language Package (CRTSQLPKG) command
SQL programming

Programming considerations for a distributed relational database application

Programming considerations for a distributed relational database application on an iSeries server fall into two main categories: those that deal with a function that is supported on the local server and those that are a result of having to connect to other servers.

Related reference

“Tips: Design distributed relational database applications” on page 19

Distributed relational database applications have different requirements from applications developed solely for use on a local database.

Naming of distributed relational database objects

SQL objects are created and maintained as iSeries server objects.

You can use either of the two naming conventions in DB2 Universal Database for iSeries programming: system (*SYS) and SQL (*SQL). The naming convention that you use affects the method for qualifying file and table names. It also affects security and the terms used on the Interactive SQL displays. Distributed relational database applications can access objects on another iSeries server using either naming convention. However, if your program accesses a non-DB2 UDB for iSeries server, only SQL names can be used. This can be specified using the NAMING parameter on the Start SQL (STRSQL) command, the OPTION parameter on one of the CRTSQLxxx commands, or by using the naming connection property for CLI/JDBC.

Related reference

Naming conventions

System (*SYS) naming convention:

When you use the system naming convention, files are qualified by library name in the form *library/file*.

Tables created using this naming convention assume the public authority of the library in which they are created. If the table name is not explicitly qualified and a default collection name is used in the DFTRDBCOL parameter of the CRTSQLxxx or CRTSQLPKG command, the default collection name is used for static SQL statements. If the file name is not explicitly qualified and the default collection name is not specified, the following rules apply:

- All SQL statements except certain CREATE statements cause SQL to search the library list (*LIBL) for the unqualified file.
- The CREATE statements resolve to unqualified objects as follows:
 - CREATE TABLE: The table name must be explicitly qualified.
 - CREATE VIEW: The view is created in the first library referred to in the subselect.
 - CREATE INDEX: The index is created in the collection or library that contains the table on which the index is being built.

SQL (*SQL) naming convention:

When you use the SQL naming convention, tables are qualified by the collection name in the form *collection.table*.

If the table name is not explicitly qualified and the default collection name is specified in the default relational database collection (DFTRDBCOL) parameter of the CRTSQLxxx or Create Structured Query Language Package (CRTSQLPKG) command, the default collection name is used. If the table name is not explicitly qualified and the default collection name is not specified, the following rules apply:

- For static SQL, the default qualifier is the user profile of the program owner.
- For dynamic SQL or Interactive SQL, the default qualifier is the user profile of the job running the statement.

Related information

Create Structured Query Language Package (CRTSQLPKG) command

Default collection name:

You can specify a default collection name to be used by an SQL program by supplying this name for the DFTRDBCOL parameter on the CRTSQLxxx command when you precompile the program.

The DFTRDBCOL parameter provides the program with the collection name as the library for an unqualified file if the *SYS naming convention is used, or as the collection for an unqualified table if the *SQL naming convention is used. If you do not specify a default collection name when you precompile the program, the rules for unqualified names apply for each naming convention. The default relational database collection name only applies to static SQL statements.

You can also use the DFTRDBCOL parameter on the Create Structured Query Language Package (CRTSQLPKG) command to change the default collection of a package. After an SQL program is compiled, you can create a new SQL package to change the default collection.

Related tasks

“Use the Create SQL Package (CRTSQLPKG) command” on page 97

You can enter the Create SQL Package (CRTSQLPKG) command to create an SQL package from a compiled distributed relational database program. You can also use this command to replace an SQL package that was created previously.

Related information

Create Structured Query Language Package (CRTSQLPKG) command

Connect to a distributed relational database

What makes a distributed relational database application *distributed* is its ability to connect to a relational database on another server.

There are two types of CONNECT statements with the same syntax but different semantics:

- CONNECT (Type 1) is used for remote unit of work.
- CONNECT (Type 2) is used for distributed unit of work.

For embedded SQL applications, the RDBCNNMTH parameter on the CRTSQLxxx commands determines the type of CONNECT. CLI and Java applications always use distributed unit of work because of the nature of the language.

Related reference

“Distributed relational database statements” on page 83

The statements included with the SQL language specifically support a distributed relational database.

Remote unit of work:

The *remote unit of work* facility provides for the remote preparation and processing of SQL statements.

An activation group at computer server A can connect to an application server at computer server B. Then, within one or more units of work, that activation group can process any number of static or dynamic SQL statements that reference objects at B. After ending a unit of work at B, the activation group can connect to an application server at computer server C, and so on.

Most SQL statements can be remotely prepared and processed with the following restrictions:

- All objects referenced in a single SQL statement must be managed by the same application server.
- All of the SQL statements in a unit of work must be run by the same application server.

Activation group states:

An activation group is in one of three states at any time.

The three states are:

- Connectable and connected
- Unconnectable and connected
- Connectable and unconnected

The following figure shows the state transitions:

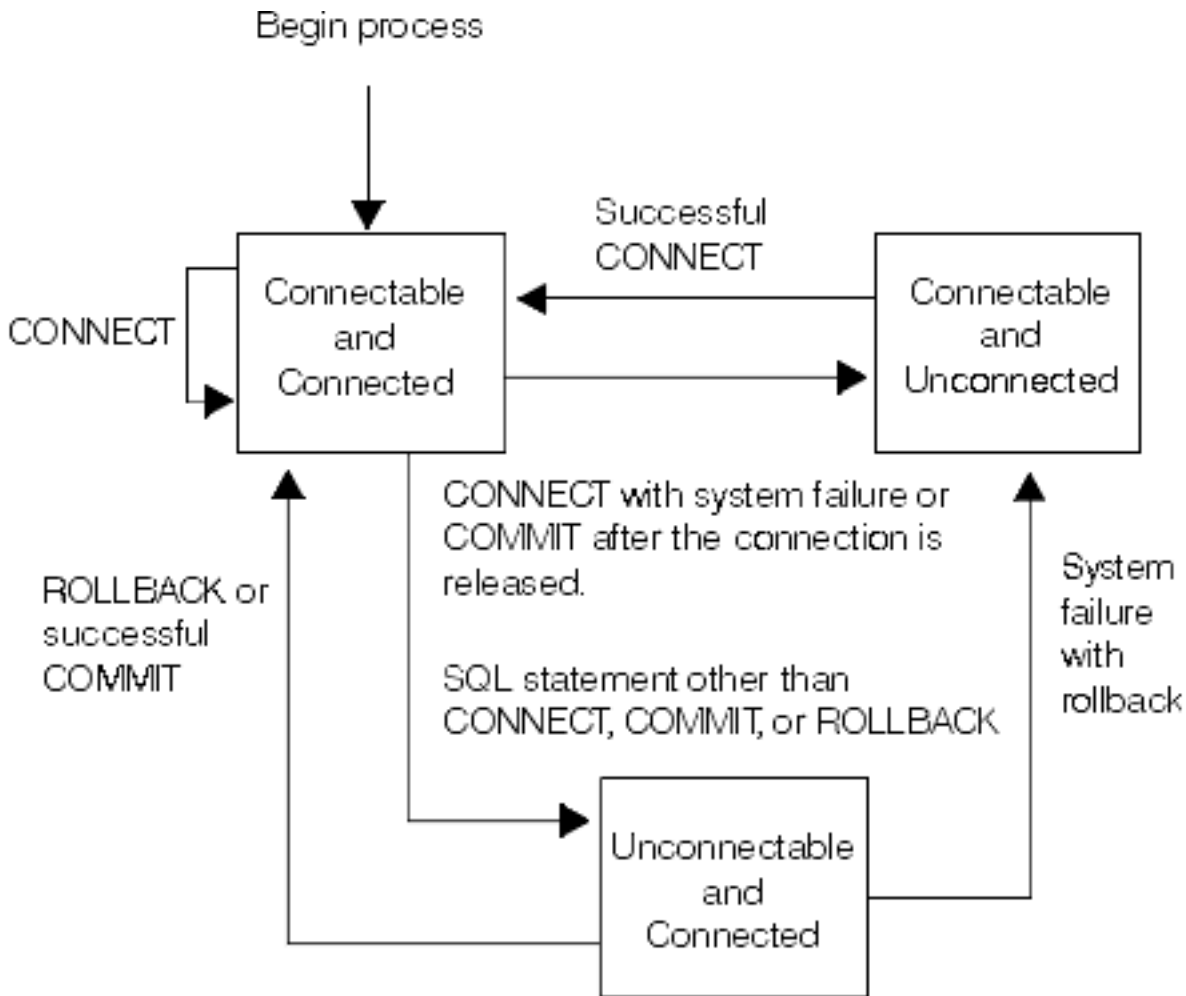


Figure 9. Remote unit of work activation group connection state transition

The initial state of an activation group is *connectable* and *connected*. The application server to which the activation group is connected is determined by the RDB parameter on the CRTSQLxxx and STRSQL commands and might involve an implicit CONNECT operation. An implicit CONNECT operation cannot occur if an implicit or explicit CONNECT operation has already successfully or unsuccessfully occurred. Thus, an activation group cannot be implicitly connected to an application server more than once.

Connectable and connected state:

An activation group is connected to an application server and CONNECT statements can be executed. The activation group enters this state when it completes a rollback or successful commit from the unconnectable and connected state, or a CONNECT statement is successfully executed from the connectable and unconnected state.

Unconnectable and connected state:

An activation group is connected to an application server, but a CONNECT statement cannot be successfully executed to change application servers. The activation group enters this state from the connectable and connected state when it executes any SQL statement other than CONNECT, COMMIT, or ROLLBACK.

Connectable and unconnected state:

An activation group is not connected to an application server. The only SQL statement that can be executed is CONNECT.

The activation group enters this state when:

- The connection was previously released and a successful COMMIT is executed.
- The connection is disconnected using the SQL DISCONNECT statement.
- The connection was in a connectable state, but the CONNECT statement was unsuccessful.

Consecutive CONNECT statements can be executed successfully because CONNECT does not remove the activation group from the connectable state. A CONNECT to the application server to which the activation group is currently connected is executed like any other CONNECT statement.

CONNECT cannot run successfully when it is preceded by any SQL statement other than CONNECT, COMMIT, DISCONNECT, SET CONNECTION, RELEASE, or ROLLBACK (unless running with COMMIT(*NONE)). To avoid an error, execute a commit or rollback operation before a CONNECT statement is executed.

Distributed unit of work:

The application-directed distributed unit of work facility also provides for the remote preparation and execution of SQL statements in the same fashion as remote unit of work.

Like remote unit of work, an activation group at computer server A can connect to an application server at computer server B and execute any number of static or dynamic SQL statements that reference objects at B before ending the unit of work. All objects referenced in a single SQL statement must be managed by the same application server. However, unlike remote unit of work, any number of application servers can participate in the same unit of work. A commit or rollback operation ends the unit of work.

Activation group states:

An activation group is always in the *connected* or *unconnected* state and has a set of zero or more connections. Each connection of an activation group is uniquely identified by the name of the application server of the connection.

An SQL connection is always in one of the following states:

- Current and held
- Current and released
- Dormant and held
- Dormant and released

Initial state of an activation group:

An activation group is initially in the connected state and has exactly one connection. The initial state of a connection is *current and held*.

The following figure shows the state transitions:

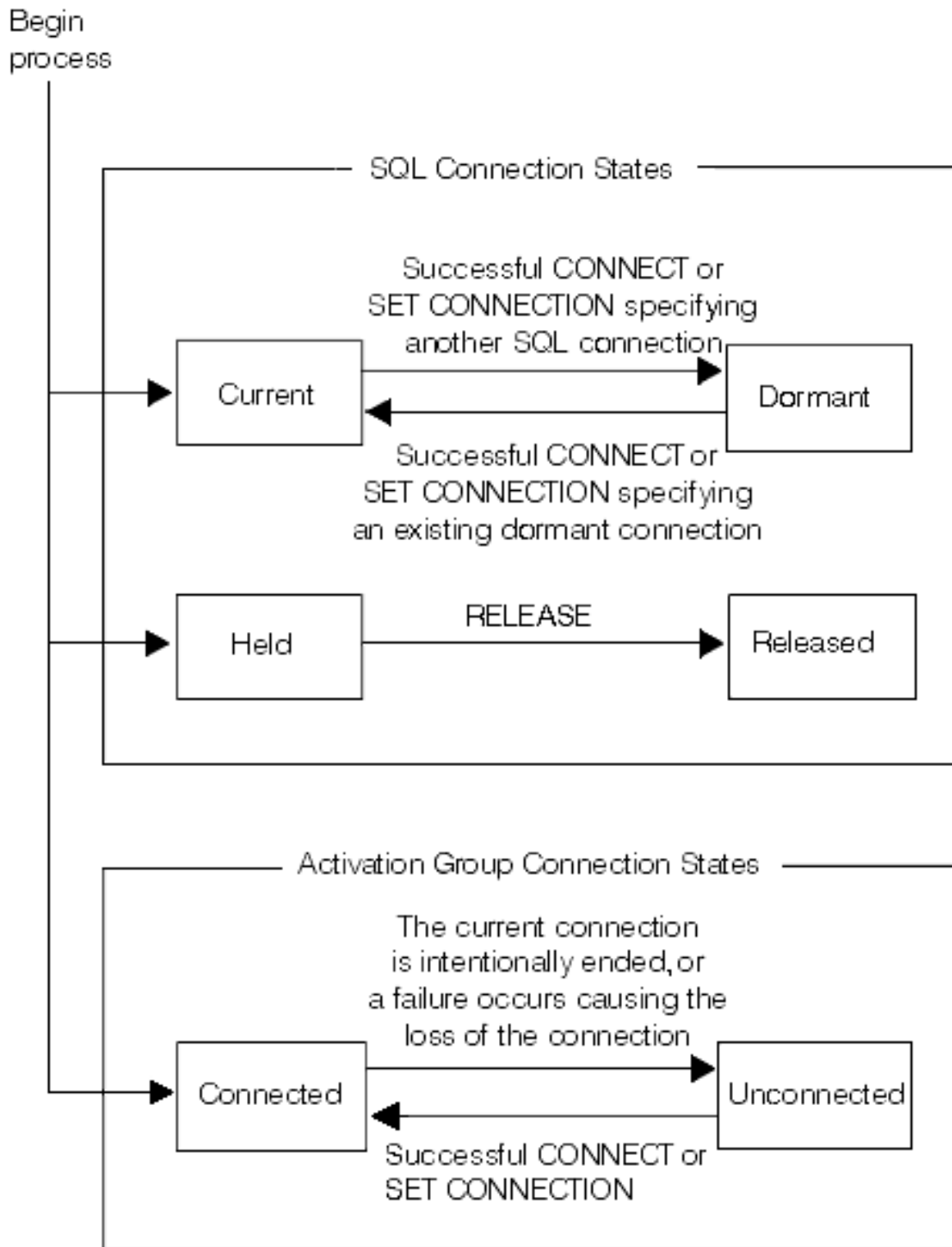


Figure 10. Application-directed distributed unit of work connection and activation group connection state transitions

Connection states:

This topic discusses the different connection states and ways to change them.

If an application processes a CONNECT statement and the server name is known to the application requester and is not in the set of existing connections of the activation group, then:

- The current connection is placed in the dormant state and held state.
- The server name is added to the set of connections and the new connection is placed in the current and held state.

If the server name is already in the set of existing connections of the activation group, an error occurs.

A connection in the dormant state is placed in the current state using the SET CONNECTION statement. When a connection is placed in the current state, the previous current connection, if any, is placed in the dormant state. No more than one connection in the set of existing connections of an activation group can be current at any time. Changing the state of a connection from current to dormant or from dormant to current has no effect on its held or released state.

A connection is placed in the released state by the RELEASE statement. When an activation group executes a commit operation, every released connection of the activation group is ended. Changing the state of a connection from held to released has no effect on its current or dormant state. Thus, a connection in the released state can still be used until the next commit operation. There is no way to change the state of a connection from released to held.

Activation group connection states:

A different application server can be established by the explicit or implicit execution of a CONNECT statement.

The following rules apply:

- An activation group cannot have more than one connection to the same application server at the same time.
- When an activation group executes a SET CONNECTION statement, the specified location name must be an existing connection in the set of connections of the activation group.
- When an activation group executes a CONNECT statement, the specified server name must not be an existing connection in the set of connections of the activation group.

If an activation group has a current connection, the activation group is in the *connected* state.

The CURRENT SERVER special register contains the name of the application server of the current connection. The activation group can execute SQL statements that refer to objects managed by that application server.

An activation group in the unconnected state enters the connected state when it successfully executes a CONNECT or SET CONNECTION statement.

If an activation group does not have a current connection, the activation group is in the *unconnected* state. The CURRENT SERVER special register contents are equal to blanks. The only SQL statements that can be executed are CONNECT, DISCONNECT, SET CONNECTION, RELEASE, COMMIT, and ROLLBACK.

An activation group in the connected state enters the unconnected state when its current connection is intentionally ended or the execution of an SQL statement is unsuccessful because of a failure that causes a rollback operation at the application server and loss of the connection. Connections are intentionally ended when an activation group successfully executes a commit operation and the connection is in the released state, or when an application process successfully executes the DISCONNECT statement.

When a connection is ended:

When a connection is ended, all resources that were acquired by the activation group through the connection and all resources that were used to create and maintain the connection are no longer allocated.

For example, when the activation group executes a RELEASE statement, any open cursors will be closed when the connection is ended during the next commit operation.

A connection can also be ended as a result of a communications failure in which case the activation group is placed in the unconnected state. All connections of an activation group are ended when the activation group ends.

Run with both RUW and DUW connection management:

Programs compiled with remote unit of work (RUW) connection management can be called by programs compiled with distributed unit of work (DUW) connection management. SET CONNECTION, RELEASE, and DISCONNECT statements can be used by the program compiled with RUW connection management to work with any of the active connections.

However, when a program compiled with DUW connection management calls a program compiled with RUW connection management, CONNECTs that are performed in the program compiled with RUW connection management will attempt to end all active connections for the activation group as part of the CONNECT.

Such CONNECTs will fail if the conversation used by active connections uses protected conversations. Furthermore, when protected conversations were used for inactive connections and the DDMCNV job attribute is *KEEP, these unused DDM conversations will also cause the connections in programs compiled with RUW connection management to fail. To avoid this situation, run with DDMCNV(*DROP) and perform a RELEASE and COMMIT prior to calling any programs compiled with RUW connection management that perform CONNECTs.

Likewise, when creating packages for programs compiled with DUW connection management after creating a package for a program compiled with RUW connection management, either run with DDMCNV(*DROP) or perform a RCLDDMCNV after creating the package for the programs compiled with DUW connection management.

Programs compiled with DUW connection management can also be called by programs compiled with RUW connection management. When the program compiled with DUW connection management performs a CONNECT, the connection performed by the program compiled with RUW connection management is not disconnected. This connection can be used by the program compiled with DUW connection management.

Implicit connection management for the default activation group:

The application requester can implicitly connect to an application server.

Implicit connection occurs when the application requester detects the first SQL statement is being issued by the first active SQL program for the default activation group and the following items are true:

- The SQL statement being issued is not a CONNECT statement with parameters.
- SQL is not active in the default activation group.

For a distributed program, the implicit connection is to the relational database specified on the RDB parameter. For a nondistributed program, the implicit connection is to the local relational database.

SQL ends any active connections in the default activation group when SQL becomes not active. SQL becomes not active when the application requester detects that the first active SQL program for the process has ended and the following conditions are all met:

- There are no pending SQL changes
- There are no connections using protected conversations
- A SET TRANSACTION statement is not active
- No programs that were precompiled with CLOSQLCSR(*ENDJOB) were run

If there are pending changes, protected conversations, or an active SET TRANSACTION statement, then SQL is placed in the exited state. If programs precompiled with CLOSQLCSR(*ENDJOB) were run, then SQL will remain active for the default activation group until the job ends.

- At the end of a unit of work if SQL is in the exited state. This occurs when you issue a COMMIT or ROLLBACK command outside of an SQL program.
- At the end of a job.

Implicit connection management for nondefault activation groups:

The application requester can implicitly connect to an application server.

Implicit connection occurs when the application requester detects the first SQL statement issued for the activation group and it is not a CONNECT statement with parameters.

For a distributed program, the implicit connection is made to the relational database specified on the RDB parameter. For a nondistributed program, the implicit connection is made to the local relational database.

Implicit disconnection can occur at the following parts of a process:

- When the activation group ends, if commitment control is not active, activation group level commitment control is active, or the job level commitment definition is at a unit of work boundary. If the job level commitment definition is active and not at a unit of work boundary then SQL is placed in the exited state.
- If SQL is in the exited state, when the job level commitment definition is committed or rolled back.
- At the end of a job.

The following example program is not distributed (no connection is required). It is a program run at a Spiffy Corporation regional office to gather local repair information into a report.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 250.

```
CRTSQLxxx PGM(SPIFFY/FIXTOTAL) COMMIT(*CHG) RDB(*NONE)
```

```
PROC: FIXTOTAL;
.
.
.
SELECT * INTO :SERVICE      A
        FROM REPAIRTOT;
EXEC SQL
COMMIT;
.
.
.
END FIXTOTAL;
```

A Statement run on the local relational database

Another program, such as the following example, can gather the same information from Spiffy dealerships in the Kansas City region. This is an example of a distributed program that is implicitly connected and disconnected:

```
CRTSQLxxx PGM(SPIFFY/FIXES) COMMIT(*CHG) RDB(KC101) RDBCNNMTH(*RUW)
```

```
PROC: FIXES;  
.  
.  
EXEC SQL  
  SELECT * INTO :SERVICE      B  
  FROM SPIFFY.REPAIR1;  
  
EXEC SQL C  
  COMMIT;  
.  
.  
END FIXES; D
```

- B** Implicit connection to application server (AS). The statement runs on the AS.
- C** End of unit of work. The application requester (AR) is placed in a connectable and connected state if the COMMIT is successful.
- D** Implicit disconnection at the end of the SQL program.

Explicit connection management:

The CONNECT statement is used to explicitly connect an application requester (AR) to an identified application server (AS). This SQL statement can be embedded within an application program or you can issue it using Interactive SQL.

The CONNECT statement is used with a TO or RESET clause. A CONNECT statement with a TO clause allows you to specify connection to a particular AS relational database. The CONNECT statement with a RESET clause specifies connection to the local relational database.

When you issue (or the program issues) a CONNECT statement with a TO or RESET clause, the AS identified must be described in the relational database directory. The AR must also be in a connectable state for the CONNECT statement to be successful.

The CONNECT statement has different effects depending on the connection management method you use. For RUW connection management, the CONNECT statement has the following effects:

- When a CONNECT statement with a TO or RESET clause is successful, the following events take place:
 - Any open cursors are closed, any prepared statements are discarded, and any held resources are released from the previous AS if the application process was placed in the connectable state through the use of COMMIT HOLD or ROLLBACK HOLD SQL statement, or if the application process is running COMMIT(*NONE).
 - The application process is disconnected from its previous AS, if any, and connected to the identified AS.
 - The name of the AS is placed in the Current Server special register.
 - Information that identifies the system module that returned the error is placed in the SQLERRP field of the SQL communication area (SQLCA) or in DB2_MODULE_DETECTING_ERROR of the SQL diagnostic area.
- If the CONNECT statement is unsuccessful for any reason, the application remains in the connectable but unconnected state. An application in the connectable but unconnected state can only run the CONNECT statement.
- Consecutive CONNECT statements can be run successfully because CONNECT does not remove the AR from the connectable state. A CONNECT to the AS to which the AR is currently connected is run like any other CONNECT statement.

- If running with commitment control, the CONNECT statement cannot run successfully when it is preceded by any SQL statement other than CONNECT, SET CONNECTION, COMMIT, ROLLBACK, DISCONNECT, or RELEASE. To avoid an error, perform a COMMIT or ROLLBACK operation before a CONNECT statement is run. If running without commitment control, the CONNECT statement is always allowed.

For DUW connection management, the CONNECT statement has the following effects:

- When a CONNECT statement with a TO or RESET clause is successful, the following events take place:
 - The name of the AS is placed in the Current Server special register.
 - Information that identifies the system module that returned the error is placed in the SQLERRP field of the SQL communication area (SQLCA) or in DB2_MODULE_DETECTING_ERROR of the SQL diagnostic area.
 - Information on the type of connection is also placed into the SQLCA and SQL diagnostic area. Encoded in these is the following information:
 - Whether the application is in a connected or unconnected state can be found in SQLERRD(5) in the SQLCA or in DB2_CONNECTION_STATE in the SQL diagnostic area.
 - Whether a remote connection uses a protected or unprotected conversation is found in SQLERRD(4) in the SQLCA or in DB2_CONNECTION_TYPE in the SQL diagnostic area.
 - Whether the connection is always read-only, always capable of updates, or whether the ability to update can change between each unit of work is found in SQLERRD(4) in the SQLCA or in DB2_CONNECTION_STATUS in the SQL diagnostic area.

See the SQL programming topic for more information about SQLERRD fields in the SQLCA and about connection information in the SQL diagnostic area.
- If the CONNECT statement with a TO or RESET clause is unsuccessful because the AR is not in the connectable state or the *server-name* is not listed in the local relational database directory, the connection state of the AR is unchanged.
- A connection to a currently connected AS results in an error.
- A connection without a TO or RESET clause can be used to obtain information about the current connection. This includes the following information:
 - Information that identifies the system module that returned the status is placed in the SQLERRP field of the SQL communication area (SQLCA) or in DB2_MODULE_DETECTING_ERROR of the SQL diagnostic area.
 - Other status information is described in the previous paragraphs discussing the contents of SQLERRD(4) and SQLERRD(5) and the corresponding information in the SQL diagnostic area.

It is a good practice for the first SQL statement run by an application process to be the CONNECT statement. However, when you have CONNECT statements embedded in your program you might want to dynamically change the AS name if the program connects to more than one AS. If you are going to run the application at multiple servers, you can specify the CONNECT statement with a host variable as shown here, so that the program can be passed the relational database name.

```
CONNECT TO : host-variable
```

Without CONNECT statements, all you need to do when you change the AS is to recompile the program with the new relational database name.

The following example shows two forms of the CONNECT statement (1 and 2) in an application program:

```
CRTSQLxxx PGM(SPIFFY/FIXTOTAL) COMMIT(*CHG) RDB(KC105)
```

```
PROC: FIXTOTAL;
EXEC SQL CONNECT TO KC105; 1

EXEC SQL
```

```

SELECT * INTO :SERVICE
      FROM REPAIRTOT;

EXEC SQL COMMIT;

EXEC SQL CONNECT TO MPLS03 USER :USERID USING :PW; 2

EXEC SQL SELECT ...

EXEC SQL COMMIT;

END FIXTOTAL;

```

The example (2) shows the use of the USER/USING form of the CONNECT statement. You must specify the user ID and password with host variables when this form of the CONNECT statement is embedded in a program. If you are using TCP/IP, a user ID and password can be extracted from a security object at connect time if you have used the Add Server Authentication Entry (ADDSVRAUTE) command with the appropriate parameters to store them.

The following example shows both CONNECT statement forms in Interactive SQL. Note that the password must be enclosed in single quotation marks.

```

Type SQL statement, press Enter.
Current connection is to relational database (RDB) KC105.
CONNECT TO KC000_____
COMMIT_____
==> CONNECT TO MPLS03 USER JOE USING 'X47K'_____
_____
_____

```

Related concepts

“Use the relational database directory” on page 29

The i5/OS licensed program uses the relational database directory to define the relational database names that can be accessed by applications running on an iSeries server, to specify if the connection uses Systems Network Architecture (SNA) or IP, and to associate these relational database names with their corresponding network parameters.

Related reference

“Control DDM conversations” on page 107

The term *connection* in this topic collection refers to the concept of an SQL connection. An SQL connection lasts from the time an explicit or implicit SQL CONNECT is done until the logical SQL connection is terminated by such means as an SQL DISCONNECT, or a RELEASE followed by a COMMIT.

SQL specific to distributed relational database and SQL CALL

During the precompile process of a distributed DB2 Universal Database for iSeries application, the i5/OS licensed program might build SQL packages to be run on an application server (AS).

After it is compiled, a distributed SQL program and package must be compatible with the servers that are being used as application receivers and application servers. The Prepare distributed relational database programs topic gives you more information about the changes to the precompile process and the addition of SQL packages.

This topic collection gives an overview of the SQL statements that are used with distributed relational database support and some things for you to consider about coexistence with other servers.

Related concepts

“Prepare distributed relational database programs” on page 90

When you write a program using SQL, you can embed the SQL statements in a host program.

Related information

SQL reference

SQL programming

Distributed relational database statements:

The statements included with the SQL language specifically support a distributed relational database.

These statements include:

- CONNECT
- SET CONNECTION
- RELEASE
- DISCONNECT
- DROP PACKAGE
- GRANT EXECUTE ON PACKAGE
- REVOKE EXECUTE ON PACKAGE

The SQL CALL statement can be used locally, but its primary purpose is to allow a procedure to be called on a remote server.

Related concepts

“Connect to a distributed relational database” on page 73

What makes a distributed relational database application *distributed* is its ability to connect to a relational database on another server.

“Authority to distributed relational database objects” on page 68

You can use either the SQL GRANT and REVOKE statements or the control language (CL) Grant Object Authority (GRTOBJAUT) and Revoke Object Authority (RVKOBJAUT) commands to grant and revoke a user’s authority to relational database objects.

“Work with SQL packages” on page 97

An SQL package is an SQL object used specifically by distributed relational database applications. It contains control structures for each SQL statement that accesses data on an application server (AS).

SQL CALL statement (stored procedures):

The SQL CALL statement is not actually specific to distributed relational databases, but a discussion of it is included here because its main value is in distributing application logic and processing.

Result sets can be generated in the stored procedure by opening one or more SQL cursors associated with SQL SELECT statements. In addition, a maximum of one array result set can also be returned. For more information about writing stored procedures that return result sets, see the descriptions of the SET RESULT SETS and CREATE PROCEDURE statements in the SQL reference topic.

The CALL statement provides a capability in a DRDA environment much like the Remote Procedure Call (RPC) mechanism does in the Open Software Foundation (OSF) Distributed Computing Environment (DCE). In fact, an SQL CALL to a program on a remote relational database actually is a remote procedure call. This type of RPC has certain advantages; for instance, it does not require the compilation of interface definitions, nor does it require the creation of stub programs.

You might want to use SQL CALL, or *stored procedures*, as the technique is sometimes called, for the following reasons:

- To reduce the number of message flows between the application requester (AR) and application server (AS) to perform a given function. If a set of SQL operations are to be run, it is more efficient for a program at the server to contain the statements and interconnecting logic.

- To allow native database operations to be performed at the remote location.
- To perform nondatabase operations (for example, sending messages or performing data queue operations) using SQL.

Note: Unlike database operations, these operations are not protected by commitment control by the server.

- To access server APIs on a remote server.

A stored procedure and application program can run in the same or different activation groups. It is recommended that the stored procedure be compiled with ACTGRP(*CALLER) specified to achieve consistency between the application program at the AR and the stored procedure at the AS. If the stored procedure is designed to return result sets, then you should not create it to run in a *NEW activation group. If you do, the cursors associated with the result sets might be prematurely closed when the procedure returns to the caller and the activation group is destroyed.

When a stored procedure is called that issues an inquiry message, the message is sent to the QSYSOPR message queue. The stored procedure waits for a response to the inquiry message. To have the stored procedure respond to the inquiry message, use the Add Reply List Entry (ADDRPYLE) command and specify *SYSRPLY on the INQMSGRPY parameter of the Change Job (CHGJOB) command in the stored procedure.

When a stored procedure and an application program run under different commitment definitions, the COMMIT and ROLLBACK statements in the application program only affect its own commitment definition. You must commit the changes in the stored procedure by other means.

Related reference

“Testing and debugging” on page 94

Testing and debugging distributed SQL programs is similar to testing and debugging local SQL programs, but certain aspects of the process are different.

Related information

SQL reference

Add Reply List Entry (ADDRPYLE) command

Change Job (CHGJOB) command

DB2 Universal Database for iSeries CALL considerations:

Stored procedures written in C that are invoked on some platforms cannot use argc and argv as parameters (that is, they cannot be of type main()). This differs from iSeries stored procedures which must use argc and argv.

For examples of stored procedures for DB2 UDB platforms, see the \SQLLIB\SAMPLES (or /sqllib/samples) subdirectory. Look for outsrv.sqc and outcli.sqc in the C subdirectory.

For UDB stored procedures called by an iSeries server, make sure that the procedure name is in uppercase. iSeries server currently folds procedure names to uppercase. This means that a procedure on the UDB server, having the same procedure name but in lowercase, will not be found. For stored procedures on an iSeries server, the procedure names are in uppercase.

Stored procedures on the iSeries server cannot have a COMMIT in them when they are created to run in the same activation group as the calling program (the proper way to create them). In UDB, a stored procedure is allowed to have a COMMIT, but the application designer should be aware that there is no knowledge on the part of DB2 UDB for iSeries that the commit occurred.

DB2 Universal Database for iSeries coexistence:

When you write and maintain programs for a distributed relational database using the SQL language, you need to consider the other servers in the distributed relational database network.

The program you are writing or maintaining might have to be compatible with the following items:

- Other iSeries servers
- Previous iSeries server releases
- Servers that are not iSeries servers

Remember that the SQL statements in a distributed SQL program run on the application server (AS). Even though the program runs on the application requester (AR), the SQL statements are in the SQL package to be run on the AS. Those statements must be supported by the AS and be compatible with the collections, tables, and views that exist on the AS. Also, the users who run the program on the AR must be authorized to the SQL package and other SQL objects on the AS.

You can convert a non-distributed embedded SQL program to a distributed embedded SQL program by creating the program again using the CRTSQLxxx command and specifying the relational database name (RDB parameter) for an AS. This compiles the program again using the distributed relational database support in DB2 Universal Database for iSeries and creates the SQL package needed on the AS.

You can write DB2 UDB for iSeries programs that run on application servers that are not iSeries servers and these other platforms might support more or fewer SQL functions. Statements that are not supported on the DB2 UDB for iSeries AR can be used and compiled on the server when the AS supports the function. SQL programs written to run on an iSeries server AS only provide the level of support described in this topic collection. See the support documentation for the other systems to determine the level of function they provide.

End DRDA units of work

You should be careful about ending SQL programs with uncommitted work. When a program ends with uncommitted work, the connection to the relational database remains active.

However, in some cases involving programs running in system-named activation groups, the system performs an automatic commit operation when the program ends.

This behavior differs from that of other systems because in the i5/OS operating system, COMMITs and ROLLBACKs can be used as commands from the command line or in a CL program. However, the preceding scenario can lead to unexpected results in the next SQL program run, unless you plan for the situation. For example, if you run Interactive SQL next (STRSQL command), the interactive session starts up in the state of being connected to the previous application server (AS) with uncommitted work. As another example, if following the preceding scenario, you start a second SQL program that does an implicit connection, an attempt is made to find and run a package for it on the AS that was last used. This might not be the AS that you intended. To avoid these surprises always commit or rollback the last unit of work before ending any application program.

Stored procedures, user-defined functions, and commitment control

When an application such as Interactive SQL is running without commitment control active (COMMIT(*NONE)) over a DRDA connection, it is possible for an invoked stored procedure or user-defined function (UDF) to start commitment control on an iSeries server.

This results in a mismatch in commitment control between the client and the server, causing the possibility of uncommitted updates when the application terminates.

You should avoid this situation. If, however, you choose to implement it, one solution is for the stored procedure or UDF running under commitment control to explicitly commit all of its database updates. If that is not done, the server will detect the pending updates during the disconnection process and automatically commit the pending work.

Coded character set identifier

Support for the national language of any country requires the proper handling of a minimum set of characters.

A cross-system support for the management of character information is provided with the IBM Character Data Representation Architecture (CDRA). CDRA defines the coded character set identifier (CCSID) values to identify the code points used to represent characters, and to convert these codes (character data), as needed to preserve their meanings.

The use of an architecture such as CDRA and associated conversion protocols is important in the following situations:

- More than one national language version is installed on the iSeries server.
- Multiple iSeries servers are sharing data between systems in different countries with different primary national language versions.
- iSeries servers and non-iSeries servers are sharing data between systems in different countries with different primary national language versions.

Tagging is the primary means to assign meaning to coded graphic characters. The tag might be in a data structure that is associated with the data object (explicit tagging), or it might be inherited from objects such as the job or the system itself (implicit tagging).

DB2 Universal Database for iSeries tags character columns with CCSIDs. A CCSID is a 16-bit number identifying a specific set of encoding scheme identifiers, character set identifiers, code page identifiers, and additional coding-related information that uniquely identifies the coded graphic character representation used. When running applications, data is not converted when it is sent to another system; it is sent as tagged along with its CCSID. The receiving job automatically converts the data to its own CCSID if it is different from the way the data is tagged.

The CDRA has defined the following range of values for CCSIDs.

- 00000** Use next hierarchical CCSID
- 00001 through 28671**
IBM-registered CCSIDs
- 28672 through 65533**
Reserved
- 65534** Refer to lower hierarchical CCSID
- 65535** No conversion done

See *Character Data Representation Architecture - Level 1, Registry* for a complete list of the CDRA CCSIDs.

The following illustration shows the parts of a CCSID.

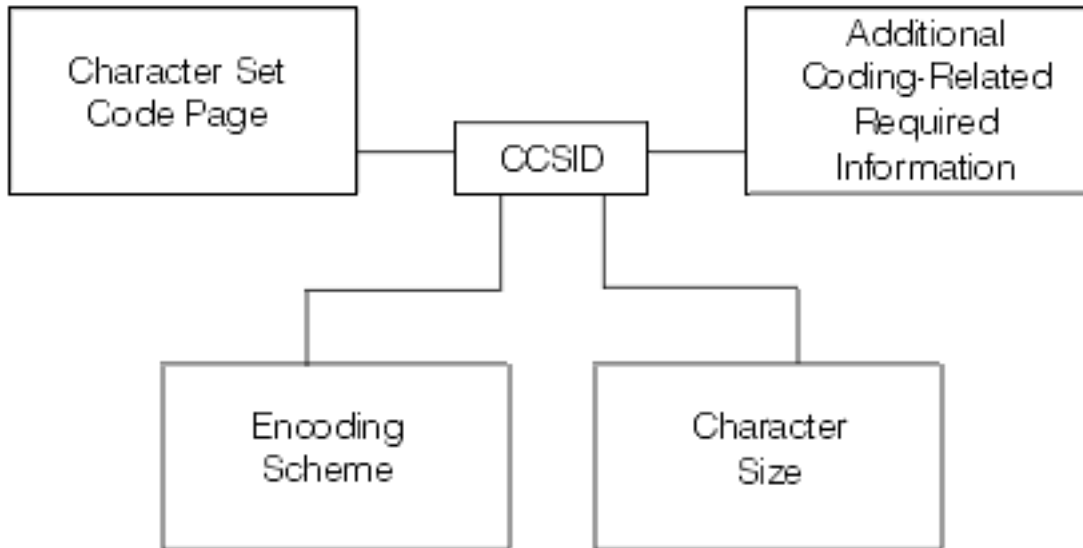


Figure 11. Coded character set identifier (CCSID)

Related concepts

SQL programming

“DRDA and CDRA support” on page 9

One of the interesting possibilities in a distributed relational database is that the database might not only span different types of computers, but those computers might be in different countries or regions. The same servers, such as iSeries servers, can encode data differently depending on the language used on the server.

Related information

i5/OS globalization

SQL reference

iSeries server support:

The default CCSID for a job on the iSeries server is specified using the Change Job (CHGJOB) command.

If a CCSID is not specified in this way, the job CCSID is obtained from the CCSID attribute of the user profile. If a CCSID is not specified on the user profile, the system gets it from the QCCSID system value. This QCCSID value is initially set to 65535. If your server is in a distributed relational database with unlike systems, it might not be able to use CCSID 65535.

All control information that flows between the application requester (AR) and application server (AS) is in CCSID 500 (a DRDA standard). This is information such as collection names, table names, and some descriptive text. Using variant characters for control information causes these names to be converted, which can affect performance. Package names are also sent in CCSID 500. Using variant characters in a package name causes the package name to be converted. This means the package is not found at run time.

After a job has been initiated, you can change the job CCSID by using the Change Job (CHGJOB) command. To do this:

1. Enter the Work with Job (WRKJOB) command to get the Work with Jobs display.
2. Select option 2 (Display job definition attributes). This locates the current CCSID value so you can reset the job to its original CCSID value later.
3. Enter the Change Job (CHGJOB) command with the new CCSID value.

The new CCSID value is reflected in the job immediately. However, if the job CCSID you change is an AR job, the new CCSID does not affect the work being done until the next CONNECT.

Attention: If you change the CCSID of an AS job, the results cannot be predicted.

Source files are tagged with the job CCSID if a CCSID is not explicitly specified on the Create Source Physical File (CRTSRCPF) or Create Physical File (CRTPF) command for source files. Externally described database files and tables are tagged with the job CCSID if a CCSID is not explicitly specified in data description specification (DDS), in interactive data definition utility (IDDU), or in the CREATE TABLE SQL statement.

For source and externally described files, if the job CCSID is 65535, the default CCSID based on the language of the operating system is used. Program described files are tagged with CCSID 65535. Views are tagged with the CCSID of its corresponding table tag or column-level tags. If a view is defined over several tables, it is tagged at the column level and assumes the tags of the underlying columns. Views cannot be explicitly tagged with a CCSID. The system automatically converts data between the job and the table if the CCSIDs are not equal and neither of the CCSIDs is equal to 65535.

When you change the CCSID of a tagged table, it cannot be tagged at the column level or have views defined on it. To change the CCSID of a tagged table, use the Change Physical File (CHGPF) command. To change a table with column-level tagging, you must create it again and copy the data to a new table using FMT(*MAP) on the Copy File (CPYF) command. When a table has one or more views defined, you must follow these steps to change the table:

1. Save the view and table along with their access paths.
2. Delete the views.
3. Change the table.
4. Restore the views and their access paths over the created table.

Source files and externally described files migrated to DB2 Universal Database for iSeries that are not tagged or are implicitly tagged with CCSID 65535 will be tagged with the default CCSID based on the language of the operating system installed. This includes files that are on the system when you install a new release and files that are restored to DB2 UDB for iSeries.

All data that is sent between an AR and an AS is sent not converted. In addition, the CCSID is also sent. The receiving job automatically converts the data to its own CCSID if it is different from the way the data is tagged. For example, consider the following application that is run on a dealership system, KC105.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 250.

```
CRTSQLxxx PGM(PARTS1) COMMIT(*CHG) RDB(KC000)
```

```
PROC: PARTS1;
  .
  .
EXEC SQL
  SELECT * INTO :PARTAVAIL
          FROM INVENTORY
          WHERE ITEM = :PARTNO;
  .
  .
END PARTS1;
```

In the preceding example, the local system (KC105) has the QCCSID system value set at CCSID 37. The remote regional center (KC000) uses CCSID 937 and all its tables are tagged with CCSID 937. CCSID processing takes place as follows:

- The KC105 system sends an input host variable (:PARTNO) in CCSID 37. (The DECLARE VARIABLE SQL statement can be used if the CCSID of the job is not appropriate for the host variable.)
- The KC000 system converts :PARTNO to CCSID 937, selects the required data, and sends the data back to KC105 in CCSID 937.
- When KC105 gets the data, it converts the data to CCSID 37 and places it in :PARTAVAIL for local use.

Related concepts

“User FAQs” on page 219

This topic collection concentrates on describing iSeries support for distributed relational databases in a network of iSeries servers (a *like* environment).

Related reference

Change Job (CHGJOB) command

Change Physical File (CHGPF) command

Copy File (CPYF) command

Create Physical File (CRTPF) command

Create Source Physical File (CRTSRCPF) command

Work with Job (WRKJOB) command

Other DRDA data conversion

Sometimes, when you are doing processing on a remote system, your program might need to convert the data from one system so that it can be used on the other. Distributed Relational Database Architecture (DRDA) support on the iSeries server converts the data automatically between other systems that use DRDA support.

When a DB2 Universal Database for iSeries application requester (AR) connects to an application server (AS), it sends information that identifies its type. Likewise, the AS sends back information to the server that identifies its processor type (for example, S/390 host or iSeries server). The two systems then automatically convert the data between them as defined for this connection. This means that you do not need to program for architectural differences between systems.

Data conversion between IBM systems with DRDA support includes data types such as:

- Floating point representations
- Zoned decimal representations
- Byte reversal
- Mixed data types
- iSeries specific data types such as:
 - DBCS-only
 - DBCS-either
 - Integer with precision and scale

DDM files and SQL

You can use iSeries distributed data management (DDM) support to help you do some distributed relational database tasks within a program that also uses SQL distributed relational database support.

It might be faster, for example, for you to use DDM and the Copy File (CPYF) command to get a large number of records rather than an SQL FETCH statement. Also, DDM can be used to get external file descriptions of the remote system data brought in during compilation for use with the distributed relational database application. To do this, you need to use DDM as described in “Initial setup” on page 24.

The following example shows how you can add a relational database directory entry and create a DDM file so that the same job can be used on the application server (AS) and application requester (AR).

Note: Either both connections must be protected or both connections must be unprotected for the conversation to be shared.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 250.

Relational Database Directory:

```
ADDRDBDIRE    RDB(KC000) +
               RMTLOCNAME(KC000)
               TEXT('Kansas City regional database')
```

DDM File:

```
CRTDDMF FILE(SPIFFY/UPDATE)
         RMTFILE(SPIFFY/INVENTORY)
         RMTLOCNAME(KC000)
         TEXT('DDM file to update local orders')
```

Here is a sample program that uses both the relational database directory entry and the DDM file in the same job on the remote server:

```
CRTSQLxxx PGM(PARTS1) COMMIT(*CHG) RDB(KC000) RDBCNNMTH(*RUW)

PROC :PARTS1;
OPEN SPIFFY/UPDATE;
.
.
.
CLOSE SPIFFY/UPDATE;
.
.
.
EXEC SQL
  SELECT * INTO :PARTAVAIL
  FROM INVENTORY
  WHERE ITEM = :PARTNO;
EXEC SQL
  COMMIT;
.
.
.
END PARTS1;
```

Related concepts

Distributed data management

Related reference

Copy File (CPYF) command

Prepare distributed relational database programs

When you write a program using SQL, you can embed the SQL statements in a host program.

The host program is the program that contains the SQL statements, written in one of the host languages: the iSeries PL/I, ILE C/400, COBOL/400, ILE COBOL/400, FORTRAN/400, RPG/400, or ILE RPG/400 programming language. In a host program you use variables referred to as host variables. These are variables used in SQL statements that are identifiable to the host program. In RPG, this is called a *field name*; in FORTRAN, PL/I, and C, this is known as a *variable*; in COBOL, this is called a *data item*.

You can code your distributed DB2 Universal Database for iSeries programs in a way similar to the coding for a DB2 UDB for iSeries program that is not distributed. You use the host language to embed the SQL statements with the host variables. Also, like a DB2 UDB for iSeries program that is not distributed, a distributed DB2 UDB for iSeries program is prepared using the certain processes.

However, a distributed DB2 UDB for iSeries program also requires that an SQL package is created on the application server (AS) to access data.

This topic collection discusses these steps in the process, outlining the differences for a distributed DB2 UDB for iSeries program.

Precompile programs with SQL statements

You must precompile and compile an application program containing embedded SQL statements before you can run it. Precompiling such programs is done by an SQL precompiler.

The SQL precompiler scans each statement of the application program source and does the following things:

- Looks for SQL statements and for the definition of host variable names
- Verifies that each SQL statement is valid and free of syntax errors
- Validates the SQL statements using the description in the database
- Prepares each SQL statement for compilation in the host language
- Produces information about each precompiled SQL statement

Application programming statements and embedded SQL statements are the primary input to the SQL precompiler. The SQL precompiler assumes that the host language statements are syntactically correct. If the host language statements are not syntactically correct, the precompiler might not correctly identify SQL statements and host variable declarations.

The SQL precompile process produces a listing and a temporary source file member. It can also produce the SQL package depending on what is specified for the OPTION and RDB parameters of the precompiler command.

Related reference

“Compile an application program” on page 93

The DB2 Universal Database for iSeries precompiler automatically calls the host language compiler after successfully precompiling the program, unless the *NOGEN precompiler option is specified.

Listing:

The output listing is sent to the printer file specified by the PRTFILE parameter of the CRTSQLxxx command.

The following items are written to the printer file:

- Precompiler options
This is a list of all the options specified with the CRTSQLxxx command and the date the source member was last changed.
- Precompiler source
This output is produced if the *SOURCE option is used for non-ILE precompiles or if the OUTPUT(*PRINT) parameter is specified for ILE precompiles. It shows each precompiler source statement with its record number assigned by the precompiler, the sequence number (SEQNBR) you see when using the source entry utility (SEU), and the date the record was last changed.
- Precompiler cross-reference
This output is produced if *XREF was specified in the OPTION parameter. It shows the name of the host variable or SQL entity (such as tables and columns), the record number where the name is defined, what the name is defined, and the record numbers where the name occurs.
- Precompiler diagnostic list
This output supplies diagnostic messages, showing the precompiler record numbers of statements in error.

Temporary source file member:

Source statements processed by the precompiler are written to QSQLTEMP in the QTEMP library (QSQLTEMP1 in the QTEMP library for programs created using CRTSQLRPGI).

In your precompiler-changed source code, SQL statements have been converted to comments and calls to the SQL interface modules: QSQRROUTE, QSQLOPEN, QSQLCLSE, and QSQLCMIT. The name of the temporary source file member is the same as the name specified in the PGM parameter of CRTSQLxxx. This member cannot be changed before being used as input to the compiler.

QSQLTEMP or QSQLTEMP1 can be moved to a permanent library after the precompile, if you want to compile at a later time. If you change the records of the temporary source file member, the compile attempted later will fail.

SQL package creation:

An object called an SQL package can be created as part of the precompile process when the CRTSQLxxx command is compiled.

Related concepts

“Work with SQL packages” on page 97

An SQL package is an SQL object used specifically by distributed relational database applications. It contains control structures for each SQL statement that accesses data on an application server (AS).

Related reference

“Compile an application program” on page 93

The DB2 Universal Database for iSeries precompiler automatically calls the host language compiler after successfully precompiling the program, unless the *NOGEN precompiler option is specified.

“Bind an application” on page 93

Before you can run your application program, a relationship between the program and any referred-to tables and views must be established. This process is called binding.

Precompiler commands:

The DB2 UDB Query Manager and SQL Development Kit licensed program has seven precompiler commands, one for each of the host languages.

Host language	Command
iSeries PL/I	CRTSQLPLI
ILE C/400 language	CRTSQLCI
COBOL/400 language	CRTSQLCBL
ILE COBOL/400 language	CRTSQLCBLI
FORTRAN/400 language	CRTSQLFTN
RPG III (part of RPG/400 language)	CRTSQLRPG
ILE RPG/400 language	CRTSQLRPGI

A separate command for each language exists so each language can have parameters that apply only to that language. For example, the options *APOST and *QUOTE are unique to COBOL. They are not included in the commands for the other languages. The precompiler is controlled by parameters specified when it is called by one of the SQL precompiler commands. The parameters specify how the input is processed and how the output is presented.

You can precompile a program without specifying anything more than the name of the member containing the program source statements as the PGM parameter (for non-ILE precompiles) or the OBJ parameter (for ILE precompiles) of the CRTSQLxxx command. SQL assigns default values for all precompiler parameters (which might, however, be overridden by any that you explicitly specify).

The following list briefly describes parameters common to all the CRTSQLxxx commands that are used to support distributed relational database.

RDB

Specifies the name of the relational database where the SQL package option is to be created. If *NONE is specified, then the program or module is not a distributed object and the Create Structured Query Language Package (CRTSQLPKG) command cannot be used. The relational database name can be the name of the local database.

RDBCNNMTH

Specifies the type of semantics to be used for CONNECT statements: remote unit of work (RUW) or distributed unit of work (DUW) semantics.

SQLPKG

Specifies the name and library of the SQL package.

USER

Specifies the user name sent to the remote server when starting the conversation. This parameter is used only if a conversation is started as part of the precompile process.

PASSWORD

Specifies the password to be used on the remote server when starting the conversation. This parameter is used only if a conversation is started as part of the precompile process.

REPLACE

Specifies if any objects created as part of the precompile process should be able to replace an existing object.

The following example creates a COBOL program named INVENT and stores it in a library named SPIFFY. The SQL naming convention is selected, and every row selected from a specified table is locked until the end of the unit of recovery. An SQL package with the same name as the program is created on the remote relational database named KC000.

```
CRTSQLCBL PGM(SPIFFY/INVENT) OPTION(*SRC *XREF *SQL)
          COMMIT(*ALL) RDB(KC000)
```

Related reference

Create Structured Query Language Package (CRTSQLPKG) command

Compile an application program

The DB2 Universal Database for iSeries precompiler automatically calls the host language compiler after successfully precompiling the program, unless the *NOGEN precompiler option is specified.

The compiler command is run specifying the program name, source file name, precompiler created source member name, text, and user profile. Other parameters are also passed to the compiler, depending on the host language.

Related concepts

“Precompile programs with SQL statements” on page 91

You must precompile and compile an application program containing embedded SQL statements before you can run it. Precompiling such programs is done by an SQL precompiler.

Embedded SQL programming

Related reference

“SQL package creation” on page 92

An object called an SQL package can be created as part of the precompile process when the CRTSQLxxx command is compiled.

Bind an application

Before you can run your application program, a relationship between the program and any referred-to tables and views must be established. This process is called binding.

The result of binding is an access plan. The access plan is a control structure that describes the actions necessary to satisfy each SQL request. An access plan contains information about the program and about the data the program intends to use. For distributed relational database work, the access plan is stored in the SQL package and managed by the server along with the SQL package.

SQL automatically attempts to bind and create access plans when the result of a successful compile is a program or service program object. If the compile is not successful or the result of a compile is a module object, access plans are not created. If, at run time, the database manager detects that an access plan is not valid or that changes have occurred to the database that might improve performance (for example, the addition of indexes), a new access plan is automatically created. If the application server (AS) is not an iSeries server, then a bind must be done again using the Create Structured Query Language Package (CRTSQLPKG) command. binding does three things:

- Revalidates the SQL statements using the description in the database.

During the bind process, the SQL statements are checked for valid table, view, and column names. If a referred to table or view does not exist at the time of the precompile or compile, the validation is done at run time. If the table or view does not exist at run time, a negative SQLCODE is returned.

- Selects the access paths needed to access the data your program wants to process.

In selecting an access path, indexes, table sizes, and other factors are considered when SQL builds an access plan. The bind process considers all indexes available to access the data and decides which ones (if any) to use when selecting a path to the data.

- Attempts to build access plans.

If all the SQL statements are valid, the bind process builds and stores access plans in the program.

If the characteristics of a table or view your program accesses have changed, the access plan might no longer be valid. When you attempt to use an access plan that is not valid, the server automatically attempts to rebuild the access plan. If the access plan cannot be rebuilt, a negative SQLCODE is returned. In this case, you might have to change the program's SQL statements and reissue the CRTSQLxxx command to correct the situation.

For example, if a program contains an SQL statement that refers to COLUMNA in TABLEA and the user deletes and recreates TABLEA so that COLUMNA no longer exists, when you call the program, the automatic rebind is unsuccessful because COLUMNA no longer exists. You must change the program source and reissue the CRTSQLxxx command.

Related concepts

"Work with SQL packages" on page 97

An SQL package is an SQL object used specifically by distributed relational database applications. It contains control structures for each SQL statement that accesses data on an application server (AS).

Related reference

"SQL package creation" on page 92

An object called an SQL package can be created as part of the precompile process when the CRTSQLxxx command is compiled.

Create Structured Query Language Package (CRTSQLPKG) command

Testing and debugging

Testing and debugging distributed SQL programs is similar to testing and debugging local SQL programs, but certain aspects of the process are different.

More than one server will eventually be required for testing. If applications are coded so that the relational database names can easily be changed by recompiling the program, changing the input parameters to the program, or making minor modifications to the program source, most testing can be accomplished using a single server.

After the program has been tested against local data, the program is then made available for final testing on the distributed relational database network. Consider testing the application locally on the server that will be the application server (AS) when the application is tested over a remote connection, so that only the program needs to be moved when the testing moves into a distributed environment.

Debugging a distributed SQL program uses the same techniques as debugging a local SQL program. You use the Start Debug (STRDBG) command to start the debugger and to put the application in debug mode. You can add breakpoints, trace statements, and display the contents of variables.

However, to debug a distributed SQL program, you must specify the value of *YES for the UPDPROD parameter. This is because i5/OS distributed relational database support uses files in library QSYS and QSYS is a production library. This allows data in production libraries to be changed on the application requester (AR). Issuing the Start Debug (STRDBG) command on the AR only puts the AR job into debug mode, so your ability to manipulate data on the AS is not changed.

While in debug mode on the AR, informational messages are entered in the job log for each SQL statement run. These messages give information about the result of each SQL statement. A list of SQL return codes and a list of error messages for distributed relational database are provided in "Troubleshooting" on page 148.

Informational messages about how the server maximizes processing efficiency of SQL statements are also issued as a result of being in debug mode. Because any maximization occurs at the AS, these types of messages will not appear in the AR job log. To get this information, the AS job must be put in debug mode.

A relatively easy way to start debug mode on the server if you are using TCP/IP is to use the QRWOPTIONS data area. However, you cannot specify a specific program to debug with this facility. For details on setup, see QRWOPTIONS data area usage. The data area can be used not only to start debug, but to start job traces, request job logs, display job output and do other things as well. You can even do the QRWOPTIONS setup on an iSeries AR, and have the options shadowed to an iSeries server server.

If both the AR and AS are iSeries servers, and they are connected with APPC, you can use the Submit Remote Command (SBMRMTCMD) command to start the debug mode in an AS job. Create a DDM file as described in the Set up DDM files topic. The communications information in the DDM file must match the information in the relational database directory entry for the relational database being accessed. Then issue the command:

```
SBMRMTCMD CMD('STRDBG UPDPROD(*YES)') DDMFILE(ddmfile name)
```

The (SBMRMTCMD) command starts the AS job if it does not already exist and starts the debug mode in that job. Use one of the methods for monitoring relational database activity to examine the AS job log to find the job.

The following method for putting the AS job into debug mode works with any AR and a DB2 Universal Database for iSeries AS with certain restrictions. It depends on being able to pause after the application makes a connection to do setup. It also assumes that what you want to trace or otherwise debug occurs after the connection is established.

- Sign on to the AS and find the AS job.
- Issue the Start Service Job (STRSRVJOB) command from the interactive job (the job you are using to find the AS job) as shown:

```
STRSRVJOB (job-number/user-ID/job-name)
```

The job name for the (STRSRVJOB) command is the name of the AS job. Issuing this command lets you issue certain commands from your interactive job that affect the AS job. One of these commands is the Start Debug (STRDBG) command.

- Issue the (STRDBG) command using a value of *YES for the UPDPROD parameter in the interactive job. This puts the AS job into debug mode to produce debug messages on the AS job log.

To end this debug session, either end your interactive job by signing off or use the End Debug (ENDDDBG) command followed by the End Service Job (ENDSRVJOB) command.

Because the AS job must be put into debug before the SQL statements are run, the application might need to be changed to allow you time to set up debug on the AS. The AS job starts as a result of the application connecting to the AS. Your application can be coded to enter a wait state after connecting to the AS until debug is started on the AS.

If you can anticipate the prestart job that will be used for a TCP/IP connection before it occurs, such as when there is only one waiting for work and there is no interference from other clients, you do not need to introduce a delay.

Related concepts

“Monitor relational database activity” on page 99

You can rely on control language (CL) commands, all of which provide similar information, but in different ways, to give you a view of work on an iSeries server.

Related tasks

“Set up DDM files” on page 40

The implementation of DRDA support on the iSeries server uses Distributed Data Management (DDM) conversations for communications. Because of this, you can use DDM in conjunction with distributed relational database processing.

Related reference

End Debug (ENDDDBG) command

End Service Job (ENDSRVJOB) command

Start Debug (STRDBG) command

Start Service Job (STRSRVJOB) command

“SQL CALL statement (stored procedures)” on page 83

The SQL CALL statement is not actually specific to distributed relational databases, but a discussion of it is included here because its main value is in distributing application logic and processing.

Submit Remote Command (SBMRMTCMD) command

Program references:

When a program is created, the i5/OS licensed program stores information about all collections, tables, views, SQL packages, and indexes referred to in SQL statements in an SQL program.

You can use the Display Program References (DSPPGMREF) command to display all object references in the program. If the SQL naming convention is used, the library name is stored in one of the following ways:

- If the SQL name is fully qualified, the collection name is stored as the name qualifier.
- If the SQL name is not fully qualified, and the DFTRDBCOL parameter is not specified, the authorization ID of the statement is stored as the name qualifier.
- If the SQL name is not fully qualified, and the DFTRDBCOL parameter is specified, the collection name specified on the DFTRDBCOL parameter is stored as the name qualifier.

If the server naming convention is used, the library name is stored in one of the following ways:

- If the object name is fully qualified, the library name is stored as the name qualifier.
- If the object is not fully qualified, and the DFTRDBCOL parameter is not specified, *LIBL is stored.
- If the SQL name is not fully qualified, and the DFTRDBCOL parameter is specified, the collection name specified on the DFTRDBCOL parameter is stored as the name qualifier.

Related reference

Display Program References (DSPPGMREF) command

Work with SQL packages

An SQL package is an SQL object used specifically by distributed relational database applications. It contains control structures for each SQL statement that accesses data on an application server (AS).

These control structures are used by the AS at run time when the application program requests data using the SQL statement.

You must use a control language (CL) command to create an SQL package because there is no SQL statement for SQL package creation. You can create an SQL package in two ways:

- Using the CRTSQLxxx command with a relational database name specified in the RDB parameter.
- Using the Create SQL Package (CRTSQLPKG) command

Related reference

“Distributed relational database statements” on page 83

The statements included with the SQL language specifically support a distributed relational database.

“SQL package creation” on page 92

An object called an SQL package can be created as part of the precompile process when the CRTSQLxxx command is compiled.

“Bind an application” on page 93

Before you can run your application program, a relationship between the program and any referred-to tables and views must be established. This process is called binding.

Use the Create SQL Package (CRTSQLPKG) command

You can enter the Create SQL Package (CRTSQLPKG) command to create an SQL package from a compiled distributed relational database program. You can also use this command to replace an SQL package that was created previously.

A new SQL package is created on the relational database defined by the RDB parameter. The new SQL package has the same name and is placed in the same library as specified on the PKG parameter of the CRTSQLxxx command.

You do not need the DB2 UDB Query Manager and SQL Development Kit licensed program to create an SQL package on an application server (AS).

Related reference

“Default collection name” on page 72

You can specify a default collection name to be used by an SQL program by supplying this name for the DFTRDBCOL parameter on the CRTSQLxxx command when you precompile the program.

Create Structured Query Language Package (CRTSQLPKG) command

Manage an SQL package

After an SQL package is created, you can manage it the same way you manage other objects on the iSeries server, with some restrictions.

You can save and restore it, send it to other servers, and grant and revoke a user’s authority to the package. You can also delete it by entering the Delete Structured Query Language Package (DLTSQLPKG) command or the DROP PACKAGE SQL statement.

When a distributed SQL program is created, the name of the SQL package and an internal consistency token are saved in the program. These are used at run time to find the SQL package and verify that the SQL package is correct for this program. Because the name of the SQL package is critical for running distributed SQL programs, an SQL package cannot be moved, renamed, duplicated, or restored to a different library.

Delete an SQL package using the Delete SQL Package (DLTSQLPKG) command:

You can use the Delete Structured Query Language Package (DLTSQLPKG) command to delete one or more SQL packages. You must enter the (DLTSQLPKG) command on the iSeries server where the SQL package being deleted is located.

You must have *OBJEXIST authority for the SQL package and at least *EXECUTE authority for the collection where it is located.

There are also several SQL methods to drop packages:

- If you have the DB2 UDB Query Manager and SQL Development Kit licensed program installed, use Interactive SQL to connect to the application server (AS) and then drop the package using the SQL DROP PACKAGE statement.
- Run an SQL program that connects and then drops the package.
- Use Query Management to connect and drop the package.

The following command deletes the SQL package PARTS1 in the SPIFFY collection:

```
DLTSQLPKG SQLPKG(SPIFFY/PARTS1)
```

To delete an SQL package on a remote iSeries server, use the Submit Remote Command (SBMRMTCMD) command to run the Delete Structured Query Language Package (DLTSQLPKG) command on the remote server. You can also use display station pass-through to sign on the remote server to delete the SQL package. If the remote server is not an iSeries server, pass through to that server using a remote workstation program and then submit the Delete SQL Package command local to that server.

Related reference

Delete Structured Query Language Package (DLTSQLPKG) command
Submit Remote Command (SBMRMTCMD) command

| Modify package authorizations:

| For any programs created on i5/OS, you can change the users that are authorized to use that package.

| This can be done using SQL's GRANT and REVOKE statements:

- | • GRANT ALL PRIVILEGES ON TABLE table-name TO user (possibly PUBLIC for user)
- | • GRANT EXECUTE ON PACKAGE package-name (usually the i5/OS program name) TO user (possibly PUBLIC for user)

| It can also be done entering GRTOBJAUT and RVKOBJAUT commands from the command line.

Use the SQL DROP PACKAGE statement:

The DROP PACKAGE statement includes the PACKAGE parameter for distributed relational database. You can issue the DROP PACKAGE statement by embedding it in a program or by using Interactive SQL.

When you issue a DROP PACKAGE statement, the SQL package and its description are deleted from the application server (AS). This has the same result as a Delete Structured Query Language Package (DLTSQLPKG) command entered on a local server. No other objects dependent on the SQL package are deleted as a result of this statement.

You must have the following privileges on the SQL package to successfully delete it:

- The system authority *EXECUTE on the referenced collection
- The system authority *OBJEXIST on the SQL package

The following example shows how the DROP PACKAGE statement is issued:

```
DROP PACKAGE SPIFFY.PARTS1
```

A program cannot issue a DROP PACKAGE statement for the SQL package it is currently using.

Related reference

SQL DROP statement

Administration

As an administrator for a distributed relational database, you are responsible for work being done on several servers.

Work that originates on your local system as an application requester (AR) can be monitored in the same way as any other work is monitored on an iSeries server.

When you are tracking units of work being done on the local system as an application server (AS), you use the same tools but look for different kinds of information.

This topic discusses ways that you can administer the distributed relational database work being done across a network. Most of the commands, processes, and other resources discussed here do not exist just for distributed relational database use, they are tools provided for the operation of any iSeries server. All administration commands, processes and resources discussed here are included with the i5/OS licensed program, along with all of the DB2 Universal Database for iSeries functions. Work management functions on the iSeries server provide effective ways to track work on several servers.

Related tasks

“Set up DDM files” on page 40

The implementation of DRDA support on the iSeries server uses Distributed Data Management (DDM) conversations for communications. Because of this, you can use DDM in conjunction with distributed relational database processing.

Monitor relational database activity

You can rely on control language (CL) commands, all of which provide similar information, but in different ways, to give you a view of work on an iSeries server.

Work with jobs in a distributed relational database

The Work with Job (WRKJOB) command presents the Work with Job menu. This menu allows you to select options to work with or to change information related to a specified job. Enter the command without any parameters to get information about the job you are currently using.

Specify a job to get the same information pertaining to it by entering its name in the command like this:

```
WRKJOB JOB(job-number/user-ID/job-name)
```

You can get the information provided by the options on the menu whether the job is on a job queue, output queue, or active. However, a job is not considered to be in the server until all of its input has been completely read in. Only then is an entry placed on the job queue. The options for the job information are:

- Job status attributes
- Job definition attributes
- Spooled file information

Information about the following options can be shown only when the job is active:

- Job run attributes
- Job log information
- Program stack information
- Job lock information

- Library list information
- Open file information
- File override information
- Commitment control status
- Communications status
- Activation groups
- Mutexes

Option 10 (Display job log) gives you information about an active job or a job on a job queue. For jobs that have ended you can usually find the same information by using option 4 (Work with spooled files). This presents the Work with Spooled Files display, where you can use option 5 to display the file named QPJOBLOG if it is on the list. The Work with Job (WRKJOB) command presents the Work with Job menu.

Related reference

Work with Job (WRKJOB) command

“Application does not complete in the expected time” on page 151

If the request takes longer than expected to complete, the first place to check is at the application requester (AR).

Work with user jobs in a distributed relational database

If you know the user profile (user name) being used by a job, you can use the Work with User Jobs (WRKUSRJOB) command to display or change job information. Enter the command without any parameters to get a list of the jobs in the server with your user profile.

You can specify any user and the job status to shorten the list of jobs by entering its name in the command like this:

```
WRKUSRJOB USER(KCDBA)
```

The Work with User Jobs display appears with names and status information of user jobs running in the server (*ACTIVE), on job queues (*JOBQ), or on an output queue (*OUTQ). The following display shows the active and ended jobs for the user named KCDBA:

```
Work with User Jobs                KC105
03/29/92 16:15:33
Type options, press Enter.
2=Change 3=Hold 4=End 5=Work with 6=Release 7=Display message
8=Work with spooled files 13=Disconnect

Opt  Job      User      Type      -----Status-----  Function
---  ---      ---      ---      ---
---  KC000      KCDBA      CMNEVK      OUTQ
---  KC000      KCDBA      CMNEVK      OUTQ
---  KC000      KCDBA      CMNEVK      OUTQ
---  KC000      KCDBA      CMNEVK      OUTQ
---  KC000      KCDBA      CMNEVK      ACTIVE
---  KC0001     KCDBA      CMNEVK      ACTIVE      * -PASSTHRU
---  KC0001     KCDBA      INTER      ACTIVE      CMD-WRKUSRJOB

Bottom
Parameters or command
====>
F3=Exit      F4=Prompt  F5=Refresh  F9=Retrieve  F11=Display schedule data
F12=Cancel   F21=Select assistance level
```

This display lists all the jobs in the server for the user, shows the status specified (*ALL in this case), and shows the type of job. It also provides you with eight options (2 through 8 and 13) to enter commands for a selected job. Option 5 presents the Work with Job display described in the preceding paragraphs.

The Work with User Jobs (WRKUSRJOB) command is useful when you want to look at the status of the DDM TCP/IP server jobs if your server is using TCP/IP. Run the following command:

```
WRKUSRJOB QUSER *ACTIVE
```

Page down until you see the jobs starting with the characters QRWT. If the server is active, you should see one job named QRWTLSTN, and one or more named QRWTSRVR (unless prestart DRDA jobs are not run on the server). The QRWTSRVR jobs are prestart jobs. If you do not see the QRWTLSTN job, run the following command to start it:

```
STRTCPSVR *DDM
```

If you see the QRWTLSTN job and not the QRWTSRVR jobs, and the use of DRDA prestart jobs has not been disabled, run the following command to start the prestart jobs:

```
STRPJ subsystem QRWTSRVR
```

Before V5R2, the subsystem that QRWTSRVR normally ran in was QSYSWRK. After V5R1, QRWTSRVR runs in QUSRWRK.

Related reference

Work with User Jobs (WRKUSRJOB) command

“Application does not complete in the expected time” on page 151

If the request takes longer than expected to complete, the first place to check is at the application requester (AR).

Work with active jobs in a distributed relational database

Use the Work with Active Jobs (WRKACTJOB) command if you want to monitor the jobs running for several users or if you are looking for a job and you do not know the job name or the user ID.

When you enter this command, the Work with Active Jobs display appears. It shows the performance and status information for jobs that are currently active on the server. All information is gathered on a job basis and grouped by subsystem.

The display here shows the Work with Active Jobs display on a typical day at the KC105 system:

```
Work with Active Jobs                KC105
03/29/92 16:17:45
CPU %: 41.7    Elapsed time: 04:37:55    Active jobs: 42

Type options, press Enter.
2=Change  3=Hold  4=End  5=Work with  6=Release  7=Display message
8=Work with spooled files  13=Disconnect ...

Opt  Subsystem/Job  User      Type  CPU %  Function      Status
---  ---           ---      ---   ---    ---           ---
---  QBATCH         QSYS     SBS   .0     DEQW          DEQW
---  QCMN           QSYS     SBS   .0     DEQW          DEQW
---  QINTER        QSYS     SBS   .0     DEQW          DEQW
---  DSP01         CLERK1   INT   .0     CMD-STSQL     DSPW
---  DSP02         CLERK2   INT   .0     * -CMDENT     DSPW

More...
Parameters or command
====>
F3=Exit    F5=Refresh  F10=Restart statistics  F11=Display elapsed data
F12=Cancel F23=More options  F24=More keys
```

When you press F11 (Display elapsed data), the following display is provided to give you detailed status information.

```

Work with Active Jobs                KC105
03/29/92 16:17:45
CPU %: 41.7    Elapsed time: 04:37:55    Active jobs: 42

```

```

Type options, press Enter.
2=Change  3=Hold  4=End  5=Work with  6=Release  7=Display message
8=Work with spooled files  13=Disconnect ...
-----Elapsed-----

```

Opt	Subsystem/Job	Type	Pool	Pty	CPU	Int	Rsp	AuxIO	CPU %
—	QBATCH	SBS	2	0	4.4			108	.0
—	QCMN	SBS	2	0	20.7			668	.0
—	KC000	EVK	2	50	.1			9	.0
—	KC0001	EVK	2	50	.1			9	.0
—	MP000	EVK	2	50	.1			14	.0
—	QINTER	SBS	2	0	7.3			4	.0
—	DSP01	INT	2	20	.1			0	.0
—	DSP02	INT	2	20	.1			0	.0

```

More...
Parameters or command
====>
F3=Exit      F5=Refresh      F10=Restart statistics  F11=Display status
F12=Cancel   F23=More options  F24=More keys

```

The Work with Active Jobs display gives you information about job priority and server usage as well as the user and type information you get from the Work with User Jobs display. You also can use any of 11 options on a job (2 through 11 and 13), including option 5, which presents you with the Work with Job display for the selected job. Use the Work with Active Jobs (WRKACTJOB) command if you want to monitor the jobs running for several users or if you are looking for a job and you do not know the job name or the user ID.

Another method to view information about job priority and server usage is to use the iSeries Navigator. To do this, follow these steps:

1. Select **Databases** in the iSeries Navigator interface.
2. Select a remote database you want to view information about.
3. Right-click and select **Properties**. This opens a properties window with the information displayed.

Related reference

Work with Active Jobs (WRKACTJOB) command

“Improve distributed relational database performance through the server” on page 144

Achieving efficient server performance requires a proper balance among server resources. Overusing any resource adversely affects performance. This topic describes the server commands that are available to help you observe the performance of your server.

“Application does not complete in the expected time” on page 151

If the request takes longer than expected to complete, the first place to check is at the application requester (AR).

Work with commitment definitions in a distributed relational database

Use the Work with Commitment Definitions (WRKCMTDFN) command if you want to work with the commitment definitions on the server.

A *commitment definition* is used to store information about commitment control when commitment control is started by the Start Commitment Control (STRCMTCTL) command. These commitment definitions might or might not be associated with an active job. Those not associated with an active job have been ended, but one or more of its logical units of work have not yet been completed.

The Work with Commitment Definitions (WRKCMTDFN) command can be used to work with commitment definitions based on the job name, status, or logical unit of work identifier of the commitment definition.

On the STATUS parameter, you can specify all jobs or only those that have a status value of *RESYNC or *UNDECIDED. *RESYNC shows only the jobs that are involved with resynchronizing their resources in an effort to reestablish a synchronization point; a *synchronization point* is the point where all resources are in consistent state.

*UNDECIDED shows only those jobs for which the decision to commit or roll back resources is unknown.

On the LUWID parameter, you can display commitment definitions that are working with a commitment definition on another server. Jobs containing these commitment definitions are communicating using an APPC-protected conversation. An LUWID can be found by displaying the commitment definition on one server and then using it as input to the Work with Commitment Definitions (WRKCMTDFN) command to find the corresponding commitment definition.

You can use the Work with Commitment Definitions (WRKCMTDFN) command to free local resources in jobs that are undecided, but only if the commitment definitions are in a Prepared (PRP) or Last Agent Pending (LAP) state. You can force the commitment definition to either commit or roll back, and thus free up held resources; control does not return to the program that issued the original commitment until the initiator learns of the action taken on the commitment definition.

You can also use the Work with Commitment Definitions (WRKCMTDFN) command to end synchronization in cases where it is determined that resynchronization will not ever complete with another server.

Related concepts

Troubleshoot transactions and commitment control

Related reference

Start Commitment Control (STRCMTCTL) command

Work with Commitment Definitions (WRKCMTDFN) command

Track request information with the job log of a distributed relational database

Every job on the iSeries server has a job log that contains information related to requests entered for a job.

The information in a job log includes:

- Commands that were used by a job
- Messages that were sent and not removed from the program message queues
- Commands in a CL program if the program was created with LOGCLPGM(*JOB) and the job specifies LOGCLPGM(*YES) or the CL program was created with LOGCLPGM(*YES)

At the end of the job, the job log can be written to a spooled file named QPJOBLOG and the original job log is deleted. You can control what information is written in the job log by specifying the LOG parameter of a job description.

The way to display a job log depends on the status of the job. If the job has ended and the job log is not yet printed, find the job using the Work with User Jobs (WRKUSRJOB) command, then select option 8 (Display spooled file). Find the spooled file named QPJOBLOG and select option 5 (Display job log). You can also display a job log by using the Work with Job (WRKJOB) command and other options on the Work with Job display.

If the batch or interactive job is still active, or is on a job queue and has not yet started, use the WRKUSRJOB command to find the job. The Work with Active Jobs (WRKACTJOB) command is used to display the job log of active jobs and does not show jobs on job queues. Select option 5 (Work with job) and then select option 10 (Display job log).

To display the job log of your own interactive job, do one of the following things:

- Enter the Display Job Log (DSPJOBLOG) command.
- Enter the Work with Job (WRKJOB) command and select option 10 (Display job log) from the Work with Job display.
- Press F10 (Display detailed messages) from the Command Entry display to display messages that are shown in the job log.

When you use the Display Job Log (DSPJOBLOG) command, you see the Job Log display. This display shows program names with special symbols, as follows:

- >> The running command or the next command to be run. For example, if a CL or high-level language program was called, the call to the program is shown.
- > The command has completed processing.
- . . The command has not yet been processed.
- ? Reply message. This symbol marks both those messages needing a reply and those that have been answered.

Related tasks

“Print a job log” on page 172

Every job on the iSeries server has a job log that contains information related to requests entered for that job. When a user is having a problem at an application requester (AR), the information in the job log might be helpful in diagnosing the problem.

Related reference

Display Job Log (DSPJOBLOG) command

Work with Active Jobs (WRKACTJOB) command

Work with Job (WRKJOB) command

Work with User Jobs (WRKUSRJOB) command

“Distributed relational database messages” on page 158

If an error message occurs at either an application server (AS) or an application requester (AR), the server message is logged on the job log to indicate the reason for the failure.

Locate distributed relational database jobs

When you are looking for information about a distributed relational database job on an application requester (AR) and you know the user profile that is used, you can find that job by using the Work with User Jobs (WRKUSRJOB) command.

You can also use this command on the application server (AS), but be aware that the user profile on the AS might be different from that used by the AR. For TCP/IP servers, the user profile that qualifies the job name will always be QUSER, and the job name will always be QRWTSRVR. The Display Log (DSPLOG) command can be used to help find the complete server job name. The message will be in the following form:

```
DDM job 031233/QUSER/QRWTSRVR servicing user XY on 10/02/97 at 22:06
```

If there are several jobs listed for the specified user profile and the relational database is accessed using DRDA, enter option 5 (Work with job) to get the Work with Job display. From this display, enter option 10 (Display job log) to see the job log. The job log shows you whether this is a distributed relational database job and, if it is, to which remote server the job is connected. Page through the job log looking for one of the following messages (depending on whether the connection is using APPC or TCP/IP):

CPI9150

DDM job started.

CPI9160

Database connection started over TCP/IP or a local socket.

The second level text for message CPI9150 and CPI9160 contains the job name for the AS job.

If you are on the AS and you do not know the job name, but you know the user name, use the Work with User Jobs (WRKUSRJOB) command. If you do not specify a user, the command returns a list of the jobs under the user profile you are using. For TCP/IP, the user profile in the job name will always be QUSER. On the Work with User Jobs display, use these columns to help you identify the AS jobs that are servicing APPC connections.

- 1 The job type column shows jobs with the type that is listed as CMNEVK for APPC communications jobs.
- 2 The status column shows if the job is active or completed. Depending on how the server is set up to log jobs, you might see only active jobs.
- 3 The job column provides the job name. The job name on the AS is the same as the device being used.

```

Work with User Jobs                                KC105
03/29/92 16:15:33
Type options, press Enter.
2=Change 3=Hold 4=End 5=Work with 6=Release 7=Display message
8=Work with spooled files 13=Disconnect

Opt  Job      User      Type  -----Status-----  Function
---  ---      ---      ---
---  KC000      KCDBA     CMNEVK  OUTQ
---  MP000      KCDBA     CMNEVK  OUTQ
---  MP000      KCDBA     CMNEVK  OUTQ
---  KC000      KCDBA     CMNEVK  OUTQ
---  KC000      KCDBA     CMNEVK  ACTIVE
---  KC0001     KCDBA     INTER   ACTIVE                    CMD-WRKUSRJOB
---  3          1          2

```

If you are looking for an active AS job and do not know the user name, the Work with Active Jobs (WRKACTJOB) command gives you a list of those jobs for the subsystems active on the server. The following example shows you some items to look for:

```

Work with Active Jobs                            KC105
03/29/92 16:17:45
CPU %: 41.7 Elapsed time: 04:37:55 Active jobs: 102

Type options, press Enter.
2=Change 3=Hold 4=End 5=Work with 6=Release 7=Display message
8=Work with spooled files 13=Disconnect

Opt  Subsystem/Job  User      Type  CPU %  Function  Status
---  ---          ---      ---  ---
---  QBATCH        QSYS      SBS    .0      DEQW
4  QCMN          QSYS      SBS    .0      WDEQ
---  KC0001        KCCLERK   EVK    .0 *    EVTW
---  5              6

```

- 4 Search the subsystem that is set up to handle the AS jobs. In this example, the subsystem for AS jobs is QCMN. The subsystem for TCP/IP server jobs is QSYSWRK prior to V5R2, and QUSRWRK after V5R1.
- 5 For APPC AS jobs, the job name is the device name of the device that is created for AS use.
- 6 The job type listed is normally EVK, started by a program start request. For TCP/IP AS jobs, the job type is PJ (unless DRDA prestart jobs are not active on the server, in which case the job type is BCI).

When you have located a job that looks like a candidate, enter option 5 to work with that job. Then select option 10 from the Work with Job Menu to display the job log. Distributed database job logs for jobs that are accessing the AS from a DB2 Universal Database for iSeries application requester contain a statement near the top that reads:

CPI3E01

Local relational database accessed by (*system name*).

After you locate a job working on the AS, you can also trace it back to the AR if the AR is an iSeries server. One of the following messages will appear in your job log; place the cursor on the message you received:

CPI9152

Target DDM job started by application requester (AR).

CPI9162

Target job assigned to handle DDM connection started by application requester (AR) over TCP/IP.

When you press the help key, the detailed message for the statement appears. The application requester (AR) job named is the job on the AR that caused this job.

Related tasks

“Print a job log” on page 172

Every job on the iSeries server has a job log that contains information related to requests entered for that job. When a user is having a problem at an application requester (AR), the information in the job log might be helpful in diagnosing the problem.

Related reference

Display Log (DSPLOG) command

Work with Active Jobs (WRKACTJOB) command

Work with User Jobs (WRKUSRJOB) command

“Application does not complete in the expected time” on page 151

If the request takes longer than expected to complete, the first place to check is at the application requester (AR).

Operate remote iSeries servers

As an administrator in a distributed relational database you might have to operate a remote iSeries server.

For example, you might have to start or stop a remote server. The iSeries server provides options that help you ensure that a remote server is operating when it needs to be. Of course, the simplest way to ensure that a remote server is operating is to allow the remote location to power on their server to meet the distributed relational database requirements. But, this is not always possible. You can set up an automatic power-on and power-off schedule or enable a remote power on to a remote server.

The server provides several ways to do this either in real time or at previously scheduled times. More often, you might need to perform certain tasks on a remote server as it is operating. The three primary ways that you can do this is by using display station pass-through, the Submit Remote Command (SBMRMTCMD) command, or stored procedures.

The Submit Remote Command (SBMRMTCMD) command submits a CL command using Distributed Data Management (DDM) support to run on the application server (AS). You first need to create a DDM file. The remote location information of the DDM file is used to determine the communications line to be used. Thus, it identifies the AS that is to receive the submitted command. The remote file associated with the DDM file is not involved when the DDM file is used for submitting commands to run on the AS.

The Submit Remote Command (SBMRMTCMD) command can submit any CL command that can run in both the batch environment and through the QCAEXEC system program; that is, the command has values of *BPGM *and* *EXEC specified for the ALLOW attribute. You can display the ALLOW attributes by using the Display Command (DSPCMD) command.

The primary purpose of the Submit Remote Command (SBMRMTCMD) command is to allow an application requester (AR) user or program to perform file management operations and file authorization activities on objects located on an AS. A secondary purpose of this command is to allow a user to perform nonfile operations (such as creating a message queue) or to submit user-written commands to run on the AS. The CMD parameter allows you to specify a character string of up to 2000 characters that represents a command to be run on the AS.

You must have the proper authority on the AS for the CL command being submitted and for the objects that the command is to operate on. If the AR user has the correct authority to do so (as determined in an AS user profile), the following actions are examples of what can be performed on remote files using the Submit Remote Command (SBMRMTCMD) command:

- Grant or revoke object authority to remote tables
- Verify tables or other objects
- Save or restore tables or other objects

Although the command can be used to do many things with tables or other objects on the remote server, using this command for some tasks is not as efficient as other methods on the iSeries server. For example, you can use this command to display the file descriptions or field attributes of remote files, or to dump files or other objects, but the output remains at the AS. To display remote file descriptions and field attributes at the AR, a better method is to use the Display File Description (DSPFD) and Display File Field Description (DSPFFD) commands with SYSTEM(*RMT) specified, and specify the names of the DDM files associated with the remote files.

See the Distributed data management topic for lists of CL commands you can submit and restrictions for the use of these commands. In addition, see Control DDM conversations for information about how DDM shares conversations.

Related concepts

Distributed data management
Schedule a system shutdown and restart

Related tasks

“Set up DDM files” on page 40
The implementation of DRDA support on the iSeries server uses Distributed Data Management (DDM) conversations for communications. Because of this, you can use DDM in conjunction with distributed relational database processing.

Related reference

“Control DDM conversations”
The term *connection* in this topic collection refers to the concept of an SQL connection. An SQL connection lasts from the time an explicit or implicit SQL CONNECT is done until the logical SQL connection is terminated by such means as an SQL DISCONNECT, or a RELEASE followed by a COMMIT.
Display Command (DSPCMD) command
Display File Description (DSPFD) command
Display File Field Description (DSPFFD) command
Submit Remote Command (SBMRMTCMD) command
System values that control IPL

Control DDM conversations

The term *connection* in this topic collection refers to the concept of an SQL connection. An SQL connection lasts from the time an explicit or implicit SQL CONNECT is done until the logical SQL connection is terminated by such means as an SQL DISCONNECT, or a RELEASE followed by a COMMIT.

Multiple SQL connections can occur serially over a single network connection or conversation. In other words, when a connection is ended, the conversation that carried it is not necessarily ended.

Note: The term *conversation* has a specific, technical meaning in SNA APPC terminology. It does not extend to TCP/IP terminology in a formal sense. However, there is a similar concept in TCP/IP (a 'network connection' in other topics on the subject). In this topic collection, the word is used with the understanding that it applies to TCP/IP network connections as well. In other topics of this topic collection, the term retains its specific APPC meaning, but it is expected that the reader can discern that meaning from the context.

When an application requester (AR) uses DRDA to connect to an application server (AS), it uses a DDM conversation. The conversation is established with the SQL CONNECT statement from the AR, but only if:

- A conversation using the same remote location values does not already exist for the AR job.
- A conversation uses the same activation group.
- If started from DDM, a conversation has the file scoped to the activation group.
- A conversation has the same conversation type (protected or unprotected).

DDM conversations can be in one of three states: active, unused, or dropped. A DDM conversation used by the distributed relational database is active while the AR is connected to the AS.

The SQL DISCONNECT and RELEASE statements are used to end connections. Connections can also be ended implicitly by the server. In addition, when running with RUW connection management, previous connections are ended when a CONNECT is performed.

After a connection ends, the DDM conversations then either become unused or are dropped. If a DDM conversation is unused, the conversation to the remote database management system is maintained by the DDM communications manager and marked as unused. If a DDM conversation is dropped, the DDM communications manager ends the conversation. The DDMCNV job attribute determines whether DDM conversations for connections that are no longer active become unused or dropped. If the job attribute value is *KEEP and the connection is to another iSeries server, the conversation becomes unused. If the job attribute value is *DROP or the connection is not to another iSeries server, the conversation is dropped.

Using a DDMCNV job attribute of *KEEP is desirable when connections to remote relational databases are frequently changed.

A value of *DROP is desirable in the following situations:

- When the cost of maintaining the conversation is high and the conversation will not be used relatively soon.
- When running with a mixture of programs compiled with RUW connection management and programs compiled with DUW connection management. Attempts to run programs compiled with RUW connection management to remote locations will fail when protected conversations exist.
- When running with protected conversations either with DDM or DRDA. Additional overhead is incurred on commits and rollbacks for unused protected conversations.

If a DDM conversation is also being used to operate on remote files through DDM, the conversation will remain active until the following conditions are met:

- All the files used in the conversation are closed and unlocked
- No other DDM-related functions are being performed
- No DDM-related function has been interrupted (by a break program, for example)
- For protected conversations, a commit or rollback was performed after ending all SQL programs and after all DDM-related functions were completed

- An AR job is no longer connected to the AS

Regardless of the value of the DDMCNV job attribute, conversations are dropped at the end of a job routing step, at the end of the job, or when the job initiates a Reroute Job (RRTJOB) command. Unused conversations within an active job can also be dropped by the Reclaim DDM Conversations (RCLDDMCNV) or Reclaim Resources (RCLRSC) command. Errors, such as communications line failures, can also cause conversations to drop.

The DDMCNV parameter is changed by the Change Job (CHGJOB) command and is displayed by Display Job (DSPJOB) command with OPTION(*DFNA). Also, you can use the Retrieve Job Attributes (RTVJOBA) command to get the value of this parameter and use it within a CL program.

Related reference

Change Job (CHGJOB) command

Display Job (DSPJOB) command

“Explicit connection management” on page 80

The CONNECT statement is used to explicitly connect an application requester (AR) to an identified application server (AS). This SQL statement can be embedded within an application program or you can issue it using Interactive SQL.

“Reclaim DDM resources”

The Reclaim Distributed Data Management Conversations (RCLDDMCNV) command reclaims all application conversations that are not currently being used by a source job, even if the DDMCNV attribute value for the job is *KEEP.

Reclaim Resources (RCLRSC) command

Reroute Job (RRTJOB) command

Retrieve Job Attributes (RTVJOBA) command

Reclaim DDM resources

The Reclaim Distributed Data Management Conversations (RCLDDMCNV) command reclaims all application conversations that are not currently being used by a source job, even if the DDMCNV attribute value for the job is *KEEP.

The command allows you to reclaim unused DDM conversations without closing all open files or doing any of the other functions performed by the Reclaim Resources (RCLRSC) command.

The Reclaim Distributed Data Management Conversations (RCLDDMCNV) command applies to the DDM conversations for the job on the application requester (AR) in which the command is entered. There is an associated application server (AS) job for the DDM conversation used by the AR job. The AS job ends automatically when the associated DDM conversation ends. For TCP/IP conversations that end, the AS job is normally a prestart job and is usually recycled rather than ended.

Although this command applies to all DDM conversations used by a job, using it does not mean that all of them will be reclaimed. A conversation is reclaimed only if it is not being actively used. If commitment control is used, a COMMIT or ROLLBACK operation might have to be done before a DDM conversation can be reclaimed.

Related reference

“Control DDM conversations” on page 107

The term *connection* in this topic collection refers to the concept of an SQL connection. An SQL connection lasts from the time an explicit or implicit SQL CONNECT is done until the logical SQL connection is terminated by such means as an SQL DISCONNECT, or a RELEASE followed by a COMMIT.

Reclaim Resources (RCLRSC) command

Reclaim Distributed Data Management Conversations (RCLDDMCNV) command

Display objects used by programs

You can use the Display Program References (DSPPGMREF) command to determine which tables, data areas, and other programs are used by a program or SQL package. This information is only available for SQL packages and compiled programs and can be displayed, printed, or written to a database output file.

When a program or package is created, the information about certain objects used in the program or package is stored. This information is then available for use with the Display Program References (DSPPGMREF) command. Information retrieved can include:

- The name of the program or package and its text description
- The name of the library or collection containing the program or package
- The number of objects referred to by the program package
- The qualified name of the server object
- The information retrieval dates
- The object type of the referenced object

For files and tables, the record contains the following additional fields:

- The name of the file or table in the program or package (possibly different from the server object name if an override was in effect when the program or package was created)

Note:

Any overrides apply only on the application requester (AR).

- The program or package use of the file or table (input, output, update, unspecified, or a combination of these four)
- The number of record formats referenced, if any
- The name of the record format used by the file or table and its record format level identifier
- The number of fields referenced for each format

Before the objects can be shown in a program, the user must have *USE authority for the program. Also, of the libraries specified by the library qualifier, only the libraries for which the user has read authority are searched for the programs.

The following table shows the objects for which the high-level languages and utilities save information.

Table 5. How high-level languages save information about objects

Language	Files	Programs	Data areas	See note
CL	Yes	Yes	Yes	1
COBOL/400 Language	Yes	Yes	No	2
PL/I	Yes	Yes	N/A	2
RPG/400 Language	Yes	No	Yes	3
DB2 UDB SQL	Yes	N/A	N/A	4

Notes:

1. All server commands that refer to files, programs, or data areas specify in the command definition that the information should be stored when the command is compiled in a CL program. If a variable is used, the name of the variable is used as the object name (for example, &FILE); If an expression is used, the name of the object is stored as *EXPR. User-defined commands can also store the information for files, programs, or data areas specified on the commands. See the description of the FILE, PGM, and DTAARA parameters on the PARM or ELEM command statements.
2. The program name is stored only when a literal is used for the program name (this is a static call, for example, CALL 'PGM1'), not when a COBOL/400 identifier is used for the program name (this is a dynamic call, for example, CALL PGM1).
3. The use of the local data area is not stored.
4. Information about SQL packages.

The stored file information contains an entry (a number) for the type of use. In the database file output of the Display Program References (DSPPGMREF) command (built when using the OUTFILE parameter), this entry is a representation of one or more codes listed here. There can only be one entry per object, so combinations are used. For example, a file coded as a 7 would be used for input, output, and update.

Code Meaning

- 1 Input
- 2 Output
- 3 Input and Output
- 4 Update
- 8 Unspecified

Related reference

Display Program References (DSPPGMREF) command
 Element definition (ELEM)

Example: Display program reference

To see what objects are used by an application requester (AR) program, you can enter a command as follows:

```
DSPPGMREF PGM(SPIFFY/PARTS1) OBJTYPE(*PGM)
```

On the requester you can get a list of all the collections and tables used by a program, but you are not able to see on which relational database they are located. They might be located in multiple relational databases. The output from the command can go to a database file or to a displayed spooled file. The output looks like this:

```
File . . . . . : QPDSPPGM                Page/Line  1/1
Control . . . . .                Columns    1 - 78
Find . . . . .

3/29/92                Display Program References
DSPPGMREF Command Input
Program . . . . . : PARTS1
Library . . . . . : SPIFFY
Output . . . . . : *
Include SQL packages . . . . . : *YES
Program . . . . . : PARTS1
Library . . . . . : SPIFFY
Text 'description'. . . . . : Check inventory for parts
Number of objects referenced . . . . . : 3
Object . . . . . : PARTS1
Library . . . . . : SPIFFY
Object type . . . . . : *PGM
Object . . . . . : QSQRUTE
Library . . . . . : *LIBL
Object type . . . . . : *PGM
Object . . . . . : INVENT
Library . . . . . : SPIFFY
Object type . . . . . : *FILE
File name in program . . . . . :
File usage . . . . . : Input
```

To see what objects are used by an application server (AS) SQL package, you can enter a command as follows:

```
DSPPGMREF PGM(SPIFFY/PARTS1) OBJTYPE(*SQLPKG)
```

The output from the command can go to a database file or to a displayed spooled file. The output looks like this:

```
File . . . . . : QPDSPPGM                Page/Line  1/1
Control . . . . . :                      Columns   1 - 78
Find . . . . . :
```

```
3/29/92                Display Program References
DSPPGMREF Command Input
Program . . . . . : PARTS1
Library . . . . . : SPIFFY
Output . . . . . : *
Include SQL packages . . . . . : *YES
SQL package . . . . . : PARTS1
Library . . . . . : SPIFFY
Text 'description'. . . . . : Check inventory for parts
Number of objects referenced . . . . . : 1
Object . . . . . : INVENT
Library . . . . . : SPIFFY
Object type . . . . . : *FILE
File name in program . . . . . :
File usage . . . . . : Input
```

Drop a collection from a distributed relational database

Attempting to delete a collection that contains journal receivers might cause an inquiry message to be sent to the QSYSOPR message queue for the application server (AS) job. The AS and application requester (AR) job wait until this inquiry is answered.

The message that appears on the message queue is:

CPA7025

Receiver (*name*) in (*library*) never fully saved. (I C)

When the AR job is waiting, it might appear as if the application is hung. Consider the following items when your AR job has been waiting for a time longer than anticipated:

- Be aware that an inquiry message is sent to QSYSOPR message queue and needs an answer to proceed.
- Have the AS reply to the message using its server reply list.

Note: Once the application is in this apparent 'hung' state, the application requesting job will wait until the inquiry message on the server has been answered. This is because journal receivers cannot be moved to another library by using the Move Object (MOV OBJ) command. They also cannot be saved and restored to different libraries. All you can do is create a new journal receiver in a different library, using the Create Journal Receiver (CRTJRNRCV) command, and attach it to the journal, using the Change Journal (CHGJRN) command. Any new journal receivers that are created by the system, using the Change Journal (CHGJRN) command with the JRNRCV(*GEN) parameter, will be created in the new library. If, when the journal is saved, the attached receiver is in another library, then when the saved version of the journal is restored, the new journal receivers will also be created in the other library.

Having the AS reply to the message using its server reply list can be accomplished by changing the job that appears to be currently hung, or by changing the job description for all AS jobs running on the server. However, you must first add an entry to the AS reply list for message CPA7025 using the Add Reply List Entry (ADDRPYLE) command:

```
ADDRPYLE SEQNBR(...) MSGID(CPA7025) RPY(I)
```


To change the job description for the job that is currently running on the AS, use the Submit Remote Command (SBMRMTCMD) command. The following example shows how the database administrator on one server in the Kansas City region changes the job description on the KC105 system (the server addressed by the TEST/KC105TST DDM file):

```
SBMRMTCMD CMD('CHGJOB JOB(KC105ASJOB) INQMSGRPY(*SYSRPYL)')
DDMFILE(TEST/KC105TST)
```

You can prevent this situation from happening on the AS more permanently by using the Change Job Description (CHGJOB) command so that any job that uses that job description uses the server reply list. The following example shows how this command is entered on the same AS:

```
CHGJOB JOB(KC105ASJOB) INQMSGRPY(*SYSRPYL)
```

This method should be used with caution. Adding CPA7025 to the server reply list affects all jobs which use the server reply list. Also changing the job description affects all jobs that use a particular job description. You might want to create a separate job description for AS jobs.

Related concepts

Journal management

Manage work

Related reference

Add Reply List Entry (ADDRPYLE) command

Change Job Description (CHGJOB) command

Change Journal (CHGJRN) command

Create Journal Receiver (CRTJRNRCV) command

Move Object (MOVOBJ) command

Submit Remote Command (SBMRMTCMD) command

Job accounting in a distributed relational database

The job accounting function on the iSeries server gathers data so you can determine who is using the server and what server resources they are using. Typical job accounting provides details on the jobs running on a server and resources used, such as use of the processing unit, printer, display stations, and database and communications functions.

Job accounting is optional and must be set up on the server. To set up resource accounting on the server you must:

1. Create a journal receiver by using the Create Journal Receiver (CRTJRNRCV) command.
2. Create the journal named QSYS/QACGJRN by using the Create Journal (CRTJRN) command. You must use the name QSYS/QACGJRN and you must have authority to add items to QSYS to create this journal. Specify the names of the journal receiver you created in the previous step on this command.
3. Change the accounting level server value QACGLVL using the Work with System Values (WRKSYSVAL) or Change System Value (CHGSYSVAL) commands.

The VALUE parameter on the Change System Value (CHGSYSVAL) command determines when job accounting journal entries are produced. A value of *NONE means the server does not produce any entries in the job accounting journal. A value of *JOB means the server produces a job (JB) journal entry. A value of *PRINT produces a direct print (DP) or spooled print (SP) journal entry for each file printed.

When a job is started, a job description is assigned to the job. The job description object contains a value for the accounting code (ACGCDE) parameter, which can be an accounting code or the default value *USRPRF. If *USRPRF is specified, the accounting code in the job's user profile is used.

You can add accounting codes to user profiles using the accounting code parameter ACGCDE on the Create User Profile (CRTUSRPRF) command or the Change User Profile (CHGUSRPRF) command. You can change accounting codes for specific job descriptions by specifying the desired accounting code for the ACGCDE parameter on the Create Job Description (CRTJOBDD) command or the Change Job Description (CHGJOBDD) command.

When a job accounting journal is set up, job accounting entries are placed in the journal receiver starting with the next job that enters the server after the Change System Value (CHGSYSVAL) command takes effect.

You can use the OUTFILE parameter on the Display Journal (DSPJRN) command to write the accounting entries to a database file that you can process.

Related concepts

Manage work

“Manage the TCP/IP server”

This topic collection describes how to manage the DRDA/DDM server jobs that communicate using sockets over TCP. It describes the subsystem in which the server runs, the objects that affect the server and how to manage those resources.

Related reference

“Account for a distributed relational database” on page 23

You need to be able to account and charge for the use of distributed data.

Change Job Description (CHGJOBDD) command

Change System Value (CHGSYSVAL) command

Change User Profile (CHGUSRPRF) command

Create Job Description (CRTJOBDD) command

Create Journal Receiver (CRTJRNRCV) command

Create Journal (CRTJRN) command

Create User Profile (CRTUSRPRF) command

Display Journal (DSPJRN) command

Work with System Value (WRKSYSVAL) command

Manage the TCP/IP server

This topic collection describes how to manage the DRDA/DDM server jobs that communicate using sockets over TCP. It describes the subsystem in which the server runs, the objects that affect the server and how to manage those resources.

The DRDA/DDM TCP/IP server that is shipped with the i5/OS licensed program does not typically require any changes to your existing system configuration in order to work correctly. It is set up and configured when you install i5/OS. At some time, you might want to change the way the system manages the server jobs to better meet your needs, solve a problem, improve the server’s performance, or simply look at the jobs on the server. To make such changes and meet your processing requirements, you need to know which objects affect which pieces of the system and how to change those objects.

This topic collection describes, at a high level, some of the work management concepts that need to be understood in order to work with the server jobs and how the concepts and objects relate to the server. In order to fully understand how to manage your iSeries server, it is recommended that you carefully review the Work management topic before you continue with this topic collection. This topic collection then shows you how the TCP/IP server can be managed and how they fit in with the rest of the system.

Related concepts

“Work management on the iSeries server” on page 25

All of the work done on the iSeries server is submitted through the work management function. On an iSeries server, you can design specialized operating environments to handle different types of work to satisfy the requirements of your server.

Manage work

Related tasks

“Set up the TCP/IP server for DRDA” on page 39

If you own a Distributed Relational Database Architecture (DRDA) application server (AS) that will be using the TCP/IP protocol, you need to set up the DDM TCP/IP server.

Related reference

“Job accounting in a distributed relational database” on page 113

The job accounting function on the iSeries server gathers data so you can determine who is using the server and what server resources they are using. Typical job accounting provides details on the jobs running on a server and resources used, such as use of the processing unit, printer, display stations, and database and communications functions.

Change Job Description (CHGJOB) command

Change System Value (CHGSYSVAL) command

Change User Profile (CHGUSRPRF) command

Create Job Description (CRTJOB) command

Create Journal Receiver (CRTJRNRCV) command

Create Journal (CRTJRN) command

Create User Profile (CRTUSRPRF) command

Display Journal (DSPJRN) command

Work with System Value (WRKSYSVAL) command

DRDA TCP/IP server terminology

The same server software is used for both DDM and DRDA TCP/IP access to DB2 Universal Database for iSeries.

For brevity, the term *DDM server* is used rather than *DRDA/DDM server* in the following discussion.

Sometimes, however, it might be referred to as the *TCP/IP server*, the *DRDA server*, or the *server* when the context makes the use of a qualifier unnecessary.

The DDM server consists of two or more jobs, one of which is what is called the *DDM listener*, because it listens for connection requests and dispatches work to the other jobs. The other job or jobs, as initially configured, are prestart jobs which service requests from the DRDA or DDM client after the initial connection is made. The set of all associated jobs, the listener and the server jobs, are collectively referred to as the *DDM server*.

The term *client* is used interchangeably with *DRDA Application Requester* (or AR) in the DRDA application environment. The term *client* will be used interchangeably with *DDM source system* in the DDM (distributed file management) application environment.

| The term *server* is used interchangeably with *DRDA Application Server* (or AS) in the DRDA application environment. The term *server* will be used interchangeably with *DDM target system* in the DDM application environment. (Note that in some contexts, the iSeries system (the hardware) is also called a server, or the iSeries server.)

TCP/IP communication support concepts for DDM

There are several concepts that pertain specifically to the TCP/IP communications support used by DRDA and DDM.

Establish a DRDA or DDM connection over TCP/IP:

To initiate a DDM server job that uses TCP/IP communications support, the DRDA Application Requester or DDM source system will connect to the well-known port number, 446 or 447. The DDM server also listens on port 448, but only for use with connections, which are not supported by DB2 Universal Database for iSeries application requesters or DDM clients.

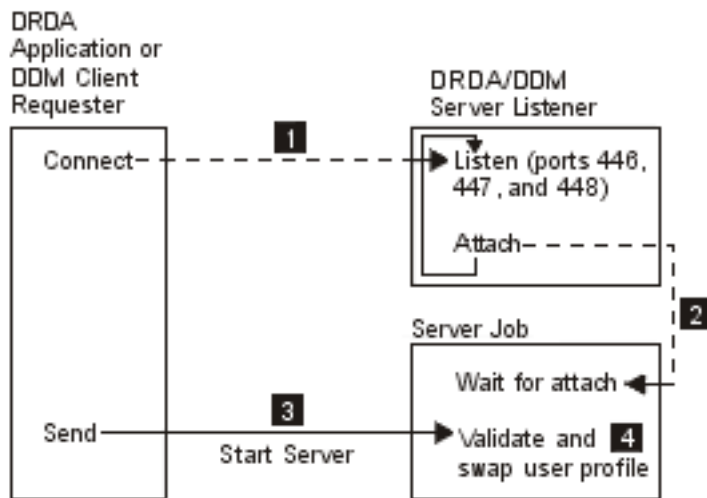


Figure 12. DRDA/DDM TCP/IP server

1. The DDM listener program must have been started (by using the Start TCP/IP Server (STRTCPSVR SERVER(*DDM)) to listen for and accept the client's connection request.

The DDM listener, on accepting this connection request, will issue an internal request to attach the client's connection to a DDM server job 2. This server job might be a prestarted job or, if the user has removed the QRWTSRVR prestart job entry from the QUSRSYS or user-defined subsystem (in which case prestart jobs are not used), a batch job that is submitted when the client connection request is processed. The server job will handle any further communications with the client.

The initial data exchange that occurs includes a request that identifies the user profile under which the server job is to run 3.

After the user profile and password (if it is sent with the user profile ID) have been validated, the server job will swap to this user profile as well as change the job to use the attributes, such as CCSID, defined for the user profile 4.

The functions of connecting to the listener program, attaching the client connection to a server job and exchanging data and validating the user profile and password are comparable to those performed when an APPC program start request is processed.

Related reference

Start TCP/IP Server (STRTCPSVR) command

DRDA and DDM listener program:

The DDM listener program runs in a batch job. There is a one-to-many relationship between it and the actual server jobs; there is one listener and potentially many DDM server jobs. The server jobs are normally prestart jobs. The listener job runs in the QSYSWRK subsystem.

The DDM listener allows client applications to establish TCP/IP connections with an associated server job by handling and routing inbound connection requests. After the client has established communications with the server job, there is no further association between the client and the listener for the duration of that connection.

The DDM listener must be active in order for DRDA Application Requesters and DDM source systems to establish connections with the DDM TCP/IP server. You can request that the DRDA listener be started automatically by either using the Change DDM TCP/IP Attributes (CHGDDMTCPA) command or through iSeries Navigator. From iSeries Navigator, navigate to the DDM settings: **Network** → **Servers** → **TCP/IP**. This will cause the listener to be started when TCP/IP is started. When starting the DRDA listener, both the QSYSWRK subsystem and TCP/IP must be active.

Related reference

Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Start TCP/IP Server (STRTCPSVR) CL command:

The Start TCP/IP Server (STRTCPSVR) command, with a SERVER parameter value of *DDM or *ALL, is used to start the listener.

Related reference

Start TCP/IP Server (STRTCPSVR) command

DDM listener restriction:

Only one DDM listener can be active at one time. Requests to start the listener when it is already active will result in an informational message to the command issuer.

Note: The DDM server will not start if the QUSER password has expired. It is recommended that the password expiration interval be set to *NOMAX for the QUSER profile. With this value the password will not expire.

Examples: Start TCP/IP Server:

This topic contains some examples of the Start TCP/IP Server (STRTCPSVR) CL command.

Example: Start all TCP/IP servers

```
STRTCPSVR SERVER(*ALL)
```

This command starts all of the TCP/IP servers, including the DDM server.

Example: Start just the DDM TCP/IP server

```
STRTCPSVR *DDM
```

This command starts only the DDM TCP/IP server.

End TCP/IP Server (ENDTCPSVR) CL command:

The End TCP/IP Server (ENDTCPSVR) command ends the DDM server.

If the DDM listener is ended, and there are associated server jobs that have active connections to client applications, the server jobs will remain active until communication with the client application is ended. Subsequent connection requests from the client application will fail, however, until the listener is started again.

Related reference

End TCP/IP Server (ENDTCPSVR) command

End TCP/IP server restrictions:

If the End TCP/IP Server (ENDTCPSVR) command is used to end the DDM listener when it is not active, a diagnostic message will be issued. This same diagnostic message will not be sent if the listener is not active when an (ENDTCPSVR) SERVER(*ALL) command is issued.

Related reference

End TCP/IP Server (ENDTCPSVR) command

Example: End TCP/IP server:

This topic contains some examples of ending TCP/IP servers.

Example: End all TCP/IP servers

```
ENDTCPSVR *ALL
```

This command ends all active TCP/IP servers.

Example: End just the DDM server

```
ENDTCPSVR SERVER(*DDM)
```

This command ends the DDM server.

Start DDM listener in iSeries Navigator:

The DDM listener can also be administered using iSeries Navigator, which is part of iSeries Access Family.

This can be done by following the path **Network** → **Servers** → **TCP/IP directory**.

DRDA/DDM server jobs

These topics discuss DRDA/DDM server jobs.

Subsystem descriptions and prestart job entries with DDM:

A subsystem description defines how, where, and how much work enters a subsystem, and which resources the subsystem uses to perform the work. The following describes how the prestart job entries in the QUSRWRK (or QSYSWRK prior to OS/400 V5R2) subsystem description affect the DDM server.

A prestart job is a batch job that starts running before an application requester (AR) initiates communications with the server. Prestart jobs use prestart job entries in the subsystem description to determine which program, class, and storage pool to use when the jobs are started. Within a prestart job entry, you must specify attributes that the subsystem uses to create and manage a pool of prestart jobs.

Prestart jobs provide increased performance when initiating a connection to a server. Prestart job entries are defined within a subsystem. Prestart jobs become active when that subsystem is started, or they can be controlled with the Start Prestart Jobs (STRPJ) and End Prestart Jobs (ENDPJ) commands.

Related reference

End Prestart Jobs (ENDPJ) command

Start Prestart Jobs (STRPJ) command

DRDA and DDM prestart jobs:

Server information that pertains to prestart jobs, such as the Display Active Prestart Jobs (DSPACTPJ) command, uses the term program start request exclusively to indicate requests made to start prestart jobs, even though the information might pertain to a prestart job that was started as a result of a TCP/IP connection request.

The following list contains the prestart job entry attributes with the initially configured value for the DDM TCP/IP server. They can be changed with the Change Prestart Job Entry (CHGPJE) command.

- Subsystem Description. The subsystem that contains the prestart job entries is QUSRWRK in V5R2. In earlier releases, it was QSYSWRK.
- Program library and name. The program that is called when the prestart job is started is QSYS/QRWTSRVR.
- User profile. The user profile that the job runs under is QUSER. This is what the job shows as the user profile. When a request to connect to the server is received from a client, the prestart job function swaps to the user profile that is received in that request.
- Job name. The name of the job when it is started is QRWTSRVR.
- Job description. The job description used for the prestart job is *USRPRF. Note that the user profile is QUSER so this will be whatever QUSER's job description is. However, the attributes of the job are changed to correspond to the requesting user's job description after the userid and password (if present) are verified.
- Start jobs. This indicates whether prestart jobs are to automatically start when the subsystem is started. These prestart job entries are shipped with a start jobs value of *YES. You can change these to *NO to prevent unnecessary jobs starting when a system IPL is performed.

Note: If the DDM server jobs are not running and the DDM listener job is batch, immediate DDM server jobs will still be run under the QSYSWRK subsystem.

- Initial number of jobs. As initially configured, the number of jobs that are started when the subsystem is started is 1. This value can be adjusted to suit your particular environment and needs.
- Threshold. The minimum number of available prestart jobs for a prestart job entry is set to 1. When this threshold is reached, additional prestart jobs are automatically started. This is used to maintain a certain number of jobs in the pool.
- Additional number of jobs. The number of additional prestart jobs that are started when the threshold is reached is initially configured at 2.
- Maximum number of jobs. The maximum number of prestart jobs that can be active for this entry is *NOMAX.
- Maximum number of uses. The maximum number of uses of the job is set to 200. This value indicates that the prestart job will end after 200 requests to start the server have been processed. In certain situations, you might need to set the MAXUSE parameter to 1 in order for the TCP/IP server to function properly. When the server runs certain ILE stored procedures, pointers to destroyed objects might remain in the prestart job environment; subsequent uses of the prestart job would cause MCH3402 exceptions. In V5R2, changes were made in i5/OS to minimize this possibility.
- Wait for job. The *YES setting causes a client connection request to wait for an available server job if the maximum number of jobs is reached.
- Pool identifier. The subsystem pool identifier in which this prestart job runs is set to 1.
- Class. The name and library of the class the prestart jobs will run under is set to QSYS/QSYSCLS20.

When the start jobs value for the prestart job entry has been set to *YES, and the remaining values are as provided with their initial settings, the following events happen for each prestart job entry:

- When the subsystem is started, one prestart job is started.
- When the first client connection request is processed for the TCP/IP server, the initial job is used and the threshold is exceeded.
- Additional jobs are started for the server based on the number defined in the prestart job entry.

- The number of available jobs will not reach below 1.
- The subsystem periodically checks the number of prestart jobs in a pool that are unused and ends excess jobs. It always leaves at least the number of prestart jobs specified in the initial jobs parameter.

Server information that pertains to prestart jobs (such as the Display Active Prestart Jobs (DSPACTPJ) command) will use the term program start request exclusively to indicate requests made to start prestart jobs, even though the information might pertain to a prestart job that was started as a result of a TCP/IP connection request.

Related tasks

“Configure the DDM server job subsystem” on page 121

By default, the DDM TCP/IP server jobs run in the QUSRWRK subsystem. Using iSeries Navigator, you can configure DDM server jobs to run all or certain server jobs in alternate subsystems based on the client’s IP address.

Related reference

Change Prestart Job Entry (CHGPJE) command

Display Active Prestart Jobs (DSPACTPJ) command

Monitor prestart jobs:

Prestart jobs can be monitored by using the Display Active Prestart Jobs (DSPACTPJ) command.

The (DSPACTPJ) command provides the following information:

- Current number of prestart jobs
- Average number of prestart jobs
- Peak number of prestart jobs
- Current number of prestart jobs in use
- Average number of prestart jobs in use
- Peak number of prestart jobs in use
- Current number of waiting connect requests
- Average number of waiting connect requests
- Peak number of waiting connect requests
- Average wait time
- Number of connect requests accepted
- Number of connect requests rejected

Related reference

Display Active Prestart Jobs (DSPACTPJ) command

Manage prestart jobs:

The information presented for an active prestart job can be refreshed by pressing the F5 key while on the Display Active Prestart Jobs display.

Of particular interest is the information about program start requests. This information can indicate to you whether you need to change the available number of prestart jobs. If you have information indicating that program start requests are waiting for an available prestart job, you can change prestart jobs using the Change Prestart Job Entry (CHGPJE) command.

If the program start requests were not being acted on fast enough, you can do any combination of the following things:

- Increase the threshold.
- Increase the Initial number of jobs (INLJOBS) parameter value.

- Increase the Additional number of jobs (ADLJOBS) parameter value.

The key is to ensure that there is an available prestart job for every sent request that starts a server job.

Related reference

Change Prestart Job Entry (CHGPJE) command

Remove prestart job entries:

If you decide that you do not want the servers to use the prestart job function, you must follow the instructions in this topic.

1. End the prestarted jobs using the End Prestart Jobs (ENDPJ) command.
Prestarted jobs ended with the (ENDPJ) command will be started the next time the subsystem is started if start jobs *YES is specified in the prestart job entry. If you only end the prestart job and do not perform the next step, any requests to start the particular server will fail.
2. Remove the prestart job entries in the subsystem description using the Remove Prestart Job Entry (RMVPJE) command.
The prestart job entries removed with the (RMVPJE) command are permanently removed from the subsystem description. After the entry is removed, new requests for the server will be successful, but will incur the performance overhead of job initiation.

Related reference

End Prestart Jobs (ENDPJ) command

Remove Prestart Job Entry (RMVPJE) command

Routing entries:

An i5/OS job is routed to a subsystem by using the routing entries in the subsystem description. The routing entry for the listener job in the QSYSWRK subsystem is present after i5/OS is installed. This job is started under the QUSER user profile, and the QSYSNOMAX job queue is used.

Prior to V5R2, the server jobs ran in the QSYSWRK subsystem. In V5R2, the server jobs run by default in QUSRWRK. The characteristics of the server jobs are taken from their prestart job entry which also comes automatically configured with i5/OS. If this entry is removed so that prestart jobs are not used for the servers, then the server jobs are started using the characteristics of their corresponding listener job.

The following list provides the initial configuration in the QSYSWRK subsystem for the listener job.

Subsystem

QSYSWRK

Job Queue

QSYSNOMAX

User QUSER

Routing Data

QRWTLSTN

Job Name

QRWTLSTN

Class QSYSCLS20

Configure the DDM server job subsystem

By default, the DDM TCP/IP server jobs run in the QUSRWRK subsystem. Using iSeries Navigator, you can configure DDM server jobs to run all or certain server jobs in alternate subsystems based on the client's IP address.

To set up the configuration:

1. Create a prestart job entry for each desired subsystem with the Add Prestart Job Entry (ADDPJE) command.
2. Start the prestart job entry you created with the Start Prestart Jobs (STRPJ) command.
3. From iSeries Navigator, expand **Network**.
4. Expand **Servers**.
5. Click **TCP/IP**.
6. Right-click **DDM** in the list of servers that are displayed in the right panel and select **Properties**.
7. On the **Subsystems** tab, add the specific client and the name of the subsystems.

In the following example, the administrator could connect and run in the QADMIN subsystem, while another server in the network could connect and run in QUSRWRK. All other clients would be rejected.

DDM Properties - Rcnasact

General Subsystems

Specify the default subsystems for server jobs which allow subsystem configuration.

Use server defaults

All clients

Subsystem:

Alternate action:

Specific clients

Client	Description	Subnet Mask	Subsystem	Alternate Action
--------	-------------	-------------	-----------	------------------

Add Edit Remove

OK Cancel Help

Related reference

Add Prestart Job Entry (ADDPJE) command

Start Prestart Jobs (STRPJ) command

“DRDA and DDM prestart jobs” on page 118

Server information that pertains to prestart jobs, such as the Display Active Prestart Jobs (DSPACTPJ) command, uses the term program start request exclusively to indicate requests made to start prestart jobs, even though the information might pertain to a prestart job that was started as a result of a TCP/IP connection request.

Identify server jobs

If you look at the server jobs started on the server, you might find it difficult to relate a server job to a certain application requester job or to a particular PC client. Being able to identify a particular job is a prerequisite to investigating problems and gathering performance data. iSeries Navigator provides support for these tasks that make the job much easier.

This topic collection provides information about how to identify server jobs before starting debug or performance investigation when you are not using iSeries Navigator.

iSeries job names:

The job name used on the iSeries consists of three parts.

- The simple job name
- User ID
- Job number (ascending order)

The DDM server jobs follow the following conventions:

- Job name is QRWTSRVR.
- User ID
 - Will always be QUSER, whether prestart jobs are used or not.
 - The job log will show which user is currently using the job.
- The job number is created by work management.

Display server jobs:

These methods can be used to help identify server jobs.

One method is to use the Work with Active Jobs (WRKACTJOB) command. Another method is to use the Work with User Jobs (WRKUSRJOB) command. A third method is to display the history log to determine which job is being used by which client user.

Related reference

Work with Active Jobs (WRKACTJOB) command

Work with User Jobs (WRKUSRJOB) command

Display active jobs using the WRKACTJOB command:

The Work with Active Jobs (WRKACTJOB) command shows all active jobs. All server jobs are displayed, as well as the listener job.

The following figures show a sample status using the (WRKACTJOB) command. Only jobs related to the server are shown in the figures. You must press F14 to see the available prestart jobs.

The following types of jobs are shown in the figures:

- 1 - Listener job
- 2 - Prestarted server jobs

```

Work with Active Jobs
AS400597
04/25/97 10:25:40
CPU %: 3.1 Elapsed time: 21:38:40 Active jobs: 77

Type options, press Enter.
 2=Change 3=Hold 4=End 5=Work with 6=Release 7=Display message
 8=Work with spooled files 13=Disconnect ...

Opt Subsystem/Job User Type CPU % Function Status
---
   QUSRWRK QSYS SBS .0 DEQW
   1
   QRWTLSTN QUSER BCH .0 SELW
   2
   QRWTSRVR QUSER PJ .0 TIMW
   QRWTSRVR QUSER PJ .0 TIMW
   QRWTSRVR QUSER PJ .0 TIMW
   QRWTSRVR QUSER PJ .0 TIMW
   QRWTSRVR QUSER PJ .0 TIMW
   .
More...

```

The following types of jobs are shown:

- PJ** The prestarted server jobs.
- SBS** The subsystem monitor jobs.
- BCH** The listener job.

The Work with Active Jobs (WRKACTJOB) command shows all active jobs. All server jobs are displayed, as well as the listener job.

Related reference

Work with Active Jobs (WRKACTJOB) command

Display active user jobs using the WRKUSRJOB command:

The Work with User Jobs (WRKUSRJOB) command USER(QUSER) STATUS(*ACTIVE) displays all active server jobs running under QUSER. This includes the DDM listener and all DDM server jobs. This command might be preferable, in that it will list fewer jobs for you to look through to find the DDM-related ones.

Related reference

Work with User Jobs (WRKUSRJOB) command

Display the history log:

Each time a client user establishes a successful connection with a server job, that job is swapped to run under the profile of that client user.

To determine which job is associated with a particular client user, you can display the history log using the Display Log (DSPLOG) command. An example of the information provided is shown in the following figure.

Display History Log Contents

```
.  
. .  
DDM job 036995/QUSER/QRWTSRVR servicing user MEL on 08/18/97 at 15:26:43.  
. .  
DDM job 036995/QUSER/QRWTSRVR servicing user REBECCA on 08/18/97 at 15:45:08.  
. .  
DDM job 036995/QUSER/QRWTSRVR servicing user NANCY on 08/18/97 at 15:56:21.  
. .  
DDM job 036995/QUSER/QRWTSRVR servicing user ROD on 08/18/97 at 16:02:59.  
. .  
DDM job 036995/QUSER/QRWTSRVR servicing user SMITH on 08/18/97 at 16:48:13.  
. .  
DDM job 036995/QUSER/QRWTSRVR servicing user DAVID on 08/18/97 at 17:10:27.  
. .  
. .  
. .
```

Press Enter to continue.

F3=Exit F10=Display all F12=Cancel

The following example shows how you can filter out uninteresting entries by using the Display Log (DSPLOG) command with the MSGID parameter:

```
DSPLOG MSGID(CPI3E34)
```

You can also prevent these records from being written to the history log by setting the appropriate options in the QRWOPTIONS data area.

Related concepts

“QRWOPTIONS data area” on page 184

When DDM or DRDA TCP/IP server jobs are initiated, they look for a data area in which the user can specify diagnostic and other options. The name is QRWOPTIONS, and it must reside in the QGPL library to take effect. It consists of a string of 48 characters.

Related reference

Display Log (DSPLOG) command

Audit the relational database directory

This topic discusses how to audit program access to the relational database directories.

Accesses to the relational database directory are recorded in the security auditing journal when either one of the items listed here is true.

- The value of the system QAUDLVL is *SYSMGT.
- The value of the user AUDLVL is *SYSMGT.

With the *SYSMGT value, the server audits all accesses made with the following commands:

- Add Relational Database Directory Entry (ADDRDBDIRE) command
- Change Relational Database Directory Entry (CHGRDBDIRE) command
- Display Relational Database Directory Entry (DSRDBDIRE) command
- Remove Relational Database Directory Entry (RMVRDBDIRE) command
- Work with Relational Database Directory Entry (WRKRDBDIRE) command

The relational database directory is a database file (QSYS/QADBXRDBD) that can be read directly without the directory entry commands.

Prior to V5R2, relational database (RDB) directory file QADBXRDBD in library QSYS was built with operational authority granted to *PUBLIC. Beginning in V5R2, that's no longer the case. Therefore, existing programs that access the RDB directory using this file might no longer run correctly. Unless you have *ALLOBJ special authority, you will have to access the logical file named QADBXRMTNM, which is built over QADBXRDBD. To audit direct accesses to this file, set auditing on with the Change Object Auditing (CHGOBJAUD) command.

Related reference

Add Relational Database Directory Entry (ADDRDBDIRE) command
Display Relational Database Directory Entry (DSPRDBDIRE) command
Change Relational Database Directory Entry (CHGRDBDIRE) command
Remove Relational Database Directory Entry (RMVRDBDIRE) command
Work with Relational Database Directory Entry (WRKRDBDIRE) command
Change Object Auditing (CHGOBJAUD) command

Data availability and protection

In a distributed relational database environment, data availability not only involves protecting data on an individual server in the network, but also ensuring that users have access to the data across the network.

The iSeries server provides the following array of functions to ensure that data on servers in a distributed relational database network is available for use:

- Save/restore
- Journal management and access path journaling
- Commitment control
- Auxiliary storage pools
- Checksum protection
- Mirrored protection and the uninterruptible power supply

While the system operator for each server is typically responsible for backup and recovery of that server's data, you should also consider aspects of network redundancy as well as data redundancy when planning your strategy to ensure the optimum availability of data across your network. The more critical certain data is to your enterprise, the more ways you should have to access that data.

Recovery support for a distributed relational database

Failures that can occur on a computer server are a server failure (when the entire server is not operating); a loss of the site due to fire, flood, or similar catastrophe; or the damage or loss of an object. For a distributed relational database, a failure on one server in the network prevents users across the entire network from accessing the relational database on that server.

If the relational database is critical to daily business activities at other locations, enterprise operations across the entire network can be disrupted for the duration of one server's recovery time. Clearly, planning for data protection and recovery after a failure is particularly important in a distributed relational database.

Each server in a distributed relational database is responsible for backing up and recovering its own data. Each server in the network also handles recovery procedures after an abnormal server end. However, backup and recovery procedures can be done by the distributed relational database administrator using display station pass-through for those servers with an inexperienced operator or no operator at all.

The most common type of loss is the loss of an object or group of objects. An object can be lost or damaged due to several factors, including power failure, hardware failures, system program errors, application program errors, or operator errors. The iSeries server provides several methods for protecting

the server programs, application programs, and data from being permanently lost. Depending on the type of failure and the level of protection chosen, most of the programs and data can be protected, and the recovery time can be significantly reduced.

You can use the following methods to protect your data and programs:

Writing data to auxiliary storage

The Force-Write Ratio (FRCRATIO) parameter on the Create File command can be used to force data to be written to auxiliary storage. A force-write ratio of one causes every add, update, and delete request to be written to auxiliary storage immediately for the table in question. However, choosing this option can reduce server performance. Therefore, saving your tables and journaling tables should be considered the primary methods for protecting the database.

Physical protection

Making sure your system is protected from sudden power loss is an important part of ensuring that your application server (AS) is available to an application requester (AR). An uninterruptible power supply, which can be ordered separately, protects the server from loss because of power failure, power interruptions, or drops in voltage, by supplying power to the server devices until power can be restored. Normally, an uninterruptible power supply does not provide power to all workstations. With the iSeries server, the uninterruptible power supply allows the server to:

- Continue operations during brief power interruptions or momentary drops in voltage.
- End operations normally by closing files and maintaining object integrity.

Data recovery after disk failures for distributed relational databases

Recovery is not possible for recently entered data if a disk failure occurs and all objects are not saved on tape or disk immediately before the failure. After previously saved objects are restored, the server is operational, but the database is not current.

Auxiliary storage pools (ASPs), checksum protection, and mirrored protection are i5/OS disk recovery functions that provide methods to recover recently entered data after a disk-related failure. These functions use additional server resources, but provide a high level of protection for servers in a distributed relational database. Because some servers might be more critical as application servers than others, the distributed relational database administrator should review how these disk data protection methods can be best used by individual servers within the network.

Related concepts

Backup and recovery

Auxiliary storage pools:

An auxiliary storage pool (ASP) is one or more physical disk units assigned to the same storage area. ASPs allow you to isolate certain types of objects on specified physical disk units.

The server ASP isolates server programs and the temporary objects that are created as a result of processing by server programs. User ASPs can be used to isolate some objects such as libraries, SQL objects, journals, journal receivers, applications, and data. The iSeries server supports up to 32 basic user ASPs, and 223 independent user ASPs. Isolating libraries or objects in a user ASP protects them from disk failures in other ASPs and reduces recovery time.

In addition to reduced recovery time and isolation of objects, placing objects in an ASP can improve performance. If a journal receiver is isolated in a user ASP, the disks associated with that ASP are dedicated to that receiver. In an environment that requires many read and write operations to the database files, this can reduce arm contention on the disks in that ASP, and can improve journaling performance.

Checksum protection in a distributed relational database:

Checksum protection guards against data loss on any disk in an auxiliary storage pool (ASP).

The checksum software maintains a coded copy of ASP data in special checksum data areas within that ASP. Any changes made to permanent objects in a checksum protected ASP are automatically maintained in the checksum data of the checksum set. If any single disk unit in a checksum set is lost, the server reconstructs the contents of the lost device using the checksum and the data on the remaining functional units of the set. In this way, if any one of the units fails, its contents can be recovered. This reconstructed data reflects the most up-to-date information that was on the disk at the time of the failure. Checksum protection can affect server performance significantly. In a distributed relational database, this might be a concern.

Mirrored protection for a distributed relational database:

Mirrored protection increases the availability of a server by duplicating different disk-related hardware components, such as a disk controller, a disk I/O processor, or a bus. The server can remain available after a failure, and service for the failed hardware components can be scheduled at a convenient time.

Different levels of mirrored protection provide different levels of server availability. For example, if only the disk units on a server are mirrored, all disk units have disk unit-level protection, so the server is protected against the failure of a single disk unit. In this situation, if a controller, I/O processor, or bus failure occurs, the server cannot run until the failing part is repaired or replaced. All mirrored units on the server must have identical disk unit-level protection and reside in the same ASP. The units in an ASP are automatically paired by the server when mirrored protection is started.

Journal management for distributed relational databases

Journal management can be used as a part of the backup and recovery strategy for relational databases and indexes.

iSeries journal support provides an audit trail and forward and backward recovery. Forward recovery can be used to take an older version of a table and apply changes logged in the journal to the table. Backward recovery can be used to remove changes logged in the journal from the table.

When a collection is created, a journal and an object called a journal receiver are created in the collection. Improved performance is gained when the journal receiver is on a different ASP from the tables. Placing the collection on a user ASP places the tables and journal and journal receivers all in the same user ASP. There is no gain in performance there. Creating a new journal receiver in a different ASP (used just for this journal's journal receivers) and attaching it with the Change Journal (CHGJRN) command will get the next server generated journal receivers all in the other user ASP, and then the user will see improved performance.

When a table is created, it is automatically journaled to the journal SQL created in the collection. You are then responsible for using the journal functions to manage the journal, journal receivers, and the journaling of tables to the journal. For example, if a table is moved into a collection, no automatic change to the journaling status occurs. If a table is restored, the normal journal rules apply. That is, if a table is journaled when it is saved, it is journaled to the same journal when it is restored on that server. If the table is not journaled at the time of the save, it is not journaled at restore time. You can stop journaling on any table using the journal functions, but doing so prevents SQL operations from running under commitment control. SQL operations can still be performed if you have specified COMMIT(*NONE), but this does not provide the same level of integrity that journaling and commitment control provide.

With journaling active, when changes are made to the database, the changes are journaled in a journal receiver before the changes are made to the database. The journal receiver always has the latest database information. All activity is journaled for a database table regardless of how the change was made.

Journal receiver entries record activity for a specific row (added, changed, or deleted), and for a table (opened, table or member saved, and so on). Each entry includes additional control information identifying the source of the activity, the user, job, program, time, and date.

The server journals some file-level changes, including moving a table and renaming a table. The server also journals member-level changes, such as initializing a physical file member, and server-level changes, such as initial program load (IPL). You can add entries to a journal receiver to identify significant events (such as the checkpoint at which information about the status of the job and the server can be journaled so that the job step can be restarted later) or to help in the recovery of applications.

For changes that affect a single row, row images are included following the control information. The image of the row after a change is made is always included. Optionally, the row image before the change is made can also be included. You control whether to journal both before and after row images or just after row images by specifying the IMAGES parameter on the Start Journal Physical File (STRJRNPf) command.

All journaled database files are automatically synchronized with the journal when the server is started (IPL time) or during the vary on of an independent ASP. If the server ended abnormally, or the independent ASP varied off abnormally, some database changes might be in the journal, but not yet reflected in the database itself. If that is the case, the server automatically updates the database from the journal to bring the tables up to date.

Journaling can make saving database tables easier and faster. For example, instead of saving entire tables every day, you can save the journal receivers that contain the changes to the tables. You can still save the entire tables on a regular basis. This method can reduce the amount of time it takes to perform your daily save operations.

The Display Journal (DSPJRN) command can be used to convert journal receiver entries to a database file. Such a file can be used for activity reports, audit trails, security, and program debugging.

Related concepts

Journal management

Related information

Change Journal (CHGJRN) command

Display Journal (DSPJRN) command

Start Journal Physical File (STRJRNPf) command

Index recovery:

An index describes the order in which rows are read from a table. When indexes are recorded in the journal, the server can recover the index to avoid spending a significant amount of time rebuilding indexes during the IPL that follows an abnormal server end or during the vary on of an independent ASP after an abnormal vary off.

When you journal tables, images of changes to the rows in the table are written to the journal. These row images are used to recover the table if the server, or independent ASP, ends abnormally. However, after an abnormal end, the server might find that indexes built over the table are not synchronized with the data in the table. If an access path and its data are not synchronized, the server must rebuild the index to ensure that the two are synchronized and usable.

When indexes are journaled, the server records images of the index in the journal to provide known synchronization points between the index and its data. By having that information in the journal, the server can recover both the data and the index, and ensure that the two are synchronized. In such cases, the lengthy time to rebuild the indexes can be avoided.

The iSeries server provides several functions to assist with index recovery. All indexes on the server have a maintenance option that specifies when the index is maintained. SQL indexes are created with an attribute of *IMMED maintenance.

In the event of a power failure or abnormal server failure, indexes that are in the process of change might need to be rebuilt to make sure they agree with the data. All indexes on the server have a recovery option that specifies when the index should be rebuilt if necessary. All SQL indexes with an attribute of UNIQUE are created with a recovery attribute of *IPL, which means these indexes are rebuilt before the i5/OS licensed program has been started. All other SQL indexes are created with the *AFTIPL recovery attribute, which means they are rebuilt after the operating system has been started or after the independent ASP has varied on. During an IPL or vary on of an independent ASP, you can see a display showing indexes that need to be rebuilt and their recovery options, and you can override these recovery options.

SQL indexes are not journaled automatically. You can use the Start Journal Access Path (STRJRNAP) command to journal any index created by SQL operations. The server save and restore functions allow you to save indexes when a table is saved by using ACCPTH(*YES) on the Save Object (SAVOBJ) or Save Library (SAVLIB) command. If you must restore a table, there is no need to rebuild the indexes. Any indexes not previously saved and restored are automatically and asynchronously rebuilt by the database.

Before journaling indexes, you must start journaling for the tables associated with the index. In addition, you must use the same journal for the index and its associated table.

Index journaling is designed to minimize additional output operations. For example, the server writes the journal data for the changed row and the changed index in the same output operation. However, you should seriously consider isolating your journal receivers in user ASPs when you start journaling your indexes. Placing journal receivers in their own user ASP provides the best journal management performance, while helping to protect them from a disk failure.

Related information

Start Journal Access Path (STRJRNAP) command

Save Object (SAVOBJ) command

Save Library (SAVLIB) command

Design tables to reduce index rebuilding time:

Table design can also help reduce index recovery time.

For example, you can divide a large master table into a history table and a transaction table. The transaction table is then used for adding new data and the history table is used for inquiry only. Each day, you can merge the transaction data into the history table and then clear the transaction file for the next day's data. With this design, the time to rebuild indexes can be shortened, because if the server abnormally ends during the day, the index to the smaller transaction table might need to be rebuilt. However, because the index to the large history table, is read-only for most of the day, it might not be out of synchronization with its data, and might not have to be rebuilt.

Consider the trade-off between using table design to reduce index rebuilding time and using server-supplied functions like access path journaling. The table design described in the previous paragraph might require a more complex application design. After evaluating your situation, you can decide to use server-supplied functions like access path journaling rather than design more complex applications.

System-managed access-path protection:

System-managed access-path protection (SMAPP) provides automatic protection for access paths.

Using the SMAPP support, you do not have to use the journaling commands, such as the Start Journal Access Path (STRJRNAP) command, to get the benefits of access path journaling. SMAPP support recovers access paths after an abnormal server end rather than rebuilding them during IPL, or the vary on of an independent ASP.

The SMAPP support is turned on with the shipped system.

The server determines which access paths to protect based on target access path recovery times provided by the user or by using a server-provided default time. The target access path recovery times can be specified as a server-wide value or on an ASP basis. Access paths that are being journaled to a user-defined journal are not eligible for SMAPP protection because they are already protected.

Related information

Start Journal Access Path (STRJRNAP) command

System-managed access-path protection

Transaction recovery through commitment control

Commitment control is an extension of the journal management function on the iSeries server. The server can identify and process a group of relational database changes as a single unit of work (transaction).

An SQL COMMIT statement guarantees that the group of operations is completed. An SQL ROLLBACK statement guarantees that the group of operations is backed out. The only SQL statements that cannot be committed or rolled back are:

- DROP COLLECTION
- GRANT or REVOKE if an authority holder exists for the specified object

Under commitment control, tables and rows used during a transaction are locked from other jobs. This ensures that other jobs do not use the data until the transaction is complete. At the end of the transaction, the program issues an SQL COMMIT or ROLLBACK statement, freeing the rows. If the server or job ends abnormally before the commit operation is performed, all changes for that job since the last time a commit or rollback operation occurred are rolled back. Any affected rows that are still locked are then unlocked. The lock levels are as follows:

***NONE**

Commitment control is not used. Uncommitted changes in other jobs can be seen.

***CHG** Objects referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements and the rows updated, deleted, and inserted are locked until the unit of work (transaction) is completed. Uncommitted changes in other jobs can be seen.

***CS** Objects referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements and the rows updated, deleted, and inserted are locked until the unit of work (transaction) is completed. A row that is selected, but not updated, is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

***ALL** Objects referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements and the rows read, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.

Table 6 on page 132 shows the record lock duration for each of these lock level values.

If you request COMMIT (*CHG), COMMIT (*CS), or COMMIT (*ALL) when the program is precompiled or when Interactive SQL is started, then SQL sets up the commitment control environment by implicitly calling the Start Commitment Control (STRCMTCTL) command. The LCKLVL parameter specified when SQL starts commitment control is the lock level specified on the COMMIT parameter on the CRTSQLxxx commands. NFYOBJ(*NONE) is specified when SQL starts commitment control. To specify a different NFYOBJ parameter, issue a (STRCMTCTL) command before starting SQL.

Note: When running with commitment control, the tables referred to in the application program by data manipulation language statements must be journaled. The tables do not have to be journaled at precompile time, but they must be journaled when you run the application.

If a remote relational database is accessing data on the server and requesting commit level repeatable read (*RR), the tables will be locked until the query is closed. If the cursor is read only, the table will be locked (*SHRNUP). If the cursor is in update mode, the table will be locked (*EXCLRD).

The journal created in the SQL collection is normally the journal used for logging all changes to SQL tables. You can, however, use the server journal functions to journal SQL tables to a different journal.

Commitment control can handle up to 131 072 distinct row changes in a unit of work. If COMMIT(*ALL) is specified, all rows read are also included in the 131 072 limit. (If a row is changed or read more than once in a unit of work, it is only counted once toward the 131 072 limit.) Maintaining a large number of locks adversely affects server performance and does not allow concurrent users to access rows locked in the unit of work until the unit of work is completed. It is, therefore, more efficient to keep the number of rows processed in a unit of work small. Commitment control allows up to 512 tables either open under commitment control or closed with pending changes in a unit of work.

The HOLD value on COMMIT and ROLLBACK statements allows you to keep the cursor open and start another unit of work without issuing an OPEN again. The HOLD value is not available when there are non-iSeries connections that are not released for a program and SQL is still in the call stack. If ALWBLK(*ALLREAD) and either COMMIT(*CHG) or COMMIT(*CS) are specified when the program is precompiled, all read-only cursors will allow blocking of rows and a ROLLBACK HOLD statement will not roll the cursor position back.

If there are locked rows (records) pending from running a SQL precompiled program or an Interactive SQL session, a COMMIT or ROLLBACK statement can be issued from the server Command Entry display. Otherwise, an implicit ROLLBACK operation occurs when the job is ended.

You can use the Work with Commitment Definitions (WRKCMTDFN) command to monitor the status of commitment definitions and free up locks and held resources involved with commitment control activities across servers.

Table 6. Record lock duration

SQL statement	COMMIT parameter	Duration of record locks	Lock type
SELECT INTO	*NONE	No locks	
	*CHG	No locks	
	*CS	Row locked when read and released	READ
	*ALL (See note 2)	From read until ROLLBACK or COMMIT	READ
FETCH (read-only cursor)	*NONE	No locks	
	*CHG	No locks	
	*CS	From read until the next FETCH	READ
	*ALL (See note 2)	From read until ROLLBACK or COMMIT	READ

Table 6. Record lock duration (continued)

SQL statement	COMMIT parameter	Duration of record locks	Lock type
FETCH (update or delete capable cursor) See note 1	*NONE	When record not updated or deleted from read until next FETCH	UPDATE
	*CHG	When record is updated or deleted from read until UPDATE or DELETE	UPDATE
	*CS	When record not updated or deleted from read until next FETCH	UPDATE
	*ALL	When record is updated or deleted from read until UPDATE or DELETE From read until ROLLBACK or COMMIT	UPDATE ³
INSERT (target table)	*NONE	No locks	UPDATE
	*CHG	From insert until ROLLBACK or COMMIT	UPDATE
	*CS	From insert until ROLLBACK or COMMIT	UPDATE
	*ALL	From insert until ROLLBACK or COMMIT	UPDATE ⁴
INSERT (tables in subselect)	*NONE	No locks	
	*CHG	No locks	
	*CS	Each record locked while being read	READ
	*ALL	From read until ROLLBACK or COMMIT	READ
UPDATE (non-cursor)	*NONE	Each record locked while being updated	UPDATE
	*CHG	From read until ROLLBACK or COMMIT	UPDATE
	*CS	From read until ROLLBACK or COMMIT	UPDATE
	*ALL	From read until ROLLBACK or COMMIT	UPDATE
DELETE (non-cursor)	*NONE	Each record locked while being deleted	UPDATE
	*CHG	From read until ROLLBACK or COMMIT	UPDATE
	*CS	From read until ROLLBACK or COMMIT	UPDATE
	*ALL	From read until ROLLBACK or COMMIT	UPDATE
UPDATE (with cursor)	*NONE	Lock released when record updated	UPDATE
	*CHG	From read until ROLLBACK or COMMIT	UPDATE
	*CS	From read until ROLLBACK or COMMIT	UPDATE
	*ALL	From read until ROLLBACK or COMMIT	UPDATE
DELETE (with cursor)	*NONE	Lock released when record deleted	UPDATE
	*CHG	From read until ROLLBACK or COMMIT	UPDATE
	*CS	From read until ROLLBACK or COMMIT	UPDATE
	*ALL	From read until ROLLBACK or COMMIT	UPDATE
Subqueries (update or delete capable cursor or UPDATE or DELETE non-cursor)	*NONE	From read until next FETCH	READ
	*CHG	From read until next FETCH	READ
	*CS	From read until next FETCH	READ
	*ALL (see note 2)	From read until ROLLBACK or COMMIT	READ

Table 6. Record lock duration (continued)

SQL statement	COMMIT parameter	Duration of record locks	Lock type
Subqueries (read-only cursor or SELECT INTO)	*NONE *CHG *CS *ALL	No locks No locks Each record locked while being read From read until ROLLBACK or COMMIT	READ READ
Notes:			
<ol style="list-style-type: none"> 1. A cursor is open with UPDATE or DELETE capabilities if the result table is not read-only (see description of DECLARE CURSOR) and if one of the following items is true: <ul style="list-style-type: none"> • The cursor is defined with a FOR UPDATE clause. • The cursor is defined without a FOR UPDATE, FOR FETCH ONLY, or ORDER BY clause and the program contains at least one of the following items: <ul style="list-style-type: none"> – Cursor UPDATE referring to the same cursor-name – Cursor DELETE referring to the same cursor-name – An EXECUTE or EXECUTE IMMEDIATE statement with ALWBLK(*READ) or ALWBLK(*NONE) specified on the CRTSQLxxx command 2. A table or view can be locked exclusively in order to satisfy COMMIT(*ALL). If a subselect is processed that includes a group by or union, or if the processing of the query requires the use of a temporary result, an exclusive lock is acquired to protect you from seeing uncommitted changes. 3. If the row is not updated or deleted, the lock is reduced to *READ. 4. An UPDATE lock on rows of the target table and a READ lock on the rows of the subselect table. 5. A table or view can be locked exclusively in order to satisfy repeatable read. Row locking is still done under repeatable read. The locks acquired and their duration are identical to *ALL. 			

Related tasks

“Work with commitment definitions in a distributed relational database” on page 102

Use the Work with Commitment Definitions (WRKCMTDFN) command if you want to work with the commitment definitions on the server.

Related reference

“Save and restore processing for a distributed relational database” on page 135

Saving and restoring data and programs allows recovery from a program or server failure, exchange of information between servers, or storage of objects or data offline. A sound backup policy at each server in the distributed relational database network ensures that a server can be restored and made available to network users quickly in the event of a problem.

Related information

DECLARE CURSOR

Start Commitment Control (STRCMTCTL) command

Troubleshoot transactions and commitment control

Work with Commitment Definitions (WRKCMTDFN) command

Save Object (SAVOBJ) command

Save Library (SAVLIB) command

Save Changed Object (SAVCHGOBJ) command

Save Save File Data (SAVSAVFDTA) command

Save System (SAVSYS) command

Restore Library (RSTLIB) command

Restore Object (RSTOBJ) command

Restore User Profiles (RSTUSRPRF) command

Restore Authority (RSTAUT) command

Restore Configuration (RSTCFG) command

Save and restore processing for a distributed relational database

Saving and restoring data and programs allows recovery from a program or server failure, exchange of information between servers, or storage of objects or data offline. A sound backup policy at each server in the distributed relational database network ensures that a server can be restored and made available to network users quickly in the event of a problem.

Saving the server on external media such as tape, protects server programs and data from disasters, such as fire or flood. However, information can also be saved to a disk file called a save file. A save file is a disk-resident file used to store data until it is used in input and output operations or for transmission to another iSeries server over communication lines. Using a save file allows unattended save operations because an operator does not need to load tapes. In a distributed relational database, save files can be sent to another server as a protection method.

When information is restored, the information is written from tape or a save file into auxiliary storage where it can be accessed by server users.

The iSeries server has a full set of commands to save and restore your database tables and SQL objects:

- The Save Library (SAVLIB) command saves one or more collections
- The Save Object (SAVOBJ) command saves one or more objects such as SQL tables, views and indexes
- The Save Changed Object (SAVCHGOBJ) command saves any objects that have changed since either the last time the collection was saved or from a specified date
- The Save Save File Data (SAVSAVFDTA) command saves the contents of a save file
- The Save System (SAVSYS) command saves the operating system, security information, device configurations, and server values
- The Restore Library (RSTLIB) command restores a collection
- The Restore Object (RSTOBJ) command restores one or more objects such as SQL tables, views and indexes
- The Restore User Profiles (RSTUSRPRF), Restore Authority (RSTAUT) and Restore Configuration (RSTCFG) commands restore user profiles, authorities, and configurations saved by a Save System (SAVSYS) command

Related tasks

“Work with commitment definitions in a distributed relational database” on page 102

Use the Work with Commitment Definitions (WRKCMTDFN) command if you want to work with the commitment definitions on the server.

Related reference

“Transaction recovery through commitment control” on page 131

Commitment control is an extension of the journal management function on the iSeries server. The server can identify and process a group of relational database changes as a single unit of work (transaction).

Related information

DECLARE CURSOR

Start Commitment Control (STRCMTCTL) command

Troubleshoot transactions and commitment control

Work with Commitment Definitions (WRKCMTDFN) command

Save Object (SAVOBJ) command

Save Library (SAVLIB) command

Save Changed Object (SAVCHGOBJ) command

Save Save File Data (SAVSAVFDTA) command

Save System (SAVSYS) command

Restore Library (RSTLIB) command
Restore Object (RSTOBJ) command
Restore User Profiles (RSTUSRPRF) command
Restore Authority (RSTAUT) command
Restore Configuration (RSTCFG) command

Save and restore indexes in the distributed relational database environment:

Restoring an SQL index can be faster than rebuilding it. Although times vary depending on a number of factors, rebuilding a database index takes approximately one minute for every 10 000 rows.

After restoring the index, the table might need to be brought up to date by applying the latest journal changes (depending on whether journaling is active). Even with this additional recovery time, you might find it faster to restore indexes rather than to rebuild them.

The server ensures the integrity of an index before you can use it. If the server determines that the index is unusable, the server attempts to recover it. You can control when an index will be recovered. If the server ends abnormally, during the next IPL the server automatically lists those tables requiring index or view recovery. You can decide whether to rebuild the index or to attempt to recover it at one of the following times:

- During the IPL
- After the IPL
- When the table is first used

Related concepts

Backup and recovery

Save and restore security information in the distributed relational database environment:

You can use a variety of CL commands to save and restore security information.

If you make frequent changes to your server security environment by updating user profiles and updating authorities for users in the distributed relational database network, you can save security information to media or a save file without a complete Save System (SAVSYS) command, a long-running process that uses a dedicated server. With the Save Security Data (SAVSECDTA) command, you can save security data in a shorter time without using a dedicated server. Data saved using the (SAVSECDTA) command can be restored using the Restore User Profiles (RSTUSRPRF) or Restore Authority (RSTAUT) command.

Included in the security information that the Save Security Data (SAVSECDTA) and Restore User Profiles (RSTUSRPRF) commands can save and restore are the server authorization entries that the DRDA TCP/IP support uses to store and retrieve remote server user ID and password information.

Related information

Save System (SAVSYS) command
Save Security Data (SAVSECDTA) command
Restore User Profiles (RSTUSRPRF) command
Restore Authority (RSTAUT) command

Save and restore SQL packages in the distributed relational database environment:

When an application program that refers to a relational database on a remote server is precompiled and bound, an SQL package is created on the application server (AS) to contain the control structures necessary to process any SQL statements in the application.

An SQL package is an iSeries object, so it can be saved to media or a save file using the Save Object (SAVOBJ) command and restored using the Restore Object (RSTOBJ) command.

An SQL package must be restored to a collection that has the same name as the collection from which it was saved, and an SQL package cannot be renamed.

Related information

Restore Object (RSTOBJ) command

Save Object (SAVOBJ) command

Save and restore relational database directories:

The relational database directory is not an iSeries object. Instead it is made up of files that are opened by the server at IPL time.

Because of this, the Save Object (SAVOBJ) command cannot be used to directly save these files. You can save the relational database directory by creating an output file from the relational database directory data. This output file can then be used to add entries to the directory again if it is damaged.

When entries have been added and you want to save the relational database directory, specify the OUTFILE parameter on the Display Relational Database Directory Entry (DSPRDBDIRE) command to send the results of the command to an output file. The output file can be saved to tape or a save file and restored to the server. If your relational database directory is damaged or your server needs to be recovered, you can restore the output file that contains relational database entry data using a control language (CL) program. The CL program reads data from the restored output file and creates the CL commands that add entries to a new relational database directory.

For example, the relational database directory for the Spiffy Corporation MP000 server is sent to an output file named RDBDIRM as follows:

```
DSPRDBDIRE OUTPUT(*OUTFILE) OUTFILE(RDBDIRM)
```

The sample CL program that follows reads the contents of the output file RDBDIRM and recreates the relational database directory using the Add Relational Database Directory Entry (ADDRDBDIRE) command. Note that the old directory entries are removed before the new entries are made.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 250.

```
/* **** */
/* - Restore RDB Entries from output file created with:          - */
/* - DSPRDBDIRE OUTPUT(*OUTFILE) OUTFILE(RDBDIRM)                - */
/* - FROM A V4R2 OR LATER LEVEL OF OS/400 or i5/OS              - */
/* **** */
PGM  PARM(&ACTIVATE)
DCL  VAR(&ACTIVATE) TYPE(*CHAR) LEN(7)

/* Declare Entry Types Variables to Compare with &RWTYPE      */
DCL &LOCAL    *CHAR 1
DCL &SNA      *CHAR 1
DCL &IP       *CHAR 1
DCL &ARD      *CHAR 1
DCL &ARDSNA   *CHAR 1
DCL &ARDIP    *CHAR 1
DCL &RWTYPE   *CHAR 1
DCL &RWRDB   *CHAR 18
DCL &RWRLOC  *CHAR 8
DCL &RWTEXT  *CHAR 50
DCL &RWDEV   *CHAR 10
DCL &RWLLOC  *CHAR 8
DCL &RWNTID  *CHAR 8
DCL &RWMODE  *CHAR 8
```

```

DCL &RWTPN    *CHAR 8
DCL &RWSLOC   *CHAR 254
DCL &RWPORT   *CHAR 14
DCL &RWDPGM   *CHAR 10
DCL &RWDLIB   *CHAR 10

DCLF FILE(RDBSAV/RDBDIRM) /* SEE PROLOG CONCERNING THIS */
IF COND(&ACTIVATE = SAVE) THEN(GOTO CMBLBL(SAVE))
IF COND(&ACTIVATE = RESTORE) THEN(GOTO CMDLBL(RESTORE))
SAVE:
CRTLIB RDBSAV
DSPRDBDIRE OUTPUT(*OUTFILE) OUTFILE(RDBSAV/RDBDIRM)
GOTO CMDLBL(END)

RESTORE:
/* Initialize Entry Type Variables to Assigned Values */
CHGVAR &LOCAL '0' /* Local RDB (one per system) */
CHGVAR &SNA '1' /* APPC entry (no ARD pgm) */
CHGVAR &IP '2' /* TCP/IP entry (no ARD pgm) */
CHGVAR &ARD '3' /* ARD pgm w/o comm parms */
CHGVAR &ARDSNA '4' /* ARD pgm with APPC parms */
CHGVAR &ARDIP '5' /* ARD pgm with TCP/IP parms */

RMVRDBDIRE RDB(*ALL) /* Clear out directory */

NEXTENT: /* Start of processing loop */
RCVF /* Get a directory entry */
MONMSG MSGID(CPF0864) EXEC(DO) /* End of file processing */
 QSYS/RCVMSG PGMQ(*SAME (*)) MSGTYPE(*EXCP) RMV(*YES) MSGQ(*PGMQ)
 GOTO CMDLBL(LASTENT)
ENDDO

/* Process entry based on type code */
IF (&RWTYPE = &LOCAL) THEN( +
 QSYS/ADDRDBDIRE RDB(&RWRDB) RMTLOCNAME(&RWRLOC) TEXT(&RWTEXT) )

ELSE IF (&RWTYPE = &SNA) THEN( +
 QSYS/ADDRDBDIRE RDB(&RWRDB) RMTLOCNAME(&RWRLOC) TEXT(&RWTEXT) +
 DEV(&RWDEV) LCLLOCNAME(&RWLLC) +
 RMTNETID(&RWNTID) MODE(&RWMODE) TNSPGM(&RWTPN) )

ELSE IF (&RWTYPE = &IP) THEN( +
 QSYS/ADDRDBDIRE RDB(&RWRDB) RMTLOCNAME(&RWSLOC *IP) +
 TEXT(&RWTEXT) PORT(&RWPORT) )

ELSE IF (&RWTYPE = &ARD) THEN( +
 QSYS/ADDRDBDIRE RDB(&RWRDB) RMTLOCNAME(&RWRLOC) TEXT(&RWTEXT) +
 ARDPGM(&RWDLIB/&RWDPGM) )

ELSE IF (&RWTYPE = &ARDSNA) THEN( +
 QSYS/ADDRDBDIRE RDB(&RWRDB) RMTLOCNAME(&RWRLOC) TEXT(&RWTEXT) +
 DEV(&RWDEV) LCLLOCNAME(&RWLLC) +
 RMTNETID(&RWNTID) MODE(&RWMODE) TNSPGM(&RWTPN) +
 ARDPGM(&RWDLIB/&RWDPGM) )

ELSE IF (&RWTYPE = &ARDIP) THEN( +
 QSYS/ADDRDBDIRE RDB(&RWRDB) RMTLOCNAME(&RWSLOC *IP) +
 TEXT(&RWTEXT) PORT(&RWPORT) +
 ARDPGM(&RWDLIB/&RWDPGM) )

GOTO CMDLBL(NEXTENT)

LASTENT:
RETURN
DLTLIB RDBSAV

```

END

ENDPGM

There is an alternate method of restoring the directory, for the case when no output file of the type described previously is available. This method is to extract the object from a saved server and restore it to some other library and then manually enter the entries in it with the Add Relational Database Directory Entry (ADDRDBDIRE) command.

The files that make up the relational database directory are saved when a Save System (SAVSYS) command is run. The physical file that contains the relational database directory can be restored from the save media to your library with the following Restore Object (RSTOBJ) command:

```
RSTOBJ    OBJ(QADBXRDBD)  SAVLIB(QSYS)
          DEV(TAP01)  OBJTYPE(*FILE)
          LABEL(Qpppppppvrmmxx0003)
          RSTLIB(your lib)
```

In this example, the relational database directory is restored from tape. The characters *ppppppp* in the LABEL parameter represent the product code of i5/OS (for example, 5722SS1 for Version 5 Release 3). The *vrmm* in the LABEL parameter is the version, release, and modification level of i5/OS. The *xx* in the LABEL parameter is the last two digits of the current server language value. For example, 2924 is for the English language; therefore, the value of *xx* is 24.

After you restore this file to your library, you can use the information in the file to manually re-create the relational database directory.

Related reference

“Example: Set up a relational database directory” on page 35

The Spiffy Corporation network provides an example to illustrate how the relational database directory is used on servers in a distributed relational database network and show how each is set up. The example assumes the use of Advanced Program-to-Program Communication (APPC) for communications, as opposed to TCP/IP, which would be simpler to set up.

Related information

Add Relational Database Directory Entry (ADDRDBDIRE) command

Display Relational Database Directory Entry (DSPRDBDIRE) command

Restore Object (RSTOBJ) command

Save Object (SAVOBJ) command

Save System (SAVSYS) command

Network redundancy considerations for a distributed relational database

Network redundancy provides different ways for users on the distributed relational database network to access a relational database on the network.

If there is only one communications path from an application requester (AR) to an application server (AS), when the communications line is down, users on the AR do not have access to the AS relational database. For this reason, network redundancy considerations are important to the distributed relational database administrator for the Spiffy Corporation. For example, consider service booking or customer parts purchasing issues for a dealership. When a customer is waiting for service or to purchase a part, the service clerk needs access to all authorized tables of enterprise information to schedule work or sell parts.

If the local server is down, no work can be done. If the local server is running but a request to a remote server is needed to process work and the remote server is down, the request can not be handled. In the Spiffy Corporation example, this might mean that a dealership cannot request parts information from a

regional inventory center. Also, if an AS that handles many AR jobs is down, none of the ARs can complete their requests. In the case of the Spiffy Corporation network, if a regional center is down, none of the application servers it supports can order parts.

Providing the region's dealerships with access to regional inventory data is important to the Spiffy Corporation distributed relational database administrator. Providing paths through the network to data can be addressed in several ways. The original network configuration for the Spiffy Corporation linked the end node dealerships to their respective network node regional centers.

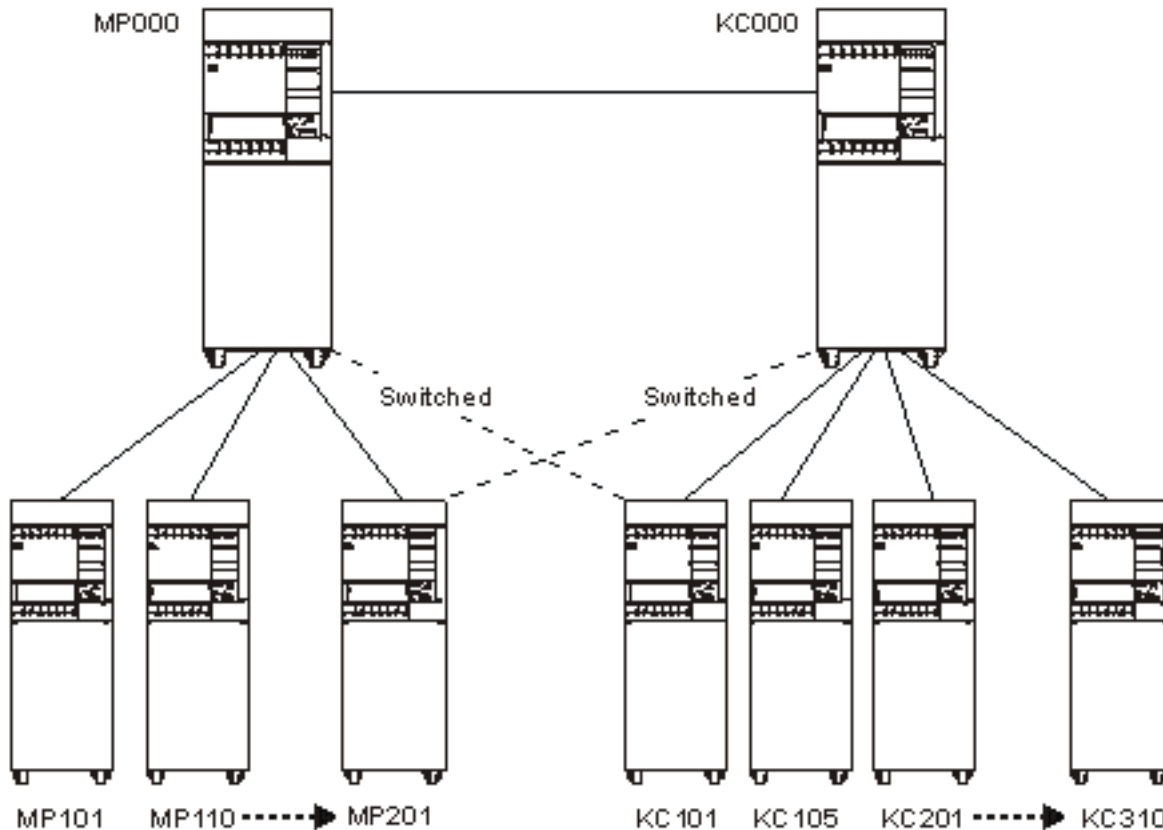


Figure 13. Alternative network paths

An alternative for some dealerships might be a switched-line connection to a different regional center. If the local regional center is unavailable to the network, access to another AS allows the requesting dealership to obtain information needed to do their work. In the first figure, some ARs served by the MP000 server establish links to the KC000 server, which can be used whenever the MP000 server is unavailable. The Vary Configuration (VRYCFG) or Work with Configuration Status (WRKCFGSTS) command can be used by a server operator or distributed relational database administrator to vary the line on when needed and vary the line off when the primary AS is available.

Another alternative might be if one of the larger area dealerships also acted as an AS for other dealerships. As shown in the second figure, an end node is only an AS to other end nodes through its network node. In the first figure, if the link to Minneapolis is down, none of the dealerships can query another (end node) for inventory. The configuration illustrated above can be changed so that one of the

dealerships is configured as an APPN network node, and lines to that dealership from other area dealerships are set up.

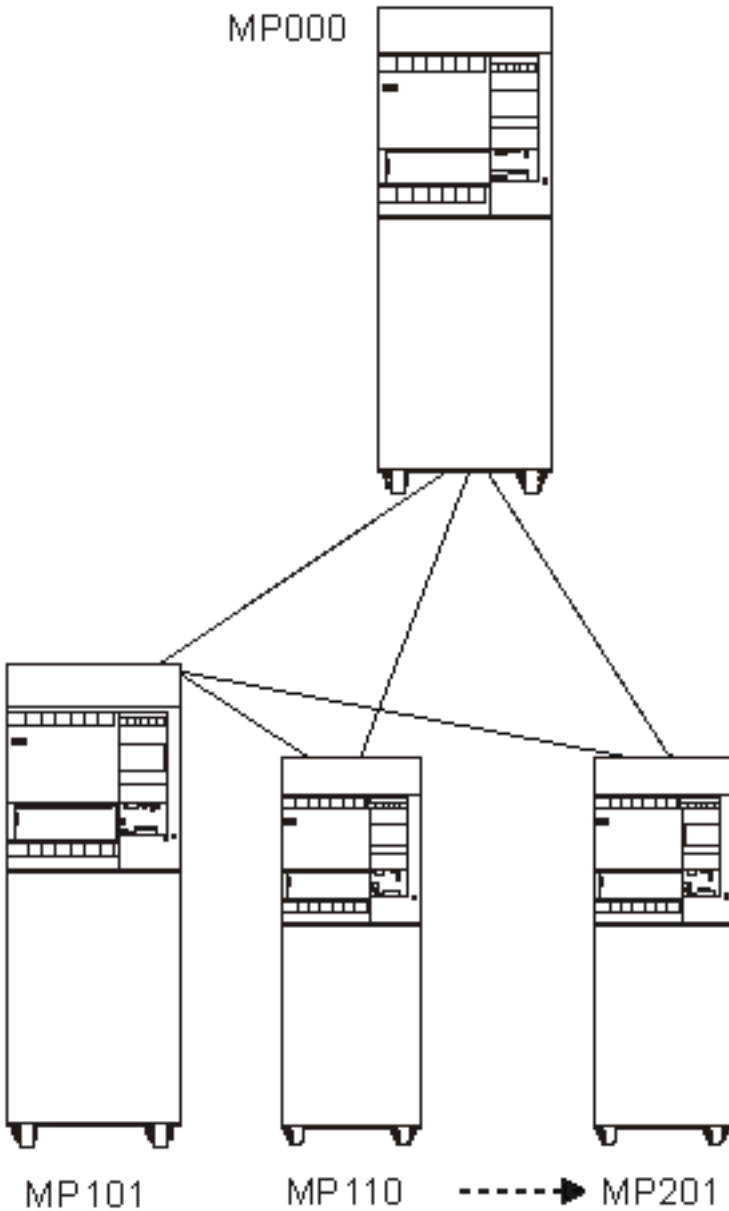


Figure 14. Alternate application server

Related information

Vary Configuration (VRYCFG) command

Work with Configuration Status (WRKCFGSTS) command

Data redundancy in your distributed relational database network

Data redundancy in a distributed relational database also provides different ways for users on the distributed relational database network to access a database on the network.

The considerations a distributed relational database administrator examines to create a data redundancy strategy are more complex than ensuring communications paths are available to the data.

Tables can be replicated across servers in the network, or a snapshot of data can be used to provide data availability. DataPropagator for iSeries, V8.1 can provide this capability.

The following figure shows that a copy of the MP000 server distributed relational database can be stored on the KC000 server, and a copy of the KC000 server distributed relational database can be stored on the MP000 server. The application requester (AR) from one region can link to the other application server (AS) to query or to update a replicated copy of their relational database.

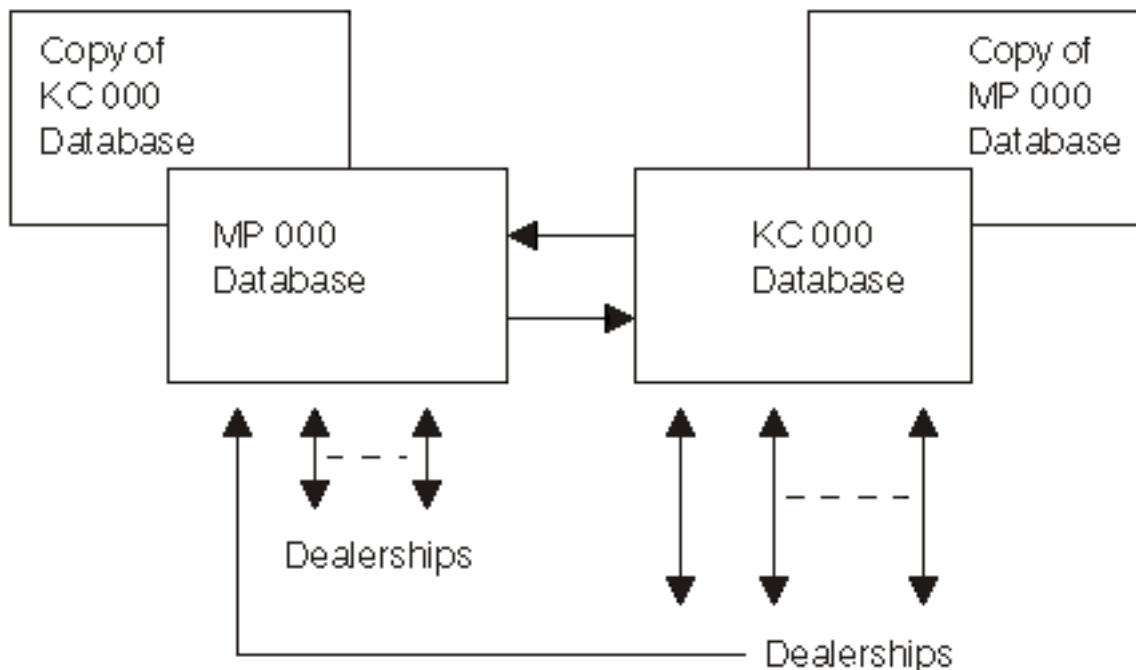


Figure 15. Data redundancy example

The administrator must decide what is the most efficient, effective strategy to allow distributed relational database processing. Alternative strategies might include these scenarios.

One alternative might be that when MP000 is unavailable, its ARs connect to the KC000 server to query a read-only snapshot of the MP000 distributed relational database so service work can be scheduled.

DataPropagator for iSeries, V8.1 can provide a read-only copy (or *snapshot*) of the tables to a remote server on a regular basis. For the Spiffy Corporation, this might be at the end or the beginning of each business day. In this example, the MP000 database snapshot provides a 24-hour-old, last-point-in-time picture for dealerships to use for scheduling only. When the MP000 server is back on line, its ARs query the MP000 distributed relational database to completely process inventory requests or other work queried on the snapshot.

Another alternative might be that Spiffy Corporation wants dealership users to be able to update a replicated table at another AS when their regional AS is unavailable.

For example, an AR that normally connects to the MP000 database could connect to a replicated MP000 database on the KC000 server to process work. When the MP000 server is available again, the MP000

relational database can be updated by applying journal entries from activity originating in its replicated tables at the KCOOO location. When these journal entries have been applied to the original MP000 tables, distributed relational database users can access the MP000 as an AS again.

Journal management processes on each regional server update all relational databases. The amount of journal management copy activity in this situation should be examined because of potential adverse performance effects at these servers.

Performance

No matter what kind of application programs you are running on a server, performance can always be a concern. For a distributed relational database, network, server, and application performance are all crucial.

Server performance can be affected by the size and organization of main and auxiliary storage. There can also be performance gains if you know the strengths and weaknesses of SQL programs.

Related concepts

“Troubleshooting” on page 148

When a problem occurs accessing a distributed relational database, it is the job of the administrator to determine the nature of the problem and whether it is a problem with the application or a problem with the local or remote system.

Improve distributed relational database performance through the network

You can improve the performance of your network in various ways.

Among them are the following ways:

- Line speed
- Pacing
- Frame size
- RU sizing
- Connection type (nonswitched versus switched)

Note: Unprotected conversations are used for DRDA connections under one of the following circumstances:

- When the connection is performed from a program using remote unit of work (RUW) connection management
- If the program that is making the connection is not running under commitment control
- If the database to which the connection is made does not support two-phase commit for the protocol that is being used

If the characteristics of the data are such that the transaction only affects one database management system, establishing the connection from a program using RUW connection management or from a program running without commitment control can avoid the overhead associated with two-phase commit flows.

Additionally, when conversations are kept active with DDMCNV(*KEEP) and those conversations are protected conversations, two-phase commit flows are sent regardless of whether the conversation was used for DRDA or DDM processing during the unit of work. Therefore, when you run with DDMCNV(*KEEP), it is better to run with unprotected conversations if possible. If running with protected conversations, you should run with DDMCNV(*DROP) and use the RELEASE statement to end the connection and the conversation at the next commit when the conversation will not be used in future units of work.

Related information

Improve distributed relational database performance through the server

Achieving efficient server performance requires a proper balance among server resources. Overusing any resource adversely affects performance. This topic describes the server commands that are available to help you observe the performance of your server.

You can use the iSeries Performance Tools licensed program to help analyze your performance. In addition, there are some system commands available to help you observe the performance of your server:

- Work with System Status (WRKSYSSTS) command
- Work with Disk Status (WRKDSKSTS) command
- Work with Active Jobs (WRKACTJOB) command

In using them, you should observe server performance during typical levels of activity. For example, statistics gathered when no jobs are running on the server are of little value in assessing server performance. To observe the server performance, complete the following steps:

1. Enter the (WRKSYSSTS), (WRKDSKSTS), or (WRKACTJOB) command.
2. Allow the server to collect data for a minimum of 5 minutes.
3. Press F5 (Refresh) to refresh the display and present the performance data.
4. Tune your server based on the new data.

Press F10 (Restart) to restart the elapsed time counter.

Use both the Work with System Status (WRKSYSSTS) and the Work with Active Jobs (WRKACTJOB) commands when observing the performance of your system. With each observation period, you should examine and evaluate the measures of server performance against the goals you have set.

Some of the typical measures include:

- Interactive throughput and response time, available from the (WRKACTJOB) display.
- Batch throughput. Observe the AuxIO and CPU% values for active batch jobs.
- Spool throughput. Observe the AuxIO and CPU% values for active writers.

Each time you make tuning adjustments, you should measure and compare all of your main performance measures. Make and evaluate adjustments one at a time.

Related tasks

“Work with active jobs in a distributed relational database” on page 101

Use the Work with Active Jobs (WRKACTJOB) command if you want to monitor the jobs running for several users or if you are looking for a job and you do not know the job name or the user ID.

Related information

Work with System Status (WRKSYSSTS) command

Work with Disk Status (WRKDSKSTS) command

Work with Active Jobs (WRKACTJOB) command

Work management

Improve distributed relational database performance through the database

Distributed relational database performance is affected by the overall design of the database. The location of distributed data, the level of commitment control you use, and the design of your SQL indexes all affect performance.

Related concepts

“Planning and design” on page 16

The first requirement for the successful operation of a distributed relational database is thorough planning. You must consider the needs and goals of your enterprise when making the decision to use a distributed relational database.

Decide DRDA data location

Because putting a network between an application and the data it needs will probably slow performance, consider these items when deciding where to put data.

- Transactions that use the data
- How often the transactions are performed
- How much data the transactions send or receive

If an application involves transactions that run frequently or that send or receive a lot of data, you should try to keep it in the same location as the data. For example, an application that runs many times a second or that receives hundreds of rows of data at a time will have better performance if the application and data are on the same server. Conversely, consider placing data in a different location than the application that needs it if the application includes low-use transactions or transactions that send or receive only moderate amounts of data at a time.

Factors that affect blocking for DRDA

A very important performance factor is whether blocking occurs when data is transferred between the application requester (AR) and the application server (AS). A group of rows transmitted as a block of data requires much less communications overhead than the same data sent one row at a time.

One way to control blocking when connected to another iSeries server is to use the SQL multiple-row INSERT and multiple-row FETCH statements. The multiple-row FETCH forces the blocking of the number of rows specified in the FOR n ROWS clause, unless a hard error or end of data is encountered. The following discussion gives rules for determining if blocking will occur for single-row FETCHs.

Conditions that inhibit the blocking of query data between the AR and the AS are also listed in the following discussion. These conditions do not apply to the use of the multiple-row FETCH statement. Any condition listed under each of the following cases is sufficient to prevent blocking.

Related reference

“Application does not complete in the expected time” on page 151

If the request takes longer than expected to complete, the first place to check is at the application requester (AR).

DB2 Universal Database for iSeries to DB2 Universal Database for iSeries blocking:

Blocking will not occur if:

- The cursor is updatable (see note 1).
- The cursor is potentially updatable (see note 2).
- The ALWBLK(*NONE) precompile option was used.
- The commitment control level is *CS and the level of OS/400 is earlier than V3R1.
- The commitment control level is *ALL and the outer subselect does not contain one of the following items:

- The DISTINCT keyword
- The UNION operator
- An ORDER BY clause and the sum of the lengths of the fields in the clause requires a sort
- A reference to a server database file (server database files are those in library QSYS named QADBxxxx, and any views built over those files)
- The row size is greater than approximately 2 KB or, if the Submit Remote Command (SBMRMTCMD) command or a stored procedure was used to extend the size of the default AS database buffer, the row size is greater than approximately half of the size of the database buffer resulting from specification of the Override with Database File (OVRDBF) command SEQONLY number-of-records parameter. (Note that for the (OVRDBF) command to work remotely, OVRSCOPE(*JOB) must be specified.)
- The cursor is declared to be scrollable (DECLARE...SCROLL CURSOR...) and a scroll option specified in a FETCH statement is one of the following: RELATIVE, PRIOR, or CURRENT (unless a multiple-row FETCH was done, as mentioned previously.)

Related information

- Submit Remote Command (SBMRMTCMD) command
- Override with Database File (OVRDBF) command

DB2 Universal Database for iSeries to non-DB2 Universal Database for iSeries blocking:

Blocking will not occur if:

- The cursor is updatable (see note 1).
- The cursor is potentially updatable (see note 2).
- The ALWBLK(*NONE) precompile option is used.
- The row size is greater than approximately 16 KB.

Non-DB2 Universal Database for iSeries to DB2 Universal Database for iSeries blocking:

Blocking will not occur if:

- The cursor is updatable (see note 1).
- The cursor is potentially updatable (see note 2).
- A precompile or bind option is used that caused the package default value to be force-single-row protocol.
 - For DB2 Universal Database for iSeries, there is no option to do this.
 - For DB2 Universal Database for VM, this is the NOBLOCK keyword on SQLPREP (the default).
 - For DB2 for OS/2® (DB2/2), this is /K=NO on SQLPREP or SQLBIND.
- The row size is greater than approximately 0.5 multiplied by QRYBLKSIZ. The default and maximum QRYBLKSIZ values are as follows:

Table 7. QRYBLKSIZ

DB2 product	Default QRYBLKSIZ	Maximum QRYBLKSIZ in Version 8
DB2 Universal Driver for SQLJ and JDBC	32 KB	32 KB
DB2 Universal Database for z/OS	32 KB	64 KB
DB2 Universal Database for VM	8 KB	32 KB
DB2 Universal Database for Linux, UNIX and Windows	32 KB	64 KB

In the latest level of Distributed Relational Database Architecture (DRDA), the server can choose to operate in a mode in which it is not limited to a fixed query block size in returning result sets.

- The cursor is defined as scrollable and a block cursor is not used in the application.

Summarization of DRDA blocking rules:

In summary, what these rules (including the notes) say is that in the absence of certain special or unusual conditions, blocking will occur in both of the following cases:

- It will occur if the cursor is read-only (see note 3) and if:
 - Either the application requester or application server is a non-DB2 Universal Database for iSeries.
 - Both the application requester and application server are DB2 Universal Database for iSeries and ALWBLK(*ALLREAD) is specified and COMMIT(*ALL) is **not** specified.
- It will occur if COMMIT(*ALL) was not specified and all of the following conditions are also true:
 - There is no FOR UPDATE OF clause in the SELECT, and
 - There are no UPDATE or DELETE WHERE CURRENT OF statements against the cursor in the program, and
 - Either the program does not contain dynamic SQL statements or a precompile/bind option was used to request limited-block protocol (/K=ALL with DB2 Universal Database for Linux, UNIX and Windows, ALWBLK(*ALLREAD) with DB2 UDB for iSeries, CURRENTDATA(NO) with DB2 Universal Database for z/OS, SBLOCK with DB2 Universal Database for VM).

Notes:

1. A cursor is updatable if it is not read-only (see note 3), and one of the following is true:
 - The select statement contained the FOR UPDATE OF clause, or
 - There exists in the program an UPDATE or DELETE WHERE CURRENT OF against the cursor.
2. A cursor is potentially updatable if it is not read-only (see note 3), and if the program includes an EXECUTE or EXECUTE IMMEDIATE statement (or when connected to a non-iSeries server, any dynamic statement), and a precompile or bind option is used that caused the package default value to be single-row protocol.
 - For DB2 Universal Database for iSeries, this is the ALWBLK(*READ) precompile option (the default).
 - For DB2, this is CURRENTDATA(YES) on BIND PACKAGE (the default).
 - For DB2 Universal Database for VM, this is the SBLOCK keyword on SQLPREP.
 - For DB2 for OS/2 (DB2/2), this is /K=UNAMBIG on SQLPREP or SQLBIND (the default).
3. A cursor is read-only if one or more of the following conditions are true:
 - The DECLARE CURSOR statement specified an ORDER BY clause but did not specify a FOR UPDATE OF clause.
 - The DECLARE CURSOR statement specified a FOR FETCH ONLY clause.
 - The DECLARE CURSOR statement specified the SCROLL keyword without DYNAMIC (i5/OS only).
 - One or more of the following conditions are true for the cursor or a view or logical file referenced in the outer subselect to which the cursor refers:
 - The outer subselect contains a DISTINCT keyword, GROUP BY clause, HAVING clause, or a column function in the outer subselect.
 - The select contains a join function.
 - The select contains a UNION operator.
 - The select contains a subquery that refers to the same table as the table of the outer-most subselect.
 - The select contains a complex logical file that had to be copied to a temporary file.

- All of the selected columns are expressions, scalar functions, or constants.
- All of the columns of a referenced logical file are input only (i5/OS only).

Factors that affect the size of DRDA query blocks

If a large amount of data is being returned on a query, performance might be improved by increasing the size of the block of query data. How this is done depends on the types of servers participating in the query.

In an unlike environment, the size of the query block is determined at the application requester by a parameter sent with the Open Query command. When an iSeries server is the application requester (AR), it always initially requests a query block size of 32 KB. It increases the size of each successive requested query block for large queries that require multiple blocks to transmit. Other types of ARs give the user a choice of what block size to use. The default query block sizes for DB2 Universal Driver for SQLJ and JDBC, DB2 Universal Database for z/OS, DB2 Universal Database for VM, and DB2 Universal Database for Linux, UNIX and Windows are 32 KB, 32 KB, 8 KB, and 32 KB, respectively. See the product documentation for the platform being used as an AR when a DB2 Universal Database for iSeries server is connected to an unlike AR.

In the DB2 UDB for iSeries to DB2 UDB for iSeries environment, the query block size is determined by the size of the buffer used by the database manager. The default size is 4 KB. This can be changed on application servers that are at the version 2 release 3, or later. In order to do this, use the Submit Remote Command (SBMRMTCMD) command to send and execute an Override with Database File (OVRDBF) command on the application server (AS). Besides the name of the file being overridden, the (OVRDBF) command should contain OVRSCOPE(*JOB) and SEQONLY(*YES nnn). The number of records desired per block replaces nnn in the SEQONLY parameter. Increasing the size of the database buffer not only can reduce communications overhead, but can also reduce the number of calls to the database manager to retrieve the rows.

You can also change the query block size using an SQL CALL statement (a stored procedure) from non-iSeries servers or between iSeries servers.

Related information

- Submit Remote Command (SBMRMTCMD) command
- Override with Database File (OVRDBF) command

Troubleshooting

When a problem occurs accessing a distributed relational database, it is the job of the administrator to determine the nature of the problem and whether it is a problem with the application or a problem with the local or remote system.

You must then resolve the problem or obtain customer support assistance to resolve the problem. To do this, you need:

- An understanding of the i5/OS licensed program support.
- A good idea of how to decide if a problem is on an application requester (AR) or an application server (AS).
- Familiarity with using i5/OS problem management functions.

For more information about diagnosing problems in a distributed relational database, see the *Distributed Relational Database Problem Determination Guide*, SC26-4782.

Related concepts

“Performance” on page 143

No matter what kind of application programs you are running on a server, performance can always be a concern. For a distributed relational database, network, server, and application performance are all crucial.

iSeries problem handling overview

The i5/OS licensed program helps you manage problems for both user- and system-detected problems that occur on local and remote iSeries servers.

Problem handling support includes:

- Messages with initial problem handling information
- Automatic alerting of system-detected problems
- Integrated problem logging and tracking
- First failure data capture (FFDC) support
- Electronic customer support service requisition
- Electronic customer support, program temporary fix (PTF) requisition

The iSeries server and its attached devices are able to detect some types of problems. These are called system-detected problems. When a problem is detected, several operations take place:

- An entry in the Product Activity Log is created
- A problem record is created
- A message is sent to the QSYSOPR message queue

Information is recorded in the error log and the problem record. When serious problems are detected, a spooled file of FFDC information is also created. The error log and the problem record might contain the following information:

- Vital product data
- Configuration information
- Reference code
- The name of the associated device
- Additional failure information

User-detected problems are usually related to program errors that can cause any of the following problems to occur:

- Job problems
- Incorrect output
- Messages indicating a program failure
- Device failure not detected by the system
- Poor performance

When a user detects a problem, no information is gathered by the server until problem analysis is run or you select the option to save information to help resolve a problem from the Operational Assistant USERHELP menu.

The iSeries server tracks both user- and system-detected problems using the problem log and problem manager. A problem state is maintained from when a problem is detected (OPENED) to when it is resolved (CLOSED) to assist you with tracking.

Related concepts

“System and communications problems” on page 170

When a problem with a system or its communications occurs, a message is generated. System-detected problems are automatically entered into the problem log, where they can be viewed and analyzed.

Isolate distributed relational database problems

A problem you encounter when running a distributed relational database application can exhibit two general symptoms: incorrect output or the application does not complete in the expected time.

The figures in this topic collection show generally how you can classify problems as application program problems, performance-related problems, and server-related problems, so you can use standard iSeries server problem analysis methods to resolve the problem.

DRDA incorrect output problems

If you receive an error message, use the error message, SQLCODE, or SQLSTATE to determine the cause of the problem.

See the following figure. The message description indicates what the problem is and provides corrective actions. If you do not receive an error message, you must determine whether distributed relational database is causing the failure. To do this, run the failing statement locally on the application server (AS) or use interactive Structured Query Language (SQL) to run the statement on the AS. If you can create the problem locally, the problem is not with distributed relational database support. Use iSeries server problem analysis methods to provide specific information for your support staff depending on the results of this operation.

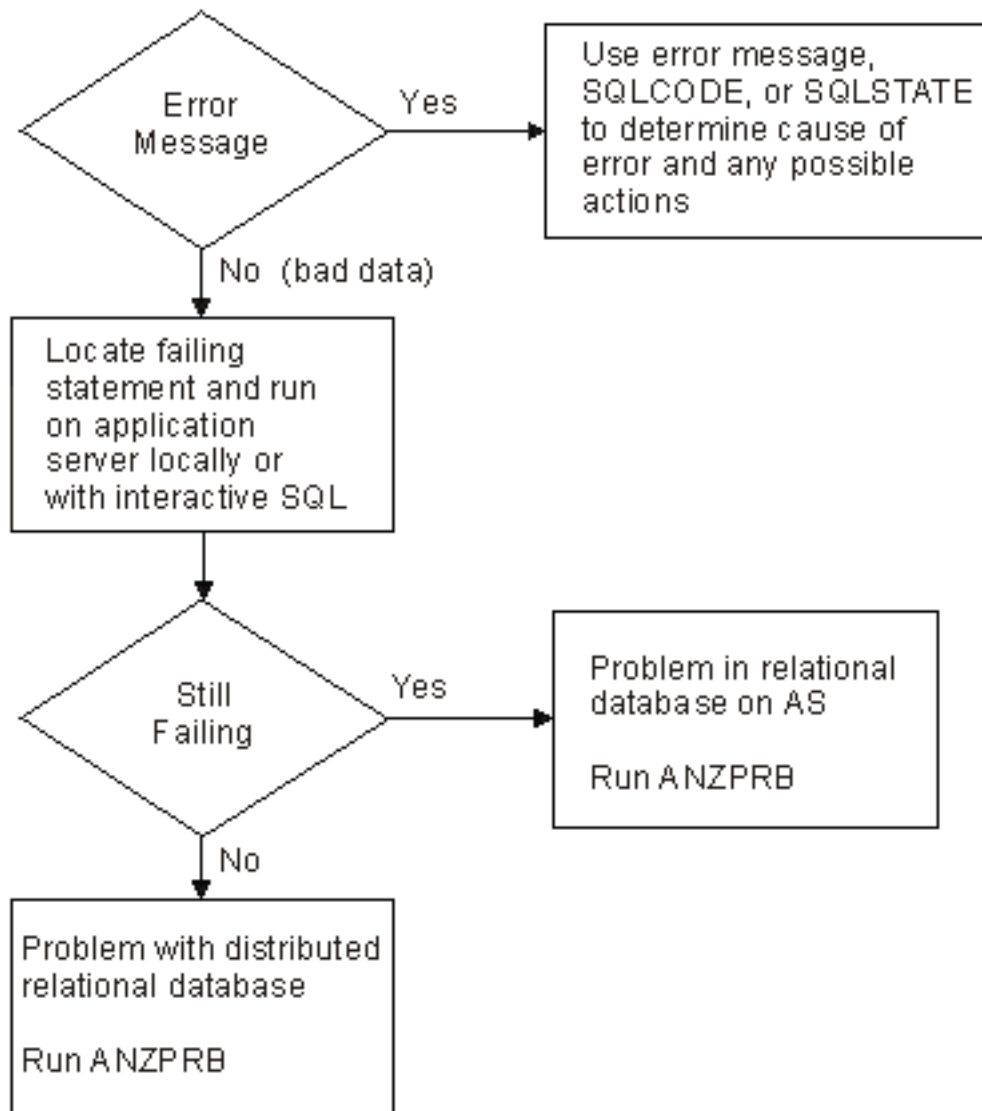


Figure 16. Resolve incorrect output problem

Application does not complete in the expected time

If the request takes longer than expected to complete, the first place to check is at the application requester (AR).

Check the job log for message SQL7969, which indicates that a connection to a relational database is complete. This tells you the application is a distributed relational database application. Check the AR for a loop by using the Work with Job (WRKJOB) command to display the program stack, and check the program stack to determine whether the system is looping. If the application itself is looping, contact the application programmer for resolution. If you see QAPDEQUE and QCNSRCV on the stack, the AR is waiting for the application server (AS). If the system is not in a communications wait state, use problem analysis procedures to show whether there is a performance problem or a wait state somewhere else.

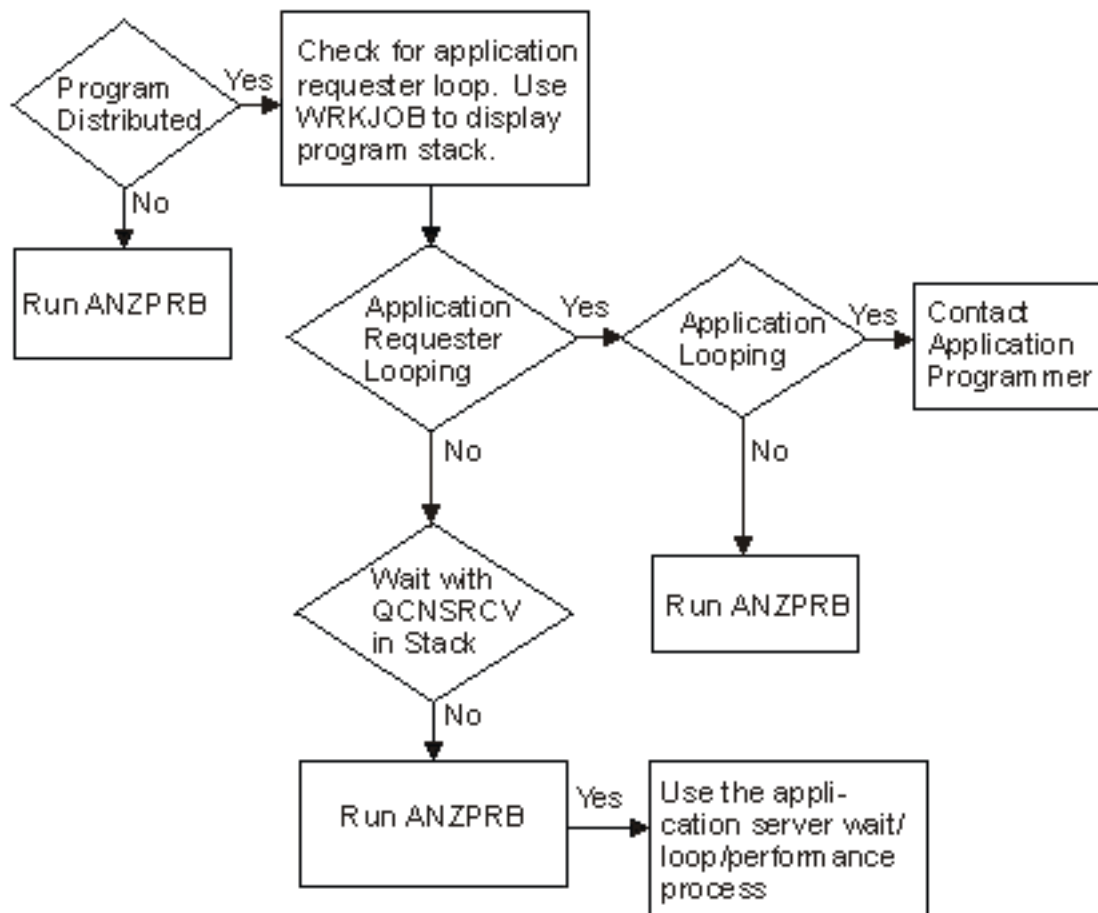


Figure 17. Resolve wait, loop, or performance problems on the application requester

You can find the AR job name by looking at the job log on the AS. When you need to check the AS job, use the Work with Job (WRKJOB), Work with Active Jobs (WRKACTJOB), or Work with User Jobs (WRKUSRJOB) command to locate the job on the AS.

From one of these job displays, look at the program stack to see if the AS is looping. If it is looping, use problem analysis to handle the problem. If it is not looping, check the program stack for WAIT with

QCNTRCV, which means the AS is waiting for the AR. If both servers are in this communications wait state, there is a problem with your network. If the AS is not in a wait state, there is a performance issue that might have to be addressed.

Two common sources of slow query performance are:

- An accessed table does not have an index. If this is the case, create an index using an appropriate field or fields as the key.
- The rows returned on a query request are not blocked. Whether the rows are blocked can cause a significant difference in query performance. It is important to understand the factors that affect blocking, and tune the application to take advantage of it.

The first time you connect to DB2 Universal Database for iSeries from a workstation using a product such as DB2 JDBC Universal Driver or DB2 Universal Database for Linux, UNIX and Windows, if you have not already created the SQL packages for the product in DB2 UDB for iSeries, the packages will be created automatically, and the NULLID collection might need to be created automatically as well. This will result in a somewhat lengthy delay in getting a response back from the server for one of the first SQL statements issued after the initial connection.

A long delay will occur if the server to which you are trying to connect over TCP/IP is not available. A several minute timeout delay will precede the message A remote host did not respond within the timeout period. An incorrect IP address in the RDB directory will cause this behavior as well.

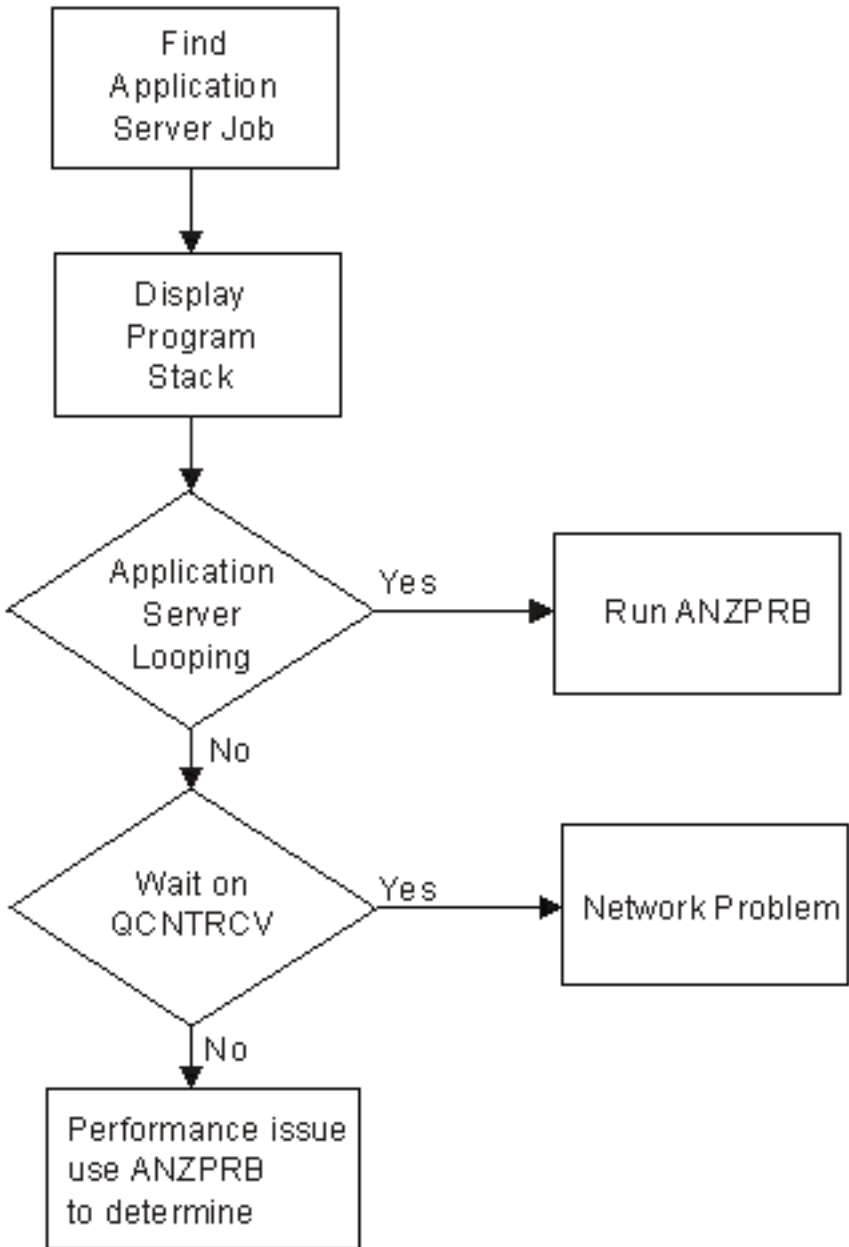


Figure 18. Resolve wait, loop, or performance problems on the application server

Related concepts

“Factors that affect blocking for DRDA” on page 145

A very important performance factor is whether blocking occurs when data is transferred between the application requester (AR) and the application server (AS). A group of rows transmitted as a block of data requires much less communications overhead than the same data sent one row at a time.

Related tasks

“Locate distributed relational database jobs” on page 104

When you are looking for information about a distributed relational database job on an application requester (AR) and you know the user profile that is used, you can find that job by using the Work with User Jobs (WRKUSRJOB) command.

“Work with jobs in a distributed relational database” on page 99

The Work with Job (WRKJOB) command presents the Work with Job menu. This menu allows you to

select options to work with or to change information related to a specified job. Enter the command without any parameters to get information about the job you are currently using.

“Work with user jobs in a distributed relational database” on page 100

If you know the user profile (user name) being used by a job, you can use the Work with User Jobs (WRKUSRJOB) command to display or change job information. Enter the command without any parameters to get a list of the jobs in the server with your user profile.

“Work with active jobs in a distributed relational database” on page 101

Use the Work with Active Jobs (WRKACTJOB) command if you want to monitor the jobs running for several users or if you are looking for a job and you do not know the job name or the user ID.

Work with distributed relational database users

Investigating a problem usually begins with the user. Users might not be getting the results they expect when running a program or they might get a message indicating a problem. Sometimes the best way to diagnose and solve a problem is to go through the procedure with a user.

The Copy screen function allows you to do this either in real time with the user or in examining a file of the displays the user saw previously.

You can also gather more information from Messages than just the line of text that appears at the bottom of a display. This topic collection discusses how you can copy displays being viewed by another user and how you can obtain more information about messages you or a user receives when doing distributed relational database work.

In addition to programming problems, you might have problems with starting the program or connecting to the server when using Advanced Program-to-Program Communication (APPC) or TCP/IP.

Copy screen

The Start Copy Screen (STRCPYSCN) command allows you to be signed on to your workstation and see the same displays being viewed by someone else at another workstation.

You must be signed on to the same iSeries server as the user. If that user is on a remote server, you can use display station pass-through to sign on that server and then enter the (STRCPYSCN) command to see the other displays. Screen images can be copied to a database file at the same time they are copied to another workstation or when another workstation cannot be used. This allows you to process this data later and prepares an audit trail for the operations that occur during a problem situation.

To copy the display image to another display station, the following requirements must be met:

- Both displays are defined to the server
- Both displays are color or both are monochrome, but not one color and the other monochrome
- Both displays have the same number of character positions horizontally and vertically

If you type your own display station ID as the sending device, the receiving display station must have the sign on display shown when you start copying screen images. Graphics are copied as blanks.

If not already signed on to the same server, use the following process to see the displays that another user sees on a remote server:

1. Enter the Start Pass-Through (STRPASTHR) command.

```
STRPASTHR RMTLOCNAME(KC105)
```

2. Log on to the application server (AS).

3. Enter the (STRCPYSCN) command.

```
STRCPYSCN SRCDEV(KC105)  
OUTDEV(*REQUESTER)  
OUTFILE(KCHELP/TEST)
```

- SRCDEV specifies the name of the source device, the display station that is sending the display image. To send your display to command to another device, enter the *REQUESTER value for this parameter.
 - OUTDEV specifies the name of the output device to which the display image is sent. In this example, the display image is sent to the display station of the person who enters the command (*REQUESTER). You can also name another display station, another device (where a third user is viewing), or to no other device (*NONE). When the *NONE value is used, specify an output file for the display images.
 - OUTFILE specifies the name of the output file that will contain an image of all the displays viewed while the command is active.
4. An inquiry message is sent to the source device to notify the user of that device that the displays will be copied to another device or file. Type a g (Go) to start sending the images to the requesting device.

The sending display station's screens are copied to the other display station. The image shown at the receiving display station trails the sending display station by one screen. If the user at the sending display station presses a key that is not active (such as the Home key), both display stations will show the same display.

While you are copying screens, the operator of the receiving display station cannot do any other work at that display station until the copying of screens is ended.

To end the copy screen function from the sending display station, enter the End Copy Screen (ENDCPYSCN) command from any command line and press the Enter key.

ENDCPYSCN

The display you viewed when you started the copy screen function is shown.

Related information

Start Pass-Through (STRPASTHR) command

Messages

The iSeries server sends a variety of system messages that indicate conditions ranging from simple typing errors to problems with server devices or programs.

Listed here are the messages you might get:

- An error message on your current display.
These messages can interrupt your job or sound an alarm. You can display these messages by entering DSPMSG on any command line.
- A message regarding a server problem that is sent to the server operator message queue and displayed on a separate Work with Messages display.
To see these messages, enter DSPMSG QSYSOPR on any server command line.
- A message regarding a server problem that is sent to the message queue specified in a device description.
To see these messages, enter DSPMSG message-queue-name on any server command line.

The server sends informational or inquiry messages for certain server events. Informational messages give you status on what the server is doing. Inquiry messages give you information about the server, but also request a reply.

In some message displays, a message is accompanied by a letter and number code such as:

CPF0083

The first two or three letters indicate the message category. Some message categories for distributed relational database are:

Table 8. Message categories

Category	Description	Library
CPA through CPZ	Messages from the operating system	QSYS/QCPFMSG
MCH	Licensed internal code messages	QSYS/QCPFMSG
SQ and SQL	Structured Query Language (SQL) messages	QSYS/QSQLMSG
TCP	TCP/IP messages	QTCP/QTCPMMSGF

The remaining four digits (five digits if the prefix is SQ) indicate the sequence number of the message. The example message ID shown indicates this is a message from the operating system, number 0083.

To obtain more information about a message on the message line of a display or in a message queue, follow these steps:

1. Move the cursor to the same line as the message.
2. Press the Help key. The Additional Message Information display is shown.

```

Additional Message Information

Message ID . . . . . : CPD6A64          Severity . . . . . : 30
Message type . . . . . : DIAGNOSTIC
Date sent . . . . . : 03/29/92         Time sent . . . . . : 13:49:06
From program . . . . . : QUIACT         Instruction . . . . . : 080D
To program . . . . . : QUIMGFLW        Instruction . . . . . : 03C5

Message . . . . . : Specified menu selection is not correct.
Cause . . . . . : The selection that you have specified is not correct for
one of the following reasons:
-- The number selected was not valid.
-- Something other than a menu option was entered on the option line.
Recovery . . . . . : Select a valid option and press the Enter or Help key
again.

Bottom
Press Enter to continue.

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

You can get more information about a message that is not showing on your display if you know the message identifier and the library in which it is located. To get this information, enter the Display Message Description (DSPMSGD) command:

```
DSPMSGD RANGE(SQL0204) MSGF(QSYS/QSQLMSG)
```

This command produces a display that allows you to select the following information about a message:

- Message text
- Field data
- Message attributes
- All of the preceding items

The text is the same message and message help text that you see on the Additional Message Information display. The field data is a list of all the substitution variables defined for the message and their attributes. The message attributes are the values (when defined) for severity, logging, level of message, default program, default reply, and dump parameters. You can use the information to help you determine what the user was doing when the message appeared.

Related information

Display Message Description (DSPMSGD) command

Message types:

On the Additional Message Information display, you see the message type and severity code for the message.

The table here shows the different message types for iSeries messages and their associated severity codes:

Table 9. Message severity codes

Message type	Severity code
Informational messages. For informational purposes only; no reply is needed. The message can indicate that a function is in progress or that a function has completed successfully.	00
Warning. A potential error condition exists. The program might have taken a default, such as supplying missing data. The results of the operation are assumed to be successful.	10
Error. An error has been found, but it is one for which automatic recovery procedures probably were applied; processing has continued. A default might have been taken to replace the wrong data. The results of the operation might not be correct. The function might not have completed; for example, some items in a list ran correctly, while other items did not.	20
Severe error. The error found is too severe for automatic recovery procedures and no defaults are possible. If the error was in the source data, the entire data record was skipped. If the error occurred during a program, it leads to an abnormal end of program (severity 40). The results of the operation are not correct.	30
Severe error: abnormal end of program or function. The operation has ended, possibly because the program was not able to handle data that was not correct or because the user canceled it.	40
Abnormal end of job or program. The job was not started or failed to start, a job-level function might not have been done as required, or the job might have been canceled.	50
System status. Issued only to the system operator message queue. It gives either the status of or a warning about a device, a subsystem, or the system.	60
Device integrity. Issued only to the system operator message queue, indicating that a device is not working correctly or is in some way no longer operational.	70
System alert and user messages. A condition exists that, although not severe enough to stop the system now, could become more severe unless preventive measures are taken.	80
System integrity. Issued only to the system operator message queue. Describes a condition where either a subsystem or system cannot operate.	90
Action. Some manual action is required, such as entering a reply or changing printer forms.	99

Distributed relational database messages:

If an error message occurs at either an application server (AS) or an application requester (AR), the server message is logged on the job log to indicate the reason for the failure.

A server message exists for each SQLCODE returned from an SQL statement supported by the DB2 Universal Database for iSeries program. The message is made available in precompiler listings, on Interactive SQL, or in the job log when running in the debug mode. However, when you are working with an AS that is not an iSeries server, there might not be a specific message for every error condition in the following cases:

- The error is associated with a function not used by the iSeries server.
For example, the special register CURRENT SQLID is not supported by DB2 UDB for iSeries, so SQLCODE -411 (SQLSTATE 56040) CURRENT SQLID cannot be used in a statement that references remote objects does not exist.
- The error is product-specific and will never occur when using DB2 UDB for iSeries.
DB2 UDB for iSeries will never have SQLCODE -925 (SQLSTATE 56021), SQL commit or rollback is invalid in an IMS™ or CICS environment.

For SQLCODEs that do not have corresponding messages, a generic message is returned that identifies the unrecognized SQLCODE, SQLSTATE, and tokens, along with the relational database name of the AS which generated the message. To determine the specific condition and how to interpret the tokens, consult the product documentation corresponding to the particular release of the connected AS.

Messages in the ranges CPx3E00 through CPx3EFF and CPI9100 through CPI91FF are used for distributed relational database server messages. The following list is not inclusive, but shows more common server messages you might see in a distributed database job log on an iSeries server.

Table 10. Distributed relational database messages

MSG ID	Description
CPA3E01	Attempt to delete *LOCAL RDB directory entry
CPC3EC5	Some parameters for RDB directory entry ignored
CPD3E30	Conflicting remote network ID specified
CPD3E35	Structure of remote location name not valid for ...
CPD3E36	Port identification is not valid
CPD3E38	Type conflict for remote location
CPD3E39	Value &3 for parameter &2 not allowed
CPD3E3B	Error occurred retrieving server authorization information for ...
CPD3ECA	RDB directory operation may not have completed
CPD3E01	DBCS or MBCS CCSID not supported.
CPD3E03	Local RDB name not in RDB directory
CPD3E05	DDM conversation path not found
CPD3E31	DDM TCP/IP server is not active
CPD3E32	Error occurred ending DDM TCP/IP server
CPD3E33	DDM TCP/IP server error occurred with reason code ...
CPD3E34	DDM TCP/IP server communications error occurred
CPD3E37	DDM TCP/IP get host by name failure
CPF3E30	Errors occurred starting DDM TCP/IP server

Table 10. Distributed relational database messages (continued)

MSG ID	Description
CPF3E31	Unable to start DDM TCP/IP server
CPF3EC6	Change DDM TCP/IP attributes failed
CPF3EC9	Scope message for interrupt RDB
CPF3E0A	Resource limits error
CPF3E0B	Query not open
CPF3E0C	FDOCA LID limit reached
CPF3E0D	Interrupt not supported
CPF3E01	DDM parameter value not supported
CPF3E02	AR cannot support operations
CPF3E04	SBCS CCSID not supported
CPF3E05	Package binding not active
CPF3E06	RDB not found
CPF3E07	Package binding process active
CPF3E08	Open query failure
CPF3E09	Begin bind error
CPF3E10	AS does not support DBCS or MC
CPF3E12	Commit/rollback HOLD not supported
CPF3E13	Commitment control operation failed
CPF3E14	End RDB request failed
CPF3E16	Not authorized to RDB
CPF3E17	End RDB request is in progress
CPF3E18	COMMIT/ROLLBACK with SQLCA
CPF3E19	Commitment control operation failed
CPF3E20	DDM conversation path not found
CPF3E21	RDB interrupt fails
CPF3E22	Commit resulted in a rollback at the application server
CPF3E23	DDM data stream violates conversation capabilities
CPF3E30	Errors occurred starting DDM TCP/IP server
CPF3E32	Server error occurred processing client request
CPF3E80	Data stream syntax error
CPF3E81	Invalid FDOCA descriptor
CPF3E82	ACCRDB sent twice
CPF3E83	Data mismatch error
CPF3E84	DDM conversational protocol error
CPF3E85	RDB not accessed
CPF3E86	Unexpected condition
CPF3E87	Permanent agent error
CPF3E88	Query already open
CPF3E89	Query not open
CPF3E99	End RDB request has occurred

Table 10. Distributed relational database messages (continued)

MSG ID	Description
CPI9150	DDM job started
CPI9152	Target DDM job started by application requester (AR)
CPI9160	DDM connection started over TCP/IP
CPI9161	DDM TCP/IP connection ended
CPI9162	Target job assigned to handle DDM connection started
CPI9190	Authorization failure on distributed database
CPI3E01	Local RDB accessed successfully
CPI3E02	Local RDB disconnected successfully
CPI3E04	Connection to relational database &1; ended
CPI3E30	DDM TCP/IP server already active
CPI3E31	DDM TCP/IP server does not support security mechanism
CPI3E32	DDM server successfully started
CPI3E33	DDM server successfully ended
CPI3E34	DDM job xxxx servicing user yyy on mm/dd/yy at hh:mm:ss (This can be suppressed with QRWOPTIONS)
CPI3E35	No DDM server prestart job entry
CPI3E36	Connection to relational database xxxx ended
SQ30082	A connection attempt failed with reason code...
SQL7992	Connect completed over TCP/IP
SQL7993	Already connected

Related concepts

“QRWOPTIONS data area” on page 184

When DDM or DRDA TCP/IP server jobs are initiated, they look for a data area in which the user can specify diagnostic and other options. The name is QRWOPTIONS, and it must reside in the QGPL library to take effect. It consists of a string of 48 characters.

Related tasks

“Track request information with the job log of a distributed relational database” on page 103

Every job on the iSeries server has a job log that contains information related to requests entered for a job.

Related information

SQL messages and codes

Handle program start request failures for APPC

When a program start request is received by an i5/OS subsystem on the application server (AS), the server attempts to start a job based on the information sent with the program start request. The application requester (AR) user’s authority to the application server (AS), existence of the requested database, and many other items are checked.

If the AS subsystem determines that it cannot start the job (for example, the user profile does not exist on the AS, the user profile exists but is disabled, or the user is not properly authorized to the requested objects on the AS), the subsystem sends a message, CPF1269, to the QSYSMSG message queue (or QSYSOPR when QSYSMSG does not exist). The CPF1269 message contains two reason codes (one of the reason codes might be zero, which can be ignored).

The nonzero reason code gives the reason why the program start request was rejected. Because the remote job was to have started on the AS, the message and reason codes are provided on the application server, and not the application requester. The user at the AR only knows that the program start request failed, not why it failed. The user on the AR must either talk to the system operator at the AS, or use display station pass-through to the AS to determine the reason why the request failed.

Related information

ICF Programming PDF

Handle connection request failures for TCP/IP

The main causes for failed connection requests at a Distributed Relational Database Architecture (DRDA) server configured for TCP/IP use is that the DDM TCP/IP server is not started, an authorization error occurred, or the machine is not running.

Server is not started or the port ID is not valid:

The error message given if the DDM TCP/IP server is not started is CPE3425.

The message is:

A remote host refused an attempted connect operation.

You can also get this message if you specify the wrong port on the Add Relational Database Directory Entry (ADDRDBDIRE) or the Change Relational Database Directory Entry (CHGRDBDIRE) command. For a DB2 Universal Database for iSeries server, the port should usually be *DRDA (the DRDA well-known port of 446). However, if you have configured port 447 for use with IPSec, you might want to use that port for transmitting sensitive data. If you are using a DRDA client that supports Secure Sockets Layer (SSL), you must connect to port 448 on the server.

To start the DDM server on the remote server, run the Start TCP/IP Server (STRTCPSVR) *DDM command. You can request that the DDM server be started whenever TCP/IP is started by running the Change DDM TCP/IP Attributes (CHGDDMTCPA) AUTOSTART(*YES) command.

Related information

Add Relational Database Directory Entry (ADDRDBDIRE) command
Change DDM TCP/IP Attributes (CHGDDMTCPA) command
Change Relational Database Directory Entry (CHGRDBDIRE) command
Secure Sockets Layer for DDM and DRDA
Start TCP/IP Server (STRTCPSVR) command

DRDA connect authorization failure:

The error messages given for an authorization failure is SQ30082.

The message text is:

Authorization failure on distributed database connection attempt.

The cause section of the message gives a reason code and a list of meanings for the possible reason codes. Reason code 17 means that there was an unsupported security mechanism (SECMEC).

DB2 Universal Database for iSeries implements several Distributed Relational Database Architecture (DRDA) SEMECs that an iSeries application requester (AR) can use:

- User ID only
- User ID with password
- Encrypted password security mechanism
- Kerberos (V5R2)

The encrypted password is sent only if a password is available at the time the connection is initiated.

The default SECMEC for an iSeries server requires user IDs with passwords. If the application requester sends a user ID with no password to a server, with the default security configuration, error message SQ30082 with reason code 17 is given.

Solutions for the unsupported security mechanism failure are:

- If the client is trusted by the server and authentication is not required, change the DDM TCP/IP server's authentication setting to password not required.
- If the client is not trusted by the server and authentication is required, change the application to send either a password or authenticated security token (for example, a Kerberos token).

The DDM TCP/IP server's authentication setting can be changed either using the Change DDM TCP/IP Attributes (CHGDDMTCPA) command or by using Operations Navigator's **Network** → **Servers** → **DDM Properties**.

You can send a password by either using the USER/USING form of the SQL CONNECT statement, or by using the Add Server Authentication Entry (ADDSVRAUTE) command to add the remote user ID and the password in a server authorization entry for the user profile under which the connection attempt is made. In V4R5 and later systems, an attempt is automatically made to send the password encrypted. Note that pre-V4R5 iSeries servers cannot send encrypted passwords, nor can they decrypt encrypted passwords of the type sent by V4R5 iSeries ARs.

Note that you have to have system value QRETSVRSEC (retain server security data) set to '1' to be able to store the remote password in the server authorization entry.

Attention: You must enter the RDB name on the Add Server Authentication Entry (ADDSVRAUTE) command in uppercase for use with DRDA or the name will not be recognized during the connection processing and the information in the authorization entry will not be used.

Related information

Add Server Authentication Entry (ADDSVRAUTE) command
Change DDM TCP/IP Attributes (CHGDDMTCPA) command

Server not available:

If a remote server is not up and running, or if you specify an incorrect IP address in the RDB directory entry for the application server (AS), you will get message CPE3447.

The message text is:

A remote host did not respond within the timeout period.

There is normally a several minute delay before this message occurs. It might appear that something is hung up or looping during that time.

Connection failures specific to Interactive SQL:

Sometimes when you are running a CONNECT statement from Interactive SQL, a general SQ30080 message is given.

The text of that message is:

Communication error occurred during distributed database processing

In order to get the details of the error, you should exit from Interactive SQL and look at the job log.

If you get message SQL7020, SQL package creation failed, when connecting for the first time (for any given level of commitment control) to a server that has only single-phase commit capabilities, the cause might be that you accessed the remote server as a read-only server and you need to update it to create the SQL package.

You can verify that by looking at the messages in the job log. The solution is to do a RELEASE ALL and COMMIT to get rid of all connections before connecting, so that the connection will be updatable.

Related tasks

“Set up SQL packages for Interactive SQL” on page 39

This topic applies only to non-iSeries Application Servers (ASs).

Not enough prestart jobs at server:

If the number of prestart jobs associated with the TCP/IP server is limited by the QRWTSRVR prestart job entry of the QSYSWRK subsystem, and all prestart jobs are being used for a connection, an attempt at a new connection will fail with these messages.

CPE3426

A connection with a remote socket was reset by that socket.

CPD3E34

DDM TCP/IP communications error occurred on recv() — MSG_PEEK.

You can avoid this problem at the server by setting the MAXJOBS parameter of the Change Prestart Job Entry (CHGPJE) command for the QTWTSRVR entry to a higher number or to *NOMAX, and by setting the ADLJOBS parameter to something other than 0.

Related information

Change Prestart Job Entry (CHGPJE) command

Application problems

The best time to handle a problem with an application is before it goes into production. However, it is impossible to anticipate all the conditions that will exist for an application when it gets into general use.

The job log of either the application requester (AR) or the application server (AS) can tell you that a package failed; the listings of the program or the package can tell you why it failed. The SQL compilers provide diagnostic tests that show the SQLCODEs and SQLSTATEs generated by the precompile process on the diagnostic listing.

For Integrated Language Environment[®] (ILE) precompiles, you can optionally specify OPTION(*XREF) and OUTPUT(*PRINT) to print a precompile source and cross-reference listing. For non-ILE precompiles, you can optionally specify *SOURCE and *XREF on the OPTIONS parameter of the Create SQL program (CRTSQLxxx) commands to print a precompile source and cross-reference listings.

Listings

The listing from the Create SQL program (CRTSQLxxx) command provides these kinds of information.

- The values supplied for the parameters of the precompile command
- The program source
- The identifier cross-references
- The messages resulting from the precompile

Precompiler listing:

```

5722ST1 V5R4M0 060210          Create SQL ILE C Object UPDATEPGM 02/10/06 14:30:10   Page  1
Source type.....C
Object name.....TST/UPDATEPGM
Source file.....*LIBL/QCSRC
Member.....*OBJ
Options.....*XREF
Listing option.....*PRINT
Target release.....*CURRENT
INCLUDE file.....*LIBL/*SRCFILE
Commit.....*CHG
Allow copy of data.....*YES
Close SQL cursor.....*ENDACTGRP
Allow blocking.....*READ
Delay PREPARE.....*NO
Generation level.....10
Margins.....*SRCFILE
Printer file.....*LIBL/QSYSPT
Date format.....*JOB
Date separator.....*JOB
Time format.....*HMS
Time separator.....*JOB
Replace.....*YES
Relational database.....RCHASLKM
User.....*CURRENT
RDB connect method.....*DUW
Default Collection.....*NONE
Package name.....*OBJLIB/*OBJ
Created object type.....*PGM
Debugging view.....*NONE
Dynamic User Profile.....*USER
Sort Sequence.....*JOB
Language ID.....*JOB
IBM SQL flagging.....*NOFLAG
ANS flagging.....*NONE
Text.....*SRCMBRTXT
Source file CCSID.....37
Job CCSID.....65535
Source member changed on 02/10/06 14:25:33

```

```

5722ST1 V5R4M0 060210          Create SQL ILE C Object UPDATEPGM 02/10/06 14:30:10   Page  2
Record*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 SEQNBR Last change
1  /*****/ 100
2  /* This program is called to update the DEPTCODE of file RWDS/DPT1 */ 200
3  /* to NULL. This is run once a month to clear out the old */ 300
4  /* data. */ 400
5  /* */ 500
6  /* NOTE: Because this program was compiled with an RDB name, it is */ 600
7  /* not necessary to do a connect, as an implicit connect will take */ 700
8  /* place when the program is called. */ 800
9  /*****/ 900
10 #include <stdio.h> 1000
11 #include <stdlib.h> 1100
12 exec sql include sqlca; 1200
13 1300
14 main() 1400
15 { 1500
16     /* Just update RWDS/DPT1, setting deptcode = NULL */ 1600
17     exec sql update RWDS/DPT1 1700
18         set deptcode = NULL; 1800
19 } 1900
* * * * * E N D   O F   S O U R C E   * * * * *

```

Figure 19. Listing from a precompiler

Figure 20. Listing from a precompiler (continued)

```
5722ST1 V5R4M0 060210          Create SQL ILE C Object UPDATEPGM 02/10/06 14:30:10   Page    3
CROSS REFERENCE
Data Names                      Define    Reference
DEPTCODE                        ****     COLUMN
18
DPT1                             ****     TABLE IN RWDS
17
RWDS                             ****     COLLECTION
17
5722ST1 V5R4M0 060210          Create SQL ILE C Object UPDATEPGM 02/10/06 14:30:10   Page    4
DIAGNOSTIC MESSAGES
MSG ID  SEV  RECORD  TEXT
SQL0088  0    17    Position 15 UPDATE applies to entire table.
SQL1103 10    17    Field definitions for file DPT1 in RWDS not found.
Message Summary
Total  Info  Warning  Error  Severe  Terminal
2     1     1       0     0       0
10 level severity errors found in source
19 Source records processed
***** END OF LISTING *****
```

CRTSQLPKG listing:

The listing from the Create Structured Query Language Package (CRTSQLPKG) command here provides several types of information:

- The values used on the parameters of the command
- The statement in error, if any
- The messages resulting from running the Create Structured Query Language Package (CRTSQLPKG) command

The following figure illustrates this information:

```

5722ST1 V5R4M0 060210          Create SQL package 02/10/06 14:30:31  Page  1
Record*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 SEQNBR  Last change
Program name.....TST/UPDATEPGM
Relational database.....*PGM
User .....*CURRENT
Replace.....*YES
Default Collection.....*PGM
Generation level.....10
Printer file.....*LIBL/QSYSVRT
Object type.....*PGM
Module list.....*ALL
Text.....*PGMTXT
Source file.....TST/QCSRC
Member.....UPDATEPGM

5722ST1 V5R4M0 060210          Create SQL package 02/10/06 14:30:31  Page  2
Record*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 SEQNBR  Last change
17  UPDATE RWDS / DPT1 SET deptcode = NULL
DIAGNOSTIC MESSAGES
MSG ID  SEV  RECORD  TEXT
SQL0204  10      17  Position 17 DPT1 in RWDS type *FILE not found.
SQL5057              SQL Package UPDATEPGM in TST created at KC000 from
module UPDATEPGM.
Message Summary
Total  Info  Warning  Error  Severe  Terminal
1      0      1         0      0        0
10 level severity errors found in source
* * * * * E N D   O F   L I S T I N G   * * * * *

```

Figure 21. Listing from CRTSQLPKG

Related information

Create Structured Query Language Package (CRTSQLPKG) command

SQLCODEs and SQLSTATEs

Program interfaces using SQL return error information to the application program when an error occurs. SQLSTATEs and their corresponding SQLCODEs are returned to the application program in either the SQL communication area (SQLCA) or the SQL diagnostic area.

An SQLCA is a collection of variables in a control block in space provided by the application that is updated by the database management system with information about the SQL statement most recently run. An SQL diagnostic area is a more complex storage area in space provided by the database manager that is designed to communicate more extensive information about the SQL statement most recently run.

When an SQL error is detected, a five-character global variable called the SQLSTATE identifies the nature of the error. In addition to the SQLSTATE, an integer SQLCODE is also available. However, the SQLCODE does not return the same return code for the same error condition among the current four IBM relational database products. SQLSTATE has been designed so that application programs can test for specific error conditions or classes of errors regardless of which DB2 product the application is connected to.

If SQL encounters a hard error while processing a statement, the SQLCODE is a negative number (for example, SQLCODE -204). If SQL encounters an exceptional but valid condition (warning) while processing a statement, the SQLCODE is a positive number (for example, SQLCODE +100). If SQL encounters no error or exceptional condition while processing a statement, the SQLCODE is 0. Every DB2 Universal Database for iSeries SQLCODE has a corresponding message in message file QSQLMSG in library QSYS. For example, SQLCODE -204 is logged as message ID SQL0204.

Because the returned error information is a valuable problem-diagnosis tool, it is a good idea to include in your application programs the instructions necessary to display some of the information contained in either the returned SQLCA or SQL diagnostic area. The message tokens discussed here are also very valuable for problem diagnosis:

- SQLSTATE
Return code.
- SQLCODE (SQLCA) or DB2_RETURNED_SQLCODE (SQL diagnostic area)
Return code.
- SQLERRD(3) (SQLCA) or ROW_COUNT (SQL diagnostic area)
The number of rows updated, inserted, or deleted by SQL.

The complete message can be viewed online by using the Display Message Description (DSPMSGD) command.

Related information

SQL reference

SQL messages and codes

Display Message Description (DSPMSGD) command

Distributed relational database SQLCODEs and SQLSTATEs:

The list provides some of the common SQLCODEs and SQLSTATEs associated with distributed relational database processing.

In these brief descriptions of the SQLCODEs (and their associated SQLSTATEs), message data fields are identified by an ampersand (&), and a number (for example, &1). The replacement text for these fields is stored in SQLERRM if the application program is using an SQLCA, or in DB2_ORDINAL_TOKEN_n (where n is the token number) if the application program is using the SQL diagnostic area. More detailed cause and recovery information for any SQLCODE can be found by using the Display Message Description (DSPMSGD) command.

Table 11. SQLCODEs and SQLSTATEs

SQLCODE	SQLSTATE	Description
+100	02000	This SQLSTATE reports a No Data exception warning due to an SQL operation on an empty table, zero rows identified in an SQL UPDATE or SQL DELETE statement, or the cursor in an SQL FETCH statement was after the last row of the result table.
+114	0A001	Relational database name &1; not the same as current server &2;
+304	01515	This SQLSTATE reports a warning on a FETCH or SELECT into a host variable list or structure that occurred because the host variable was not large enough to hold the retrieved value. The FETCH or SELECT does not return the data for the indicated SELECT item, the indicator variable is set to -2 to indicate the return of a NULL value, and processing continues.
+331	01520	Character conversion cannot be performed.
+335	01517	Character conversion has resulted in substitution characters.
+551	01548	Not authorized to object & in &2 type &3.
+552	01542	Not authorized to &1;
+595	01526	Commit level &1; has been escalated to &2; lock.

Table 11. SQLCODEs and SQLSTATEs (continued)

SQLCODE	SQLSTATE	Description
+802	01519	This SQLSTATE reports an arithmetic exception warning that occurred during the processing of an SQL arithmetic function or arithmetic expression that was in the SELECT list of an SQL select statement, in the search condition of a SELECT or UPDATE or DELETE statement, or in the SET clause of an UPDATE statement. For each expression in error, the indicator variable is set to -2 to indicate the return of a NULL value. The associated data variable remains unchanged, and processing continues.
+863	01539	Only SBCS characters allowed to relational database &1;
+990	01587	This SQLSTATE reports a pending response or a mixed outcome from at least one participant during the two-phase process.
+30104	01615	Bind option ignored.
-114	42961	Relational database &1; not the same as current server &2;
-144	58003	Section number &1; not valid. Current high section number is &3; Reason &2;
-145	55005	Recursion not supported for heterogeneous application server.
-175	58028	The commit operation failed.
-189	22522	Coded Character Set identifier &1; is not valid.
-191	22504	A mixed data value is invalid.
-250	42718	Local relational database not defined in the directory.
-251	2E000 42602	Character in relational database name &1; is not valid.
-300	22024	A NUL-terminated input host variable or parameter did not contain a NUL.
-302	22001	Conversion error on input host variable &2;
-330	22021	Character conversion cannot be performed.
-331	22021	Character conversion cannot be performed.
-332	57017	Character conversion between CCSID &1; and CCSID &2; not valid.
-334	22524	Character conversion resulted in truncation.
-351 -352	56084	An unsupported SQLTYPE was encountered in a select-list or input-list.
-426	2D528	Operation invalid for application run environment. This SQLSTATE reports the attempt to use EXCSQLIMM or EXCSQLSTT to execute a COMMIT in a dynamic COMMIT restricted environment.
-427	2D529	Operation invalid for application run environment.
-501 -502 -507	24501	Execution failed due to an invalid cursor state. The identified cursor is not open.
-510	42828	This SQLSTATE reports an attempt to DELETE WHERE CURRENT OF CURSOR or UPDATE WHERE CURRENT OF CURSOR on a cursor that is fetching rows using a blocking protocol.
-525	51015	Statement is in error.
-551	42501	Not authorized to object &1; in &2; type *&3;
-552	42502	Not authorized to &1;

Table 11. SQLCODEs and SQLSTATEs (continued)

SQLCODE	SQLSTATE	Description
-683	42842	FOR DATA clause or CCSID clause not valid for specified type.
-752	0A001	Application process is not in a connectable state. Reason code &1;
-802	22003 22012	A numeric value is out of range and division by zero is invalid.
-805	51002	SQL package &1; in &2; not found.
-818	51003	Consistency tokens do not match.
-842	08002	The connection already exists.
-862	55029	Local program attempted to connect to remote relational database.
-871	54019	Too many CCSID values specified.
-900	08003	The connection does not exist.
-918	51021	SQL statements cannot be executed until the application process executes a rollback operation.
-922	42505	This SQLSTATE reports a failure to authenticate the user during connection processing to an application server.
-925 -926	2D521	SQL COMMIT or ROLLBACK statements are invalid in the current environment.
-950	42705	Relational database &1; not in relational directory.
-952	57014	Processing of the SQL statement was ended by ENDRDBRQS command.
-969	58033	Error occurred when passing request to application requester driver program.
-7017	42971	Commitment control is already active to a DDM target.
-7018	42970	COMMIT HOLD or ROLLBACK HOLD is not allowed.
-7021	57043	Local program attempting to run on application server.
-30000	58008	Distributed Relational Database Architecture (DRDA) protocol error.
-30001	57042	Call to distributed SQL program not allowed.
-30020	58009	Distributed Relational Database Architecture (DRDA) protocol error.
-30021	58010	Distributed relational database not supported by remote server.
-30040	57012	DDM resource &2; at relational database &1; unavailable.
-30041	57013	DDM resources at relational database &1; unavailable.
-30050	58011	DDM command &1; is not valid while bind process in progress.
-30051	58012	bind process with specified package name and consistency token not active.
-30052	42932	Program preparation assumptions are incorrect.
-30053	42506	Not authorized to create package for owner&1;
-30060	08004	User not authorized to relational database &1;
-30061	08004	Relational database &1; not found.
-30070	58014	Distributed Data Management (DDM) command &1; not supported.
-30071	58015	Distributed Data Management (DDM) object &1; not supported.

Table 11. *SQLCODEs and SQLSTATEs (continued)*

SQLCODE	SQLSTATE	Description
-30072	58016	Distributed Data Management (DDM) parameter &1; not supported.
-30073	58017	Distributed Data Management (DDM) parameter value &1; not supported.
-30074	58018	Distributed Data Management (DDM) reply message &1; not supported.
-30080	08001	Communication error occurred during distributed database processing.
-30082	08001	Authorization failure on distributed database connection attempt.
-30090	25000 2D528 2D529	Change request not valid for read-only application server.
-30104	56095	Invalid bind option. This SQLSTATE reports that one or more bind options were not valid at the server. The bind operation terminates. The first bind option in error is reported in SQLERRMC.
-30105	56096	Conflicting bind options. The bind operation terminates. The bind options in conflict are reported in SQLERRMC.
Unrecognized by AR	58020	SQLSTATE value not defined for the error or warning.

Related information

SQL messages and codes

Display Message Description (DSPMSGD) command

System and communications problems

When a problem with a system or its communications occurs, a message is generated. System-detected problems are automatically entered into the problem log, where they can be viewed and analyzed.

You can run problem analysis on logged problems at any time by entering the Analyze Problem (ANZPRB) command from any system command line. This command takes you through an analysis procedure and stores additional problem-related information in the problem log.

Use the Work with Problems (WRKPRB) command to view the problem log. The following displays show the two views of the problem log:

```

Work with Problems
System: KC000
Position to . . . . . Problem ID

Type options, press Enter.
2=Change 4=Delete 5=Display details 6=Print details
8=Work with problem 9=Work with alerts 12=Enter notes

Opt Problem ID Status Problem Description
— 9114350131 READY User detected a hardware problem on a differen
— 9114326436 OPENED System cannot call controller . No lines avail
— 9114326281 OPENED Line failed during insertion into the token-r
— 9114324416 OPENED Device failed, recovery stopped.
— 9114324241 OPENED System cannot call controller . No lines avail
— 9114324238 OPENED System cannot call controller . No lines avail
— 9114324234 OPENED System cannot call controller . No lines avail
— 9114324231 OPENED System cannot call controller . No lines avail
— 9114324227 OPENED System cannot call controller . No lines avail
— 9114324224 OPENED System cannot call controller . No lines avail
— 9114324218 OPENED System cannot call controller . No lines avail
More...
F3=Exit F5=Refresh F6=Print list F11=Display dates and times
F12=Cancel F16=Report prepared problems F24=More keys

```

Press F11 on the first view to see the following display:

```

Work with Problems
System: KC000
Position to . . . . . Problem ID

Type options, press Enter.
2=Change 4=Delete 5=Display details 6=Print details
8=Work with problem 9=Work with alerts 12=Enter notes

Opt Problem ID Date Time Origin
— 9114350131 03/29/92 14:36:05 APPN.KC000
— 9114326436 03/29/92 07:41:59 APPN.KC000
— 9114326281 03/29/92 07:39:17 APPN.KC000
— 9114324416 03/29/92 07:06:42 APPN.KC000
— 9114324241 03/29/92 07:03:38 APPN.KC000
— 9114324238 03/29/92 07:03:35 APPN.KC000
— 9114324234 03/29/92 07:03:31 APPN.KC000
— 9114324231 03/29/92 07:03:27 APPN.KC000
— 9114324227 03/29/92 07:03:24 APPN.KC000
— 9114324224 03/29/92 07:03:20 APPN.KC000
— 9114324218 03/29/92 07:03:14 APPN.KC000
More...
F3=Exit F5=Refresh F6=Print list F11=Display descriptions F12=Cancel
F14=Analyze new problem F16=Report prepared problems F18=Work with alerts

```

iSeries problem log support allows you to display a list of all the problems that have been recorded on the local server. You can also display detailed information about a specific problem such as the following items:

- Product type and serial number of device with a problem
- Date and time of the problem
- Part that failed and where it is located
- Problem status

From the problem log you can also analyze a problem, report a problem, or determine any service activity that has been done.

Related concepts

“iSeries problem handling overview” on page 149

The i5/OS licensed program helps you manage problems for both user- and system-detected problems that occur on local and remote iSeries servers.

Related information

Analyze Problem (ANZPRB) command

Work with Problem (WRKPRB) command

Get data to report a failure

This topic collection describes the kinds of data that you can print to help you diagnose a problem in a distributed relational database on iSeries servers.

This data is produced by the i5/OS licensed program. You can also use system operator messages and the application program (along with its data) to diagnose problems.

Print a job log

Every job on the iSeries server has a job log that contains information related to requests entered for that job. When a user is having a problem at an application requester (AR), the information in the job log might be helpful in diagnosing the problem.

One easy way to get this information is to have the user sign off with the command:

```
SIGNOFF *LIST
```

This command prints a copy of the user’s job log, or places it in an output queue for printing.

Another way to print the job log is by specifying LOG(4 00 *SECLVL) on the application job description. After the job is finished, all messages are logged to the job log for that specific job. You can print the job log by locating it on an output queue and running a print procedure.

The job log for the application server (AS) might also be helpful in diagnosing problems.

Related tasks

“Track request information with the job log of a distributed relational database” on page 103

Every job on the iSeries server has a job log that contains information related to requests entered for a job.

“Locate distributed relational database jobs” on page 104

When you are looking for information about a distributed relational database job on an application requester (AR) and you know the user profile that is used, you can find that job by using the Work with User Jobs (WRKUSRJOB) command.

Find job logs from TCP/IP server prestart jobs

When the connection ends that is serviced by one of the QRWTSRVR prestart jobs associated with the distributed data management (DDM) TCP/IP server, the prestart job is recycled for use by another connection. When this happens, the job log associated with the ended connection is cleared.

However, in certain cases the job log is spooled to a printer file before it is cleared. The job log is not printed to a spooled file if the client user ID and password were not successfully validated. If validation was successful, these are the conditions under which the job log is printed to a spooled file:

- If the operating system is at V5R1 or later and the server job’s message logging text level is *SECLVL or *MSG.
- If the server request handler routing program detects that a serious error occurred in processing the request that ended the connection.
- If the prestart job was being serviced (by use of the Start Service Job (STRSRVJOB) command).
- If the QRWOPTIONS data area on the client or server specified a job log output condition that was satisfied by the server job.

You might want to get job log information for several reasons. It is obviously useful for diagnosing errors. It can also be useful for analyzing performance problems. For example, if you want to get SQL optimizer data that is generated when running under debug, you can either manually start a service job and run the Start Debug (STRDBG) command, or you can set one or more applicable options in the QRWOPTIONS data area to cause the job log to be retained.

The logs of jobs in which failures occur during the connection phase will not be saved by the process described above. Jobs that are saved by that process will not be stored under the prestart job ID. To find them, run the following command:

```
WRKJOB userid/QPRTJOB
```

where *userid* is the user ID used on the CONNECT to the application server (AS). You can find that user ID if you do not know it with the Display Log (DSPLOG) command on the AS.

You can filter out unwanted messages by use of parameters like this:

```
DSPLOG PERIOD(('11:00')) MSGID(CPI3E34)
```

Look for the following message. Note, however, that if the QRWOPTIONS data area has been used to suppress this message (CPI3E34), it will not appear in the history log.

```
DDM job xxxx servicing user yyy on ddd at ttt.
```

Related concepts

“QRWOPTIONS data area” on page 184

When DDM or DRDA TCP/IP server jobs are initiated, they look for a data area in which the user can specify diagnostic and other options. The name is QRWOPTIONS, and it must reside in the QGPL library to take effect. It consists of a string of 48 characters.

Related information

Start Service Job (STRSRVJOB) command

Display Log (DSPLOG) command

Start Debug (STRDBG) command

Print the product activity log

The Product Activity Log on the iSeries server is a record of machine checks, device errors, and tape statistics. It also contains FFDC information including the first 1000 bytes of each FFDC dump. By reviewing these errors you might be able to determine the nature of a problem.

To print the product activity log for a server on which you are signed on, follow these steps:

1. Type the Print Error Log (PRERRLOG) command on any command line and press F4 (Prompt). The Print Error Log display is shown.
2. Type the parameter value for the kind of log information you want to print and press the Enter key. The log information is sent to the output queue identified for your job.
3. Enter the Work with Job (WRKJOB) command. The Work with Job display is shown.
4. Select the option to work with spooled files. The Work with Job Spooled Files display is shown.
5. Look for the log file you just created at or near the bottom of the spooled file list.
6. Type the work with printing status option in the *Opt* column next to the log file. The Work with Printing Status display is shown.
7. On the Work with Printing Status display, use the change status option to change the status of the file and specify the printer to print the file.

Related information

Print Error Log (PRERRLOG) command

Work with Job (WRKJOB) command

Job tracing

Sometimes a problem cannot be tracked to a specific program. In these cases, Start Trace (STRTRC) and Trace Job (TRCJOB) commands can be used for tracing module flow, i5/OS data acquisition and CL commands executed.

These tools should be used when the problem analysis procedures do not supply sufficient information about the problem. For distributed database applications, these commands are also useful for capturing distributed database requests and response data streams.

If you need to get a job trace of the application server job, you need to start a service job at the server.

Related concepts

“Start a service job to diagnose application server problems” on page 181

When an application uses Distributed Relational Database Architecture (DRDA), the SQL statements are run in the application server job. Because of this, you might need to start debug or a job trace for the application server job that is running on the i5/OS operating system. The technique for doing this differs based on the use of either Advanced Program-to-Program Communication (APPC) or TCP/IP.

“Communications trace” on page 175

If you get a message in the CPF3Exx range or the CPF91xx range when using Distributed Relational Database Architecture (DRDA) to access a distributed relational database, you should run a communications trace.

Trace job:

The Trace Job (TRCJOB) command is the older of the two tracing tools. As the trace records are generated, the records are stored in an internal trace storage area. When the trace is ended, the trace records can be written to a spooled printer file (QPSRVTRC) or directed to a database output file.

A sample trace scenario is as follows:

```
TRCJOB SET(*ON) TRCTYPE(*ALL) MAXSTG(2000)
        TRCFULL(*WRAP) EXITPGM($$CFTRC)
CALL QCMD
TRCJOB SET(*OFF) OUTPUT(*PRINT)
WRKOUTQ output-queue-name
```

You will see a spooled file with a name of QPSRVTRC. The spooled file contains your trace.

Related concepts

“Interpret trace job and FFDC data” on page 228

This topic collection provides additional problem-analysis information. It is useful to specialists responsible for problem determination. It is also for suppliers of software products designed to conform to the Distributed Relational Database Architecture who want to test connectivity to an iSeries server.

Related information

Trace Job (TRCJOB) command

Start trace:

You can also use the Start Trace (STRTRC) command to perform traces. The STRTRC command is more flexible and less intrusive than the Trace Job (TRCJOB) command. It allows tracing across multiple jobs and shows more in-depth details about the module flow.

As the trace records are generated, the records are stored in an internal trace storage area that is identified by a session ID. When the trace is ended using End Trace (ENDTRC), the trace records are placed in a user-specified library as a set of database files. These files can then be written to a spooled printer file (QPSRVTRCJ) or directed to a database output file by issuing the PRTRC.

A sample trace scenario is as follows:

```
STRTRC SSNID(DRDATRACE) JOB((*ALL/QUSER/QRWTSRVR)) MAXSTG(160000)
TRCFULL(*STOPTRC)
```

Run the failing DRDA scenario:

```
ENDTRC SSNID(DRDATRACE) DTALIB(TRACELIB)
PRTRTC DTAMBR(DRDATRACE) DTALIB(TRACELIB)
```

Related information

End Trace (ENDTRC) command

Start Trace (STRTRC) command

Communications trace

If you get a message in the CPF3Exx range or the CPF91xx range when using Distributed Relational Database Architecture (DRDA) to access a distributed relational database, you should run a communications trace.

The following list shows common messages you might see in these ranges.

Table 12. Communications trace messages

MSG ID	Description
CPF3E80	DDM data stream syntax error.
CPF91xx	DDM protocol error.
CPF3E83	Invalid FD0:CA descriptor.
CPF3E84	Data mismatch error.

You can perform two types of communications traces. The first is the standard communications trace. The second is the TRCTCPAPP function. The TRCTCPAPP function provides for intelligible traces where IPsec or the secure sockets protocol has encrypted the data streams. It captures the data before encryption and after decryption. However, it also works well for getting traces of unencrypted data streams. It is required for getting traces of intra-system DRDA flows where LOOPBACK is used.

Related concepts

“Job tracing” on page 174

Sometimes a problem cannot be tracked to a specific program. In these cases, Start Trace (STRTRC) and Trace Job (TRCJOB) commands can be used for tracing module flow, i5/OS data acquisition and CL commands executed.

Related tasks

“TCP/IP communications trace” on page 177

One of the uses of the trace tool is to show the clear text of a transmission in an environment where the data is encrypted.

Related information

“Connection security protocols for DDM and DRDA” on page 60

Several connection security protocols are supported by the current DB2 UDB for iSeries implementation of distributed data management (DDM) or Distributed Relational Database Architecture (DRDA) over TCP/IP.

Standard communications trace:

The communications trace function lets you start or stop a trace of data on communications configuration objects. After you have run a trace of data, you can format the data for printing or viewing. You can view the printer file only in the output queue.

Communication trace options run under system service tools (SST). SST lets you use the configuration objects while communications trace is active. You can trace and format data for any of the communications types that you can use in a distributed database network.

You can run the iSeries communications trace from any display that is connected to the server. Anyone with a special authority (SPCAUT) of *SERVICE can run the trace on iSeries server. Communications trace supports all line speeds.

You should use communications trace in the following situations:

- The problem analysis procedures do not supply sufficient information about the problem.
- You suspect that a protocol violation is the problem.
- You suspect a line noise to be the problem.
- The error messages indicate that there is a Systems Network Architecture (SNA) bind problem.

You must have detailed knowledge of the line protocols that you use to correctly interpret the data that is generated by a communications trace. For information about interpreting DRDA data streams, see "Example: Analyze the RW trace data" on page 230.

Whenever possible, start the communications trace before varying on the lines. This gives you the most accurate sample of your line as it varies on.

To run an APPC trace and to work with its output, you have to know on what line, controller, and device you are running.

To format and avoid unwanted data in the output of a TCP/IP trace, you can specify the IP addresses of the source and application servers (ASs). Sometimes it is sufficient to just specify the port number instead, which is easier.

The following commands start, stop, print, and delete communications traces:

Start Communications Trace (STRCMNTRC) command

Starts a communications trace for a specified line or network interface description. Specify *MAX for value of Beginning bytes in Number of bytes to trace parameter. A communications trace continues until you run the End Communications Trace (ENDCMNTRC) command.

End Communications Trace (ENDCMNTRC) command

Ends the communications trace running on the specified line or network interface description.

Print Communications Trace (PRTCMNTRC) command

Moves the communications trace data for the specified line or network interface description to a spooled file or an output file. Specify *YES for the format SNA data only parameter.

Delete Communications Trace (DLTCMNTRC) command

Deletes the communications trace for a specified line or network interface description.

Related information

Communications Management PDF

Delete Communications Trace (DLTCMNTRC) command

End Communications Trace (ENDCMNTRC) command

Print Communications Trace (PRTCMNTRC) command

Start Communications Trace (STRCMNTRC) command

Find your line, controller, and device descriptions:

Use the Work with Configuration Status (WRKCFGSTS) command to find the controller and device under which your application server job starts.

For example:

```
WRKCFGSTS CFGTYPE(*DEV)
          CFGD(*LOC)
          RMTLOCNAME(DB2ESYS)
```

The value for the RMTLOCNAME keyword is the application server's name.

The Work with Configuration Status (WRKCFGSTS) command shows all of the devices that have the specified server name as the remote location name. You can tell which device is in use because you can vary on only one device at a time. Use option 8 to work with the device description and then option 5 to display it. The attached controller field gives the name of your controller. You can use the (WRKCFGSTS) command to work with the controller and device descriptions. For example:

```
WRKCFGSTS CFGTYPE(*CTL)
          CFGD(PCXZZ1205) /* workstation */
WRKCFGSTS CFGTYPE(*CTL)
          CFGD(LANSLKM) /* AS/400 on token ring */
```

The CFGD values are the controller names that are acquired from the device descriptions in the first example in this topic.

The output from the Work with Configuration Status (WRKCFGSTS) command also includes the name of the line description that you need when working with communications traces. If you select option 8 and then option 5 to display the controller description, the active switched line parameter displays the name of the line description. The LAN remote adapter address gives the token-ring address of the remote server.

Another way to find the line name is to use the Work with Line Descriptions (WRKLIND) command, which lists all of the line descriptions for the server.

Related information

Work with Configuration Status (WRKCFGSTS) command

Work with Line Descriptions (WRKLIND) command

TCP/IP communications trace

One of the uses of the trace tool is to show the clear text of a transmission in an environment where the data is encrypted.

The trace data is captured before encryption at the sender, and after encryption at the receiver. However, the trace tool is useful in other environments as well. You can only use this function when you are using TCP/IP for communication.

To use the Trace TCP/IP Application (TRCTCPAPP) command, you must have a user profile with *SERVICE special authority. To start the trace, enter the following line:

```
TRCTCPAPP *DDM
```

If you want to restrict the trace to a certain port, for example port 448 for SSL, follow this example:

```
TRCTCPAPP *DDM *ON RMTNETADR(*INET *N '255.255.255.255' 448)
```

After the communication that you are tracing has finished, run the following command and look at the resulting spooled file:

```
TRCTCPAPP *DDM *OFF
```

If you traced more than one connection, you will need to locate and match your spool files to each QRWTSRVR job. The spool file name is QZBSTRC and the job is QRWxxxxxx, where xxxxxx is the job number placed in the user data for the spool file.

Restriction for use with *DDM application

When you use the Trace TCP/IP Application (TRCTCPAPP) command with the *DDM application, the maximum amount of data you can trace for a single sent or received message is limited to 6000 bytes.

Related concepts

"Communications trace" on page 175

If you get a message in the CPF3Exx range or the CPF91xx range when using Distributed Relational Database Architecture (DRDA) to access a distributed relational database, you should run a communications trace.

Related information

Trace TCP/IP Application (TRCTCPAPP) command

TCP/IP communication trace formatting:

The Trace TCP/IP Application (TRCTCPAPP) command can be used to break down DRDA and DDM flows into an easier-to-read logical representation. It also displays the information in ASCII which can be of help in unlike environments.

To request this formatting, enter the following while ending the communications trace:

```
TRCTCPAPP APP(*DDM) SET(*OFF) ARGLIST('lv1=2')
```

Here is an example of an unformatted trace, edited to fit the width of this topic:

```
0080D0010001007A 200100162110D9C3 C8C1E2D5E3E24040
4040404040404040 *..}.....RCHASNTS *
0006210F2407000D 002FD8E3C4E2D8D3 F4F0F0000C112ED8
E2D8F0F5F0F3F000 *.....QTDSQL400....QSQ05030.*
0A00350006119C00 2500062121241E00 062120241E0010D1
2A01000000000000 *.....J.....*
0000000000001621 35C1D7D7D54BD3D7 F0F6F6C1C2B9191C
F706F90005213BF1 *.....APPN.LP066AB...7.9....1*
```

This is the same trace, formatted using TRCTCPAPP:

```
-Datastream-----
DATA:                (ASCII)                (EBCDIC)
0080D0010001007A    200100162110D9C3    .8'.....a...b.ẽã .0}.....RC
C8C1E2D5E3E24040    4040404040404040    ç ë+ëë..... HASNTS
0006210F2407000D    002FD8E3C4E2D8D3    .Lb.f"....ëëääëë< .....QTDSQL
F4F0F0000C112ED8    E2D8F0F5F0F3F000    .....ëëë..... 400....QSQ05030.
0A00350006119C00    2500062121241E00    C...L.õ...LbbfK. ....ä.....
062120241E0010D1    2A01000000000000    LbafK..çk..... .....J.....
0000000000001621    35C1D7D7D54BD3D7    .....b. &&+.<& .....APPN.LP
F0F6F6C1C2B9191C    F706F90005213BF1    ... â".c.L...bB. 066AB¾..7.9....1
-Parsed-----
```

RECV(AS) RQSDSS - Request Data Stream Structure

LL: 128 CORR: 0001 CHAINED: n CONT ON ERR: n SAME CORR FOR NEXT DSS: n

NM: ACCRDB - Access RDB

LL: 122 CP: 2001

NM: RDBNAM - Relational Database Name

LL: 22 CP: 2110

ASCII: ẽãç ë+ëë.....

EBCDIC: RCHASNTS

NM: RDBACCCL - RDB Access Manager Class
 LL: 6 CP: 210F
 CODE POINT DATA: 2407
 NAME: SQLAM - SQL Application Manager
 NM: TYPDEFNAM - Data Type Definition Name
 LL: 13 CP: 002F
 ASCII: éèâëé<...
 EBCDIC: QTDSQL400
 NM: PRDID - Product-Specific Identifier
 LL: 12 CP: 112E
 DATA: (ASCII) (EBCDIC)
 D8E2D8F0F5F0F3F0 ëëé..... QSQ05030
 NM: TYPDEFOVR - TYPDEF Overrides
 LL: 10 CP: 0035
 NM: CCSIDSBC - CCSID for Single-Byte Characters
 LL: 6 CP: 119C
 DATA: (ASCII) (EBCDIC)
 0025
 NM: STTDECDL - Statement Decimal Delimiter
 LL: 6 CP: 2121
 CODE POINT DATA: 241E
 NAME: DFTPKG - Package Default
 NM: STTSTRDEL - Statement String Delimiter
 LL: 6 CP: 2120
 CODE POINT DATA: 241E
 NAME: DFTPKG - Package Default
 NM: SXXPRDDTA - Extended Product Data
 LL: 16 CP: D12A
 DATA: (EBCDIC)
 0100000000000000 00000000
 NM: CRRTKN - Correlation Token
 LL: 22 CP: 2135
 DATA: (ASCII) (EBCDIC)
 C1D7D7D54BD3D7F0 F6F6C1C2B9191CF7 &&+.<&... â".c. APPN.LP066AB¾.7

06F9 L. .9

NM: TRGDFTRT - Target Default Value Return

LL: 5 CP: 213B

BOOLEAN: TRUE

Related information

Trace TCP/IP Application (TRCTCPAPP) command

Find first-failure data capture data

This topic describes tips on how to locate first-failure data capture (FFDC) data on an iSeries server. The information is most useful if the failure causing the FFDC data output occurred on the application server (AS). The FFDC data for an application requester (AR) can usually be found in one of the spooled files associated with the job running the application program.

Note: No FFDC data is produced unless the QSFWERRLOG system value is set to *LOG.

1. Execute a Display Messages (DSPMSG) QSYSOPR command and look for a Software problem detected in Qccxyyyy message in the QSYSOPR message log. (cc in the program name is usually RW, but could be CN or SQ.) The presence of this message indicates that FFDC data was produced. You can use the help key to get details on the message. The message help gives you the problem ID, which you can use to identify the problem in the list presented by the Work with Problems (WRKPRB) command. You might be able to skip this step because the problem record, if it exists, might be at or near the top of the list.
2. Enter the Work with Problems (WRKPRB) command and specify the program name (Qccxyyyy) from the Software problem detected in Qccxyyyy message. Use the program name to filter out unwanted list items. When a list of problems is presented, specify option 5 on the line containing the problem ID to get more problem details, such as symptom string and error log ID.
3. When you have the error log ID, enter the Start System Service Tools (STRSST) command. On the first screen, select Start a service tool. On the next screen, enter 1 to select Error log utility. On the next screen, enter 2 to select Display or print by error log ID. In the next screen, you can:
 - Enter the error log ID.
 - Enter Y to get the hexadecimal display.
 - Select the Print or Display option.

The Display option gives 16 bytes per line instead of 32. This can be useful for online viewing and printing screens on an 80-character workstation printer. If you choose the Display option, use F6 to see the hexadecimal data after you press Enter.

The hexadecimal data contains the first one KB of the FFDC dump data, preceded by some other data. The start of the FFDC data is identified by the FFDC data index. The name of the target job (if this is on the application server) is before the data index. If the FFDC dump spool file has not been deleted, use this fully qualified job name to find the spool file. If the spool file is missing, either:

- Use the first one KB of the dump stored in the error log.
- Re-create the problem if the one KB of FFDC data is insufficient.

When interpreting FFDC data from the error log, the FFDC data in the error log is not formatted for reading as well as the data in the spooled files. Each section of the FFDC dump in the error log is prefixed by a four-byte header. The first two bytes of the header are the length of the following section (not counting the prefix). The second two bytes, which are the section number, correspond to the section number in the index.

Related reference

“FFDC dump output description” on page 238

This information describes the data areas and types of information available in a first-failure data capture (FFDC) dump output.

Related information

Display Messages (DSPMSG) command

Work with Problem (WRKPRB) command

Start System Service Tools (STRSST) command

Start a service job to diagnose application server problems

When an application uses Distributed Relational Database Architecture (DRDA), the SQL statements are run in the application server job. Because of this, you might need to start debug or a job trace for the application server job that is running on the i5/OS operating system. The technique for doing this differs based on the use of either Advanced Program-to-Program Communication (APPC) or TCP/IP.

Related concepts

“Job tracing” on page 174

Sometimes a problem cannot be tracked to a specific program. In these cases, Start Trace (STRTRC) and Trace Job (TRCJOB) commands can be used for tracing module flow, i5/OS data acquisition and CL commands executed.

Service jobs for APPC servers

When the DB2 Universal Database for iSeries application server recognizes a special transaction program name (TPN), it causes the application server to send a message to the system operator and then wait for a reply.

This allows you to issue a Start Service Job (STRSRVJOB) command that allows job trace or debug to be started for the application server job.

To stop the DB2 UDB for iSeries application server job and restart it in debug mode, follow these steps:

1. Specify QCNTRVC as the TPN at the application requester. There is a different method of doing this for each platform. This topic collection describes the different methods. When the i5/OS application receives a TPN of QCNTRVC, it sends a CPF9188 message to QSYSOPR and waits for a G (for go) reply.
2. Before entering the G reply, use the Start Service Job (STRSRVJOB) command to start a service job for the application server job and put it into debug mode. (Request help on the CPF9188 message to display the jobname.)
3. Enter the Start Debug (STRDBG) command.
4. After starting debug for the application server job, reply to the QSYSOPR message with a G. After receiving the G reply, the application server continues with normal DRDA processing.
5. After the application runs, look at the application server job log to see the SQL debug messages.

Related concepts

“Create your own TPN and set QCNTRVC”

To create your own TPN and set QCNTRVC, do the following tasks.

Related information

Start Service Job (STRSRVJOB) command

Start Debug (STRDBG) command

Create your own TPN and set QCNTRVC

To create your own TPN and set QCNTRVC, do the following tasks.

Related tasks

“Service jobs for APPC servers” on page 181

When the DB2 Universal Database for iSeries application server recognizes a special transaction program name (TPN), it causes the application server to send a message to the system operator and then wait for a reply.

Set QCNTRVC as a TPN on a DB2 Universal Database for iSeries application requester:

Specify the QCNTRVC on the TNSPGM parameter of the Add Relational Database Directory Entry (ADDRDBDIRE) or Change Relational Database Directory Entry (CHGRDBDIRE) command.

It might be helpful to make a note of the special transaction program name (TPN) in the text of the RDB directory entry as a reminder to change it back when you are finished with debugging.

Related information

Add Relational Database Directory Entry (ADDRDBDIRE) command

Change Relational Database Directory Entry (CHGRDBDIRE) command

Create your own TPN for debugging a DB2 Universal Database for iSeries application server job:

You can create your own TPN by compiling a CL program that contains debug statements and a TFRCTL QSYS/QCNTEDDM statement at the end. The advantage of this is that you do not need any manual intervention when making the connection.

See the following code for an example program:

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 250.

```
PGM
MONMSG CPF0000
STRDBG UPDPROD(*YES) PGM(CALL/QRWTEEXEC) MAXTRC(9999)
ADDBKP STMT(CKUPDATE) PGMVAR((*CHAR (SQLDA@))) OUTFMT(*HEX) +
      LEN(1400)
ADDTRC PGMVAR((DSLENGTH ( )) (LNTH ( )) (FDODTA_LNTH ( )))
TRCJOB *ON TRCTYPE(*DATA) MAXSTG(2048) TRCFULL(*STOPTRC)
TFRCTL QSYS/QCNTEDDM
ENDPGM
```

The TPN name in the RDB directory entry of the application requester (AR) is the name that you supply. Use the text field to provide a warning that the special TPN is in use, and be sure to change the TPN name back when you have done debugging.

Be aware that when you change the TPN of an RDB, all connections from that AR will use the new TPN until you change it back. This might cause surprises for unsuspecting users, such as poor performance, long waits for operator responses, and the filling up of storage with debug data.

Set QCNTRVC as a TPN on a DB2 Universal Database for VM application requester:

Change the UCOMDIR NAMES file to specify QCNTRVC in the TPN tag.

For example:

```
:nick.RCHASLAI :tpn.QCNTRVC
                :luname.VM4GATE RCHASLAI
                :modename.MODE645
                :security.NONE
```

Then issue SET COMDIR RELOAD USER.

Set QCNTRVC as a TPN on a DB2 Universal Database for z/OS application requester:

Update the SYSIBM.LOCATIONS table to specify QCNTRVC in the TPN column for the row that contains the RDB-NAME of the DB2 Universal Database for iSeries application server.

Set QCNTRVC as a TPN on a DB2 Universal Database for Linux, UNIX and Windows application requester:

If you are working with DB2 Universal Database for Linux, UNIX and Windows and would like instructions on how to set up the TPN on this family of products, there is a Web page to help you.

See the Web page DB2 Universal Database for Linux, UNIX and Windows. There you can find the several books specific to different versions.

Service jobs for TCP/IP servers

The DDM TCP/IP server does not use transaction program names (TPNs) as the Advanced Program-to-Program Communication (APPC) server does. However, the use of prestart jobs by the TCP/IP server provides a way to start a service job in that environment.

Note, however, that with the introduction of the function associated with the QRWOPTIONS data area usage, you might not need to start a service job in many cases. That feature allows one to start traces and do other diagnostic functions. You might still need to start a service job if you need a trace of the connection phase of the job.

You can use the Display Log (DSPLOG) command to find the CPI3E34 message reporting the name of the server job being used for a given connection if the following statements are true:

- You do not need to trace the actions of the server during the connect operation
- You choose not to use the QRWOPTIONS function
- You have the ability to delay execution of the application requester (AR) job until you can do some setup on the server, such as from Interactive SQL

You can then use the Start Service Job (STRSRVJOB) command.

If you do need to trace the connect statement, or do not have time to do manual setup on the server after the connection, you will need to anticipate what prestart job will be used for the connection before it happens. One way to do that is to prevent other users from connecting during the time of your test, if possible, and end all of the prestart jobs except one.

You can force the number of prestart jobs to be 1 by setting the following parameters on the Change Prestart Job Entry (CHGPJE) command for QRWTSRVR running in QSYSWRK to the values specified here:

- Initial number of jobs: **1**
- Threshold: **1**
- Additional number of jobs: **0**
- Maximum number of jobs: **1**

If you use this technique, be sure to change the parameters back to values that are reasonable for your environment; otherwise, users will get the message that 'A connection with a remote socket was reset by that socket' when trying to connect when the one prestart job is busy.

Related concepts

“QRWOPTIONS data area” on page 184

When DDM or DRDA TCP/IP server jobs are initiated, they look for a data area in which the user can specify diagnostic and other options. The name is QRWOPTIONS, and it must reside in the QGPL library to take effect. It consists of a string of 48 characters.

Related information

Start Service Job (STRSRVJOB) command
 Change Prestart Job Entry (CHGPJE) command
 Display Log (DSPLOG) command

QRWOPTIONS data area

When DDM or DRDA TCP/IP server jobs are initiated, they look for a data area in which the user can specify diagnostic and other options. The name is QRWOPTIONS, and it must reside in the QGPL library to take effect. It consists of a string of 48 characters.

Note: The information in the data area must be entered in uppercase in CCSID 37 or 500.

The format of the data area is as follows:

Table 13. Data area format

Columns	Contents
1-15	Client IP address in dotted decimal format for use when I is specified as a switch value (ignored otherwise).
16	Reserved area ignored by server (can contain a character for human usability)
17-26	User profile name for comparison when U is specified as a switch value (ignored otherwise)
27	Switch to cause job log to be kept if set to A, I or U (see notes 1 and 2)
28	Switch to cause DSPJOB output to be printed if set to A, I or U (see notes 1 and 2)
29	Switch to cause job to be traced if set to A, I or U (see notes 1 and 2).
30	Switch to cause debug to be started for job if set to A, I or U (see note 1).
31	Switch to invoke the Change Query Attributes (CHGQRYA) command with a QRYOPTLIB value if set to A, I or U. The QRYOPTLIB value is extracted from columns 39-48 which must contain the name of the library containing the proper QAQQINI file (see Note 1) Note: If an I or A is specified in this column, QUSER must have *JOBCTL special authority for it to take effect.
32	Switch to shadow client debug options if set to A, I or U (see note 1).
33	Switch to use old TRCJOB instead of new STRTRC for job trace if set to T and column 29 requests tracing. Note: If this column is set to T, TRCJOB will be used for the job trace. Set it to blank or S to use STRTRC.
34	Set this to N to suppress CPI3E34 messages in the history log (This is available in OS/400 V5R1 only with PTF SI02613)
35	Switch to start special subroutine trace if set to A, I, or U (see notes 1 and 2).
36-38	Reserved
39-48	General data area (contains library name if the Change Query Attributes (CHGQRYA) command is triggered by the appropriate value in column 31)

Notes:

- These are the switch values that activate the function corresponding to the column in which they appear:
 - A activates the function for all uses of the server job.
 - I activates the function if the client IP address specified in columns 1-15 matches that used on the connect attempt.

- U activates the function if the user ID specified in columns 17-26 matches that used on the connect attempt.
2. To find the spooled files resulting from this function, use Work with Job command (WRKJOB user-profile/QPRTJOB), where user-profile is the user ID used on the connect request. Take option 4 and you should see one or more of these files.

Table 14. File list from WRKJOB user-profile/QPRTJOB command

File	Device or queue	User data
QPJOBLOG	QEZJOBLOG	QRWTSRVR
QPDSJOB	PRT01	
QPSRVTRC	PRT01	

3. The file containing the special DRDA subroutine trace will be created in library QGPL, with a name in this format: QRWDBmmddy, where mm represents the month, dd the day, and y the last digit of the year, in which the trace was recorded. Not all server programs are traced.

Related tasks

“Display the history log” on page 124

Each time a client user establishes a successful connection with a server job, that job is swapped to run under the profile of that client user.

Related reference

“Distributed relational database messages” on page 158

If an error message occurs at either an application server (AS) or an application requester (AR), the server message is logged on the job log to indicate the reason for the failure.

“Find job logs from TCP/IP server prestart jobs” on page 172

When the connection ends that is serviced by one of the QRWTSRVR prestart jobs associated with the distributed data management (DDM) TCP/IP server, the prestart job is recycled for use by another connection. When this happens, the job log associated with the ended connection is cleared.

“Service jobs for TCP/IP servers” on page 183

The DDM TCP/IP server does not use transaction program names (TPNs) as the Advanced Program-to-Program Communication (APPC) server does. However, the use of prestart jobs by the TCP/IP server provides a way to start a service job in that environment.

Related information

Change Query Attributes (CHGQRYA) command

Work with Job (WRKJOB) command

Example: CL command to create the data area:

This example requests the functions indicated in the table.

```
CRTDTAARA DTAARA(QGPL/QRWOPTIONS) TYPE(*CHAR) LEN(48)
          VALUE('9.5.114.107 :MYUSERID AAUIU TN INILIBRARY')
          TEXT('DRDA TCP SERVER DIAGNOSTIC OPTIONS')
```

Note: Because the proper spacing in the example is critical, the contents of the VALUE parameter are repeated in table form.

Table 15. Explanation of data elements in VALUE parameter of CRTDTAARA example

Columns	Contents	Explanation
1-11	9.5.114.107	IP address for use with switch in column 30.
16	:	Character to mark the end of the IP address field.
17-24	MYUSERID	User ID for use with switches in columns 29 and 31.
27	A	Spool the server job log (for QRWTSRVR) for all users.

Table 15. Explanation of data elements in VALUE parameter of CRTDTAARA example (continued)

Columns	Contents	Explanation
28	A	Spool the DSPJOB output for all uses of the server.
29	U	Trace the job with the (TRCJOB) command if the user ID on the connect request matches what is specified in columns 17 through 26 ('MYUSERID' in this example) of the data area.
30	I	Start debug with the Start Debug (STRDBG) command (specifying no program) if the client IP address ('9.5.114.107' in this example) matches what is specified in columns 1 through 15 of the data area.
31	U	Invoke the command Change Query Attributes (CHGQRYA) QRYOPTLIB(INILIBRARY) if the user ID on the connect request matches what is specified in columns 17 through 26 ('MYUSERID' in this example) of the data area. Note: The library name is taken from columns 39 through 48 of the data area.
32		Do not shadow client debug options to server.
33	T	Use the old TRCJOB facility for job traces.
34	N	Do not place CPI3E34 messages in the history log.
35		Do not initiate a subroutine trace.
39-48	INILIBRARY	Library used with switch 31.

Related information

Change Query Attributes (CHGQRYA) command

Start Debug (STRDBG) command

Examples: Application programming

This example application for distributed relational database use is written in RPG/400, COBOL/400, Java and ILE C/400 programming languages. This example shows how to use a distributed relational database for functional specification tasks.

Example: Business requirement for distributed relational database

The application for the distributed relational database in this example is parts stock management in an automobile dealer or distributor network.

This program checks the level of stock for each part in the local part stock table. If this is below the re-order point, the program then checks on the central tables to see whether there are any existing orders outstanding and what quantity has been shipped against each order.

If the net quantity (local stock, plus orders, minus shipments) is still below the re-order point, an order is placed for the part by inserting rows in the appropriate tables on the central server. A report is printed on the local server.

Technical Notes

Commitment control

This program uses the concept of local and remote logical units of work (LUW). Because this program uses remote unit of work, it is necessary to close the current LUW on one server (COMMIT) before beginning a new unit of work on another server.

Cursor repositioning

When an LUW is committed and the application connects to another database, all cursors are closed. This application requires the cursor reading the part stock file to be re-opened at the next part number. To achieve this, the cursor is defined to begin where the part number is greater than the current value of part number, and to be ordered by part number.

Note: This technique will not work if there are duplicate rows for the same part number.

| **Example: Program definitions**

| **Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 250.

|

```

| /*****
| /* Local Database */
| /*****/
| CREATE COLLECTION DRDA
|
| CREATE TABLE DRDA/PART_STOCK (PART_NUM CHAR(5) NOT NULL,
|                                PART_UM CHAR(2) NOT NULL,
|                                PART_QUANT INTEGER NOT NULL WITH DEFAULT,
|                                PART_ROP INTEGER NOT NULL,
|                                PART_EOQ INTEGER NOT NULL,
|                                PART_BIN CHAR(6) NOT NULL WITH DEFAULT
|                                )
|
| CREATE UNIQUE INDEX DRDA/PART_STOCI ON DRDA/PART_STOCK (PART_NUM ASC)
|
| /*****
| /* Remote Database */
| /*****/
| CREATE COLLECTION DRDA
|
| CREATE TABLE DRDA/PART_ORDER (ORDER_NUM SMALLINT NOT NULL,
|                                ORIGIN_LOC CHAR(4) NOT NULL,
|                                ORDER_TYPE CHAR(1) NOT NULL,
|                                ORDER_STAT CHAR(1) NOT NULL,
|                                NUM_ALLOC SMALLINT NOT NULL WITH DEFAULT,
|                                URG_REASON CHAR(1) NOT NULL WITH DEFAULT,
|                                CREAT_TIME TIMESTAMP NOT NULL,
|                                ALLOC_TIME TIMESTAMP,
|                                CLOSE_TIME TIMESTAMP,
|                                REV_REASON CHAR(1)
|                                )
|
| CREATE UNIQUE INDEX DRDA/PART_ORDEI ON DRDA/PART_ORDER (ORDER_NUM ASC)
|
| CREATE TABLE DRDA/PART_ORDLN (ORDER_NUM SMALLINT NOT NULL,
|                                ORDER_LINE SMALLINT NOT NULL,
|                                PART_NUM CHAR(5) NOT NULL,
|                                QUANT_REQ INTEGER NOT NULL,
|                                LINE_STAT CHAR(1) NOT NULL
|                                )
|
| CREATE UNIQUE INDEX PART_ORDLI ON DRDA/PART_ORDLN (ORDER_NUM ASC,
|                                                    ORDER_LINE ASC)
|
| CREATE TABLE DRDA/SHIPMENTLN (SHIP_NUM SMALLINT NOT NULL,
|                                SHIP_LINE SMALLINT NOT NULL,
|                                ORDER_LOC CHAR(4) NOT NULL,
|                                ORDER_NUM SMALLINT NOT NULL,
|                                ORDER_LINE SMALLINT NOT NULL,
|                                PART_NUM CHAR(5) NOT NULL,
|                                QUANT_SHIP INTEGER NOT NULL,
|                                QUANT_RECV INTEGER NOT NULL WITH DEFAULT
|                                )
|
| CREATE UNIQUE INDEX SHIPMENTLI ON DRDA/SHIPMENTLN (SHIP_NUM ASC,
|                                                    SHIP_LINE ASC)

```

Figure 22. Create a collection and tables

```

| /*****
| /* Local Database */
| /*****
| /*****
| /* PART_STOCK */
| /*****
| INSERT INTO DRDA/PART_STOCK VALUES('14020','EA',038,050,100,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('14030','EA',043,050,050,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('14040','EA',030,020,030,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('14050','EA',010,005,015,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('14060','EA',110,045,090,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('14070','EA',130,080,160,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('18020','EA',013,025,050,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('18030','EA',015,005,010,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('21010','EA',029,030,050,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('24010','EA',025,020,040,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('24080','EA',054,050,050,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('24090','EA',030,025,050,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('24100','EA',020,015,030,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('24110','EA',052,050,080,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('25010','EA',511,300,600,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('36010','EA',013,005,010,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('36020','EA',110,030,060,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('37010','EA',415,100,200,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('37020','EA',010,020,040,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('37030','EA',154,055,060,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('37040','EA',223,120,120,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('43010','EA',110,020,040,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('43020','EA',067,050,050,' ')
|
| INSERT INTO DRDA/PART_STOCK VALUES('48010','EA',032,030,060,' ')
|
| /*****

```

Figure 23. Insert data into the tables

```

| /* Remote Database */
| /*****
| /*****
| /* PART_ORDER TABLE */
| /*****
| INSERT INTO DRDA/PART_ORDER VALUES(1,'DB2B','U','0',0,' ','1991-03-12-17.00.00',
| NULL,NULL,NULL)
|
| INSERT INTO DRDA/PART_ORDER VALUES(2,'SQLA','U','0',0,' ','1991-03-12-17.01.00',
| NULL,NULL,NULL)
|
| INSERT INTO DRDA/PART_ORDER VALUES(3,'SQLA','U','0',0,' ','1991-03-12-17.02.00',
| NULL,NULL,NULL)
|
| INSERT INTO DRDA/PART_ORDER VALUES(4,'SQLA','U','0',0,' ','1991-03-12-17.03.00',
| NULL,NULL,NULL)
|
| INSERT INTO DRDA/PART_ORDER VALUES(5,'DB2B','U','0',0,' ','1991-03-12-17.04.00',
| NULL,NULL,NULL)
|
| /*****
| /* PART_ORDLN TABLE */
| /*****
| INSERT INTO DRDA/PART_ORDLN VALUES(1,1,'24110',005,'0')
|
| INSERT INTO DRDA/PART_ORDLN VALUES(1,2,'24100',021,'0')
|
| INSERT INTO DRDA/PART_ORDLN VALUES(1,3,'24090',018,'0')
|
| INSERT INTO DRDA/PART_ORDLN VALUES(2,1,'14070',004,'0')
|
| INSERT INTO DRDA/PART_ORDLN VALUES(2,2,'37040',043,'0')
|
| INSERT INTO DRDA/PART_ORDLN VALUES(2,3,'14030',015,'0')
|
| INSERT INTO DRDA/PART_ORDLN VALUES(3,2,'14030',025,'0')
|
| INSERT INTO DRDA/PART_ORDLN VALUES(3,1,'43010',003,'0')
|
| INSERT INTO DRDA/PART_ORDLN VALUES(4,1,'36010',013,'0')
|
| INSERT INTO DRDA/PART_ORDLN VALUES(5,1,'18030',005,'0')
|
| /*****
| /* SHIPMENTLN TABLE */
| /*****
| INSERT INTO DRDA/SHIPMENTLN VALUES(1,1,'DB2B',1,1,'24110',5,5)
|
| INSERT INTO DRDA/SHIPMENTLN VALUES(1,2,'DB2B',1,2,'24100',10,1)
|
| INSERT INTO DRDA/SHIPMENTLN VALUES(2,1,'SQLA',2,1,'14070',4,4)
|
| INSERT INTO DRDA/SHIPMENTLN VALUES(2,2,'SQLA',2,2,'37040',45,25)
|
| INSERT INTO DRDA/SHIPMENTLN VALUES(2,3,'SQLA',2,3,'14030', 5,5)
|
| INSERT INTO DRDA/SHIPMENTLN VALUES(3,1,'SQLA',2,3,'14030', 5,5)
|

```

Figure 24. Insert data into the tables (continued)

Example: RPG program

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 250.

```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:48          PAGE 1
SOURCE FILE . . . . . DRDA/QRPGSRC
MEMBER . . . . . DDBPT6RG
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
100 *****
200 *
300 *   DESCRIPTIVE NAME = D-DB SAMPLE APPLICATION          *
400 *                   REORDER POINT PROCESSING          *
500 *                   AS/400                             *
600 *
700 *   FUNCTION = THIS MODULE PROCESS THE PART_STOCK TABLE AND *
800 *                   FOR EACH PART BELOW THE ROP (REORDER POINT) *
900 *                   CREATES A SUPPLY ORDER AND PRINTS A REPORT. *
1000 *
1100 *
1200 *   INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION: *
1300 *
1400 *           LOCADB          LOCAL DB NAME          *
1500 *           REMODB          REMOTE DB NAME          *
1600 *
1700 *   TABLES = PART-STOCK          - LOCAL          *
1800 *           PART_ORDER          - REMOTE          *
1900 *           PART_ORDLN          - REMOTE          *
2000 *           SHIPMENTLN          - REMOTE          *
2100 *
2200 *   INDICATORS = *IN89          - '0' ORDER HEADER NOT DONE *
2300 *                   '1' ORDER HEADER IS DONE          *
2400 *           *IN99          - '1' ABNORMAL END (SQLCOD<0) *
2500 *
2600 *   TO BE COMPILED WITH COMMIT(*CHG) RDB(remotedbname) *
2700 *
2800 *   INVOKE BY : CALL DDBPT6RG PARM(localdbname remotedbname) *
2900 *
3000 *   CURSORS WILL BE CLOSED IMPLICITLY (BY CONNECT) BECAUSE *
3100 *   THERE IS NO REASON TO DO IT EXPLICITLY          *
3200 *
3300 *****
3400 *
3500 FQPRINT 0 F 33 OF PRINTER          03/29/92
3600 F*          03/29/92
3700 I*          03/29/92
3800 IMISC DS          03/29/92
3900 I B 1 20SHORTB          03/29/92
4000 I B 3 60LONGB          03/29/92
4100 I B 7 80INDNUL          03/29/92
4200 I 9 13 PRTTBL          03/29/92
4300 I 14 29 LOCTBL          03/29/92
4400 I I 'SQLA' 30 33 LOC          03/29/92
4500 I*          03/29/92
4600 I*          03/29/92
4700 C*          03/29/92
4800 C *LIKE DEFN SHORTB NXTORD NEW ORDER NR          03/29/92
4900 C *LIKE DEFN SHORTB NXTORL ORDER LINE NR          03/29/92
5000 C *LIKE DEFN SHORTB RTCOD1 RTCOD NEXT_PART          03/29/92
5100 C *LIKE DEFN SHORTB RTCOD2 RTCOD NEXT_ORD_          03/29/92
5200 C *LIKE DEFN SHORTB CURORD ORDER NUMBER          03/29/92
5300 C *LIKE DEFN SHORTB CURORL ORDER LINE          03/29/92
5400 C *LIKE DEFN LONGB QUANTI FOR COUNTING          03/29/92
5500 C *LIKE DEFN LONGB QTYSTC QTY ON STOCK          03/29/92
5600 C *LIKE DEFN LONGB QTYORD REORDER QTY          03/29/92

```

Figure 25. RPG program example


```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:48          PAGE    2
SOURCE FILE . . . . . DRDA/QRPGSRC
MEMBER . . . . . DDBPT6RG
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
5700    C          *LIKE      DEFN LONGB      QTYROP      REORDER POINT      03/29/92
5800    C          *LIKE      DEFN LONGB      QTYREQ      QTY ORDERED        03/29/92
5900    C          *LIKE      DEFN LONGB      QTYREC      QTY RECEIVED       03/29/92
6000    C*
6100    C*
6200    C*****
6300    C*      PARAMETERS      *
6400    C*****
6500    C*
6600    C          *ENTRY      PLIST
6700    C          PARM          LOCADB 18      LOCAL DATABASE      03/29/92
6800    C          PARM          REMODB 18      REMOTE DATABASE     03/29/92
6900    C*
7000    C*
7100    C*****
7200    C*      SQL CURSOR DECLARATIONS      *
7300    C*****
7400    C*
7500    C* NEXT PART WHICH STOCK QUANTITY IS UNDER REORDER POINTS QTY
7600    C/EXEC SQL
7700    C+      DECLARE NEXT_PART CURSOR FOR
7800    C+          SELECT PART_NUM,
7900    C+          PART_QUANT,
8000    C+          PART_ROP,
8100    C+          PART_EQQ
8200    C+          FROM PART_STOCK
8300    C+          WHERE PART_ROP > PART_QUANT
8400    C+          AND PART_NUM > :PRTTBL
8500    C+          ORDER BY PART_NUM ASC
8600    C/END-EXEC
8700    C*
8800    C* ORDERS WHICH ARE ALREADY MADE FOR CURRENT PART
8900    C/EXEC SQL
9000    C+      DECLARE NEXT_ORDER_LINE CURSOR FOR
9100    C+          SELECT A.ORDER_NUM,
9200    C+          ORDER_LINE,
9300    C+          QUANT_REQ
9400    C+          FROM PART_ORDLN A,
9500    C+          PART_ORDER B
9600    C+          WHERE PART_NUM = :PRTTBL
9700    C+          AND LINE_STAT <> 'C'
9800    C+          AND A.ORDER_NUM = B.ORDER_NUM
9900    C+          AND ORDER_TYPE = 'R'
10000   C/END-EXEC
10100   C*
10200   C*****
10300   C*      SQL RETURN CODE HANDLING      *
10400   C*****
10500   C/EXEC SQL
10600   C+      WHENEVER SQLERROR GO TO DBERRO
10700   C/END-EXEC
10800   C/EXEC SQL
10900   C+      WHENEVER SQLWARNING CONTINUE
11000   C/END-EXEC
11100   C*
11200   C*

```

```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:48          PAGE    3
SOURCE FILE . . . . . DRDA/QRPGRSRC
MEMBER . . . . . DDBPT6RG
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
11300 C*****
11400 C*   PROCESS - MAIN PROGRAM LOGIC                               *
11500 C*   MAIN PROCEDURE WORKS WITH LOCAL DATABASE                 *
11600 C*****
11700 C*
11800 C*CLEAN UP TO PERMIT RE-RUNNING OF TEST DATA
11900 C           EXSR CLEANU
12000 C*
12100 C*
12200 C           RTCOD1      DOUEQ100
12300 C*
12400 C/EXEC SQL
12500 C+           CONNECT TO :LOCADB
12600 C/END-EXEC
12700 C/EXEC SQL
12800 C+           OPEN      NEXT_PART
12900 C/END-EXEC
13000 C/EXEC SQL
13100 C+           FETCH     NEXT_PART
13200 C+           INTO      :PRTTBL,
13300 C+           :QTYSTC,
13400 C+           :QTYROP,
13500 C+           :QTYORD
13600 C/END-EXEC
13700 C           MOVE SQLCOD      RTCOD1
13800 C/EXEC SQL
13900 C+           COMMIT
14000 C/END-EXEC
14100 C           RTCOD1      IFNE 100
14200 C           EXSR CHECKO
14300 C           ENDIF
14400 C*
14500 C           ENDDO
14600 C*
14700 C           GOTO SETLR
14800 C*
14900 C*
15000 C*****
15100 C*   SQL RETURN CODE HANDLING ON ERROR SITUATIONS                 *
15200 C*****
15300 C*
15400 C           DBERRO      TAG
15500 C*           *-----*
15600 C           EXCPTERRLIN
15700 C           MOVE *ON      *IN99
15800 C/EXEC SQL
15900 C+           WHENEVER SQLERROR CONTINUE
16000 C/END-EXEC
16100 C/EXEC SQL
16200 C+           ROLLBACK
16300 C/END-EXEC
16400 C/EXEC SQL

```

```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:48          PAGE 4
SOURCE FILE . . . . . DRDA/QRPGSRC
MEMBER . . . . . DDBPT6RG
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
16500 C+          WHENEVER SQLERROR GO TO DBERRO          03/29/92
16600 C/END-EXEC
16700 C*
16800 C*
16900 C          SETLR      TAG          03/29/92
17000 C*          *-----*          03/29/92
17100 C/EXEC SQL
17200 C+          CONNECT    RESET          03/29/92
17300 C/END-EXEC
17400 C          MOVE *ON      *INLR          03/29/92
17500 C*
17600 C*****          03/29/92
17700 C*          THE END OF THE PROGRAM          *          03/29/92
17800 C*****          03/29/92
17900 C*
18000 C*
18100 C*****          03/29/92
18200 C* SUBROUTINES TO WORK WITH REMOTE DATABASES          *          03/29/92
18300 C*****          03/29/92
18400 C*
18500 C*
18600 C          CHECKO      BEGSR          03/29/92
18700 C*          *-----*          03/29/92
18800 C*****          03/29/92
18900 C* CHECKS WHAT IS CURRENT ORDER AND SHIPMENT STATUS FOR THE PART *          03/29/92
19000 C* IF ORDERED AND SHIPPED IS LESS THAN REORDER POINT OF PART, *          03/29/92
19100 C* PERFORMS A SUBROUTINE WHICH MAKES AN ORDER.          *          03/29/92
19200 C*****          03/29/92
19300 C*
19400 C          MOVE 0          RTCOD2          03/29/92
19500 C          MOVE 0          QTYREQ          03/29/92
19600 C          MOVE 0          QTYREC          03/29/92
19700 C*
19800 C/EXEC SQL          03/29/92
19900 C+          CONNECT    TO      :REMO DB          03/29/92
20000 C/END-EXEC          03/29/92
20100 C/EXEC SQL          03/29/92
20200 C+          OPEN      NEXT_ORDER_LINE          03/29/92
20300 C/END-EXEC          03/29/92
20400 C*
20500 C          RTCOD2      DOWNE100          03/29/92
20600 C*
20700 C/EXEC SQL          03/29/92
20800 C+          FETCH      NEXT_ORDER_LINE          03/29/92
20900 C+          INTO      :CURORD,          03/29/92
9 ...+... 0
21000 C+          :CURORL,          03/29/92
21100 C+          :QUANTI          03/29/92
21200 C/END-EXEC          03/29/92
21300 C*

```

```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:48          PAGE 5
SOURCE FILE . . . . . DRDA/QRPGSRC
MEMBER . . . . . DDBPT6RG
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
21400 C          SQLCOD  IFEQ 100                                03/29/92
21500 C          MOVE 100      RTCOD2                            03/29/92
21600 C          ELSE                                           03/29/92
21700 C          ADD QUANTI  QTYREQ                             03/29/92
21800 C*                                                03/29/92
21900 C/EXEC SQL                                           03/29/92
22000 C+          SELECT  SUM(QUANT_RECV)                        03/29/92
22100 C+          INTO    :QUANTI:INDNUL                       03/29/92
22200 C+          FROM    SHIPMENTLN                            03/29/92
22300 C+          WHERE   ORDER_LOC = :LOC                       03/29/92
22400 C+          AND     ORDER_NUM = :CURORD                    03/29/92
22500 C+          AND     ORDER_LINE = :CURORL                   03/29/92
22600 C/END-EXEC                                           03/29/92
22700 C*                                                03/29/92
22800 C          INDNUL  IFGE 0                                03/29/92
22900 C          ADD QUANTI  QTYREC                             03/29/92
23000 C          ENDIF                                       03/29/92
23100 C*                                                03/29/92
23200 C          ENDIF                                       03/29/92
23300 C          ENDDO                                        03/29/92
23400 C*                                                03/29/92
23500 C/EXEC SQL                                           03/29/92
23600 C+          CLOSE NEXT_ORDER_LINE                         03/29/92
23700 C/END-EXEC                                           03/29/92
23800 C*                                                03/29/92
23900 C          QTYSTC  ADD QTYREQ  QUANTI                    03/29/92
24000 C          SUB QUANTI  QTYREC                             03/29/92
24100 C*                                                03/29/92
24200 C          QTYROP  IFGT QUANTI                            03/29/92
24300 C          EXSR ORDERP                                    03/29/92
24400 C          ENDIF                                       03/29/92
24500 C*                                                03/29/92
24600 C/EXEC SQL                                           03/29/92
24700 C+          COMMIT                                       03/29/92
24800 C/END-EXEC                                           03/29/92
24900 C*                                                03/29/92
25000 C          ENDSR          CHECKO                           03/29/92
25100 C*                                                03/29/92
25200 C*                                                03/29/92
25300 C          ORDERP  BEGSR                                  03/29/92
25400 C*          *-----*                                    03/29/92
25500 C*****                                                03/29/92
25600 C* MAKES AN ORDER. IF FIRST TIME, PERFORMS THE SUBROUTINE, WHICH * 03/29/92
25700 C* SEARCHES FOR NEW ORDER NUMBER AND MAKES THE ORDER HEADER. * 03/29/92
25800 C* AFTER THAT MAKES ORDER LINES USING REORDER QUANTITY FOR THE * 03/29/92
25900 C* PART. FOR EVERY ORDERED PART WRITES A LINE ON REPORT. * 03/29/92
26000 C*****                                                03/29/92

```

```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:48          PAGE 7
SOURCE FILE . . . . . DRDA/QRPGSRC
MEMBER . . . . . DDBPT6RG
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
26100 C* 03/29/92
26200 C *IN89 IFEQ *OFF FIRST ORDER ? 03/29/92
26300 C EXSR STORD 03/29/92
26400 C MOVE *ON *IN89 ORD.HEAD.DONE 03/29/92
26500 C EXCPTHEADER WRITE HEADERS 03/29/92
26600 C ENDIF 03/29/92
26700 C* 03/29/92
26800 C ADD 1 NXTORL NEXT ORD.LIN 03/29/92
26900 C/EXEC SQL 03/29/92
27000 C+ INSERT 03/29/92
27100 C+ INTO PART_ORDLN 03/29/92
27200 C+ (ORDER_NUM, 03/29/92
27300 C+ ORDER_LINE, 03/29/92
27400 C+ PART_NUM, 03/29/92
27500 C+ QUANT_REQ, 03/29/92
27600 C+ LINE_STAT) 03/29/92
27700 C+ VALUES (:NXTORD, 03/29/92
27800 C+ :NXTORL, 03/29/92
27900 C+ :PRTTBL, 03/29/92
28000 C+ :QTYORD, 03/29/92
28100 C+ '0') 03/29/92
28200 C/END-EXEC 03/29/92
28300 C* 03/29/92
28400 C *INOF IFEQ *ON 03/29/92
28500 C EXCPTHEADER 03/29/92
28600 C END 03/29/92
28700 C EXCPTDETAIL 03/29/92
28800 C* 03/29/92
28900 C ENDSR ORDERP 03/29/92
29000 C* 03/29/92
29100 C* 03/29/92
29200 C STORD BEGSR 03/29/92
29300 C* *-----* 03/29/92
29400 C***** 03/29/92
29500 C* SEARCHES FOR NEXT ORDER NUMBER AND MAKES AN ORDER HEADER * 03/29/92
29600 C* USING THAT NUMBER. WRITES ALSO HEADERS ON REPORT. * 03/29/92
29700 C***** 03/29/92
29800 C* 03/29/92
29900 C/EXEC SQL 03/29/92
30000 C+ SELECT (MAX(ORDER_NUM) + 1) 03/29/92
30100 C+ INTO :NXTORD 03/29/92
30200 C+ FROM PART_ORDER 03/29/92
30300 C/END-EXEC 03/29/92
30400 C/EXEC SQL 03/29/92
30500 C+ INSERT 03/29/92
30600 C+ INTO PART_ORDER 03/29/92
30700 C+ (ORDER_NUM, 03/29/92
30800 C+ ORIGIN_LOC, 03/29/92

```

```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:48          PAGE    8
SOURCE FILE . . . . . DRDA/QRPGSRC
MEMBER . . . . . DDBPT6RG
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
30900    C+                ORDER_TYPE,                03/29/92
31000    C+                ORDER_STAT,                03/29/92
31100    C+                CREAT_TIME)                03/29/92
31200    C+          VALUES (:NXTORD,                03/29/92
31300    C+                :LOC,                03/29/92
31400    C+                'R',                03/29/92
31500    C+                'O',                03/29/92
31600    C+                CURRENT_TIMESTAMP)          03/29/92
31700    C/END-EXEC
31800    C                ENDSR                STRORD          03/29/92
31900    C*
32000    C*
32100    C          CLEANU  BEGSR
32200    C*          *-----*
32300    C*****
32400    C* THIS SUBROUTINE IS ONLY REQUIRED IN A TEST ENVIRONMENT 03/29/92
32500    C* TO RESET THE DATA TO PERMIT RE-RUNNING OF THE TEST 03/29/92
32600    C*****
32700    C*
32800    C/EXEC SQL
32900    C+          CONNECT TO :REMO DB          03/29/92
33000    C/END-EXEC
33100    C/EXEC SQL
33200    C+          DELETE
33300    C+                FROM PART_ORDLN
33400    C+                WHERE ORDER_NUM IN
33500    C+                (SELECT ORDER_NUM
33600    C+                FROM PART_ORDER
33700    C+                WHERE ORDER_TYPE = 'R')          03/29/92
33800    C/END-EXEC
33900    C/EXEC SQL
34000    C+          DELETE
34100    C+                FROM PART_ORDER
34200    C+                WHERE ORDER_TYPE = 'R'          03/29/92
34300    C/END-EXEC
34400    C/EXEC SQL
34500    C+          COMMIT
34600    C/END-EXEC
34700    C*
34800    C                ENDSR                CLEANU          03/29/92
34900    C*
35000    C*
35100    C*****
35200    O* OUTPUTLINES FOR THE REPORT *
35300    O*****
35400    O*
35500    OQPRINT E 2          HEADER
35600    O                + 0 '***** ROP PROCESSING'          03/29/92

```

SOURCE FILE DRDA/QRPGSRC

MEMBER DDBPT6RG

```

SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
35700      0                + 1 'REPORT *****'                                03/29/92
35800      0*
35900      QQPRINT  E 2          HEADER                                          03/29/92
36000      0                + 0 ' ORDER NUMBER = '                                03/29/92
36100      0                NXTORDZ + 0                                          03/29/92
36200      0*
36300      QQPRINT  E 1          HEADER                                          03/29/92
36400      0                + 0 '-----'                                          03/29/92
36500      0                + 0 '-----'                                          03/29/92
36600      0*
36700      QQPRINT  E 1          HEADER                                          03/29/92
36800      0                + 0 ' LINE '                                          03/29/92
36900      0                + 0 'PART '                                          03/29/92
37000      0                + 0 'QTY '                                           03/29/92
37100      0*
37200      QQPRINT  E 1          HEADER                                          03/29/92
37300      0                + 0 ' NUMBER '                                         03/29/92
37400      0                + 0 'NUMBER '                                         03/29/92
37500      0                + 0 'REQUESTED '                                       03/29/92
37600      0*
37700      QQPRINT  E 11         HEADER                                          03/29/92
37800      0                + 0 '-----'                                          03/29/92
37900      0                + 0 '-----'                                          03/29/92
38000      0*
38100      QQPRINT  EF1         DETAIL                                          03/29/92
38200      0                NXTORLZ + 4                                          03/29/92
38300      0                PRTTBL  + 4                                          03/29/92
38400      0                QTYORD1 + 4                                          03/29/92
38500      0*
38600      QQPRINT  T 2          LRN99                                          03/29/92
38700      0                + 0 '-----'                                          03/29/92
38800      0                + 0 '-----'                                          03/29/92
38900      QQPRINT  T 1          LRN99                                          03/29/92
39000      0                + 0 'NUMBER OF LINES '                                03/29/92
39100      0                + 0 'CREATED = '                                       03/29/92
39200      0                NXTORLZ + 0                                          03/29/92
39300      0*
39400      QQPRINT  T 1          LRN99                                          03/29/92
39500      0                + 0 '-----'                                          03/29/92
39600      0                + 0 '-----'                                          03/29/92
39700      0*
39800      QQPRINT  T 2          LRN99                                          03/29/92
39900      0                + 0 '*****'                                          03/29/92
40000      0                + 0 ' END OF PROGRAM '                                03/29/92
40100      0                + 0 '*****'                                          03/29/92
40200      0*
40300      QQPRINT  E 2          ERRLIN                                          03/29/92
40400      0                + 0 '*** ERROR **'                                       03/29/92
40500      0                + 0 '*** ERROR **'                                       03/29/92
40600      0                + 0 '*** ERROR **'                                       03/29/92
40700      QQPRINT  E 1          ERRLIN                                          03/29/92
40800      0                + 0 '* SQLCOD:'                                         03/29/92
40900      0                SQLCODM + 0                                          03/29/92
41000      0                33 '*'                                          03/29/92
41100      QQPRINT  E 1          ERRLIN                                          03/29/92
41200      0                + 0 '* SQLSTATE:'                                       03/29/92
41300      0                SQLSTT  + 2                                          03/29/92
41400      0                33 '*'                                          03/29/92
41500      QQPRINT  E 1          ERRLIN                                          03/29/92
41600      0                + 0 '*** ERROR **'                                       03/29/92
41700      0                + 0 '*** ERROR **'                                       03/29/92
41800      0                + 0 '*** ERROR **'                                       03/29/92

```

Example: COBOL program

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 250.


```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:35          PAGE    1
SOURCE FILE . . . . . DRDA/QLBLSRC
MEMBER . . . . . DDBPT6CB
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
100  IDENTIFICATION DIVISION.
200  *-----
300  PROGRAM-ID. DDBPT6CB.                                03/29/92
400  *****                                              03/29/92
500  *  MODULE NAME = DDBPT6CB                             03/29/92
600  *
700  *  DESCRIPTIVE NAME = D-DB SAMPLE APPLICATION
800  *                      REORDER POINT PROCESSING
900  *                      AS/400                          03/29/92
1000 *                      COBOL
1100 *
1200 *  FUNCTION = THIS MODULE PROCESS THE PART_STOCK TABLE AND
1300 *                      FOR EACH PART BELOW THE ROP (REORDER POINT)
1400 *                      CHECKS THE EXISTING ORDERS AND SHIPMENTS,      03/29/92
1500 *                      CREATES A SUPPLY ORDER AND PRINTS A REPORT.    03/29/92
1600 *
1700 *  DEPENDENCIES = NONE                                          03/29/92
1800 *
1900 *  INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:
2000 *
2100 *          LOCAL-DB      LOCAL DB NAME                    03/29/92
2200 *          REMOTE-DB     REMOTE DB NAME                   03/29/92
2300 *
2400 *  TABLES = PART-STOCK      - LOCAL                      03/29/92
2500 *          PART_ORDER        - REMOTE                     03/29/92
2600 *          PART_ORDLN        - REMOTE                     03/29/92
2700 *          SHIPMENTLN       - REMOTE                     03/29/92
2800 *
2900 *  CRTSQLCBL SPECIAL PARAMETERS                             03/29/92
3000 *  PGM(DDBPT6CB) RDB(remotedbname) OPTION(*APOST *APOSTSQL) 03/29/92
3100 *
3200 *  INVOKE BY : CALL DDBPT6CB PARM(localdbname remotedbname) 03/29/92
3300 *
3400 *****                                              03/29/92
3500 ENVIRONMENT DIVISION.
3600 *-----
3700 INPUT-OUTPUT SECTION.
3800 FILE-CONTROL.
3900     SELECT RELAT ASSIGN TO PRINTER-QPRINT.              03/29/92
4000 DATA DIVISION.
4100 *-----
4200 FILE SECTION.
4300 *-----                                              03/29/92
4400 FD RELAT
4500     RECORD CONTAINS 33 CHARACTERS
4600     LABEL RECORDS ARE OMITTED
4700     DATA RECORD IS REPREC.
4800     01 REPREC          PIC X(33).
4900 WORKING-STORAGE SECTION.
5000 *-----                                              03/29/92
5100 *  PRINT LINE DEFINITIONS                                    03/29/92
5200     01 LINE0          PIC X(33) VALUE SPACES.
5300     01 LINE1          PIC X(33) VALUE

```

Figure 26. COBOL program example

```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:35          PAGE    2
SOURCE FILE . . . . . DRDA/QLBLSRC
MEMBER . . . . . DDBPT6CB
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
5400          '***** ROP PROCESSING REPORT *****'.
5500          01 LINE2.
5600             05 FILLER          PIC X(18) VALUE ' ORDER NUMBER = '.
5700             05 MASK0          PIC ZZZ9.
5800             05 FILLER          PIC X(11) VALUE SPACES.
5900          01 LINE3          PIC X(33) VALUE
6000             '-----'.
6100          01 LINE4          PIC X(33) VALUE
6200             ' LINE PART QTY '.
6300          01 LINE5          PIC X(33) VALUE
6400             ' NUMBER NUMBER REQUESTED '.
6500          01 LINE6.
6600             05 FILLER          PIC XXXX VALUE SPACES.
6700             05 MASK1          PIC ZZZ9.
6800             05 FILLER          PIC XXXX VALUE SPACES.
6900             05 PART-TABLE      PIC XXXXX.
7000             05 FILLER          PIC XXXX VALUE SPACES.
7100             05 MASK2          PIC Z,ZZZ,ZZ.ZZ.
7200          01 LINE7.
7300             05 FILLER          PIC X(26) VALUE
7400             'NUMBER OF LINES CREATED = '.
7500             05 MASK3          PIC ZZZ9.
7600             05 FILLER          PIC XXX VALUE SPACES.
7700          01 LINE8          PIC X(33) VALUE
7800             '***** END OF PROGRAM *****'.
7900          * MISCELLANEOUS DEFINITIONS                                03/29/92
8000          01 WHAT-TIME      PIC X VALUE '1'.
8100             88 FIRST-TIME      VALUE '1'.
8200          01 CONTL          PIC S9999 COMP-4 VALUE ZEROS.                03/29/92
8300          01 CONTD          PIC S9999 COMP-4 VALUE ZEROS.                03/29/92
8400          01 RTCODE1        PIC S9999 COMP-4 VALUE ZEROS.                03/29/92
8500          01 RTCODE2        PIC S9999 COMP-4.                            03/29/92
8600          01 NEXT-NUM       PIC S9999 COMP-4.                            03/29/92
8700          01 IND-NULL       PIC S9999 COMP-4.                            03/29/92
8800          01 LOC-TABLE      PIC X(16).
8900          01 ORD-TABLE      PIC S9999 COMP-4.                            03/29/92
9000          01 ORL-TABLE      PIC S9999 COMP-4.                            03/29/92
9100          01 QUANT-TABLE     PIC S9(9) COMP-4.                            03/29/92
9200          01 QTY-TABLE      PIC S9(9) COMP-4.                            03/29/92
9300          01 ROP-TABLE      PIC S9(9) COMP-4.                            03/29/92
9400          01 EOQ-TABLE      PIC S9(9) COMP-4.                            03/29/92
9500          01 QTY-REQ        PIC S9(9) COMP-4.                            03/29/92
9600          01 QTY-REC        PIC S9(9) COMP-4.                            03/29/92
9700          * CONSTANT FOR LOCATION NUMBER                            03/29/92
9800          01 XPARM.                                                03/29/92
9900             05 LOC          PIC X(4) VALUE 'SQLA'.                    03/29/92
10000         * DEFINITIONS FOR ERROR MESSAGE HANDLING                    03/29/92
10100         01 ERROR-MESSAGE.                                        03/29/92
10200             05 MSG-ID.                                            03/29/92
10300             10 MSG-ID-1      PIC X(2)                                03/29/92
10400                 VALUE 'SQ'.                                        03/29/92
10500             10 MSG-ID-2      PIC 99999.                            03/29/92

```

```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:35          PAGE    3
SOURCE FILE . . . . . DRDA/QLBLSRC
MEMBER . . . . . DDBPT6CB
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
10600      *****
10700      *      SQLCA INCLUDE      *
10800      *****
10900      EXEC SQL INCLUDE SQLCA      END-EXEC.
11000
11100      LINKAGE SECTION.
11200      *-----*
11300      01 LOCAL-DB      PIC X(18).
11400      01 REMOTE-DB      PIC X(18).
11500
11600      PROCEDURE DIVISION USING LOCAL-DB REMOTE-DB.
11700      *-----*
11800      *****
11900      *      SQL CURSOR DECLARATION *
12000      *****
12100      * RE-POSITIONABLE CURSOR : POSITION AFTER LAST PART_NUM
12200      EXEC SQL DECLARE NEXT_PART CURSOR FOR
12300      SELECT PART_NUM,
12400      PART_QUANT,
12500      PART_ROP,
12600      PART_EOQ
12700      FROM PART_STOCK
12800      WHERE PART_ROP > PART_QUANT
12900      AND PART_NUM > :PART-TABLE
13000      ORDER BY PART_NUM ASC
13100      END-EXEC.
13200      * CURSOR FOR ORDER LINES
13300      EXEC SQL DECLARE NEXT_ORDER_LINE CURSOR FOR
13400      SELECT A.ORDER_NUM,
13500      ORDER_LINE,
13600      QUANT_REQ
13700      FROM PART_ORDLN A,
13800      PART_ORDER B
13900      WHERE PART_NUM = :PART-TABLE
14000      AND LINE_STAT <> 'C'
14100      AND A.ORDER_NUM = B.ORDER_NUM
14200      AND ORDER_TYPE = 'R'
14300      END-EXEC.
14400      *****
14500      *      SQL RETURN CODE HANDLING*
14600      *****
14700      EXEC SQL WHENEVER SQLERROR GO TO DB-ERROR END-EXEC.
14800      EXEC SQL WHENEVER SQLWARNING CONTINUE END-EXEC.
14900
15000      MAIN-PROGRAM-PROC.
15100      *-----*
15200      PERFORM START-UP THRU START-UP-EXIT.
15300      PERFORM MAIN-PROC THRU MAIN-EXIT UNTIL RTCODE1 = 100.
15400      END-OF-PROGRAM.

```

```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:35          PAGE    4
SOURCE FILE . . . . . DRDA/QLBLSRC
MEMBER . . . . . DDBPT6CB
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
15500      *-----
15600      ****
15700      EXEC SQL CONNECT RESET END-EXEC.
15800      ****
15900      CLOSE RELAT.
16000      GOBACK.
16100      MAIN-PROGRAM-EXIT. EXIT.
16200      *-----
16300
16400      START-UP.
16500      *-----
16600      OPEN OUTPUT RELAT.
16700      ****
16800      EXEC SQL COMMIT END-EXEC.
16900      ****
17000      PERFORM CLEAN-UP THRU CLEAN-UP-EXIT.
17100      *****
17200      * CONNECT TO LOCAL DATABASE *
17300      *****
17400      ****
17500      EXEC SQL CONNECT TO :LOCAL-DB END-EXEC.
17600      ****
17700      START-UP-EXIT. EXIT.
17800      *-----
17900      EJECT
18000      MAIN-PROC.
18100      *-----
18200      EXEC SQL OPEN NEXT_PART END-EXEC.
18300      EXEC SQL
18400          FETCH NEXT_PART
18500          INTO :PART-TABLE,
18600              :QUANT-TABLE,
18700              :ROP-TABLE,
18800              :EQQ-TABLE
18900      END-EXEC.
19000      IF SQLCODE = 100
19100          MOVE 100 TO RTCODE1
19200          PERFORM TRAILER-PROC THRU TRAILER-EXIT
19300      ELSE
19400          MOVE 0 TO RTCODE2
19500          MOVE 0 TO QTY-REQ
19600          MOVE 0 TO QTY-REC
19700      * --- IMPLICIT "CLOSE" CAUSED BY COMMIT ---
19800      ****
19900      EXEC SQL COMMIT END-EXEC
20000      ****
20100      *****
20200      * CONNECT TO REMOTE DATABASE *
20300      *****

```

```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:35          PAGE    5
SOURCE FILE . . . . . DRDA/QLBLSRC
MEMBER . . . . . DDBPT6CB
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
20400      ****                                03/29/92
20500      EXEC SQL CONNECT TO :REMOTE-DB END-EXEC                                03/29/92
20600      ****                                03/29/92
20700      EXEC SQL OPEN NEXT_ORDER_LINE END-EXEC                                03/29/92
20800      PERFORM UNTIL RTCODE2 = 100
20900          EXEC SQL                                03/29/92
21000              FETCH NEXT_ORDER_LINE
21100              INTO :ORD-TABLE,
21200                  :ORL-TABLE,
21300                  :QTY-TABLE
21400          END-EXEC
21500          IF SQLCODE = 100
21600              MOVE 100 TO RTCODE2
21700              EXEC SQL CLOSE NEXT_ORDER_LINE END-EXEC
21800          ELSE
21900              ADD QTY-TABLE TO QTY-REQ
22000              EXEC SQL
22100                  SELECT SUM(QUANT_RECV)                                03/29/92
22200                  INTO :QTY-TABLE:IND-NULL
22300                  FROM SHIPMENTLN                                03/29/92
22400                  WHERE ORDER_LOC = :LOC
22500                  AND ORDER_NUM = :ORD-TABLE
22600                  AND ORDER_LINE = :ORL-TABLE
22700              END-EXEC
22800              IF IND-NULL NOT < 0
22900                  ADD QTY-TABLE TO QTY-REC
23000              END-IF
23100          END-IF
23200          END-PERFORM
23300          IF ROP-TABLE > QUANT-TABLE + QTY-REQ - QTY-REC
23400              PERFORM ORDER-PROC THRU ORDER-EXIT
23500          END-IF
23600      END-IF.
23700      ****                                03/29/92
23800      EXEC SQL COMMIT END-EXEC.                                03/29/92
23900      ****                                03/29/92
24000      *****
24100      * RECONNECT TO LOCAL DATABASE *                                03/29/92
24200      *****                                03/29/92
24300      ****                                03/29/92
24400      EXEC SQL CONNECT TO :LOCAL-DB END-EXEC.                                03/29/92
24500      ****                                03/29/92
24600      MAIN-EXIT. EXIT.
24700      *-----
24800      ORDER-PROC.
24900      *-----
25000          IF FIRST-TIME
25100              MOVE '2' TO WHAT-TIME
25200              PERFORM CREATE-ORDER-PROC THRU CREATE-ORDER-EXIT.            03/29/92
25300          ADD 1 TO CNTL.

```

```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:35          PAGE 7
SOURCE FILE . . . . . DRDA/QLBLSRC
MEMBER . . . . . DDBPT6CB
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
25400          EXEC SQL
25500              INSERT
25600                  INTO      PART_ORDLN                      03/29/92
25700                      (ORDER_NUM,
25800                      ORDER_LINE,
25900                      PART_NUM,
26000                      QUANT_REQ,
26100                      LINE_STAT)
26200              VALUES (:NEXT-NUM,
26300                      :CONTL,
26400                      :PART-TABLE,
26500                      :EQQ-TABLE,
26600                      '0')
26700          END-EXEC.
26800          PERFORM DETAIL-PROC THRU DETAIL-EXIT.
26900          ORDER-EXIT. EXIT.
27000          *-----
27100
27200          CREATE-ORDER-PROC.
27300          *-----
27400          *GET NEXT ORDER NUMBER
27500          EXEC SQL
27600              SELECT (MAX(ORDER_NUM) + 1)
27700              INTO  :NEXT-NUM:IND-NULL
27800              FROM  PART_ORDER
27900          END-EXEC.
28000          IF IND-NULL < 0
28100              MOVE 1 TO NEXT-NUM.
28200          EXEC SQL
28300              INSERT
28400              INTO      PART_ORDER
28500                      (ORDER_NUM,
28600                      ORIGIN_LOC,
28700                      ORDER_TYPE,
28800                      ORDER_STAT,
28900                      CREAT_TIME)
29000              VALUES (:NEXT-NUM,
29100                      :LOC, 'R', '0',
29200                      CURRENT_TIMESTAMP)
29300          END-EXEC.
29400          MOVE NEXT-NUM TO MASK0.
29500          PERFORM HEADER-PROC THRU HEADER-EXIT.
29600          CREATE-ORDER-EXIT. EXIT.
29700          *-----
29800
29900          DB-ERROR.
30000          *-----
30100          PERFORM ERROR-MSG-PROC THRU ERROR-MSG-EXIT.
30200          *****
30300          *      ROLLBACK THE LUW *

```

```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:35          PAGE      8
SOURCE FILE . . . . . DRDA/QLBLSRC
MEMBER . . . . . DDBPT6CB
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
30400      *****
30500      EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
30600      ****
30700      EXEC SQL ROLLBACK WORK END-EXEC.
30800      ****
30900      PERFORM END-OF-PROGRAM THRU MAIN-PROGRAM-EXIT.
31000      * -- NEXT LINE INCLUDED TO RESET THE "GO TO" DEFAULT --
31100      EXEC SQL WHENEVER SQLERROR GO TO DB-ERROR END-EXEC.
31200
31300      ERROR-MSG-PROC.
31400      *-----
31500      MOVE SQLCODE TO MSG-ID-2.
31600      DISPLAY 'SQL STATE =' SQLSTATE ' SQLCODE =' MSG-ID-2.
31700      * -- ADD HERE ANY ADDITIONAL ERROR MESSAGE HANDLING --
31800      ERROR-MSG-EXIT. EXIT.
31900      *-----
32000
32100      *****
32200      * REPORT PRINTING *
32300      *****
32400      HEADER-PROC.
32500      *-----
32600      WRITE REPREC FROM LINE1 AFTER ADVANCING PAGE.
32700      WRITE REPREC FROM LINE2 AFTER ADVANCING 3 LINES.
32800      WRITE REPREC FROM LINE3 AFTER ADVANCING 2 LINES.
32900      WRITE REPREC FROM LINE4 AFTER ADVANCING 1 LINES.
33000      WRITE REPREC FROM LINE5 AFTER ADVANCING 1 LINES.
33100      WRITE REPREC FROM LINE3 AFTER ADVANCING 1 LINES.
33200      WRITE REPREC FROM LINE0 AFTER ADVANCING 1 LINES.
33300      HEADER-EXIT. EXIT.
33400      *-----
33500      DETAIL-PROC.
33600      *-----
33700      ADD 1 TO CONTD.
33800      IF CONTD > 50
33900      MOVE 1 TO CONTD
34000      PERFORM HEADER-PROC THRU HEADER-EXIT
34100      END-IF
34200      MOVE CONTL TO MASK1.
34300      MOVE EOQ-TABLE TO MASK2.
34400      WRITE REPREC FROM LINE6 AFTER ADVANCING 1 LINES.
34500      DETAIL-EXIT. EXIT.
34600      *-----
34700      TRAILER-PROC.
34800      *-----
34900      MOVE CONTL TO MASK3.
35000      WRITE REPREC FROM LINE3 AFTER ADVANCING 2 LINES.
35100      WRITE REPREC FROM LINE7 AFTER ADVANCING 2 LINES.
35200      WRITE REPREC FROM LINE3 AFTER ADVANCING 2 LINES.
35300      WRITE REPREC FROM LINE8 AFTER ADVANCING 1 LINES.
35400      TRAILER-EXIT. EXIT.
35500      *-----

```

```

5738PW1 V2R1M1 920327          SEU SOURCE LISTING          03/29/92 17:12:35          PAGE    8
SOURCE FILE . . . . . DRDA/QLBLSRC
MEMBER . . . . . DDBPT6CB
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
35600      *****
35700      * THIS PARAGRAPH IS ONLY REQUIRED IN A TEST ENVIRONMENT*
35800      * TO RESET THE DATA TO PERMIT RE-RUNNING OF THE TEST *
35900      *****
36000      CLEAN-UP.
36100      *-----
36200      *****
36300      * CONNECT TO REMOTE DATABASE *
36400      *****
36500      ****
36600      EXEC SQL CONNECT TO :REMOTE-DB END-EXEC.
36700      ****
36800      *-----DELETED ORDER ROWS FOR RERUNABILITY
36900      EXEC SQL
37000      DELETE
37100      FROM PART_ORDLN
37200      WHERE ORDER_NUM IN
37300      (SELECT ORDER_NUM
37400      FROM PART_ORDER
37500      WHERE ORDER_TYPE = 'R')
37600      END-EXEC.
37700      EXEC SQL
37800      DELETE
37900      FROM PART_ORDER
38000      WHERE ORDER_TYPE = 'R'
38100      END-EXEC.
38200      ****
38300      EXEC SQL COMMIT END-EXEC.
38400      ****
38500      CLEAN-UP-EXIT. EXIT.
38600      *-----
*****
* * * * E N D O F S O U R C E * * * *

```

Example: C program using embedded SQL

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 250.


```

| /*****
| /* PROGRAM NAME: SAMPEMBC */
| /* */
| /* DESCRIPTIVE NAME: Sample embedded C application using DRDA */
| /* */
| /* FUNCTION: This module processes the PART_STOCK table and */
| /* for each part below the ROP (REORDER POINT) */
| /* creates a supply order. */
| /* */
| /* LOCAL TABLES: PART_STOCK */
| /* */
| /* REMOTE TABLES: PART_ORDER, PART_ORDLN, SHIPMENTLN */
| /* */
| /* COMPILE OPTIONS: */
| /* CRTSQLCI OBJ(SAMPEMBC) COMMIT(*CHG) RDB(rdbname) OBJTYPE(*PGM) */
| /* RDBCNNMTH(*RUW) */
| /* */
| /* INVOKED BY: */
| /* CALL PGM(SAMPEMBC) PARM('lcldbname' 'rmtdbname') */
| *****/
| #include <stdlib.h>
| #include <string.h>
| #include <stdio.h>
|
| EXEC SQL INCLUDE SQLCA;
|
| /*****
| /* SQL Variables */
| *****/
| EXEC SQL BEGIN DECLARE SECTION;
| char loc[4] = "SQLA"; /* dealer's database name */
| char remote_db[18] = " "; /* sample remote database */
| char local_db[18] = " "; /* sample local database */
| char part_table[5] = " "; /* part number in table part_stock */
| long quant_table; /* quantity in stock, tbl part_stock */
| long rop_table; /* reorder point , tbl part_stock */
| long eqq_table; /* reorder quantity , tbl part_stock */
| short next_num; /* next order nbr, table part_order */
| short ord_table; /* order nbr. , tbl order_line */
| short orl_table; /* order line , tbl order_line */
| long qty_table; /* ordered quantity , tbl order_line */
| long line_count = 0; /* total number of order lines */
| short ind_null; /* null indicator for qty_table */
| short contl = 0; /* continuation line, tbl order_line */
| EXEC SQL END DECLARE SECTION;
|
| /*****
| /* Other Variables */
| *****/
| char first_time, what_time;
| long qty_rec = 0, qty_req = 0;
|
| EXEC SQL WHENEVER SQLERROR GOTO error_tag;
| EXEC SQL WHENEVER SQLWARNING CONTINUE;

```

Figure 27. C program example using embedded SQL

```

| /*****
| /* Function Declarations
| /*****
| /*****
| /* Function For Declaring Cursors
| /*****
| declare_cursors() {
|
|     /* SQL Cursor declaration and reposition for local UW
|     EXEC SQL DECLARE NEXT_PART CURSOR FOR
|         SELECT PART_NUM, PART_QUANT, PART_ROP, PART_EOQ
|         FROM DRDA/PART_STOCK
|         WHERE PART_ROP > PART_QUANT AND
|             PART_NUM > :part_table
|         ORDER BY PART_NUM;
|
|     /* SQL Cursor declaration and connect for RUW
|     EXEC SQL DECLARE NEXT_OLINE CURSOR FOR
|         SELECT A.ORDER_NUM, ORDER_LINE, QUANT_REQ
|         FROM DRDA/PART_ORDLN A,
|             DRDA/PART_ORDER B
|         WHERE PART_NUM = :part_table AND
|             LINE_STAT <> 'C' AND
|             A.ORDER_NUM = B.ORDER_NUM AND
|             ORDER_TYPE = 'R';
|
|     /* upline exit function in connectable state
|     goto function_exit;
|
| error_tag:
|     error_function();
|
| function_exit:
|     ;
| } /* function declare_cursor
|
| /*****
| /* Function For Reseting Tables
| /*****
| reset_tables() {
|
|     /* Clean up for rerunability in test environment
|     EXEC SQL CONNECT TO :remote_db;
|     EXEC SQL DELETE FROM DRDA/PART_ORDLN
|         WHERE ORDER_NUM IN
|             (SELECT ORDER_NUM
|              FROM DRDA/PART_ORDER
|              WHERE ORDER_TYPE = 'R');
|     EXEC SQL DELETE FROM DRDA/PART_ORDER
|         WHERE ORDER_TYPE = 'R';
|     /* Exit function in connectable state
|     EXEC SQL COMMIT;
|     goto function_exit;
|
| error_tag:
|     error_function();
|
| function_exit:
|     ;
| } /* function delete_for_rerun
|

```

```

| /*****
| /* Function For Declaring Cursors */
| /*****
| calculate_order_quantity() {
|
|     /* Set current connection to local database */
|     EXEC SQL CONNECT TO :local_db;
|     /* available qty = Stock qty + qty in order - qty received */
|     EXEC SQL OPEN NEXT_PART;
|     EXEC SQL FETCH NEXT_PART
|         INTO :part_table, :quant_table, :rop_table, :eqq_table;
|
|     if (sqlca.sqlcode == 100) {
|         printf("-----\n");
|         printf("NUMBER OF LINES CREATED = %d\n",line_count);
|         printf("-----\n");
|         printf("***** END OF PROGRAM *****\n");
|         rop_table = 0;          /* no (more) orders to process */
|     }
|     else {
|         qty_rec = 0;
|         qty_req = 0;
|         EXEC SQL COMMIT;
|         EXEC SQL CONNECT TO :remote_db;
|         EXEC SQL OPEN NEXT_OLINE;
|         do {
|             EXEC SQL FETCH NEXT_OLINE
|                 INTO :ord_table, :orl_table, :qty_table;
|             qty_rec = qty_rec + qty_table;
|         } while(sqlca.sqlcode != 100);
|         EXEC SQL CLOSE NEXT_OLINE;
|         EXEC SQL SELECT SUM(QUANT_RECV)
|             INTO :qty_table:ind_null
|             FROM DRDA/SHIPMENTLN
|             WHERE ORDER_LOC = :loc AND
|                   ORDER_NUM = :ord_table AND
|                   ORDER_LINE = :orl_table;
|         if (ind_null != 0)
|             qty_rec = qty_rec + qty_table;
|     } /* end of else branch */
|
|     goto function_exit;
|
| error_tag:
|     error_function();
|
| function_exit:
|     ;
| } /* end of calculate_order_quantity */
|

```

```

| /*****
| /* Function For Declaring Cursors */
| /*****
| process_order() {
|
| /* insert order and order_line in remote database */
| if (contl == 0) {
|     EXEC SQL SELECT (MAX(ORDER_NUM) + 1)
|         INTO :next_num
|         FROM DRDA/PART_ORDER;
|     EXEC SQL INSERT INTO DRDA/PART_ORDER
|         (ORDER_NUM, ORIGIN_LOC, ORDER_TYPE, ORDER_STAT, CREAT_TIME)
|         VALUES (:next_num, :loc, 'R', '0', CURRENT_TIMESTAMP);
|     printf("***** ROP PROCESSING *****\n");
|     printf("ORDER NUMBER = %d \n\n",next_num);
|     printf("-----\n");
|     printf("  LINE      PART      QTY      \n");
|     printf("  NBR       NBR       REQUESTED\n");
|     printf("-----\n");
|     contl = contl + 1;
| } /* if contl == 0 */
|
| EXEC SQL INSERT INTO DRDA/PART_ORDLN
|     (ORDER_NUM, ORDER_LINE, PART_NUM, QUANT_REQ, LINE_STAT)
|     VALUES (:next_num, :contl, :part_table, :eqq_table, '0');
| line_count = line_count + 1;
| printf("  %d      %.5s      %d\n",line_count,part_table,eqq_table);
| contl = contl + 1;
| /* Exit function in connectable state */
| EXEC SQL COMMIT;
|
| goto function_exit;
|
| error_tag:
|     error_function();
|
| function_exit:
|     ;
| } /* end of function process_order */
|

```

```

| /*****
| /* Function For Declaring Cursors
| /*****
| error_function() {
|
|     printf("*****\n");
|     printf("*      SQL ERROR      *\n");
|     printf("*****\n");
|     printf("SQLCODE   = %d\n",sqlca.sqlcode);
|     printf("SQLSTATE   = %5s",sqlca.sqlstate);
|     printf("\n*****\n");
|     EXEC SQL WHENEVER SQLERROR CONTINUE;
|     EXEC SQL ROLLBACK;
|     /* Reset Current Connection To Local Database
|     EXEC SQL CONNECT RESET;
|
|     exit(999);
| } /* end of function error_function
|
| /*****
| /* Mainline
| /*****
| main(int argc, char *argv[]) {
|
|     memcpy(local_db,argv[1],strlen(argv[1]));
|     memcpy(remote_db,argv[2],strlen(argv[2]));
|
|     /* Initialization
|     declare_cursors();
|     reset_tables();
|
|     /* Main Work
|     do {
|         calculate_order_quantity();
|         if (rop_table > quant_table + qty_req - qty_rec) {
|             process_order();
|             quant_table = qty_req = qty_rec = 0;
|         }
|     } while (sqlca.sqlcode == 0);
|     EXEC SQL COMMIT;
|     /* Reset Current Connection To Local Database
|     EXEC SQL DISCONNECT :local_db;
|
|     exit(0);
| } /* end of main

```

Example: Java program

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 250.

```

| /*****
| /* PROGRAM NAME: SampJava */
| /* */
| /* DESCRIPTIVE NAME: Sample java application using DRDA */
| /* */
| /* FUNCTION: This module processes the PART_STOCK table and */
| /* for each part below the ROP (REORDER POINT) */
| /* creates a supply order. */
| /* */
| /* LOCAL TABLES: PART_STOCK */
| /* */
| /* REMOTE TABLES: PART_ORDER, PART_ORDLN, SHIPMENTLN */
| /* */
| /* COMPILE OPTIONS: */
| /* javac SampJava.java */
| /* */
| /* INVOKED BY: */
| /* java SampJava lcldbname rmtdbname */
| /*****
| import java.sql.*;
|
| public class SampJava {
|     private static String JDBCdriver = "com.ibm.db2.jcc.DB2Driver";
|     private static String part_table = " "; /* part number in table part_stock */
|     private static long line_count = 0; /* total number of order lines */
|     private static long eqq_table = 0; /* reorder quantity , tbl part_stock */
|     private static long quant_table = 0; /* quantity in stock, tbl part_stock */
|     private static long rop_table = 0; /* reorder point , tbl part_stock */
|     private static int contl = 0; /* continuation line, tbl order_line */
|     private static short next_num = 0; /* next order nbr,table part_order */
|
|     /*****
|     /* Method For Reseting Environment */
|     /*****
|     private static void resetTables(Connection rmtConn) throws SQLException {
|
|         Statement stmt1 = rmtConn.createStatement();
|
|         /* Clean up for rerunability in test environment */
|         stmt1.executeUpdate("DELETE FROM DRDA.PART_ORDLN WHERE ORDER_NUM IN " +
|             " (SELECT ORDER_NUM FROM DRDA.PART_ORDER " +
|             " WHERE ORDER_TYPE = 'R')");
|         stmt1.executeUpdate("DELETE FROM DRDA.PART_ORDER WHERE ORDER_TYPE = 'R');
|         stmt1.close();
|         rmtConn.commit();
|
|     } /* function delete_for_rerun */
|
|

```

```

| /*****
| /* Method For Calculating Order Quantity */
| /*****
| private static void calculateOrderQuantity(Connection lclConn, Connection rmtConn, String loc)
| throws SQLException {
|     PreparedStatement prpStmt1;
|     PreparedStatement prpStmt2;
|     ResultSet rsltSet1;
|     ResultSet rsltSet2;
|     short ord_table = 0;          /* order nbr.          , tbl order_line */
|     short orl_table = 0;         /* order line       , tbl order_line */
|
|     prpStmt1 = lclConn.prepareStatement("SELECT PART_NUM, PART_QUANT, PART_ROP, PART_EQQ " +
|                                       "          FROM DRDA.PART_STOCK WHERE PART_ROP > PART_QUANT AND " +
|                                       "          PART_NUM > ? ORDER BY PART_NUM");
|
|     prpStmt1.setString(1,part_table);
|     rsltSet1 = prpStmt1.executeQuery();
|     if (rsltSet1.next() == false) {
|         System.out.println("-----");
|         System.out.println("NUMBER OF LINES CREATED = " + line_count);
|         System.out.println("-----");
|         System.out.println("***** END OF PROGRAM *****");
|         rop_table = 0;          /* no (more) orders to process */
|     }
|     else {
|         /* available qty = Stock qty + qty in order - qty received */
|         part_table = rsltSet1.getString(1);
|         quant_table = rsltSet1.getLong(2);
|         rop_table = rsltSet1.getLong(3);
|         eqq_table = rsltSet1.getLong(4);
|         long qty_rec = 0;
|
|         prpStmt2 = rmtConn.prepareStatement("SELECT A.ORDER_NUM, ORDER_LINE, QUANT_REQ " +
|                                           "          FROM DRDA.PART_ORDLN A, DRDA.PART_ORDER B " +
|                                           "          WHERE PART_NUM = ? AND LINE_STAT <> 'C' AND " +
|                                           "          A.ORDER_NUM = B.ORDER_NUM AND ORDER_TYPE = 'R'");
|
|         prpStmt2.setString(1,part_table);
|         rsltSet2 = prpStmt2.executeQuery();
|         while (rsltSet2.next()) {
|             ord_table = rsltSet2.getShort(1);
|             orl_table = rsltSet2.getShort(2);
|             long qty_table = rsltSet2.getLong(3);
|             qty_rec = qty_rec + qty_table;
|         }
|         rsltSet2.close();
|
|
|
|

```

```

| prpStmt2 = rmtConn.prepareStatement("SELECT SUM(QUANT_RECV) FROM DRDA.SHIPMENTLN " +
|                                     "      WHERE ORDER_LOC = ? AND ORDER_NUM = ? AND " +
|                                     "      ORDER_LINE = ?");
|
|     prpStmt2.setString(1,loc);
|     prpStmt2.setShort(2,ord_table);
|     prpStmt2.setShort(3,orl_table);
|     rsltSet2 = prpStmt2.executeQuery();
|     rsltSet2.next();
|     long qty_table = rsltSet2.getLong(1);
|     qty_rec = qty_rec + qty_table;
|     rsltSet2.close();
|     prpStmt2.close();
| }
| rsltSet1.close();
| prpStmt1.close();
|
| } /* end of calculate_order_quantity                                     */
|
| /*****
| /* Method For Processing Orders                                     */
| /*****
| private static void processOrder(Connection rmtConn, String loc) throws SQLException {
|     PreparedStatement prpStmt1;
|     ResultSet rsltSet1;
|
|     /* insert order and order_line in remote database                                     */
|     if (cont1 == 0) {
|         prpStmt1 = rmtConn.prepareStatement("SELECT (MAX(ORDER_NUM) + 1) FROM DRDA.PART_ORDER");
|         rsltSet1 = prpStmt1.executeQuery();
|         rsltSet1.next();
|         next_num = rsltSet1.getShort(1);
|         rsltSet1.close();
|         prpStmt1 = rmtConn.prepareStatement("INSERT INTO DRDA.PART_ORDER (ORDER_NUM, ORIGIN_LOC,
| ORDER_TYPE, ORDER_STAT, CREAT_TIME) " +
|                                     "      VALUES (?, ?, 'R', 'O', CURRENT_TIMESTAMP)");
|
|         prpStmt1.setShort(1,next_num);
|         prpStmt1.setString(2,loc);
|         prpStmt1.executeUpdate();
|         System.out.println("***** ROP PROCESSING *****");
|         System.out.println("ORDER NUMBER = " + next_num);
|         System.out.println("-----");
|         System.out.println("  LINE      PART      QTY      ");
|         System.out.println("  NBR      NBR      REQUESTED");
|         System.out.println("-----");
|         cont1 = cont1 + 1;
|     } /* if cont1 == 0                                               */
|
|
|
|

```



```

| prpStmt1 = rmtConn.prepareStatement("INSERT INTO DRDA.PART_ORDLN (ORDER_NUM, ORDER_LINE,
| PART_NUM, QUANT_REQ, LINE_STAT) " +
|                                     "          VALUES (?, ?, ?, ?, '0')");
|
| prpStmt1.setShort(1,next_num);
| prpStmt1.setInt(2,cont1);
| prpStmt1.setString(3,part_table);
| prpStmt1.setLong(4,eqq_table);
| prpStmt1.executeUpdate();
| line_count = line_count + 1;
| System.out.println("  " + line_count + "          " + part_table + "          " + eqq_table + "");
| cont1 = cont1 + 1;
| prpStmt1.close();
|
| } /* end of function processOrder                                     */
|
|
| /*****
| /* Method For Displaying Errors                                     */
| /*****
| private static void errorFunction(SQLException e, Connection lclConn, Connection rmtConn) {
|
| System.out.println("*****");
| System.out.println("*      SQL ERROR      *");
| System.out.println("*****");
| System.out.println("SQLCODE    = " + e.getErrorCode());
| System.out.println("SQLSTATE   = " + e.getSQLState());
| System.out.println("*****");
| try {
|     lclConn.rollback();
|     rmtConn.rollback();
| }
| catch (SQLException uowErr) {
| }
|
| } /* end of function errorFunction                                 */
|
|

```

```

| /*****
|  /* Mainline
|  *****/
| public static void main(String[] args) {
|     String User = "myuser";
|     String Password = "mypwd";
|     String lclUrl = null;
|     String rmtUrl = null;
|     String loc = "SQLA";          /* dealer's database name          */
|     Connection lclConn = null;
|     Connection rmtConn = null;
|
|     try {
|         Class.forName(JDBCdriver).newInstance();
|     }
|     catch (Exception e) {
|         System.out.println("Error: Failed to load DB2 driver.");
|         System.exit(1);
|     }
|
|     try {
|         lclUrl = "jdbc:db2:" + args[0];
|         lclConn = DriverManager.getConnection(lclUrl, User, Password);
|         rmtUrl = "jdbc:db2:" + args[1];
|         rmtConn = DriverManager.getConnection(rmtUrl, User, Password);
|     }
|     catch (Exception e) {
|         System.out.println("Error: Failed to get database connections.");
|         System.exit(1);
|     }
|
|     try {
|         /* Initialization
|         resetTables(rmtConn);
|
|         /* Main Work
|         do {
|             calculateOrderQuantity(lclConn, rmtConn, loc);
|             if (rop_table > quant_table) {
|                 processOrder(rmtConn, loc);
|                 quant_table = 0;
|             }
|         } while (rop_table != 0);
|
|         /* End Work
|         lclConn.commit();
|         rmtConn.commit();
|     }
|     catch (SQLException e) {
|         e.printStackTrace();
|         errorFunction(e, lclConn, rmtConn);
|         System.exit(1);
|     }
| }
|
|

```

Example: Program output

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 250.

```
***** ROP PROCESSING *****
ORDER NUMBER = 6
-----
LINE   PART   QTY
NBR   NBR   REQUESTED
-----
1      14020   100
2      14030   50
3      18020   50
4      21010   50
5      37020   40
-----
NUMBER OF LINES CREATED = 5
-----
***** END OF PROGRAM *****
```

Figure 28. Example: Program output

User FAQs

This topic collection concentrates on describing iSeries support for distributed relational databases in a network of iSeries servers (a *like* environment).

Many distributed relational database implementations exist in a network of different DRDA-supporting platforms. This topic collection provides a list of tips and techniques you might need to consider when using the iSeries server in an *unlike* DRDA environment.

This topic collection describes some conditions you need to consider when working with another specific IBM product. It is not intended to be a comprehensive list. Many problems or conditions like the ones described here depend significantly on your application. You can get more information about the differences between the various IBM platforms from the *IBM SQL Reference Volume 2*, SC26-8416, or the *DRDA Application Programming Guide*, SC26-4773.

Related reference

“iSeries server support” on page 87

The default CCSID for a job on the iSeries server is specified using the Change Job (CHGJOB) command.

Connect to a distributed relational database

When you connect from a non-iSeries application requester (AR) to a DB2 Universal Database for iSeries application server (AS), columns tagged with CCSID 65535 are not converted. If the files that contain these columns do not contain any columns that have a CCSID explicitly identified, the CCSID of all character columns can be changed to another CCSID value. To change the CCSID, use the Change Physical File (CHGPF) command. If you have logical files built over the physical file, follow the directions given in the recovery topic of the error message (CPD322D) that you get.

iSeries system value QCCSID

The iSeries server is shipped with a QCCSID system value set to 65535.

Data tagged with this CCSID is not to be converted by the receiving server. You might not be able to connect to an unlike server when your iSeries server application requester (AR) is using this CCSID. Also, you might not be able to use source files that are tagged with this CCSID to create applications on unlike servers.

The CCSID used at connection time is determined by the job CCSID. When a job begins, its CCSID is determined by the user profile the job is running under. The user profile can, and as a default does, use the system value QCCSID.

If you are connecting to a server that does not support the server default CCSID, you need to change your job CCSID. You can change the job CCSID by using the Change Job (CHGJOB) command. However, this solution is only for the job you are currently working with. The next time you will have to change the job CCSID again.

A more permanent solution is to change the CCSID designated by the user profiles used in the distributed relational database. When you change the user profiles, you affect only those users that need to have their data converted. If you are working with a DB2 Universal Database for iSeries application server (AS), you need to change the user profile that the AS uses.

If an unlike application requester connects to a DB2 UDB for iSeries application server using job CCSID 65535, the job will be switched to use the job default CCSID. The job default CCSID is determined by the job's language identifier (LANGID). For better performance, the job's CCSID should be switched to a value other than 65535 in this case. For example, the CCSID value can be changed to the value of the user profile under which the server jobs are run.

The default CCSID value in a user profile is *SYSVAL. This references the QCCSID system value. You can change the QCCSID system value that is used by all user profiles with the Change System Value (CHGSYSVAL) command. If you do this, you would want to select a CCSID that represents most (if not all) of the users on your server.

If you suspect that you are working with a server that does not support a CCSID used by your job or your server, look for the following indicators in a job log, SQLCA or SQL diagnostic area:

Message

SQ30073

SQLCODE or DB2_RETURNED_SQLCODE

-30073

SQLSTATE

58017

Text Distributed Data Management (DDM) parameter X'0035' not supported.

Message

SQL0332

SQLCODE or DB2_RETURNED_SQLCODE

-332

SQLSTATE

57017

Text Total conversion between CCSID &1 and CCSID &2 not valid.

Related concepts

i5/OS globalization

Related reference

Change Job (CHGJOB) command

Change System Value (CHGSYSVAL) command

CCSID conversion considerations for DB2 Universal Database for z/OS and DB2 Universal Database server for VM database managers

One of the differences between a DB2 Universal Database for iSeries and other DB2 databases is that the iSeries system supports a larger set of CCSIDs. This can lead to errors when the other database managers attempt to perform character conversion on the data (SQLCODE -332 and SQLSTATE 57017).

Certain fields in the DB2 UDB SQL catalog tables might be defined to have a DBCS-open data type. This is a data type that allows both double-byte character set (DBCS) and single-byte character set (SBCS) characters. The CCSID for these field types is based on the default CCSID shipped with the server.

When these fields are selected from a DB2 Universal Database for z/OS or DB2 Universal Database Server for VM application requester (AR), the SELECT statement might fail because the DB2 Universal Database for z/OS and DB2 Universal Database Server for VM databases might not support the conversion to this CCSID.

To avoid this error, you must change the DB2 Universal Database for z/OS database or the DB2 Universal Database Server for VM AR to run with either one of the following items:

- The same mixed-byte CCSID as the DBCS-OPEN fields in the iSeries SQL catalog tables.
- A CCSID that the server allows conversion of data to when the data is from the mixed-byte CCSID of the DBCS-OPEN fields in the iSeries SQL catalog tables. This CCSID might be a single-byte CCSID if the data in the iSeries SQL catalog tables DBCS-OPEN fields is all single-byte data.

This requires some analysis of the CCSID conversions supported on the DB2 Universal Database for z/OS or DB2 Universal Database Server for VM so you can make the correct changes to your server. See the *DB2 UDB for z/OS Administration Guide* for specific information on how to handle this error.

Why am I getting a SQL5048N message when I attempt to connect from DB2 Universal Database for Linux, UNIX and Windows?

The definition of message SQL5048N states that the release level of the database client is not supported by the release level of the database server. However, the message can sometimes be misleading.

There are several common causes for this problem:

1. You will see this error message if you have only the Client Application Enabler installed. In this case, the client system must be connected to the iSeries server through a gateway server. Direct connection is not supported.
2. You can also get this error if someone has made errors while manually configuring the connection.

Use the Client Configuration Assistant (CCA) to avoid getting SQL5048N.

Another potential cause for the problem concerns the collection NULLID. DB2 Universal Database for Linux, UNIX and Windows, IBM DB2 Universal Driver for SQLJ and JDBC, and other application requesters use the collection NULLID for building their needed SQL packages. The collection and packages are created upon first connection. If the user profile does not have sufficient authority to create the collection, another profile with higher authority should initially connect to get these objects created.

If you believe there is another cause for the error, see the Authorized Problem Analysis Report Web site. Enter **APAR II12722** in the **Search** field.

Do iSeries files have to be journaled?

Journaling is not required if the client application is using an isolation level of no-commit (NC) or uncommitted read (UR), and if the DB2 UDB SQL function determines that the query data can be blocked. In that case commitment control is not enabled, which makes journaling unnecessary.

The answer to this question is closely related to the question in “When will query data be blocked for better performance?”

Examples of methods of changing isolation levels are:

- The DB2 Universal Database for Linux, UNIX and Windows precompiler uses the ISOLATION UR parameter to specify uncommitted read.
- The DB2 Universal Database for Linux, UNIX and Windows command line processor (CLP) uses the command DBM CHANGE SQLISL TO UR to specify uncommitted read.
- The DB2 Universal Database for Linux, UNIX and Windows command line processor (CLP) uses the command DBM CHANGE SQLISL TO NC to specify no-commit.
- JDBC clients set their connection property isolation level to TRANSACTION_READ_UNCOMMITTED to specify uncommitted read.

When will query data be blocked for better performance?

The query data will be blocked if none of these conditions are true.

- The cursor is updatable (see note 1).
- The cursor is potentially updatable (see note 2).
- The BLOCKING NO precompiler or bind option was used on SQLPREP or SQLBIND.

Unless you force single-row protocol with the BLOCKING NO precompile/bind option, blocking will occur in both of the following cases:

- The cursor is read-only (see note 3).
- All of the following are true:
 - There is no FOR UPDATE OF clause in the SELECT, and
 - There are no UPDATE or DELETE WHERE CURRENT OF statements against the cursor in the program, and
 - Either the program does not contain dynamic SQL statements or BLOCKING ALL was used.

Notes:

1. A cursor is updatable if it is not read-only (see note 3), and one of the following items is true:
 - The select statement contained the FOR UPDATE OF clause, or
 - There exists in the program an UPDATE or DELETE WHERE CURRENT OF against the cursor.
2. A cursor is potentially updatable if it is not read-only (see note 3), and if the program includes any dynamic statement, and the BLOCKING UNAMBIG precompile or bind option was used on SQLPREP or SQLBIND.
3. A cursor is read-only if one or more of the following conditions is true:
 - The DECLARE CURSOR statement specified an ORDER BY clause but did not specify a FOR UPDATE OF clause.
 - The DECLARE CURSOR statement specified a FOR FETCH ONLY clause.
 - One or more of the following conditions are true for the cursor or a view or logical file referenced in the outer subselect to which the cursor refers:
 - The outer subselect contains a DISTINCT keyword, GROUP BY clause, HAVING clause, or a column function in the outer subselect.
 - The select contains a join function.
 - The select contains a UNION operator.
 - The select contains a subquery that refers to the same table as the table of the outer-most subselect.
 - The select contains a complex logical file that had to be copied to a temporary file.

- All of the selected columns are expressions, scalar functions, or constants.
- All of the columns of a referenced logical file are input only.

How do you interpret an SQLCODE and the associated tokens reported in a DBM SQL0969N error message?

The client support used with DB2 Universal Database for Linux, UNIX and Windows returns message SQL0969N when reporting host SQLCODEs and tokens for which it has no equivalent code.

Here is an example of message SQL0969N:

```
SQL0969N There is no message text corresponding to SQL error
"-7008" in the Database Manager message file on this workstation.
The error was returned from module "QSQOPEN" with original
tokens "TABLE1  PRODLIB1  3".
```

Use the Display Message Description (DSPMSGD) command to interpret the code and tokens:
 DSPMSGD SQL7008 MSGF(QSQLMSG)

Select option 1 (Display message text) and the server presents the Display Formatted Message Text display. The three tokens in the message are represented by &1, &2, and &3 in the display. The reason code in the example message is 3, which points to Code 3 in the list at the bottom of the display.

```

                Display Formatted Message Text
System:  RCHASLAI
Message ID . . . . . :  SQL7008
Message file . . . . . :  QSQLMSG
Library . . . . . :  QSYS

Message . . . . . :  &1 in &2 not valid for operation.
Cause . . . . . :  The reason code is &3. A list of reason codes follows:
-- Code 1 indicates that the table has no members.
-- Code 2 indicates that the table has been saved with storage free.
-- Code 3 indicates that the table is not journaled, the table is
journaled to a different journal than other tables being processed under
commitment control, or that you do not have authority to the journal.
-- Code 4 indicates that the table is in a production library but the user
is in debug mode with UPDPROD(*NO); therefore, production tables may not be
updated.
-- Code 5 indicates that a table, view, or index is being created into a
production library but the user is in debug mode with UPDPROD(*NO);
therefore, tables, views, or indexes may not be created.
More...
Press Enter to Continue.

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

Related reference

Display Message Description (DSPMSGD) command

How can the host variable type in WHERE clauses affect performance?

One potential source of performance degradation on an iSeries server is the client's use in a C program of a floating point variable for a comparison in the WHERE clause of a SELECT statement.

If the operating system has to do a conversion of the data for that column, that will prevent it from being able to use an index on that column. You should always try to use the same type for columns, literals, and host variables used in a comparison. If the column in the database is defined as packed or zoned decimal, and the host variable is of some other type, that can present a problem in C.

Related concepts

Programming techniques for database performance

Can I use a library list for resolving unqualified table and view names?

The iSeries server supports a limited capability to use the operating system naming option when accessing DB2 Universal Database for iSeries data from a non-iSeries DRDA client program such as those that use the DB2 Universal Database for Linux, UNIX and Windows product.

Previously, only the SQL naming option has been available when connecting from unlike DRDA clients. System naming changes several characteristics of DB2 UDB for iSeries. For example:

1. The library list is searched for resolving unqualified table and view names.
2. When running a CREATE SQL statement, an unqualified object will be created in the current library.
3. A slash (/) instead of a period (.) is used to separate qualified objects' names from the library or collection in which they reside.
4. Certain authorization characteristics are changed.

For details, read about server naming in the SQL reference. For more information about the implications regarding authorization, see Planning and design.

DB2 Universal Database for Linux, UNIX and Windows supports the specification of generic bind options on two of its program preparation commands, the precompile (PREP) command and the (BIND) command. i5/OS naming can be specified on either of them as in the following examples drawn from a Windows batch file:

For DB2 Universal Database for Linux, UNIX and Windows, Version 8 and later:

```
DB2 PREP %1.SQC BINDFILE OS400NAMING SYSTEM ...
DB2 BIND %1.BND OS400NAMING SYSTEM ...
```

For DB2 Universal Database for Linux, UNIX and Windows, version earlier than Version 8:

```
DB2 PREP %1.SQC BINDFILE GENERIC 'OS400NAMING SYSTEM' ...
DB2 BIND %1.BND GENERIC 'OS400NAMING SYSTEM' ...
```

Note that on the Windows development platform, single quotation marks are used around the generic option name/value pair. On an AIX or UNIX platform, quotation marks should be used.

Note: For OS/400 V4R5 and V5R1, the name of the option is AS400NAMING, not OS400NAMING.

The only valid value for the OS400NAMING option besides SYSTEM is SQL, which is the default value, and the only possible option from a non-iSeries client prior to the introduction of this feature.

If you use the OS400NAMING option on the (BIND) command but not on the (PREP) command, then you might need to code a parameter on the (PREP) command that indicates a bind file should be created in spite of SQL errors detected by the server platform. In the case of DB2 Universal Database for Linux, UNIX and Windows, use the SQLError CONTINUE parameter for this purpose. The capability is described as "limited" because in certain situations, the client-side software might parse an SQL statement intended for execution on the remote server. If a slash instead of a period is used to separate a schema ID from a table ID, as is required for server naming, the statement might be rejected as having improper syntax.

Can a user of DB2 Universal Database for Linux, UNIX and Windows specify that the NLSS sort sequence table of the DRDA job on the iSeries server be used instead of the usual EBCDIC sequence?

The iSeries server recognizes a generic bind option that allows one who prepares a program to be run from DB2 Universal Database for Linux, UNIX and Windows or any other client that supports generic bind options to request that the iSeries server use the NLSS sort sequence associated with the server job in which the client's request is run.

This function is enabled by PTF SI00174 in V5R1. It is in the base operating system for subsequent releases.

If you choose to take advantage of this enhancement, you need to recreate any SQL packages on DB2 Universal Database for which the new sort sequence option is desired by using the generic bind option SORTSEQ with a value of JOBRUN from the client system.

The bind option enables a user to specify that the NLSS sort sequence table of the DRDA job on the iSeries server should be used instead of the usual EBCDIC sequence. Previously, only the default *HEX option, which causes the EBCDIC sequence to be used, has been available when connecting from unlike DRDA clients.

This feature is available from DRDA application requesters that support the DRDA generic bind function. It has undergone limited testing using DB2 Universal Database for Linux, UNIX and Windows 6.1 FixPak 1 running on Windows as a client development platform and run environment. DB2 Universal Database for Linux, UNIX and Windows supports the specification of generic bind options on two of its program preparation commands, the precompile (PREP) command and the (BIND) command. JOBRUN sort sequence can be specified on either of them as in the following examples drawn from a Windows batch file:

```
DB2 PREP %1.SQC BINDFILE SORTSEQ JOBRUN...
DB2 BIND %1.BND SORTSEQ JOBRUN...
```

Note: On the Windows development platform, single quotation marks are used around the generic option name/value pair, but on an AIX or UNIX platform, quotation marks should be used.

The only other valid value for the SORTSEQ option is HEX, which is the default value, and only possible option from a non-iSeries client before the introduction of this feature.


Why are no rows returned when I perform a query?

One potential cause of this problem is a failure to add an entry for the iSeries server in the DB2 Universal Database for Linux, UNIX and Windows Database Communication Services Directory.

What level of DB2 Universal Database for Linux, UNIX, Windows is required to interact with DB2 Universal Database for iSeries?

These Fix Packs are required for interaction.

- DB2 Universal Database for Linux, UNIX and Windows Version 7 Fix Pack 10
- DB2 Universal Database for Linux, UNIX and Windows Version 8 Fix Pack 4

You can get these Fix Packs from the DB2 Universal Database for Linux, UNIX and Windows  Web site.

How can I get scrollable cursor support enabled from DB2 Universal Database for Linux, Unix and Windows Version 8 to the iSeries?

You must be using FixPak 4 or later on the client.

If you are using FixPak 4, you must do *one* of the following actions:

- Issue this command:

```
UPDATE CLI CFG FOR SECTION iSeries dbname USING CURSORTYPES 1
```

Where *iSeries dbname* is the name of your iSeries database.

- Edit the db2cli.ini file using this syntax:

```
CURTYPES = 1
```

Other tips for interoperating in unlike environments

This topic collection provides additional information for using DB2 Universal Database for iSeries with DB2 Universal Database for Linux, UNIX and Windows. These tips were developed from experiences testing with the products on an OS/2 platform, but it is believed that they apply to all environments to which they have been ported.

DB2 Connect™ as opposed to DB2 Universal Database for Linux, UNIX and Windows

Users are sometimes confused over what products are needed to perform the DRDA application server function as opposed to the application requester (client) function. The AR is sometimes referred to as DB2 Connect; and both the AR and AS as DB2 Universal Database for Linux, UNIX and Windows. DB2 UDB refers to the following products:

- DB2 Universal Database for AIX
- DB2 Universal Database for HP-UX
- DB2 Universal Database for Linux
- DB2 Universal Database for Sun Solaris
- DB2 Universal Database for Windows

Proper configuration and maintenance level

Be sure to follow the installation and configuration instructions given in the product manuals carefully. Make sure that you have the most current level of the products. Apply the appropriate fix packs if not.

Table and collection naming

SQL tables accessed by DRDA applications have three-part names: the first part is the database name, the second part is a collection ID, and the third part is the base table name. The first two parts are optional. DB2 UDB for iSeries qualifies table names at the second level by a collection (or library) name. Tables reside in the DB2 UDB for iSeries database.

Before V5R2 and the advent of independent auxiliary storage pools, there was only one database for each iSeries server. However, in DB2 UDB, tables are qualified by a user ID (that of the creator of the table), and reside in one of possibly multiple databases on the platform. DB2 Universal Database for Linux, UNIX and Windows has the same notion of using the user ID for the collection ID.

A dynamic query from DB2 Universal Database for Linux, UNIX and Windows to DB2 UDB for iSeries will use the user ID of the target side job (on the iSeries server) for the default collection name, if the name of the queried table was specified without a collection name. This might not be what is expected by the user and can cause the table to be not found.

A dynamic query from DB2 UDB for iSeries to DB2 UDB would have an implied table qualifier if it is not specified in the query in the form *qualifier.table-name*. The second-level UDB table qualifier defaults to the user ID of the user making the query.

You might want to create the DB2 UDB databases and tables with a common user ID. Remember, for UDB there are no physical collections as there are in DB2 UDB for iSeries; there is only a table qualifier, which is the user ID of the creator.

APPC communications setup

i5/OS communications must be configured properly, with a controller and device created for the workstation when you use APPC with either DB2 Universal Database for Linux, UNIX and Windows as an AR, or with DB2 UDB as an AS.

Set up the RDB directory

When adding an entry in the RDB directory for each DB2 UDB database that an iSeries server will connect to, use the Add Relational Database Directory Entry (ADDRDBDIRE) command. The RDB name is the UDB database name.

When using APPC communications, the remote location name is the name of the workstation.

When using TCP/IP, the remote location name is the domain name of the workstation, or its IP address. The port used by the UDB DRDA server is typically not 446, the well-known DRDA port that the iSeries server uses (*DDM).

Consult the UDB product documentation to determine the port number. A common value used is 50000. An example DSPRDBDIRE screen showing a properly configured RDB entry for a UDB server follows.

```
Display Relational Database Detail
  Relational database . . . . . : SAMPLE
  Remote location:
    Remote location . . . . . : 9.5.36.17
    Type . . . . . : *IP
    Port number or service name . . . : 50000
  Text . . . . . : My UDB server
```

How do I create the NULLID packages used by DB2 UDB for Linux, UNIX and Windows and IBM DB2 Universal Driver for SQLJ and JDBC?

Before using DB2 Universal Database for Linux, UNIX and Windows to access data on DB2 UDB for iSeries, you must create SQL packages on the iSeries server for application programs and for the DB2 Universal Database for Linux, UNIX and Windows utilities.

The DB2 (PREP) command can be used to process an application program source file with embedded SQL. This processing will create a modified source file containing host language calls for the SQL statements and it will, by default, create an SQL package in the database you are currently connected to.

To bind DB2 Universal Database for Linux, UNIX and Windows to a DB2 UDB for iSeries server:

1. CONNECT TO rdbname
2. Bind path@ddcs400.lst BLOCKING ALL SQLERROR CONTINUE MESSAGES DDCS400.MGS GRANT PUBLIC

Replace 'path' in the path@ddcs400.lst parameter above with the default path C:\SQLLIB\BND\
(c:/sqllib/bin/ on non-INTEL platforms), or with your value if you did not install to the default directory.

3. CONNECT RESET

How do I set up the Interactive SQL packages?

To use Interactive SQL, you need the DB2 UDB Query Manager and SQL Development Kit product installed on i5/OS. To access data on DB2 Universal Database:

1. When starting a session with STRSQL, use session attributes of NAMING(*SQL), DATFMT(*ISO), and TIMFMT(*ISO). Other formats besides *ISO work, but not all, and what is used for the date format (DATFMT) must also be used for the time format (TIMFMT).
2. Note the correspondence between COLLECTIONS on the iSeries server, and table qualifier (the creator's user ID) for UDB.
3. For the first interactive session, you must do this sequence of SQL statements to get a package created on UDB: (1) RELEASE ALL, (2) COMMIT, and (3) CONNECT TO rdbname (where 'rdbname' is replaced with a particular database).

As part of your setup for the use of Interactive SQL, you might also want to use the statement GRANT EXECUTE ON PACKAGE QSQL400.QSQLabcd TO PUBLIC (or to specific users), so that others can use the SQL PKG created on the PC for Interactive SQL. The actual value for abcd in the following GRANT statement can be determined from the following table, which gives the package names for various sets of options that are in effect when the package is created. For example, you would use the statement GRANT EXECUTE ON PACKAGE QSQL400.QSQL0200 TO *some-user* if the following options were in use when you created the package: *ISO for date, *ISO for time, *CS for commitment control, a single quotation mark for string delimiter, and single byte for character subtype.

Position	Option	Value
a	Date Format	0 = ISO, JIS date format 1 = USA date format 2 = EUR date format
b	Time Format	0 = JIS time format 1 = USA time format 2 = EUR, ISO time format
c	Commitment Control Decimal Delimiter	0 = *CS commitment control period decimal delimiter 1 = *CS commitment control comma decimal delimiter 2 = *RR commitment control period decimal delimiter 3 = *RR commitment control comma decimal delimiter
d	String Delimiter Default Character Subtype	0 = single quotation mark string delimiter, single byte character subtype 1 = single quotation mark string delimiter, double byte character subtype 2 = quotation marks string delimiter, single byte character subtype 3 = quotation marks string delimiter, double byte character subtype

Close of queries

DB2 Universal Database for Linux, UNIX and Windows now provides an option to request that read locks be released when queries are closed either implicitly or explicitly. It is not considered an error if the server does not honor the request, which is the case for iSeries servers. DB2 Universal Database for Linux, UNIX and Windows provides another new option to specify whether the server should close the query implicitly for a non-scrollable cursor when there are no more rows to read. Previously the server made this decision. The iSeries AS supports this new feature in V5R3.

What is the maximum length of user IDs and passwords in a heterogeneous environment?

DB2 UDB for iSeries running as the application requester (AR) allows user IDs and passwords longer than ten characters when running to an unlike application server (AS). The exact limits are specified in the description of the specific interface being used. For example, see the SQL reference topic for limits on the SQL CONNECT statement.

Create Interactive SQL packages on DB2 UDB Server for VM

On DB2 Universal Database Server for VM, a collection name is synonymous with a user ID. To create packages to be used with Interactive SQL or iSeries Query Manager on an DB2 Universal Database Server for VM application server, create a user ID of QSQL400 on the i5/OS operating system. This user ID can be used to create all the necessary packages on the DB2 Universal Database Server for VM application server. Users can then use their own user IDs to access DB2 Universal Database Server for VM through Interactive SQL or iSeries Query Manager on i5/OS.

Interpret trace job and FFDC data

This topic collection provides additional problem-analysis information. It is useful to specialists responsible for problem determination. It is also for suppliers of software products designed to conform to the Distributed Relational Database Architecture who want to test connectivity to an iSeries server.

Related reference

Trace Job (TRCJOB) command

Example: Analyze the RW trace data

This discussion examines the elements that make up the data stream in the example.

The example in Figure 29 on page 229 shows the data stream received during a distributed SQL query function. This particular trace was run at the application requester (AR) end of the connection. Therefore, the associated program module that produced the data is QRWSQRY.

For more information about the interpretation of Distributed Relational Database Architecture (DRDA) data streams, see the *Distributed Relational Database Architecture Reference* and the *Distributed Data Management Level 4.0 Architecture Reference* books. These documents are available on the Web at www.opengroup.org/dbiop/index.htm.

The trace data follows the colon (:) marking the end of the trace point identifier. In this example, the first six bytes of the data stream contain the DDM data stream structure (DSS) header. The first two bytes of this DSS header are a length field. The third byte, X'D0' is the registered SNA architecture identifier for all DDM data. The fourth byte is the format identifier (explained in more detail later). The fifth and sixth bytes contain the DDM request correlation identifier.

The next two bytes, X'0010' (decimal 16) give the length of the next DDM object, which in this case is identified by the X'2205' which follows it and is the code point for the OPNQRYRM reply message.

Following the 16-byte reply message is a six-byte DSS header for the reply objects that follow the reply message. The first reply object is identified by the X'241A' code point. It is a QRYDSC object. The second reply object in the example is a QRYDTA structure identified by the X'241B' code point (split between two lines in the trace output). As with the OPNQRYRM code point, the preceding two bytes give the length of the object.

Looking more closely at the QRYDTA object, you can see a X'FF' following the X'241B' code point. This represents a null SQLCAGRP (the form of an SQLCA or SQL diagnostic area that flows on the wire). The null form of the SQLCAGRP indicates that it contains no error or warning information about the associated data. In this case, the associated data is the row of data from an SQL SELECT operation. It follows the null SQLCAGRP. Because rows of data as well as SQLCAGRPs are nullable, however, the first byte that follows the null SQLCAGRP is an indicator containing X'00' that indicates that the row of data is not null. The meaning of the null indicator byte is determined by the first bit. A 1 in this position indicates null. However, all 8 bits are usually set on when an indicator represents a null object.

The format of the row of data is indicated by the preceding QRYDSC object. In this case, the QRYDSC indicates that the row contains a nullable SMALLINT value, a nullable CHAR(3) value, and a non-nullable double precision floating point value. The second byte past the null SQLCAGRP is the null indicator associated with the SMALLINT field. It indicates the field is not null, and the X'0001' following it is the field data. The nullable CHAR(3) that follows is present and contains 111. The floating point value that follows next does not have a X'00' byte following it, since it is defined to be not nullable.

A second row of data with a null SQLCAGRP follows the first, which in turn is followed by another six-byte DSS header. The second half of the format byte (X'2') contained in that header indicates that the corresponding DSS is a REPLY. The format byte of the previous DSS (X'53') indicated that it was an OBJECT DSS. The ENDQRYRM reply message carried by the third DSS requires that it be contained in a REPLY DSS. The ENDQRYRM code point is X'220B'. This reply message contains a severity code of X'0004', and the name of the RDB that returned the query data ('DB2ESYS').

Following the third DSS in this example is a fourth and final one. The format byte of it is X'03'. The 3 indicates that it is an OBJECT DSS, and the 0 that precedes it indicates that it is the last DSS of the chain (the chaining bits are turned off).

The object in this DSS is an SQLCARD containing a non-null SQLCAGRP. The first byte following the X'2408' SQLCARD code point is the indicator telling us that the SQLCAGRP is not null. The next four bytes, X'00000064', represents the +100 SQLCODE which means that the query was ended by the encounter of a 'row not found' condition. The rest of the fields correspond to other fields in an SQLCA. The rest of the fields correspond to other fields in an SQLCA or SQL diagnostic area. The mapping of SQLCAGRP fields to SQLCA and SQL diagnostic area fields can be found in the *Distributed Relational Database Architecture Reference* book. This document is available at the Open Group Web site



(www.opengroup.org/dbiop/index.htm)

Description of RW trace points

This topic collection discusses the RW trace points.

RWff RC—Receive Data Stream Trace Point:

This data stream contains a DDM response from an application server (AS) program. The DSS headers are present in this data stream.

This is the trace point that is shown in Figure 29 on page 229.

The IDs of the Distributed Relational Database Architecture (DRDA) function that is being performed (ff) are provided here.

ff	DRDA Function
AC	Access RDB.
OQ	Open query.
CQ	Continue query.
EQ	Close query.
PS	Prepare SQL statement.
XS	Execute SQL statement.
XI	Execute SQL statement immediately.
DT	Describe Table statement.
DS	Describe Statement statement.
SY	TCP/IP SYnc point operation.

RWff SN—Send Data Stream Trace Point:

This data stream contains either a distributed data management (DDM) request from an application requester (AR) program, or a DDM response from an application server (AS) program, as that request or response exists before it is given to the lower level CN component for addition of headers and transmission across the wire.

Besides content, the main difference between the trace information for receiving data streams and sending data streams is that for the latter, the six-byte DSS header information is missing. For the first DSS in a send data stream trace area, the header is omitted entirely, and for subsequent ones, six bytes of zeros are present which will be overlaid by the header when it is constructed later by a CN component module.

The IDs of the Distributed Relational Database Architecture (DRDA) function that is being performed are the same as those listed for RWff RC—Receive Data Stream Trace Point.

Related reference

“RWff RC—Receive Data Stream Trace Point” on page 231

This data stream contains a DDM response from an application server (AS) program. The DSS headers are present in this data stream.

RWQY S1—Partial Send Data Stream Trace Point 1:

This trace point occurs in the NEWBLOCK routine of the QRWTQRY module when a new query block is needed in the building of QRYDTA in the like environment.

In the like environment a query block need not be filled up before it is transmitted, and it is always put on the wire at this point so that the buffer space can be reused. DSS headers are absent as in other send data streams.

RWQY S2—Partial Send Data Stream Trace Point 2:

This trace point occurs in the NEWBLOCK routine of the QRWTQRY module when a new query block is needed in the building of QRYDTA in the unlike environment.

In the unlike environment all query blocks except the last one must be filled up before construction of a new one can be started, and they are not transmitted until all are built.

RWQY BP—Successful Fetch Trace Point:

This trace point occurs in the FETCH routine of the QRWTQRY module when a call to the SQFCHCRS macro results in a non-null pointer to a BPCA structure, implying that one or more records were returned in the BPCA buffer.

The data dumped is the BPCA structure (not the associated buffer), which among other things indicates how many records were returned.

RWQY NB—Unsuccessful Fetch Trace Point:

This trace point occurs in the FETCH routine of the QRWTQRY module when a call to the SQFCHCRS macro results in a null pointer to a BPCA structure, implying that no records were returned in the BPCA buffer.

The data dumped is the SQLSTATE.

RWQY P0—Result Set Pseudo-Open:

The associated information is the package list entry.

RWQY AR—Array Result Set Processed:

The associated information is the array result set control block.

RWQY DA—Array Result Set SQLDA:

The associated information is the array result set SQLDA.

RWQY DO—Debug Options:

The associated information is a modified version of the QRWOPTIONS string.

RWQY L1 and RWEX L1—Saved in Outbound LOB Table Trace Point:

These trace points record the address and other information about a large object (LOB) column saved by QRWTQRY or QRWTEEXEC for later transmission to an application requester.

RWQY L2 and RWEX L2—Built in Datastream from LOB Table Trace Point:

These trace points record the address and other information about a large object (LOB) column copied by QRWTQRY or QRWTEEXEC to the communications buffer.

RWQY L0 and RWEX L0—Saved in Inbound LOB Table Trace Point:

These trace points record the address and other information about a large object (LOB) column saved by QRWTQRY or QRWTEEXEC for later construction of an SQL descriptor area (SQLDA) for input to the database management system (DBMS).

RWAC RQ—Access RDB Request Trace Point:

This trace point occurs on entry to either the QRWSARDB module at a DRDA application requester (AR), or the QRWTARDB module at an application server (AS).

The content varies accordingly. If the trace is taken at an AS, the content of the data is a two-byte DDM code point identifying the DDM command to be executed by QRWTARDB, followed by the English name of the command, which can be SXXDSCT for disconnection, SXXCLNUP for cleanup, or ACCRDB for a connection. If the trace is taken at the AR, the content of the data is as follows:

OFFSET	TYPE	CONTENT
0	BIN(8)	FUNCTION CODE
1	CHAR(8)	INTERPRETATION OF FUNCTION CODE
9	BIT(8)	BIT FLAGS
10	CHAR(1)	COMMIT SCOPE
11	CHAR(1)	SQLHOLD value
12	CHAR(1)	CMTFAIL value
13	BIN(15)	Index of last AFT entry processed by RWRDBCMT

The function codes are:

0	'CONNECT '	==>	CONNECT
1	'DISCONN'	==>	DISCONNECT
2	'CLEANUP '	==>	CLEANUP
3	'RELEASE '	==>	RELEASE
4	'EXIT '	==>	EXIT
5	'PRECOMT '	==>	PRE-COMMIT
6	'POSTCMT '	==>	POST-COMMIT
7	'PREROLLB'	==>	PRE-ROLLBACK
8	'POSTROLL'	==>	POST-ROLLBACK
9	'FORCED D'	==>	FORCED DISCONNECT

RWAC cb—Access RDB Control Block Trace Points:

The trace points identify control blocks that are associated with functions that are provided by the QRWSARDB module.

- cb** Name of control block
- LV** Local variables.
- DD** Commit definition directory.
- CD** Commit definition control block.
- RI** TSSCNAFT 'remote info' structure.
- CB** Access RDB control block.
- DE** RDB directory entry.

TE Active file table entry.

RWSY FN: SYNCxxx [TYPE:x] -- Source TCP SYNC/RESYNC Trace Point:

This source-side trace point records various commands and replies flown in the execution of TCP/IP two-phase commit operations.

The segment of the data represented by the preceding 'xxx' can be:

- CTL, representing a control command
- RSY, representing a resync command
- CRD, representing reply data from a control command
- RRD, representing reply data from a resync command

For the CTL and RSY records, there is also a TYPE code associated with the commands. It is not a printable character, so it is observable only in the hexadecimal data part of the record. It follows the string 'TYPE:'.

RWSY xx: yyyyyyy... -- Target TCP SYNC/RESYNC Trace Point:

This target-side trace point records various information. The type of information is identified by the two characters represented by xx.

The details are in the variable length yyyyyyy string.

- Type RC records the command received: SYNCCTL or SYNCRSY.
- Type RW records the parameter structure WrwSYData.
- Type LG records a received synclog (can be multiple occurrences).
- Type SN records the send buffer when no errors occurred.
- Type GE records the local variables at time of a general exception.
- Type TE records the send buffer and local variables when a request to TN component failed (two occurrences of record).
- Type CP records the send buffer and local variables when a conversation protocol error was detected (two occurrences of record).

RW_ff_m—Application Requester Driver Control Block Trace Point:

This trace point displays the contents of the Application Requester Driver (ARD) control blocks for the different types of ARD calls that can be made. It displays three different types of control blocks: input formats, output formats, and SQLCAs.

The type of call and type of control block being displayed is encoded in the trace point ID. The form of the ID is RW_ff_m, where ff is the call-type ID, and m is the control block type code. The call-type IDs (ff) and control block type codes (m) are as follows:

ff	Call Type	m	Ctl Blk Type
--	-----	-	-----
CN	Connect	I	Input Format
DI	Disconnect	O	Output Format
BB	Begin bind	C	SQLCA
BS	Bind Statement		
EB	End bind		
PS	Prepare Statement		
PD	Prepare and Describe Statement		
XD	Execute Bound Statement with Data		
XB	Execute Bound Statement without Data		
XP	Execute Prepared Statement		
XI	Execute Immediate		

OC Open Cursor
FC Fetch from Cursor
CC Close Cursor
DS Describe a Statement
DT Describe an Object

First-failure data capture

The iSeries server provides a way for you to capture and report error information for the distributed relational database. This function is called first-failure data capture (FFDC).

The primary purpose of FFDC support is to provide extensive information on errors detected in the DDM components of the i5/OS operating system from which an Authorized Program Analysis Report (APAR) can be created.

You can also use this function to help you diagnose some system-related application problems. By means of this function, key structures and the DDM data stream are automatically dumped to the spooled file. The goal of this automatic dumping of error information on the first occurrence of an error is to minimize the need to have to create the failure again to report it for service support. FFDC is active in both the application requester and the application server.

One thing you should keep in mind is that not all negative SQLCODEs result in dumps; only those that might indicate an APAR situation are dumped.

A first-failure data capture dump

System-detected internal failures trigger first-failure data capture (FFDC) data to be dumped. FFDC output can be disabled by setting the QSFWERRLOG system value to *NOLOG, but it is strongly recommended that you do not disable the FFDC dump process.

If an FFDC dump has occurred, the informational message, *Software problem detected in Qxxxxxxx. (where Qxxxxxxx is an operating system module identifier), is logged in the QSYSOPR message queue.

To see output from an FFDC dump operation, use the Work with Spooled Files (WRKSPLF) command and view QPSRVDMP. The information contained in the dump output are:

- DDM function
- Specific information on the failing DDM module
- DDM source or target main control block
- DDM internal control structures
- DDM communication control blocks
- Input and output parameter list for the failing DDM module if at the application requester
- The request and reply data stream

The first 1 KB of data is put in the error log. However, the data put in the spooled file is always complete and easier to work with. If multiple DDM conversations have been established, the dump output might be contained in more than one spooled file because of a limit of only 32 entries per spooled file. In this case, there will be multiple software problem messages in the QSYSOPR message queue that are prefixed with an asterisk (*).

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 250.

A 5738SS1 V2R1M1 AS/400 DUMP B 090454/SRR/SRRS1 02/27/91 15:12:52 PAGE 1
DUMP TAKEN FOR DETECTED ERROR

C QRWSQRY LIBRARY- S
..LICENSED PROGRAM- 5738SS1 V2R1M1
..FUNCTION- 5001
..LOAD- 0000
..PTF-

D QRWSQRY LIBRARY- S
..LICENSED PROGRAM- 5738SS1 V2R1M1
..FUNCTION- 5001
..LOAD- 0000
..PTF-
.SYMPTOM STRING-

E F G
5738 MSGCPF3E86 F/QRWSQRY RC10000002

H 01 I
.SPAC- 01
000000 F0F17EC9 D5C4E740 F0F27EC6 C3E34E40 F0F37EC5 D4E2C740 F0F47ED7 D9D4E240 *01=INDX 02=FCT+ 03=EMSG 04=PRMS *
000020 F0F57EE2 D5C4C240 F0F67ED9 C3E5C240 F0F77EC1 D9C4C240 F0F87ED8 C4E3C140 *05=SNDB 06=RCVB 07=ARDB 08=QDTA *
000040 F0F97EC9 D5C4C140 F1F07EE2 D8C3C140 F1F17EE6 D9C3C140 F1F27ED9 C6D4E340 *09=INDA 10=SQCA 11=WRCA 12=RFMT *
000060 F1F37EC1 C6E34040 F1F47EE2 D4C3C240 F1F57EE3 E2D3D240 F1F67EE5 C1D9E240 *13=AFT 14=SMCB 15=TSLK 16=VAR5 *
000080 4DD9C5E2 E340C9E2 40C3C3C2 6BD7C3C2 E26BE2C1 E36BD7D4 C1D76BD9 C3E5C240 *(REST IS CCB,PCBS,SAT,PMAP,RCVB *
0000A0 D7C5D940 C3C3C25D *PER CCB) *

L 02 J
.SPAC- 02
000000 200C1254 0102F5F8 F0F0F9 * 58009 *
.SPAC- 04
000000 D8D7C1D9 D4E20000 D67FC01D A60065A0 00000000 F0F10000 00000434 00000000 *QPAMS 0" 01 *
000020 D9C3C8C1 E2F2F6F6 40404040 40404040 4040E2D9 D9404040 40404040 40404040 *RCHAS266 SRR *
000040 40404040 D7E3F140 40404040 40404040 40404040 4040700F 70DB33C0 00BB0005 * PT1 *

05
000000 00000000 0056D051 00010050 200C0044 2113D9C3 C8C1E2F2 F6F64040 40404040 * & RCHAS266 *
000020 40404040 E2D9D940 40404040 40404040 40404040 4040D7E3 F1404040 40404040 * SRR PT1 *
000040 40404040 40404040 700F70DB 33C000BB 00050008 21140000 7FFF0021 D0030001 * *
000060 001B2412 00100010 0676D004 00000671 E4D00001 0007147A 000002 * *
.SPAC- 06
000000 0016D052 00010010 22050006 11490000 00062102 24170052 D0530001 0022241A * *
000020 0F76D004 00002600 03020000 0A000009 71E05400 01D00001 0671F0E0 0000002A * *
000040 241BFF00 0001F0F0 F1000000 013FF000 00000000 00FF0000 02F0F0F2 00000002 * 001 0 002 *
000060 40000000 00000000 0010D052 0001000A 220B0006 11490004 0069D003 00010063 * *
0000E0 FF * *

K 07
.SPAC- 07
000000 D9C3C8C1 E2F2F6F6 40404040 40404040 4040D9C3 C8C1E2F2 F6F64040 40404040 *RCHAS266 RCHAS266 *
000020 40404040 E2D9D940 40404040 40404040 40404040 4040D7E3 F1404040 40404040 * SRR PT1 *
000040 40404040 40404040 700F70DB 33C000BB D8E3C4E2 D8D3F4F0 F0D8E2D8 F0F2F0F1 * QTDSQL400QS00201*
000060 F1002500 00000000 25000000 000010F0 F4F5F1F7 F461E2D9 D961C4E2 F3F7F840 *1 045174/SRR/DS378 *
000080 40404040 40404040 40404040 40404040 40404040 40404040 40404040 * *
LINES 0000A0 TO 00015F SAME AS ABOVE
000160 40404040 40404040 40404040 4040A000 2434E2D9 D9404040 40404040 00000000 * SRR *
000180 C1D7D7D5 4BD9C3C8 C1E2F3F7 F8A7CCA7 54137200 40404000 00000000 00000000 *APPN.RCHAS378x x *
0001A0 00000000 00000000 * *

```

.SPACE-          09
000000  E2D8D3C4 C1404040 00000060 00010001 01F40002 00000400 00000040 40404040 *SQLDA          4          *
000020  80000000 00000000 007FC01E 11000334 00000000 00000000 00000000 00000000 *              *
000040  00080000 00250000 00000000 00000000 00000000 00000000 00000000 00000000 *              *
.SPACE-          10
000000  E2D8D3C3 C1404040 00000088 FFF8ABC 00041254 01020000 00000000 00000000 *SQLCA          *
000020  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *              *
000040  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *              *
000060  00000000 00000000 00000000 00000000 00000000 00000000 40404040 40404040 *              *
000080  404040F5 F8F0F0F9 *              * 58009          *
.SPACE-          11
000000  E2D8D3C3 C1404048 00000088 00000000 00000000 00000000 00000000 00000000 *SQLCA          *
000020  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *              *
000040  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *              *
000060  00000000 00000000 00000000 00000000 00000000 00000000 40404040 40404040 *              *
000080  404040F0 F0F0F0F0 *              * 00000          *
.SPACE-          13
000000  00001BB0 00310001 F0F0F0F0 F0F0F0F0 00000000 00000000 00000000 00000000 *              00000000          *
000020  00000470 000002C0 7023C382 57000048 80000000 00000000 007FA083 A3000820 *              *
000040  80000000 00000000 007FA083 E7000100 D9C3C8C1 E2F2F6F6 40404040 40404040 *              RCHAS266          *
000060  40405CD3 D6C34040 40404040 5CD5C5E3 C1E3D940 D9C3C8C1 E2F2F6F6 5CD3D6C3 * *LOC          *NETATR RCHAS266*LOC*
LINES 0000A0 TO 001B9F SAME AS ABOVE
001BA0 00000000 00000000 00000000 00000000 *              *
.SPACE-          14
000000  E2D4C3C2 20000100 00000010 F0F9F0F4 F5F461E2 D9D961E2 D9D9E2F1 00000000 *SMCB          090454/SRR/SRRS1          *
000020  00000000 00000000 E5F0F2D9 F0F1D4F0 F1D9C3C8 C1E2F3F7 F8000000 00800000 *              V02R01M01RCHAS378          *
000040  0302C3D5 E2E2D5D9 C3E5D8D3 F7F9F7F1 80000000 00000000 007FA083 E9000106 * CNSSNRCVQL7971          *
000060  F1000000 00710000 00000000 00000000 00000470 000002C0 7023C382 57000048 *1              *
.SPACE-          15
000000  00000000 00000000 007FA083 E60019FF 00000000 00000000 00000000 00000000 *              *
000020  00000000 00400000 *              *
.SPACE-          16
000000  00000000 00000000 00000000 00000002 00000017 000000E1 00000000 00000071 *              *
000020  00000000 00007FFF 00000003 00170000 001B0000 FF000000 00002410 00F0F060 *              *
000040  E70400 *              *X          *
.SPACE-          17
000000  E2C3C3C2 5CD3D6C3 40404040 40405CD5 C5E3C1E3 D9405CD3 D6C34040 4040D9C3 *SCCB*LOC          *NETATR *LOC          RC*
000020  C8C1E2F2 F6F65CD3 D6C34040 404007F6 C4C24040 40405CC4 D9C4C140 40404040 *HAS266*LOC          6DB          *DRDA          *
000040  40404040 40404040 4000001E 00110000 00000000 00000000 00000000 00000000 *              *
000060  00000000 00000000 00000000 00000000 *              *
.SPACE-          18
000000  E2D7C3C2 00000000 007FA083 A3000810 00000470 000002C0 7023C382 57000048 *SPCB          *
000020  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *              *
000040  00000000 00000000 00000000 00000000 *              *
.SPACE-          19
000000  C5E7C3C2 00000076 00000003 00000079 00000009 00000082 00000010 00000092 *EXCB          *
000020  00000008 00000000 00000018 00200003 00030003 00030003 00030001 00030003 *              *
000040  00000000 00000000 00000000 00000000 00000000 0000C4C4 D4E5F0F2 D9F0F1D4 *              *
DDMV02R01M*000060 F0F1F0F4 F5F1F7F4 61E2D9D9 61C4E2F3 F7F8D9C3 C8C1E2F2 F6F6 *              *
01045174/SRR/DS378RCHAS266          *
.SPACE-          20
000000  00000030 000002B6 00000430 0000043E 00010000 00000000 00000000 00000000 *              *
000020  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *              *
000040  80000000 00000000 007FA083 D2000100 00000000 0000029A 0000005C 22050000 *              *
000060  00060000 02B60000 00B00000 00000000 00000000 00000000 00000000 00000000 *              *
LINES 0000E0 TO 00017F SAME AS ABOVE

```

```

.SPACE-                21
000000 0016D052 00010010 22050006 11490000 00062102 24170052 D0530001 0022241A * *
000020 0F76D004 00002600 03020000 0A000009 71E05400 01D00001 0671F0E0 0000002A * *
000040 241BFF00 0001F0F0 F1000000 013FF000 00000000 00FF0000 02F0F0F2 00000002 * 001 0 002 *
000060 40000000 00000000 00100052 0001000A 220B0006 11490004 0069D003 00010063 * *
000080 24080000 000064F0 F2F0F0F0 D8E2D8C6 C5E3C3C8 00D9C3C8 C1E2F2F6 F6404040 * 02000QSQFETCH RCHAS266 *
.SPACE-                22
000000 E2C3C3C2 5CD3D6C3 40404040 40405CD5 C5E3C1E3 D9405CD3 D6C34040 4040D9C3 *SCCB*LOC *NETATR *LOC RC*
000020 C8C1E2F2 F6F65CD3 D6C34040 404007F0 F0F14040 4040E77D F0F7C6F0 C6F0C6F1 *HAS266*LOC 001 X'07F0F0F1*
000040 7D404040 40404040 40000014 00110000 00000000 00000000 00000000 00000000 * *
000060 00000000 00000000 00000000 00000000 00008F00 00000700 F0F0F100 00000000 * 001 *
.SPACE-                23
000000 C5E7C3C2 00000076 00000003 00000079 00000009 00000082 00000010 00000092 *EXCB b k*
000020 00000008 00000000 00000018 00200003 00030003 00030003 00030001 00030003 * *
000040 00000000 00000000 00000000 00000000 00000000 0000C4C4 D4E5F0F2 D9F0F1D4 * DDMV02R01M*
000060 F0F1F0F4 F5F1F7F2 61E2D9D9 61C4E2F3 F7F8D9C3 C8C1E2F2 F6F6 *01045172/SRR/DS378RCHAS266 *
.SPACE-                24
000000 00000030 0000005C 00000000 000000CC 00010000 00000000 00000000 00000000 * *
000020 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 * *
000040 80000000 00000000 007FA083 A4000100 00000000 00000000 0000005C D2010000 * *K *
.SPACE-                25
000000 0010D002 0001000A D2010006 11490000 E200D11 5AE5F0F2 D9F0F1D4 F0F1000C * V02R01M01 *
000020 116DD9C3 C8C1E2F2 F6F60014 115EF0F4 F5F1F7F2 61E2D9D9 61C4E2F3 F7F80064 * RCHAS266 045172/SRR/DS378 *
000040 14041403 00031423 00031405 00031406 00031407 00031444 00031458 00011457 * *
000060 0003140C 00031419 0003141E 00031422 0003240F 000314A0 00041432 00031433 * *
END OF DUMP
***** END OF LISTING *****

```

Related reference

Work with Spooled Files (WRKSPLF) command

FFDC dump output description

This information describes the data areas and types of information available in a first-failure data capture (FFDC) dump output.

Like the figure in the “A first-failure data capture dump” on page 235, the following data areas and types of information are available in the FFDC dump output:

1. Each FFDC dump output will differ in content, but the format is generally the same. An index (I) is provided to help you understand the content and location of each section of data.
2. Each section of data is identified by SPACE and a number; for example: SPACE- ... 01. The sections of data present in your dump output are dependent on the operation and its progress at the time of failure.
3. Each section of data is given a name; for example SQCA. SQCA is the section name for data from the DB2 UDB Query Manager and SQL Development Kit SQL diagnostic area. To locate the SQL diagnostic area data, find SQCA in the index (I). In the sample dump index, SQCA is shown to be in data section 10 (10=SQCA). To view the SQL diagnostic area data, go to the SPACE- 10 heading.
4. There are two basic classes of modules that can be dumped:
 - Application requester (AR) modules
 - Application server (AS) modules

The sample dump output is typical of a dump from an AR module. AR dump outputs typically have a fixed number of data sections identified in the index. (For example, in the sample dump output SPACE- 01 through 16 are listed.) In addition, they have a variable number of other data sections. These sections are not included in the index. (For example, in the sample dump output, SPACE- 17 through 25 are not listed in the index.)

Application server dump output is usually simpler because they consist only of a fixed number of data sections, all of which are identified in the index.

5. There are index entries for all data sections whether the data section actually exists in the current dump output or not. For example, in the sample dump output, there is no SPACE- 08. In the index, 08 equals QDTA (query data). The absence of SPACE- 08 means that no query data was returned, so none could be dumped.

6. In the sample dump output, the last entry in the index is "(REST IS CCB, PCBS, SAT, PMAP, RCVB, PER CCB)". This entry means that SPACE- 17 and upward contain one or more communications control blocks (CCB), each containing:

- Zero, one, or more path control blocks (SPCB); there is normally just one.
- Exchange server attributes control block (EXCB)
- Parser map space
- Receive buffer for the communications control block

The data section number is increment by one from 17 onward as each control block is dumped. For example, in the sample dump output, data sections SPACE- 17 through SPACE- 21 are for the first data control block dumped (CCB 1), while data sections SPACE- 22 through SPACE- 25 are for the second data control block dumped (CCB 2), as shown here:

```
17      CCB (Eyecatcher is :'SCCB:'. For an application server module, the eyecatcher is :'TCCB:'.)
18      PCB for CCB 1 (Eyecatcher is :'SPBC:'.)
19      SAT for CCB 1 (Eyecatcher is :'EXCB:'.)
20      PMAP for CCB 1 (No eyecatcher.)
21      RCVB for CCB 1 (No eyecatcher.)
22      CCB 2 (Eyecatcher is :'SCCB:'.)
--      (No PCB for CCB 2 because the conversation is not active.)
23      SAT for CCB 2 (Eyecatcher is :'EXCB:'.)
24      PMAP for CCB 2 (No eyecatcher.)
25      RCVB for CCB 2 (No eyecatcher.)
A      Name and release information of the server on which the dump was taken.
B      Name of job that created the dump output.
C      Name of module in the operating system suspected of failure.
D      Name of module that detected the failure.
```

Symptom string-contents:

- E Message identifier.
- F Name of module suspected of causing the FFDC dump.
- G Return code (RC), identifying the point of failure.

The first digit after RC indicates the number of dump files associated with this failure. There can be multiple dump files depending on the number of conversations that were allocated. In the sample dump output, the digit is "1", indicating that this is the first (and possible the only) dump file associated with this failure.

You might have four digits (not zeros) at the rightmost end of the return code that indicate the type of error.

- The possible codes for errors detected by the AR are:
 - 0001 Failure occurred in connecting to the remote database
 - 0002 More-to-receive indicator was on when it should not have been
 - 0003 AR detected an unrecognized object in the data stream received from the AS
 - 0097 Error detected by the AR DDM communications manager

0098 Conversation protocol error detected by the DDM component of the AR

0099 Function check

- The possible codes for errors detected by the AS are:

0099 Function check

4415 Conversational protocol error

4458 Agent permanent error

4459 Resource limit reached

4684 Data stream syntax not valid

4688 Command not supported

4689 Parameter not supported

4690 Value not supported

4691 Object not supported

4692 Command check

8706 Query not open

8708 Remote database not accessed

8711 Remote database previously accessed

8713 Package bind process active

8714 FDO:CA descriptor is not valid

8717 Abnormal end of unit of work

8718 Data and/or descriptor does not match

8719 Query previously opened

8722 Open query failure

8730 Remote database not available

H SPACE- number identifying a section of data. The number is related to a data section name by the index. Data section names are defined under **I** below.

I An index and definition of SPACE- numbers (defined in **H**) to help you understand the content and location of each section of data. The order of the different data sections might vary between dump output from different modules. The meaning of the data section names are:

- AFT: DDM active file table, containing all conversation information.
- ARDB: Access remote database control block, containing the AR and AS connection information.
- ARDP: ARD program parameters at start of user space.
- BDTA: Buffer processing communications area (BPCA) and associated data record from SELECT INTO statement.
- Bind: SQL bind template
- BPCA: BPCA structure (without data records)
- DATA: Data records associated with the BPCA. It is possible that the records in this section do not reflect the total BPCA buffer contents. Already-processed records might not be included.
- DOFF: Offset within query data stream (QRYDTA) where the error was detected.
- EICB: Error information control block
- EMSG: Error message associated with a function check or DDM communications manager error.

- FCT: DDM function code point (2 bytes)
- FDOB: FDO:CA descriptor input to the parser in an execute operation.
- FDTA: FDO:CA data structure consisting of:
 - A four-byte field defining the length of the FDO:CA data stream (FDODTA)
 - The FDODTA
- HDRS: Communications manager command header stack.
- IFMT: ARD program input format.
- INDA: Input SQLDA containing user-defined SQLDA for insert, select, delete, update, open, and execute operations.
- INDX: The index that maps the data section name to the data section SPACE- code. Not all of the entries in the index have a corresponding data section. The dump data is based on the error that occurs and the progress of the operation at the time of the error. A maximum of 32 entries can be dumped in one spooled file.
- INST: SQL statement
- ITKN: Interrupt token.
- OFMT: ARD program output format.
- PKGN: Input package name, consistency token and section number.
- PMAP: Parser map in an AS dump output.
- PRMS: DDM module input or output parameter structure.
- PSOP: Input parser options.
- QDTA: Query data structure consisting of:
 - A four-byte field defining the length of the query data stream (QRYDTA)
 - The QRYDTA
- RCVB: Received data stream. The contents depend on the following conditions:
 - If the dump occurs on the application server, the section contains the DDM request data that was sent from the application requester.
 - If the dump occurs on the application requester, the section contains the DDM reply data that was sent from the application server. If this section is not present, it is possible the received data might be found in the receive buffer in the variable part of the dump.
- RDBD: Relational database directory.
- RFMT: Record format structure.
- RMTI: Remote location information in the commitment control block.
- RTDA: Returned SQLDA (from ARD program).
- SMCB: DDM source master control block, containing pointers to other DDM connection control blocks and internal DDM control blocks.
- SNDB: Send data stream. The contents depend on the following conditions:
 - If the dump occurs on the application requester, the buffer contains the DDM request that was sent to the application server or that was being prepared to send.

Note the four bytes of zeros that are at the beginning of SPACE- 05 in the example. When zeros are present, they are not part of the data stream. They represent space in the buffer that is used only in the case that a DDM large object has to be sent with a DDM request. The DDM request stream is shifted left four bytes in that case.
 - If the dump occurs on the application server, the buffer contains the DDM reply data that was being prepared to send to the application requester.
- SQCA: Output SQL diagnostic area being returned to the user.
- SQDA: SQLDA built by the FDO:CA parser.
- TBNM: Input remote database table name.

- TMCB: Target main control block.
- TSLK: Target or source connection control block, containing pointers to the DDM active file table and other internal DDM control blocks.
- VARS: Local variables for the module being dumped.
- WRCA: Warning SQLCA returned only for an open operation (OPNQRYRM).
- XSAT: Exchange server attributes control block.
- Remainder: Multiple conversation control blocks for all the DDM conversations for the job at the time of the error. Each conversation control block contains the following items:
 - Path control blocks, containing information about an established conversation. There can be multiple path control blocks for one conversation control block.
 - One exchange server information control block, containing information about the application requester and application server.
 - One DDM parser map area, containing the locations and values for all the DDM commands, objects, and replies.
 - One receive buffer, containing the requested data stream received by the application server.
 The data section number is incremented by one as each control block is dumped.

J The eyecatcher area. Information identifying the type of data in some of the areas that were dumped.

K The logical unit of work identifier (LUWID) for the conversation in progress at the time of the failure can be found in the access RDB control block. This data area is identified by the string 'ARDB' in the FFDC index. In this example, it is in SPACE- 07. The LUWID begins at offset 180. The network identifier (NETID) is APPC. A period separates it from the logical unit (LU) name, RCHAS378, which follows. Following the LU name is the six-byte LUW instance number X'A7CCA7541372'.

Related tasks

"Find first-failure data capture data" on page 180

This topic describes tips on how to locate first-failure data capture (FFDC) data on an iSeries server. The information is most useful if the failure causing the FFDC data output occurred on the application server (AS). The FFDC data for an application requester (AR) can usually be found in one of the spooled files associated with the job running the application program.

DDM error codes

These error codes are included in the FFDC dumps (L in the sample dump output) that identify DDM error conditions. These conditions might or might not be defined by the DDM architecture.

Command check codes:

If FCT+ (SPACE- 02) contains 1254 in bytes 3 and 4, look for one of these codes in byte 6.

- 01** Failure to connect to the relational database (RDB).
- 02** State of the DDM data stream is incorrect.
- 03** Unrecognized object in the data stream.
- 04** Statement CCSID received from SQL not recognized.
- 05** EXCSQLSTT OUTEXP value is inconsistent with the SQL statement being executed.
- 06** DDM command or object sent to application server (AS) violates i5/OS extension to DRDA2 architecture.
- 07** DDM reply or object received from AS violates DRDA2 architecture.
- 08** SQLDA data pointer is NULL when it should not be.
- 09** Product data structure not valid.

- 0A XLATECC failure.
- 0B EXTJOBBDI failure.
- 0C Get ASP from name failure.
- 0D Get RDB name from ASP failure.
- 0E Unexpected error data.
- 0F DDM/DRDA request not recognized.
- 10 An expected LOB was not received.
- 11 LOB length mismatch between placeholder and received data.
- 12 LOB usage mismatch.
- 13 XMIT Mode wrong for LOBs.
- 14 Buffer extension failure.
- 15 Negative SQLCODE on fetch after successful open.
- 16 Space allocation error.
- 17 Mismatch in result set reply (SQRY).
- 18 Unexpected RM in result set reply (SQRY).
- 19 Error building reply.
- 1A SQ component returned SQL code -30020.
- 1B Error updating SQL diagnostic area.
- 1C Error building reply.
- 88 No records in BPCA.
- 89 Unexpected BGNBND object.
- 8A Unsupported large DDM object header size.
- 8B LOB table error.
- 8C Request for LOB and none available.
- 8D SET_LELAST error 1.
- 8E SET_LELAST error 2.
- 8F Unexpected non-zero QRYINSID.
- 90 Non zero QRYINSID.
- 91 OPNQFL on P-open.
- 92 OPNQFL on Normal open.
- 97 DDM communications manager detected an error.
- 98 Conversation protocol error detected by the DDM module.
- 99 Function check. Look for EMSG section, normally in SPACE- 03.
- FF Error on SQ open (TQRY).

Conversational protocol error code descriptions:

If FCT+ (SPACE- 02) contains 1245 in bytes 3 and 4, look for one of these codes in byte 6.

- 01 RPYDSS received by target communications manager.

- 02 Multiple DSSs sent without chaining, or multiple DSS chains sent.
- 03 OBJDSS sent when not allowed.
- 04 Request correlation identifier of an RQSDSS is less than or equal to the previous RQSDSS request correlation identifier in the chain.
If two RQSDSSs have the same request correlation identifier, the PRECCNVRM must be sent in RPYDSS with a request correlation identifier of minus 1.
- 05 Request correlation identifier of an OBJDSS does not equal the request correlation identifier of the preceding RQSDSS.
- 06 EXCSAT was not the first command after the connection was established.
- DA SQLDA not doubled to accommodate labels.
- DF FDODSC was received but no accompanying FDODTA.
- E0 No OPNQRY (open query) reply message.
- E1 RDBNAM on ENDQRYRM (end query reply message) is not valid.
- E2 An OPEN got QRYDTA (query answer set data) without a QRYDSC (query answer set description).
- E3 Unexpected OPNQRY reply object.
- E4 Unexpected CXXQRY reply object.
- E5 QRYDTA on OPEN, single row.
- E6 RM after OPNQRYRM is not valid.
- E7 No interrupt reply message.
- E8 LOB request where application server (AS) does not support.
- E9 Light SQLDA received when standard version expected.
- FD Null SQLCARD (SQLCA reply data) following error RM.
- FE Null QRYDTA row follows null SQLCA.
- FF Expected SQLCARD missing.

DDM syntax error code descriptions:

If FCT+ (SPACE- 02) contains 124C in bytes 3 and 4, look for one of these codes in byte 6.

- 01 DSS header length less than 6.
- 02 DSS header length does not match the number of bytes of data found.
- 03 DSS head C-byte not X'D0'.
- 04 DSS header F-bytes either not recognized or not supported.
- 05 DSS continuation specified, but not found. For example, DSS continuation is specified on the last DSS, and the SEND indicator has been returned by the SNA LU 6.2 communications program.
- 06 DSS chaining specified, but no DSS found. For example, DSS chaining is specified on the last DSS, and the SEND indicator has been returned by the SNA LU 6.2 communications program.
- 07 Object length less than 4. For example, a command parameter length is specified as 2, or a command length is specified as 3.
- 08 Object length does not match the number of bytes of data found. For example, an RQSDSS with a length 150 contains a command whose length is 125, or an SRVDGN (server diagnostic information) parameter specifies a length of 200, but there are only 50 bytes left in the DSS.







- 09 Object length greater than maximum allowed. For example, the RECCNT parameter specifies a length of 5, but this indicates that only half of the hours field is present instead of the complete hours field.
- 0A Object length less than minimum required. For example, the SVRCOD parameter specifies a length of 5, but the parameter is defined to have a fixed length of 6.
- 0B Object length not allowed. For example, the FILEXPDT parameter is specified with a length of 11, but this indicates that only half of the hours field is present instead of the complete hours field.
- 0C Incorrect large object extended length field (see the description of DSS). For example, an extended length field is present, but it is only 3 bytes long. It is defined as being a multiple of 2 bytes long.
- 0D Object code point index not supported. For example, a code point of X'8032' is encountered, but X'8' is a reserved code point index.
- 0E Required object not found. For example, a CLRFIL command does not have an FILNAM parameter present, or an MODREC command is not followed by a RECORD command data object.
- 0F Too many command data objects sent. For example, an MODREC command is followed by two RECORD command data objects, or a DELREC command is followed by a RECORD object.
- 10 Mutually exclusive objects present. For example, a CRTDIRF command specifies both a DCLNAM and a FILNAM parameter.
- 11 Too few command data objects sent. For example, an INSRECEF command that specified RECCNT(5) is followed by only four RECORD command data objects.
- 12 Duplicate object present. For example, a LSTFAT command has two FILNAM parameters specified.
- 13 Specified request correlation identifier not valid. Use PRCCNVRM with a PRCCNVCD of X'04' or X'05' instead of this error code. This error code is being maintained for compatibility with Level 1 architecture.
- 14 Required value not found.
- 15 Reserved value not allowed. For example, an INSRECEF command specified an RECCNT(0) parameter.
- 16 DSS continuation less than or equal to 2. For example, the length bytes for the DSS continuation have a value of 1.
- 17 Objects not in required order. For example, a RECAL object contains a RECORD object followed by a RECNBR object that is not in the specified order.
- 18 DSS chaining bit not a binary 1, but DSSFMT bit 3 is set to a binary 1. Next DSS has the same request correlator.
- 19 Previous DSS indicated current DSS has the same request correlation, but the request correlation identifiers are not the same.
- 1A DSS chaining bit not a binary 1, but error continuation is requested.
- 1B Mutually exclusive parameter values specified. For example, an OPEN command specified PRPSHD(TRUE) and FILSHR(READER).
- 1D Code point not a valid command. For example, the first code point in RQSDSS either is not in the dictionary or is not a code point for a command.





Related information for distributed database programming

Listed here are the product manuals and IBM Redbooks (in PDF format), Web sites and information center topics that relate to the distributed database programming topic. You can view or print any of the PDFs.

iSeries server information

These iSeries books and information center topics contain information you might need.

- ADTS/400: Data File Utility  provides the application programmer, programmer or help desk aide with information about the Application Development Tools data file utility (DFU) to create programs to enter data into files, update files, inquire into files and run DFU programs. This manual also provides the workstation operator with activities and material to learn about DFU.
- The Backup and recovery topic provides the system programmer with information about the different media available to save and restore system data, as well as a description of how to record changes made to database files and how that information can be used for system recovery and activity report information.
- The CL programming topic provides a wide-ranging discussion of programming topics, including a general discussion of objects and libraries, control language (CL) programming, controlling flow and communicating between programs, working with objects in CL programs, and creating CL programs. Other topics include predefined and immediate messages and message handling, defining and creating user-defined commands and menus, and application testing, including debug mode, breakpoints, traces, and display functions.
- Communications Management  contains information about working with communications status, communications-related work management topics, communications errors, performance, line speed and subsystem storage.
- DB2 Universal Database for iSeries Query Management Programming  provides the application programmer with information about determining the database files to be queried for a report, defining a structured query language (SQL) query definition, and using and writing procedures that use query management commands. This manual also includes information about how to use the query global variable support and understanding the relationship between the query management function of the operating system and the IBM Query for iSeries.
- The Distributed data management topic provides the application programmer or system programmer with information about remote file processing. It describes how to define a remote file to i5/OS distributed data management (DDM), how to create a DDM file, which file utilities are supported through DDM, and the requirements of i5/OS DDM as related to other systems.
- DSNX Support  provides information for configuring an iSeries server to use the remote management support (distributed host command facility), the change management support (distributed systems node executive) and the problem management support (alerts).
- Local Device Configuration  provides the system operator or system administrator with information about how to do an initial local hardware configuration and how to change that configuration. It also contains conceptual information for device configuration, and planning information for device configuration on the 9406, 9404, and 9402 System Units.
- SNA Distribution Services  provides the system programmer or network administrator with information about configuring a communications network for systems network architecture distribution services (SNADS) and the Virtual Machine/Multiple Virtual Storage (VM/MVS) bridge. In addition, object distribution functions, document library services and system distribution directory services are also discussed.

- ICF Programming  provides the application programmer with the information needed to write application programs that use iSeries communications and ICF files. It also contains information about data description specifications (DDS) keywords, system-supplied formats, return codes, file transfer support, and programming examples.
- LAN, Frame-Relay and ATM Support  contains information about using an iSeries server in a token-ring network, an Ethernet network, or bridged network environment.
- Remote Work Station Support  provides information about how to set up and use remote workstation support, such as display station pass-through, distributed host command facility, and 3270 remote attachment. It also provides information for the application programmer or system programmer about configuration commands and defining lines, controllers, and devices.
- The Security topic provides the system programmer (or someone who is assigned the responsibilities of a security officer) with information about system security concepts, planning for security, and setting up security on the system.
- The SQL programming topic provides the application programmer, programmer, or database administrator with an overview of how to design, write, test and run SQL statements. It also describes interactive Structured Query Language (SQL).
- The SQL reference topic provides the application programmer, programmer, or database administrator with detailed information about SQL statements and their parameters.
- X.25 Network Support  contains information about using iSeries servers in an X.25 network.

Distributed relational database library

The books in this topic provide background and general support information for IBM Distributed Relational Database Architecture (DRDA) implementations.

- *DRDA: Every Manager's Guide*, GC26-3195, provides concise, high-level education on distributed relational database and distributed file. This book describes how IBM supports the development of distributed data systems, and discusses some current IBM products and announced support for distributed data. The information in this book is intended to help executives, managers, and technical personnel understand the concepts of distributed data.
- *DRDA: Planning for Distributed Relational Database*, SC26-4650, helps you plan for distributed relational data. It describes the steps to take, the decisions to make, and the options from which to choose in making those decisions. The book also covers the distributed relational database products and capabilities that are now available or that have been announced, and it discusses IBM's stated direction for supporting distributed relational data in the future. The information in this book is intended for planners.
- *DRDA: Connectivity Guide*, SC26-4783, describes how to interconnect IBM products that support Distributed Relational Database Architecture. It explains concepts and terminology associated with distributed relational database and network systems. This book tells you how to connect unlike systems in a distributed environment. The information in the Connectivity Guide is not included in any product documentation. The information in this book is intended for system administrators, database administrators, communication administrators, and system programmers.
- *DRDA: Application Programming Guide*, SC26-4773, describes how to design, build, and modify application programs that access IBM's relational database management systems. This manual focuses on what a programmer should do differently when writing distributed relational database applications for unlike environments. Topics include program design, preparation, and execution, as well as performance considerations. Programming examples written in IBM C are included. The information in this manual is designed for application programmers who work with at least one of IBM's high-level languages and with Structured Query Language (SQL).
- *DRDA: Problem Determination Guide*, SC26-4782, helps you define the source of problems in a distributed relational database environment. This manual contains introductory material on each product, for people not familiar with those products, and gives detailed information on how to

diagnose and report problems with each product. The guide describes procedures and tools unique to each host system and those common among the different systems. The information in this book is intended for the people who report distributed relational database problems to the IBM Support Center.

- *IBM SQL Reference, Volume 2, SC26-8416*, makes references to DRDA and compares the facilities of:
 - IBM SQL relational database products
 - IBM SQL
 - ISO-ANSI SQL (SQL92E)
 - X/Open SQL (XPG4-SQL)
 - ISO-ANSI SQL Call Level Interface (CLI)
 - X/Open CLI
 - Microsoft Open Database Connectivity (ODBC) Version 2.0

Other IBM distributed relational database platform libraries

This topic describes other IBM distributed relational database platform libraries.

DB2 Connect and Universal Database

If you are working with DB2 Connect and Universal Database and would like more information, see the Web page [DB2 Universal Database for Linux, UNIX and Windows !\[\]\(96cc62f861fdd6e50510c0224a756dff_img.jpg\)](#). There you can find the following books:

- *DB2 Connect Enterprise Edition Quick Beginning*
- *DB2 Connect Personal Edition Quick Beginning*
- *DB2 Connect User's Guide*
- *DB2 UDB Administration Guide*
- *DB2 UDB Command Reference*
- *DB2 UDB for OS/2 Quick Beginnings*
- *DB2 UDB for UNIX Quick Beginnings*
- *DB2 UDB for Windows NT[®] Quick Beginnings*
- *DB2 UDB Messages Reference*
- *DB2 UDB Personal Edition Quick Beginnings*
- *DB2 UDB SQL Getting Started*
- *DB2 UDB SQL Reference*
- *DB2 UDB Troubleshooting Guide*

DB2 for z/OS and OS/390[®]

If you are working with DB2 for z/OS and OS/390 and would like more information, see the Web page [DB2 for z/OS and OS/390 !\[\]\(e8fb589d58dad1692debababa5e928b6_img.jpg\)](#). There you can find the following books:

- *DB2 for z/OS and OS/390 Command Reference*
- *DB2 for z/OS and OS/390 Messages and Codes*
- *DB2 for z/OS and OS/390 Reference for Remote DRDA*
- *DB2 for z/OS and OS/390 SQL Reference*
- *DB2 for z/OS and OS/390 Utility Guide and Reference*

DB2 Server for VSE & VM

If you are working with DB2 Server for VSE & VM and would like more information, see the Web page

DB2 Server for VSE & VM . There you can find the following books:

- *DB2 and Data Tools for VSE and VM*
- *DB2 for VM Control Center Installation*
- *DB2 Server Data Spaces Support for VM/ESA[®]*
- *DB2 Server for VM Application Programming*
- *DB2 Server for VM Database Administration*
- *DB2 Server for VM Database Services Utilities*
- *DB2 Server for VM Diagnosis Guide*
- *DB2 Server for VM Interactive SQL Guide*
- *DB2 Server for VM Master Index and Glossary*
- *DB2 Server for VM Messages and Codes*
- *DB2 Server for VM Operation*
- *DB2 Server for VM System Administration*
- *DB2 Server for VM/VSE Training Brochure*
- *DB2 Server for VSE & VM Quick Reference*
- *DB2 Server for VSE & VM SQL Reference*
- *DB2 Server for VSE & VM LPS*
- *DB2 Server for VSE & VM Data Restore*
- *SBOF for DB2 Server for VM*

Architecture books

This topic describes different architecture books.

- *Character Data Representative Architecture: Details, SC09-2190*

This manual includes a CD-ROM, which contains the two CDRA publications in online BOOK format, conversion tables in binary form, mapping source for many of the conversion binaries, a collection of code page and character set resources, and character naming information as used in IBM. The CD also includes a viewing utility to be used with the provided material. Viewer works with OS/2, Windows 3.1, and Windows 95.

- *Character Data Representative Architecture: Overview, GC09-2207*
- *DRDA V3 Vol. 1: Distributed Relational Database Architecture*

This Technical Standard is one of three volumes documenting the Distributed Relational Database Architecture Specification. This volume describes the connectivity between relational database managers that enables applications programs to access distributed relational data. It describes the necessary connection between an application and a relational database management system in a distributed environment; the responsibilities of the participants and when flow should occur; and the formats and protocols required for distributed database management system processing. It does not describe an API for distributed database management system processing. This document is available on the Open Group Web site at www.opengroup.org/dbiop/index.htm.

- *DRDA V3 Vol. 2: Formatted Data Object Content Architecture*





This document is one of three Technical Standards documenting the Distributed Relational Database Architecture, Version 3. This volume describes the functions and services that make up the Formatted Data Object Content Architecture (FD:OCA). This architecture makes it possible to bridge the connectivity gap between environments with different data types and data representations methods. FD:OCA is embedded in DRDA. This document is available on the Open Group Web site at www.opengroup.org/dbiop/index.htm.

- *DRDA V3 Vol. 3: Distributed Data Management Architecture*

This document is one of three Technical Standards documenting the Distributed Relational Database Architecture (DRDA). This volume describes the architected commands, parameters, objects, and messages of the DDM data stream. This data stream accomplishes the data interchange between the various pieces of the DDM model. This document is available on the Open Group Web site at www.opengroup.org/dbiop/index.htm.

IBM Redbooks

This topic describes the IBM Redbooks that are available for distributed relational database.

- Distributed Relational Database: Using DDCS/6000 DRDA Support with DB2 and DB2/400, GG24-4155-00 
- DRDA Client/Server for VM and VSE Setup for System and Performance Management, GG24-4275-00 
- Getting Started with DB2 Stored Procedures: Give Them a Call through the Network, SG24-4693-01 
- WOW! DRDA Supports TCP/IP: DB2 Server for OS/390 and DB2 Universal Database, SG24-2212-00 

Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

| SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS
 | PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER
 | EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR
 | CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
 | NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

| UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR
 | ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

- | 1. LOSS OF, OR DAMAGE TO, DATA;
- | 2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC
 | CONSEQUENTIAL DAMAGES; OR
- | 3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

| SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT,
 | INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS
 | OR EXCLUSIONS MAY NOT APPLY TO YOU.

Appendix. Notices

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

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

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

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

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

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

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

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

- | The licensed program described in this information and all licensed material available for it are provided
- | by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement,
- | IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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

COPYRIGHT LICENSE:

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

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This Distributed database programming publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i5/OS.

Trademarks

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

- | Advanced Peer-to-Peer Networking
- | AIX
- | C/400
- | CICS
- | COBOL/400
- | DataJoiner
- | DataPropagator
- | DB2
- | DB2 Connect
- | DB2 Universal Database
- | Distributed Relational Database Architecture
- | DRDA
- | DXT
- | e(logo)server
- | Extended Services
- | i5/OS
- | IBM
- | IBM (logo)
- | IMS
- | Informix
- | Integrated Language Environment
- | iSeries
- | MVS
- | NetView
- | OS/2
- | OS/390
- | OS/400
- | PS/2
- | Redbooks
- | RPG/400
- | RS/6000
- | S/390
- | PS/2
- | Redbooks
- | RPG/400
- | RS/6000
- | S/390
- | SQL/DS
- | VM/ESA
- | z/OS

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

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

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

| Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

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

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.



Printed in USA