DB2® REXX SQL for VM/ESA®

Reference

Version 6 Release 1

SC09-2676-00

DB2® REXX SQL for VM/ESA®

Reference

Version 6 Release 1

SC09-2676-00

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page vii.

This book is also provided as an online book that can be viewed with the IBM® BookManager® READ and IBM Library Reader™ licensed programs.

| **First Edition (December 1998)**

| This edition, SC09-2676, applies to Version 6 Release 1, of the IBM DB2® REXX SQL for VM/ESA, which is a feature of the IBM
| DATABASE 2™ Server for VSE & VM Program 5648-A70-02, and to all subsequent releases and modifications until otherwise
| indicated in new editions. Make sure you are using the correct edition for the level of the product.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change or addition.

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

A form for readers’ comments is provided at the back of this publication. If the form has been removed, address your comments to: IBM Canada Ltd. Laboratory, Information Development, 2G/345/1150/TOR, 1150 Eglinton Ave East, North York, Ontario, Canada M3C 1H7.

You can also send your comments by facsimile to (416) 448-6161 addressed to the attention of the RCF Coordinator. If you have
| access to Internet, you can send your comments electronically to **torrcf@ca.ibm.com**; IBMLink™, to **toribm(torrcf)**; IBM/PROFS®, to **torolab4(torrcf)**; IBMMAIL, to **ibmmail(caibmwt9)**; or through our home page at
| **http://www.software.ibm.com/data/db2/vse-vm/**

If you choose to respond through Internet, please include either your entire Internet network address, or a postal address.

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

Contents

Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user’s responsibility.

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

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 the IBM Canada Ltd., Department 071, 1150 Eglinton Avenue East, North York, Ontario 3MC 1H7, Canada. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

This publication 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.

Programming Interface Information

This manual documents intended Programming Interfaces that allow the customer to write programs to obtain services of DB2 RXSQL.

Trademarks

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

IBM
BookManager
CICS/VSE
DATABASE 2
DataPropagator
DB2
Distributed Relational Database Architecture
DRDA
IBMLink
IMS
Library Reader
OS/390
PROFS
SQL/DS

About This Manual

This manual describes how the DB2 REXX SQL for VM/ESA (DB2 RXSQL) allows REXX programs to access the IBM DATABASE 2® Server for VM in a VM environment.

Who Should Use This Manual

This manual assumes that readers are familiar with the concepts of relational databases, the facilities of the DB2 Server for VM relational database manager, and the elements of the REXX programming language.

Terms Used in This Manual

Term	Meaning
ISQL	Interactive Structured Query Language
IUCV	Inter-user communication vehicle
LUW	Logical unit of work
Request	Either a DB2 RXSQL statement or command
REXX	Restructured Extended Executor Language
RXSQL	DB2 REXX SQL for VM/ESA
SAA	Systems Application Architecture
user ID	Userid, user identification
VM/ESA	Virtual Machine/Enterprise Systems Architecture Version 2 Release 2 or later
XEDIT	System Product editor

Terminology Changes

In past documentation, **database** was used in a general sense to refer to the database management system as well as to stored objects and storage devices. However, using **database** in this context has proven to be misleading. Accordingly, the following terminology has been adopted and is used consistently in the manuals:

New Term	Meaning
Application Server	Refers to the database management system including the accumulated data.
Application Requester	Refers to the facility that transforms a request from the application into a form suitable for communicating with an application server.
Database	Refers only to the collection of data.

How This Manual Is Organized

This manual contains the chapters and appendixes described here, and a summary of changes, glossary, bibliography, and index.

Chapter 1	Introduces the DB2 RXSQL interface, describes how the interface relates to REXX and DB2 Server for VM, and lists some sample programs to help you learn the interface.
Chapter 2	Provides information on the use of the general-purpose programs supplied with DB2 RXSQL.
Chapter 3	Introduces concepts needed for SQL applications.
Chapter 4	Explains how to write Dynamic and Extended Dynamic RXSQL applications.
Chapter 5	Outlines how to code RXSQL requests.
Chapter 6	Contains detailed descriptions of each DB2 RXSQL request.
Appendix A	Lists DB2 RXSQL return codes and error messages.
Appendix B	Illustrates typical DB2 RXSQL request functions within sample programs.
Appendix C	Contains performance and diagnosis information.
Appendix D	Describes how DB2 RXSQL uses CMS work units.
Appendix E	Explains how DB2 RXSQL requests can be executed in the DB2 RXSQL subcommand environment.
Appendix F	Explains the DB2 RXSQL run-time environment.
Appendix G	Describes how to run DB2 RXSQL in single user mode.
Appendix H	Outlines the considerations for running RXSQL using the Distributed Relational Database Architecture™ (DRDA®) protocol.
Appendix I	Explains the incompatibilities by release.

If You Need More Information

Prerequisite IBM Publications

For information on the REXX programming language, refer to:

- *VM/ESA: REXX/VM Reference*, SC24-5770
- *VM/ESA: REXX/VM User's Guide*, SC24-5465.

The following manual is to be used in conjunction with the *DB2 REXX SQL for VM/ESA Reference* manual.

- *DB2 Server for VSE & VM SQL Reference*, SC09-2671.

See "Bibliography" on page 171 for related VM and DB2 Server for VM publications.

Related IBM Publications

Various IBM publications are mentioned throughout this document. Refer to them if you require additional information on related IBM products.

Only the short titles of the manuals are given in the text. For the long titles and their corresponding document numbers, see the Bibliography.

Syntax Notation Conventions

Throughout this manual, syntax is described using the structure defined below.

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

  The >>—— symbol indicates the beginning of a statement or command.

  The ——> symbol indicates that the statement syntax is continued on the next line.

  The >—— symbol indicates that a statement is continued from the previous line.

  The ——>< symbol indicates the end of a statement.

  Diagrams of syntactical units that are not complete statements start with the >—— symbol and end with the ——> symbol.

- Some SQL statements, Interactive SQL (ISQL) commands, or database services utility (DBS Utility) commands can stand alone. For example:

```
►►──SAVE────────────────────────────────────────────────────►◄
```

  Others must be followed by one or more keywords or variables. For example:

```
►►──SET AUTOCOMMIT OFF──────────────────────────────────────►◄
```

- Keywords may have parameters associated with them which represent user-supplied names or values. These names or values can be specified as either constants or as user-defined variables called *host_variables* (*host_variables* can only be used in programs).

```
►►──DROP SYNONYM──synonym───────────────────────────────────►◄
```

- Keywords appear in either uppercase (for example, SAVE) or mixed case (for example, CHARacter). All uppercase characters in keywords must be present; you can omit those in lowercase.
- Parameters appear in lowercase and in italics (for example, *synonym*).
- If such symbols as punctuation marks, parentheses, or arithmetic operators

  are shown, you must use them as indicated by the syntax diagram.

- All items (parameters and keywords) must be separated by one or more blanks.
- Required items appear on the same horizontal line (the main path). For example, the parameter *integer* is a required item in the following command:

```
►►──SHOW DBSPACE──integer────────────────────────────────────────────►◄
```

  This command might appear as:

```
SHOW DBSPACE 1
```

- Optional items appear below the main path. For example:

```
►►──CREATE──┬────────┬──INDEX────────────────────────────────────────►◄
            └─UNIQUE─┘
```

  This statement could appear as either:

```
CREATE INDEX
```

  or

```
CREATE UNIQUE INDEX
```

- If you can choose from two or more items, they appear vertically in a stack.

  If you must choose one of the items, one item appears on the main path. For example:

```
►►──SHOW LOCK DBSPACE──┬─ALL─────┬───────────────────────────────────►◄
                       └─integer─┘
```

  Here, the command could be either:

```
SHOW LOCK DBSPACE ALL
```

  or

```
SHOW LOCK DBSPACE 1
```

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

```
►►──BACKWARD──┬─────────┬────────────────────────────────────────────►◄
              ├─integer─┤
              └─MAX─────┘
```

  Here, the command could be:

```
BACKWARD
```

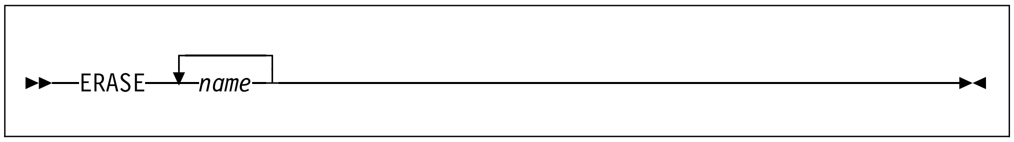
  or

```
BACKWARD 2
```

or

```
BACKWARD MAX
```

- The repeat symbol indicates that an item can be repeated. For example:



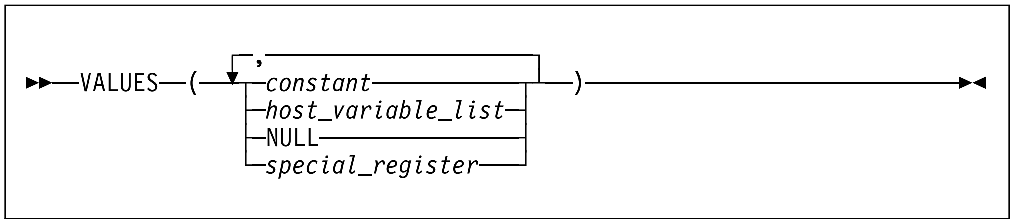
This statement could appear as:

```
ERASE NAME1
```

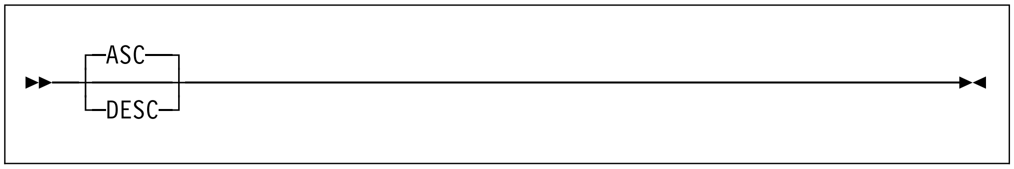
or

```
ERASE NAME1 NAME2
```

A repeat symbol above a stack indicates that you can make more than one choice from the stacked items, or repeat a choice. For example:

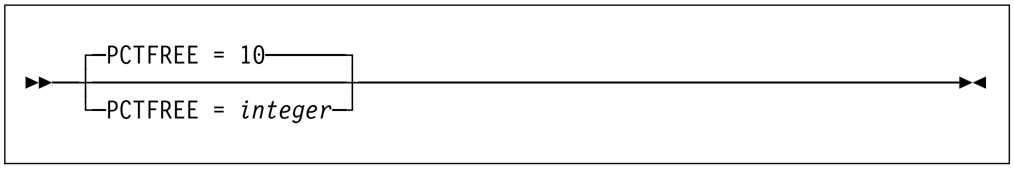


- If an item is above the main line, it represents a default, which means that it will be used if no other item is specified. In the following example, the ASC keyword appears above the line in a stack with DESC. If neither of these values is specified, the command would be processed with option ASC.



- When an optional keyword is followed on the same path by an optional default parameter, the default parameter is assumed if the keyword is not entered. However, if this keyword is entered, one of its associated optional parameters must also be specified.

  elIn the following example, if you enter the optional keyword PCTFREE =, you also have to specify one of its associated optional parameters. If you do not enter PCTFREE =, the database manager will set it to the default value of 10.



- Words that are only used for readability and have no effect on the execution of the statement are shown as a single uppercase default. For example:

```
              ┌─PRIVILEGES─┐
►►──REVOKE ALL──┴────────────┴──────────────────────────────────────►◄
```

Here, specifying either REVOKE ALL or REVOKE ALL PRIVILEGES means the same thing.

- Sometimes a single parameter represents a fragment of syntax that is expanded below. In the following example, **fieldproc_block** is such a fragment and it is expanded following the syntax diagram containing it.

```
►►──┬──────────────────────────────┬──┤ fieldproc_block ├───────────►◄
    └─NOT NULL──┬──────────────┬───┘
                ├─UNIQUE───────┤
                └─PRIMARY KEY──┘
```

```
fieldproc_block:
├──FIELDPROC──program_name──┬─────────────────────────┬──────────────┤
                            │       ┌─,────────┐      │
                            └─(─────▼─constant─┴───)──┘
```

Summary of Changes for DB2 Version 6 Release 1

This is a summary of the technical changes to the DB2 Server for VSE & VM Version 6 Release 1 database management system. All manuals are affected by some or all of the changes discussed here. This summary does not list incompatibilities between releases of the DB2 Server for VSE & VM product; see either the *DB2 Server for VSE & VM SQL Reference*, *DB2 Server for VM System Administration*, or the *DB2 Server for VSE System Administration* manuals for a discussion of incompatibilities. Version 6 Release 1 of the DB2 Server for VSE & VM database management system is intended to run on the Virtual Machine/Enterprise Systems Architecture (VM/ESA®) Version 2 Release 2 or later environment and on the Virtual Storage Extended/Enterprise Systems Architecture (VSE/ESA™) Version 2 Release 2 or later environment.

Enhancements, New Functions, and New Capabilities

DRDA® RUOW Application Requestor for VSE (Online)

DRDA Remote Unit of Work Application Requestor provides read and update capability in one location in a single unit of work.

This support provides CICS/VSE® online application programs with the ability to execute SQL statements to access and manipulate data managed by any remote application server that implements the DRDA architecture. Online application programs that access remote application servers need to be preprocessed to create a bind file and then bound (using CBND) to the remote application server. Online application programs that access a local application server are preprocessed as in previous releases.

See the following DB2 Server for VSE & VM manuals for further information:

- *DB2 Server for VSE System Administration*
- *DB2 Server for VSE & VM SQL Reference*
- *DB2 Server for VSE Database Administration*
- *DB2 Server for VSE Application Programming*
- *DB2 Server for VSE Installation*

Stored Procedures

The ability to use stored procedures provides distributed solutions that let more people access data faster.

A stored procedure is a user-written application program compiled and stored at the server. When the database is running in multiple user mode, local applications or remote DRDA applications can invoke the stored procedure. SQL statements are local to the server and issued by a stored procedure so they do not incur the high network costs of distributed statements. Instead, a single network send and receive operation is used to invoke a series of SQL statements contained in a stored procedure.

See the following DB2 Server for VSE & VM manuals for further information:

- *DB2 Server for VM System Administration*
- *DB2 Server for VM Database Administration*
- *DB2 Server for VSE & VM SQL Reference*
- *DB2 Server for VSE & VM Operation*

TCP/IP Support for DB2 Server for VM

TCP/IP support allows:

- VM applications to use SQLDS-private protocol to connect to VM databases over TCP/IP.
- VM applications to use DRDA protocol to connect to DB2 family databases (and any other database that supports DRDA connections) over TCP/IP.
- non-VM applications to use DRDA-protocol to access VM database over TCP/IP.

TCP/IP support for DB2 Server for VM integrated with the DB2 Server for VM application server means a system easier to configure and maintain.

The database manager will optionally secure TCP/IP connections using any external security manager that supports the RACROUTE interface.

New Code Page and Euro Symbol Code Page Support

The following CCSIDs are now supported:

- 1112: Latvian/Lithuanian
- 1122: Estonian
- 1123: Ukrainian
- 1130: Vietnamese
- 1132: Lao
- 1148: E-International
- 1140: E-English
- 1141: E-German
- 1144: E-Italian
- 1147: E-French

Additional support has been added for conversions from Unicode (UCS-2) to host CCSIDs.

For a complete list of CCSIDs supported refer to the *DB2 Server for VM System Administration* and *DB2 Server for VSE System Administration* manuals.

DataPropagator™ Capture

DataPropagator Capture is part of the DB2 Family of DataPropagator products. DataPropagator Capture is updated for Version 6 Release 1 compatibility.

QMF for VM, QMF for VSE, and QMF for Windows®

IBM Query Management Facility (QMF™) is now an separately priced feature of DB2 Server for VSE & VM. QMF is a tightly integrated, powerful, and reliable tool that performs query and reporting for IBM's DB2 relational database Management System Family. It offers an easy-to-learn, interactive interface. Users with little or no data processing experience can easily retrieve, create, update, insert, or delete data that is stored in DB2.

QMF offers a total solution that includes accessing large amounts of data and sharing central repositories of queries and enterprise reports. It also allows you to implement tightly-controlled, distributed, or client-server solutions. In addition, you can use QMF to publish reports to the World Wide Web that you can view with your favorite web browser.

Using QMF, users can access a wide variety of data sources, including operational or warehouse data from many platforms: DB2 for VSE, VM, OS/390® and Windows. Via IBM Data Joiner, you can access non-relational data, such as IMS™ and VSAM, as well as data from other vendor platforms.

RDS Above the Line

The RDS component will load and execute above the 16 megabyte line. This support frees up approximately 1.5 megabytes of storage below the 16 megabyte line (or approximately 2.5 megabytes, if DRDA is installed) when compared to Version 5 Release 1. No installation or migration changes are required for this support to be used (except for the definition of VM Shared Segments and for users who execute the database server with AMODE(24)). If sufficient storage is available, the RDS component will be automatically loaded above the 16 megabyte line. When using VM Shared Segments, the RDS Segment should be defined above the 16 megabyte line.

VM users who wish to run the database server in 24-bit addressing mode (i.e. use the AMODE(24) parameter) **must** use a virtual storage size no greater than 16 megabytes. See the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* for release to release incompatibility information.

Combining of NLS Feature Installation Tapes with Base Product Installation Tape

All available NLS features for DB2 Server for VSE, DB2 Server for VM, Control Center for VSE and REXX SQL for VM have been combined with the respective base product installation tape. Customers interested in an NLS feature language will no longer need to order an additional NLS feature tape because all NLS languages will be available to all customers. In all cases, the default language as shipped is American English. The installation and migration processes have been changed to allow you to choose the default language. Refer to the *DB2 Server for VM Program Directory*, *DB2 Server for VSE Installation*, *DB2 for VSE Control Center Installation and Operations Guide*, and *DB2 REXX SQL for VM/ESA Installation* for the details of how these changes affect the installation process and how you can choose to have a different default language.

Control Center Feature

DB2 Server for VSE & VM Version 6 Release 1 enhances the new Control Center feature as follows:

For both VM/ESA and VSE/ESA:

- Access to the Query Management Facility (QMF)

For VM/ESA:

- Compatibility with DB2 Server for VM Version 6 Release 1 initialization parameters and operator commands
- Shared File System Support (SFS) in a VM/ESA environment
- CA-DYNAM/T Interface Support in a VM/ESA environment
- Data Restore Incremental Backup Support in a VM/ESA environment

For VSE/ESA:

- Control Center code installation on any library
- Ability to use while viewing a list of tables online
- Ability to create, reorganize, unload, reload, move and copy tables in batch mode
- Ability to update table statistics in batch mode
- Ability to drop tables online

Data Restore Feature

The Data Restore feature provides archiving and recovery functions in addition to those provided in DB2 for VSE & VM. Data Restore is enhanced in Version 6 Release 1 with incremental database archiving support. The support allows you to archive only the areas of the database that have been updated since the last database archive, instead of having to archive the entire database. This can provide significant savings for customers with large databases which are updated infrequently, or where only a small fraction of the database is updated frequently.

DB2 REXX SQL Feature

The DB2 REXX SQL feature provides a REXX interface for VM customers to allow SQL calls to be executed from REXX programs. The DB2 REXX SQL feature is updated for Version 6 Release 1 compatibility.

Reliability, Availability, and Serviceability Improvements

Migration Considerations

Migration is supported from SQL/DS™ Version 3 and DB2 Server for VSE & VM Version 5. Migration from SQL/DS Version 2 Release 2 or earlier releases is not supported. Refer to the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* manual for migration considerations.

Library Enhancements

Some general library enhancements include:

- The following books have been removed from the library:
  - *DB2 Server for VM Operation*
  - *DB2 Server for VSE Operation*
  - *DB2 Server for VM Interactive SQL Guide and Reference*
  - *DB2 Server for VSE Interactive SQL Guide and Reference*
  - *DB2 Server for VM Database Services Utility*
  - *DB2 Server for VSE Database Services Utility*
- The following books have been added to the library:
  - *DB2 Server for VSE & VM Operation*
  - *DB2 Server for VSE & VM Interactive SQL Guide and Reference*
  - *DB2 Server for VSE & VM Database Services Utility*

Refer to the new *DB2 Server for VSE & VM Overview* for a better understanding of the benefits DB2 Server for VSE & VM can provide.

Chapter 1. Getting Started with DB2 RXSQL

This chapter introduces the DB2 RXSQL interface by describing the elements of DB2 RXSQL, and how to access it.

What is DB2 RXSQL?

The DB2 REXX SQL for VM/ESA (DB2 RXSQL) allows REXX programs to access the DB2 Server for VM relational database management system.

SQL, the language used to interface to the database manager, can be imbedded in procedural languages such as assembler, C, COBOL, FORTRAN, and PL/I. DB2 RXSQL extends this support to REXX programs and allows SQL statements to be used in DB2 RXSQL requests contained in REXX programs.

DB2 RXSQL contains an expanded interface that resembles the standard SQL language (except for user descriptors), as outlined in the *SAA Database Level 2 Reference* manual.

Unlike the other host languages mentioned, DB2 RXSQL requests are not preprocessed. Compiling your REXX programs has no effect on the DB2 RXSQL requests contained in them.

DB2 RXSQL supports the use of both Dynamic and Extended Dynamic SQL statements.

DB2 RXSQL performs some initial checking on these statements, transforms them into standard run-time SQL operations, and passes them to the database manager. Results of these SQL operations are returned to your program in REXX variables. This includes both data and resulting status information.

When you are coding DB2 RXSQL operations in a REXX program, you can use REXX variables to do the following:

- Pass input values to SQL statements
- Receive values fetched from the database
- Receive information about the outcome of the SQL statements after they are processed.

DB2 RXSQL consists of two modules:

1. RXSQL, which loads a CMS nucleus extension
2. EXECSQL, which passes control to RXSQL.

DB2 RXSQL also includes a DB2 RXSQL LOADLIB which gets loaded as a CMS nucleus extension.

What You Get with DB2 RXSQL

With DB2 RXSQL, you receive DB2 RXSQL-supplied programs, sample REXX programs, and online HELP information. You can use these along with the DB2 RXSQL requests that come with DB2 RXSQL to create your own applications.

DB2 RXSQL-Supplied Programs

RXCASE	Sets a global variable for the case setting of SQL statements to be used by the RXSELECT and RXSQLEX EXECs.
RXSELECT	Retrieves rows from the database and displays them in a temporary file, S$Q$L S$T$M$T, using XEDIT.
RXSQLANG	Sets global variables for determining languages that the HELP text and messages will be in when displayed using the RXSQLHLP EXEC.
RXSQLEX	Issues an SQL EXECUTE IMMEDIATE statement to execute a valid SQL statement.
RXSQLHLP	Provides HELP information pertaining to DB2 RXSQL return codes, DB2 RXSQL messages, and HELP topics provided by the database manager. The HELP topics provided by the database manager may also be displayed using the ISQL HELP command when using ISQL.
RXSQLOP	Executes a valid SHOW or COUNTER operator command and displays the output in the file S$Q$L O$P, using XEDIT.
RXSQLVL	Displays or stacks the current release level of DB2 RXSQL that is installed.

For further information, refer to Chapter 2, "RXSQL-Supplied Programs" on page 5.

Sample Programs

These programs are examples of how to use DB2 RXSQL statements in your own REXX programs.

EMPCRE	Creates a table and a view that are used by the rest of the sample programs. It also loads data into the table.
EMPSEL	Selects data from the table.
EMPPRP	Creates a package using Extended Dynamic SQL and generates the EMPDCL program.
EMPDCL	Declares a cursor and associates it with a statement in the package created by the EMPPRP program.
EMPSELX	Selects data from the table using Extended Dynamic SQL.
EMPPRPM	Creates a package for updating the table using Extended Dynamic SQL and generates the EMPDCLM program.
EMPDCLM	Declares cursors for statements in the package created by the EMPPRPM program.
EMPUPD	Updates the table interactively.

Refer to Appendix B, “Sample Programs with Examples of RXSQL Requests” on page 127 for more information.

DB2 RXSQL Requests

DB2 RXSQL supports a variety of commands and statements which are listed in Chapter 6, “RXSQL Request Descriptions” on page 61.

If you are unfamiliar with SQL you can read Chapter 3, “Concepts” on page 15 for an introduction to the way that DB2 RXSQL is used within REXX programs.

Online HELP

Use the CMS HELP command to obtain HELP information on these topics:

RXCASE
RXCMDS
RXSELECT
RXSQLANG
RXSQLEX
RXSQLHLP
RXSQLOP
RXSQLVL
RXUSAGE.

For further information, see “HELP RXSQL MENU” on page 13.

How to Access DB2 RXSQL

Before using DB2 RXSQL, ensure that it is installed and that you have access to the DB2 Server for VM database manager.

Link the Appropriate Disks

If DB2 RXSQL and the DB2 Server for VM program are installed on minidisks, access the DB2 Server for VM production minidisk and the DB2 RXSQL production minidisk by using the CP LINK and ACCESS commands.

If DB2 RXSQL and the DB2 Server for VM product are installed on minidisks using the defaults specified in the *DB2 REXX SQL for VM/ESA Installation* manual, then you can access the DB2 Server for VM and DB2 RXSQL production minidisks by typing the following commands:

```
CP LINK SQLMACH 195 195 RR
ACCESS 195 Q
CP LINK SQLMACH 198 198 RR
ACCESS 198 P
```

The name of the database machine (SQLMACH) and the minidisk numbers (195 and 198) may be different if DB2 RXSQL or the database manager are not installed using the defaults. Ask your database administrator about your installation's setup.

If the database manager and DB2 RXSQL are installed using SFS directories, type the following ACCESS commands:

```
ACCESS VMSYS:SQLMACH.SQL.PRODUCTION Q
ACCESS VMSYS:SQLMACH.RXSQL.PRODUCTION P
```

The names of the SFS directories, VMSYS:SQLMACH.SQL.PRODUCTION and VMSYS:SQLMACH.RXSQL.PRODUCTION may be different if DB2 RXSQL is not installed using the defaults.

Connect to the Application Server

If you are accessing the DB2 Server for VM application server for the first time, type the following command, substituting the name of your application server to establish a connection to it.

```
SQLINIT DBNAME(server_name)
```

The application server is now available for use by DB2 RXSQL or other applications that use the DB2 Server for VM relational database manager.

Access the CMS Message Repository

DB2 RXSQL uses the CMS message repository for storing its error messages. Consequently, you must issue the following CMS command before beginning an DB2 RXSQL session:

```
SET LANGUAGE (ADD ELO USER
```

Chapter 2. RXSQL-Supplied Programs

This chapter describes the use of the general-purpose programs supplied with DB2 RXSQL and the DB2 RXSQL online help that is invoked using the CMS HELP command. The programs and online help can be invoked from the command line, or from within your own programs.

There are two different kinds of online help provided with RXSQL. You can look up information about DB2 Server for VM and RXSQL error messages using the RXSQLHLP EXEC. The other way to find help information is by using the CMS HELP command. CMS HELP for RXSQL provides you with information on RXSQL-supplied programs, the syntax of the RXSQL requests and the RXSQL statement equivalents for each SQL statement.

Two of the RXSQL-supplied programs (RXSELECT and RXSQLEX) can be used to execute simple queries or perform simple data manipulations. These operations must be simple because you can not prepare an SQL statement for later execution, or use a cursor.

If you are writing an application that requires more complexity than these programs provide, see Chapter 3, “Concepts” on page 15. You can also refer to Appendix B, “Sample Programs with Examples of RXSQL Requests” on page 127 which illustrates how to use RXSQL requests in REXX programs. Chapter 6, “RXSQL Request Descriptions” on page 61, provides you with a detailed description of each RXSQL request.

RXCASE EXEC

```
►►──EXEC RXCASE──┬─String─┬──────────────────────────────────────────────►◄
                 └─Upper──┘
```

The RXCASE program sets a global variable indicating the case setting to be used by the RXSELECT EXEC and the RXSQLEX EXEC. If RXCASE is `Upper`, the SQL statement is folded to upper case by the RXSELECT EXEC and the RXSQLEX EXEC. If RXCASE is `String`, the SQL statement remains as originally typed.

Note

1. If RXCASE `String` is not issued, your SQL statements will be folded to uppercase.

Examples

The following two examples show two different forms of output from RXSELECT when the two options for RXCASE are used.

Example 1

RXSELECT folds the SQL statement to uppercase.

```
   RXCASE
Upper
   RXSELECT EMPNO, LASTNAME, JOB, SALARY FROM RXEMP WHERE JOB = 'manager'
```

The following output is displayed in the temporary file S$Q$L S$T$M$T.

```
SELECT EMPNO, LASTNAME, JOB, SALARY FROM RXEMP WHERE JOB='MANAGER'
EMPNO  LASTNAME JOB       SALARY
------ -------- -------- --------
002130 SAMS     MANAGER  41700.00
003326 GOODBAR  MANAGER  40360.00
ELO2121I ******** End-of-Data **********
```

Example 2

RXSELECT leaves the SQL statement as typed. No data is selected from the table because the data is stored in uppercase and does not fit the search specification.

```
   RXCASE
String
   RXSELECT EMPNO, LASTNAME, JOB, SALARY FROM RXEMP WHERE JOB = 'manager'
```

The following output is displayed in the temporary file S$Q$L S$T$M$T.

```
Select EMPNO, LASTNAME, JOB, SALARY FROM RXEMP WHERE JOB='manager'
EMPNO LASTNAME JOB SALARY
----- -------- --- ------
ELO2121I ******** End-of-Data **********
```

RXSELECT EXEC

```
►►──EXEC RXSELECT select_statement──────────────────────────────────────────►◄
```

select_statement
Any valid SQL SELECT statement.

Note: The FOR UPDATE OF clause is not supported by the RXSELECT EXEC. If you want to use a Positioned Update or Positioned Delete statement you must write a DB2 RXSQL application.

See the *DB2 Server for VSE & VM SQL Reference* manual for more information about SELECT statements.

RXSELECT retrieves the first 100 rows from the database and displays them, using XEDIT, in a temporary file called S$Q$L S$T$M$T. If there are more than 100 rows in the result, type:

```
MORE
```

or,

```
MORE n
```

where *n* is the number of rows you want to display, from within XEDIT.

While you are in XEDIT, you can query the database again. Type:

```
RXSELECT new_select_statement
```

on the XEDIT command line, where *new_select_statement* is a new SQL SELECT statement. The new query overrides the previous query.

The RXSELECT EXEC folds the entire SELECT statement to uppercase unless you have issued the RXCASE EXEC with the `String` option.

When the RXSELECT EXEC returns data from a GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column, and the DBCS option from the LASTING GLOBALV file for the DB2 Server for VM user is set to `'YES'`, RXSELECT concatenates the DBCS shift-out delimiter in front of the data and appends the DBCS shift-in character following the data.

Notes

1. The following commands are defined as XEDIT synonyms:

   SELECT for RXSELECT
   MORE for RXMORE
   SQLHELP for RXSQLHLP

2. A COMMIT or a ROLLBACK statement is issued from the RXSELECT EXEC before it terminates. This may affect your program if it calls the RXSELECT EXEC during a logical unit of work.

Example

Select the columns EMPNO, LASTNAME, WORKDEPT, JOB, and SALARY from the table RXEMP.

```
RXSELECT EMPNO, LASTNAME, WORKDEPT, JOB, SALARY FROM RXEMP
```

The following output is displayed in the temporary file S$Q$L S$T$M$T.

```
SELECT EMPNO, LASTNAME, WORKDEPT, JOB, SALARY FROM RXEMP
EMPNO  LASTNAME WORKDEPT JOB        SALARY
------ -------- -------- -------- --------
002130 SAMS     B12      MANAGER  41700.00
002300 HEDGLEY  B09      ANALYST  37900.00
001010 LOWRY    D14      ANALYST  38240.00
000990 SCHENKER A07      OPERATOR 30190.00
002020 RAINEY   D11      DESIGNER 32560.00
001840 CORDON   B09      FILEREP  28090.00
002330 FABER    A10      CLERK    27800.00
009236 DOBSON   D08      WRITER   37600.00
002574 MERCIER  A11      WRITER   33400.00
003567 SCHMIDT  C04      CLERK    25790.00
002419 ATWALA   A07      OPERATOR 37940.00
003326 GOODBAR  B12      MANAGER  40360.00
003589 GOULD    D07      WRITER   39250.00
ELO2121I ******** End-of-Data **********
```

RXSQLANG EXEC

```
►►──EXEC RXSQLANG──┬─────────────────────────────────┬──┬─────────────────────────────────┬───────►◄
                   ├─RXlang(language_identifier)─────┤  ├─SQlang(language_identifier)─────┤
                   ├─RXlang(langid)──────────────────┤  ├─SQlang(langid)──────────────────┤
                   └─RXlang( )───────────────────────┘  └─SQlang( )───────────────────────┘
```

RXlang

(*language_identifier***)**
A 40-character string that has the same value as that found in the LANGUAGE column of the SQLDBA.ELOLANGUAGE table.

(*langid***)**
A one-to-five character short form for the language (for example, AMENG or KANJI) that has the same value as found in the LANGID column of the SQLDBA.ELOLANGUAGE table.

( )
If neither *language_identifier* nor *langid* is supplied, then RXSQLANG erases your default RXlang variable from the LASTING GLOBALV file.

SQlang

(*language_identifier***)**
A 40-character string that has the same value as that found in the LANGUAGE column of the SQLDBA.SYSLANGUAGE catalog table.

(*langid***)**
A one-to-five character short form for the language (for example, AMENG or KANJI) that has the same value as found in the LANGID column of the SQLDBA.SYSLANGUAGE catalog table.

( )
If neither *language_identifier* nor *langid* is supplied, then RXSQLANG erases your default SQlang variable from the LASTING GLOBALV file.

RXSQLANG provides a means to override the system defaults for the languages in which HELP information is displayed by DB2 RXSQL. During installation of both the database manager and DB2 RXSQL, a system default language is established. Using RXSQLANG you can specify your own default language for either or both DB2 RXSQL and DB2 Server for VM HELP information. The RXSQLHLP EXEC uses the default languages specified to display HELP information.

RXSQLANG verifies that the parameters you supply specify national languages that have been installed on your system. To determine which DB2 RXSQL national languages have been installed check the SQLDBA.ELOLANGUAGE table. This table is provided by DB2 RXSQL. To check which DB2 Server for VM languages have been installed check the SQLDBA.SYSLANGUAGE catalog table.

The national languages that you choose do not affect any other users or change the system defaults. The selection remains in effect until you issue RXSQLANG again with new parameters.

Note

1. RXSQLANG stores variables in the CMS global variables file (LASTING GLOBALV).

Examples

Example 1

Set the default language for RXSQL HELP information to American English using the *langid*.

```
RXSQLANG RX(AMENG)
```

Example 2

Set the default language for DB2 Server for VM HELP information to American English using the *language_identifier*.

```
RXSQLANG SQ(ENGLISH)
```

RXSQLEX EXEC

```
►►──EXEC RXSQLEX statement────────────────────────────────────────────────►◄
```

statement
: A valid SQL statement that can be processed by an SQL EXECUTE IMMEDIATE statement. See the *DB2 Server for VSE & VM SQL Reference* manual for a list of valid statements.

DB2 RXSQL passes *statement* to the database manager where it is processed as an EXECUTE IMMEDIATE statement.

If no error occurs after the statement is processed, a COMMIT WORK statement is executed. If an error does occur a ROLLBACK statement is issued.

If the SQL statement is an INSERT, an UPDATE, or a DELETE, and it affects more than one row in the database, the following prompt appears:

```
Enter ROLLBACK or CANCEL to cancel the changes.
```

Cancel your changes by typing `ROLLBACK` or `CANCEL`, or accept your changes by pressing enter.

The SQL statement is folded to uppercase by DB2 RXSQL unless you select the `String` option for RXCASE.

Example

Grant select authority on the RXEMP table to all users.

```
RXSQLEX GRANT SELECT ON RXEMP TO PUBLIC
```

RXSQLHLP EXEC

```
►►──EXEC RXSQLHLP topic──────────────────────────────────────────────────►◄
```

topic
One of the following:

- A DB2 RXSQL return code
- A DB2 RXSQL message identifier of the format:

  `ELOnnnns`

  where *nnnn* is the message identifier number and *s* is the error severity
- An SQLCODE returned from the database manager
- `CONTENTS` lists all of the valid DB2 Server for VM topics that you can enter.
- Any DB2 Server for VM HELP topic that may be displayed using the ISQL HELP command. Refer to the *DB2 Server for VSE & VM Interactive SQL Guide and Reference* manual for more information.

RXSQLHLP can display both DB2 RXSQL and DB2 Server for VM HELP information if they are installed on your system.

The RXSQLHLP program uses the default language installed on your system, or the one you specified using the RXSQLANG EXEC. See “RXSQLANG EXEC” on page 8 for more information.

After the results are displayed in an XEDIT file, you can view additional topics by typing:

```
SQLHELP newtopic
```

on the XEDIT command line, where *newtopic* is any HELP topic as described above.

Notes

1. The following commands are defined as XEDIT synonyms:

   SELECT for RXSELECT,
   MORE for RXMORE
   SQLHELP for RXSQLHLP.
2. If the help tables for either DB2 Server for VM or DB2 RXSQL are not found, then RXSQLHLP returns an error condition.

Example

Display the HELP information for the topic '+100'.

```
RXSQLHLP +100
```

The following output is displayed in the temporary file S$Q$L H$E$L$P.

```
============================================================
DB2 RXSQL HELP +100
============================================================
ELO2112I DB2 RXSQL HELP text is not available
for topic '+100' for language S001.
============================================================
DB2 for VM HELP '+100'
============================================================
TOPIC NAME:  +100

+100       There are no (or no more) rows that satisfy the
          condition.

EXPLANATION:  For a query that uses a cursor, the cursor is
empty or all rows have been selected.  For a query that does
not use a cursor, no row was found that satisfied the WHERE
condition.  An INSERT via SELECT statement may return this
SQLCODE if the SELECT statement does not retrieve any rows.

SQLSTATE 02000.

SYSTEM ACTION:  Normal completion.

USER RESPONSE:  Take suitable action based on the SQLCODE
descriptive text.
```

RXSQLOP EXEC

```
►►──EXEC RXSQLOP operator_command──────────────────────────────────────►◄
```

operator_command
One of the following:

- A DB2 Server for VM SHOW operator command
- A DB2 Server for VM COUNTER operator command

SHOW and COUNTER monitor the DB2 Server for VM system. See the *DB2 Server for VSE & VM Operation* manual for more information on operator commands.

Using XEDIT, RXSQLOP displays the results in the file S$Q$L O$P.

To process additional commands, type:

```
SQLOP newcommand
```

on the XEDIT command line, where *newcommand* is a DB2 Server for VM SHOW or COUNTER operator command.

This program will issue an error message if invoked from within a logical unit of work because the DB2 Server for VM database manager does not allow operator commands to be issued from within a logical unit of work.

Note

1. Operator commands cannot be issued when using the DRDA protocol.

Example

Issue a DB2 Server for VM operator command.

```
RXSQLOP COUNTER *
```

Output similar to the following is displayed in the temporary file S$Q$L O$P.

```
===========================================================
SQLOP COUNTER *
===========================================================
Counter values at  DATE='05-30-91'  TIME='11:18:01'
Calls to RDS                   RDSCALL :  1219
Calls to DBSS                  DBSSCALL:  6862
LUWs started                   BEGINLUW:  268
LUWs rolled back               ROLLBACK:  69
System checkpoints taken       CHKPOINT:  1
Maximum locks exceeded         LOCKLMT :  0
Lock escalations               ESCALATE:  0
Waits for lock                 WAITLOCK:  0
Deadlocks detected             DEADLCK :  0
Looks in page buffer           LPAGBUFF:  13576
DBSPACE page reads             PAGEREAD:  3117
DBSPACE page writes            PAGWRITE:  20
Looks in directory buffer      LDIRBUFF:  1092
Directory block reads          DIRREAD :  414
Directory block writes         DIRWRITE:  99
Log page reads                 LOGREAD :  3
Log page writes                LOGWRITE:  13
Total DASD reads               DASDREAD:  3534
Total DASD writes              DASDWRIT:  132
Total DASD I/O                 DASDIO  :  3666
ARI0065I Operator command processing is complete.
```

RXSQLVL EXEC

```
►►──EXEC RXSQLVL──┬────────┬──────────────────────────────────────────────────────►◄
                  ├─RELMOD─┤
                  └─LIFO───┘
```

The RXSQLVL program may be executed to determine the release level of DB2 RXSQL that is installed.

RELMOD
LIFO
Places an entry on the program stack in the format `V R M` where V is the version, R is the release and M is the modification. Both parameters are equivalent.

If you do not specify a parameter, a message is displayed on the console indicating the version, release, and modification level of the installed DB2 RXSQL system.

Note

1. The RXSQLVL program should only be called with the RELMOD or LIFO parameters from another REXX program.

Example

Determine the release level of RXSQL that is installed.

```
RXSQLVL
```

The following output is displayed on the screen.

```
ELO2102I *** DB2 RXSQL Version 5 Release 1 Modification 0 ***
```

HELP RXSQL MENU

This is not a program, but online HELP provided by DB2 RXSQL that can be displayed using the CMS HELP command. It is invoked from the command line like the DB2 RXSQL-supplied programs. The syntax is as follows:

```
►►──HELP RXSQL MENU──────────────────────────────────────────────────────►◄
```

HELP RXSQL MENU displays a HELP panel with all the DB2 RXSQL HELP topics. You can find out

- how to use the DB2 RXSQL-supplied programs
- syntax of all the DB2 RXSQL operations
- the DB2 RXSQL statements that can be used for each SQL statement.

The topics you can choose from are as follows:

RXCASE
Information for invoking the RXCASE program

RXCMDS
Syntax for all the DB2 RXSQL operations

RXSELECT
Information on invoking the RXSELECT program

RXSQLANG
Information on invoking the RXSQLANG program

RXSQLEX
Information on invoking the RXSQLEX program

RXSQLHLP
Information on invoking the RXSQLHLP program

RXSQLOP
Information on invoking the RXSQLOP program

RXSQLVL
Information on invoking the RXSQLVL program

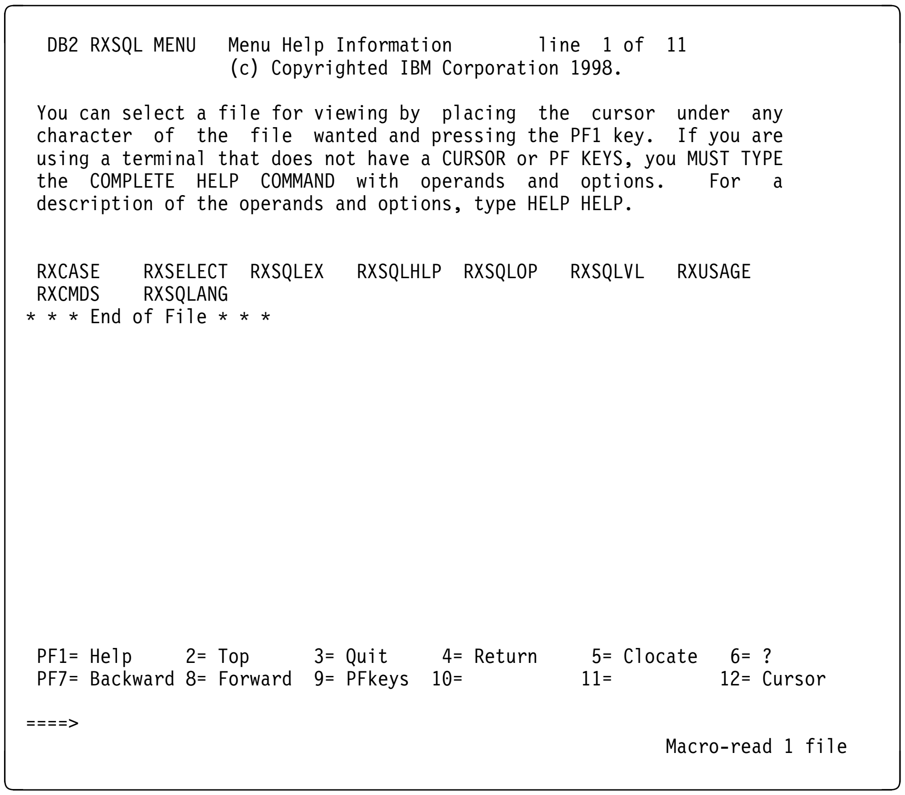
RXUSAGE
Information about which DB2 RXSQL statements can be used for each SQL statement.

Example

Display the RXSQL help menu.

```
HELP RXSQL MENU
```

The menu will look like this:



```
  DB2 RXSQL MENU   Menu Help Information       line  1 of  11
                   (c) Copyrighted IBM Corporation 1998.

 You can select a file for viewing by  placing  the  cursor  under  any
 character  of  the  file  wanted and pressing the PF1 key.  If you are
 using a terminal that does not have a CURSOR or PF KEYS, you MUST TYPE
 the  COMPLETE  HELP  COMMAND  with  operands  and  options.    For   a
 description of the operands and options, type HELP HELP.


 RXCASE    RXSELECT  RXSQLEX   RXSQLHLP  RXSQLOP   RXSQLVL   RXUSAGE
 RXCMDS    RXSQLANG
* * * End of File * * *


















 PF1= Help     2= Top      3= Quit     4= Return     5= Clocate   6= ?
 PF7= Backward 8= Forward  9= PFkeys  10=          11=          12= Cursor

====>
                                                           Macro-read 1 file
```

Chapter 3. Concepts

This chapter introduces some SQL and RXSQL concepts along with some DB2 RXSQL requests and the way that these requests work in your REXX programs. Also included in this chapter are illustrations on how to use Dynamic and Extended Dynamic SQL. A more detailed discussion on how to use Dynamic and Extended Dynamic statements in RXSQL is in the following chapter.

The concepts section is divided into two sections: SQL Concepts and RXSQL Concepts. The SQL concepts sections introduces some basic SQL concepts that are needed to use SQL in any programming language, while the RXSQL concepts section introduces some terms and concepts which are specific to RXSQL.

SQL Concepts

This section should be read along with the “Concepts” chapter in the *DB2 Server for VSE & VM SQL Reference* manual.

To understand this section you should be familiar with SQL queries and the SQL statements that can be issued interactively.

When you are using Dynamic SQL or Extended Dynamic SQL to access data, you will be using one of two types of RXSQL statement sequences, for cursor statements or for non-cursor statements.

Cursors

A *cursor* is a pointer to a row in an active set. An active set is composed of columns and rows of one or more base tables that the database manager selects (a result table) or generates (a put block) based on information in a SELECT statement or an INSERT statement respectively. In RXSQL, a cursor is defined by preparing a SELECT or INSERT statement, and optionally declaring it to give it a *cursor_name*. The cursor is then referenced in subsequent RXSQL statements (OPEN, FETCH or PUT, CLOSE) by the *cursor_name* if it was declared, or by its *prepare_name* if it was not declared.

Note: In other host languages supported by the database manager a cursor must be declared before it can be referenced on subsequent statements.

There are two types of cursors. If your program is retrieving data, the cursor is called a *query_cursor* because the active set or result table is defined by a SELECT statement. If your program is inserting data into a table, the cursor is called an *insert_cursor* because the active set or put block is defined by an INSERT statement.

When a cursor is opened, it is pointing to the top of the active set. Your program must open the cursor by issuing the OPEN statement, and then advance it row by row by issuing FETCH for a *query_cursor* or PUT for an *insert_cursor*. Generally, your program continues retrieving or inserting rows until the last row has been retrieved or all the data has been inserted. Then your program closes the cursor and commits the changes, if any.

You can also update or delete data with a cursor using a Positioned UPDATE or Positioned DELETE statement.

Positioned UPDATE or Positioned DELETE Statements

The Positioned UPDATE or Positioned DELETE statement is used to update or delete a row to which a cursor is currently pointing. This is different from the Searched UPDATE or Searched DELETE where each row that matches the search condition is updated or deleted.

Positioned UPDATE or Positioned DELETE statements use a cursor while Searched UPDATE or Searched DELETE statements do not require a cursor.

To code a Positioned UPDATE or Positioned DELETE statement, your program must first define a *query_cursor* where the SELECT statement has a FOR UPDATE OF *column_names* clause. The cursor is then used to retrieve each row of data using the FETCH statement. If the retrieved row is to be updated or deleted your program issues an UPDATE or DELETE statement with the WHERE CURRENT OF *cursor_name* clause.

Note: Coding the Positioned DELETE operation is different in other host languages supported by the database manager. See the *DB2 Server for VSE & VM SQL Reference* manual for more information on the Positioned DELETE operation.

Cursor and Non-Cursor SQL Statements

Some database operations require a cursor while others do not. The sequence needed for executing SQL statements with cursor operations is different from the sequence needed for non-cursor operations. The SQL statements that always require a cursor in RXSQL include OPEN, SELECT, Positioned UPDATE, Positioned DELETE, and CLOSE. The INSERT statement can be used as a non-cursor statement.

Note: This does not apply to other host languages that support the SELECT INTO statement, which does not require a cursor.

However, if many rows are to be inserted, it is more efficient to use an *insert_cursor* so the rows will be inserted in blocks rather than one row at a time.

The SQL statements which do not require a cursor in RXSQL include the following:

ACQUIRE DBSPACE	ALTER TABLE	ALTER DBSPACE
COMMIT	COMMENT ON	CONNECT
CREATE INDEX	CREATE SYNONYM	CREATE PACKAGE
CREATE TABLE	CREATE VIEW	Searched DELETE
DROP	EXPLAIN	GRANT
INSERT	LABEL ON	LOCK DBSPACE
LOCK TABLE	REVOKE	ROLLBACK
Searched UPDATE	UPDATE STATISTICS	

Blocking

Blocking is the process of retrieving or inserting rows of data in groups rather than one row at a time. If there are many rows to be retrieved or inserted, blocking usually improves performance. However, you should be aware that problem determination is affected by blocking. If you are inserting rows in blocks, an error condition is not detected for a PUT until the block is transmitted to the database manager. This occurs when a block is full or when CLOSE is invoked. To determine the row being inserted when the error was encountered, you must analyze the SQLCA variables as defined in the *DB2 Server for VSE & VM SQL Reference*.

Dynamic FETCH or PUT statements retrieve or insert data in blocks because blocking is the default for RXSQL. However, blocking is turned off when:

- You use the FOR UPDATE OF *cursor_name* clause in the SELECT statement
- You use the INSERT statement with any of the non-cursor sequences discussed in the following sections.
- The prepared INSERT or SELECT statement includes a LONG VARCHAR or LONG VARGRAPHIC column to be transmitted

If you wish to use FETCH or PUT without blocking, you must use DB2 RXSQL Extended Dynamic statements to create a package with the NOBLOCK option.

Illustrations of Using Dynamic SQL in RXSQL

Background

The default installation procedure for DB2 RXSQL includes installing a package for DB2 RXSQL to use. This package contains forty empty sections to be used by DB2 RXSQL when your program executes dynamic SQL statements. These sections are referenced by DB2 RXSQL with *statement_names* S1, S2, S40 and their associated *cursor_names* C1, C2, ..., C40.

When your program prepares a dynamic statement, DB2 RXSQL uses an available section with its corresponding *statement_name* Sn in the RXSQL package. DB2 RXSQL maps the *statement_name* which your program defines to Sn. If your dynamic statement involves a cursor operation, DB2 RXSQL maps your cursor to the cursor Cn associated with statement Sn in the DB2 RXSQL package. The *statement_name* and *cursor_name* which you define in your program are known to RXSQL, but these are not passed to the database manager. When you invoke a DB2 RXSQL request referencing the dynamic statement previously defined, DB2 RXSQL passes the request to the database manager referring only to the statement Sn or its associated cursor Cn.

The following diagrams illustrate how to use Dynamic SQL. They do not contain the complete programming syntax, but are intended to illustrate the statements needed in your program, and how RXSQL processes them to manipulate data stored by the database manager. It is assumed that all of these examples belong to one program and are executed in the sequence that they are illustrated.

The examples that are inline with the text illustrate how to code RXSQL requests, but they are not complete. For example, they do not illustrate error handling. For a complete example of how to code RXSQL applications see Appendix B, “Sample Programs with Examples of RXSQL Requests” on page 127.

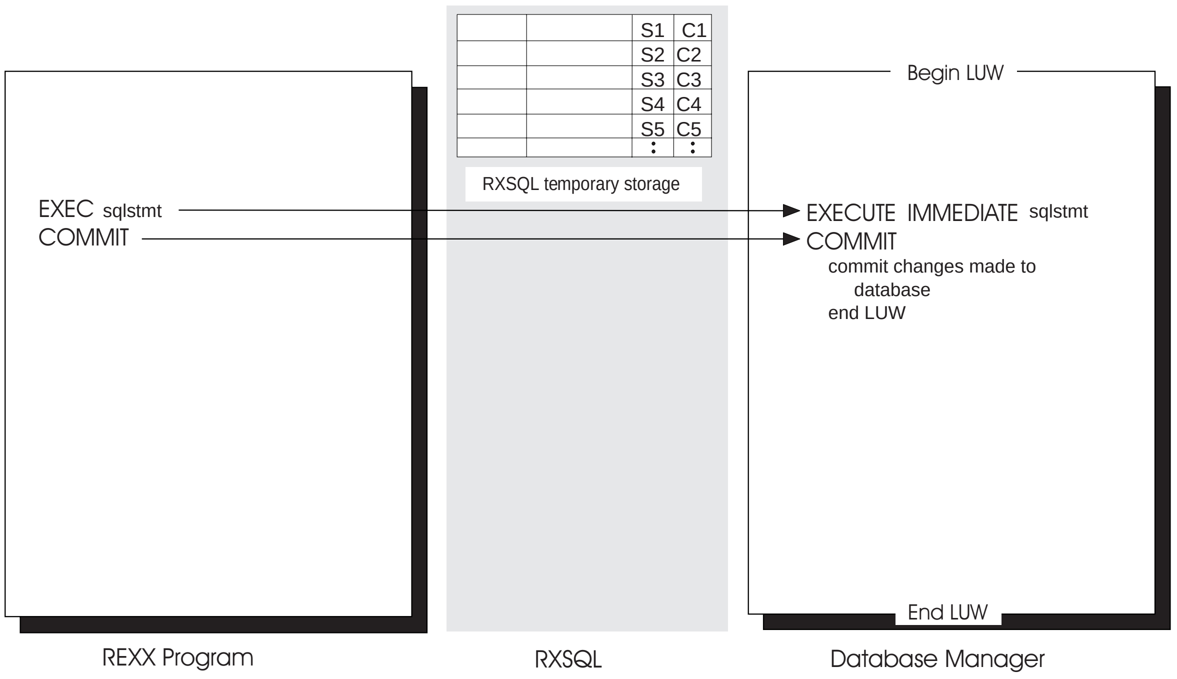


Figure 1. EXEC

RXSQL passes the SQL statement in the EXEC (or EXECUTE IMMEDIATE) statement directly to the database manager.

Rexx_host_variables are not allowed in SQL statements executed by the EXEC statement, but they can be used on the CALL or EXECUTE statement.

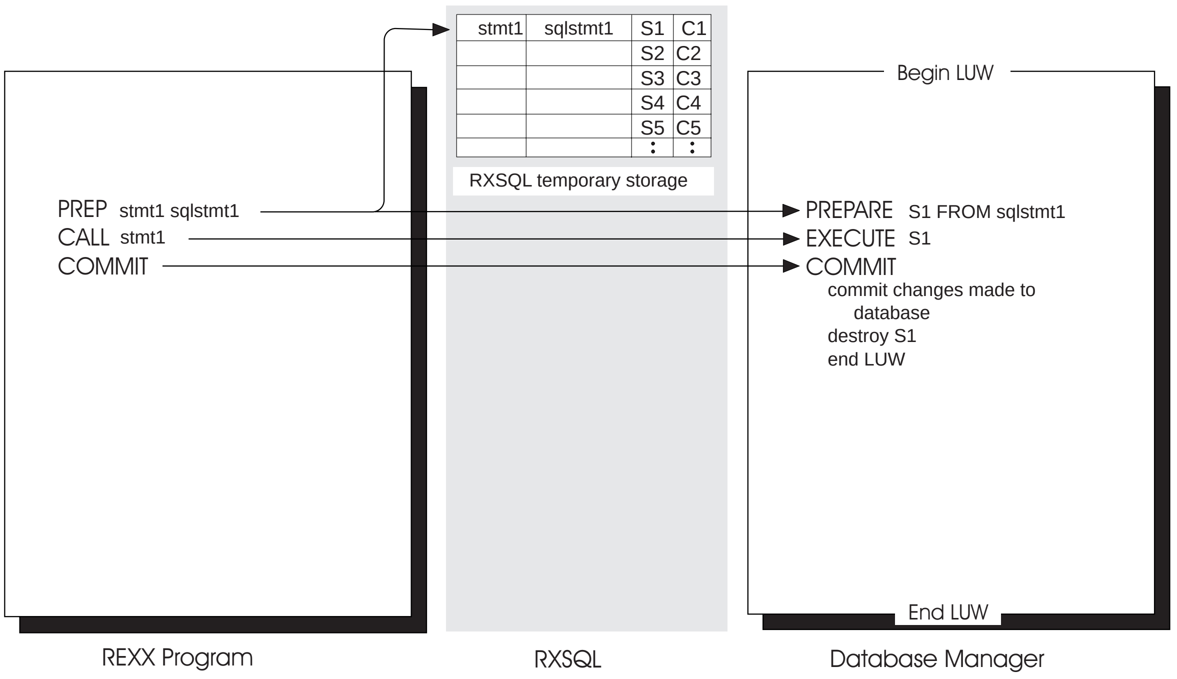


Figure 2. CALL

RXSQL stores the *statement_name* stmt1 and the statement value sqlstmt1 in a temporary storage area. Prepared statements can be opened or called until program control is returned to CMS, or until your program issues a PURGE command.

When DB2 RXSQL issues a PREPARE statement to the database manager, this statement remains active in the database manager for the duration of the LUW only. However, the prepared statement remains in DB2 RXSQL temporary storage.

If the SQL statement has *variable_names*, their values are passed to the database manager when the CALL statement is executed.

The following example illustrates a RXSQL CALL statement without any variables:

```
/*                                                          */
/*  For all of these examples assume that A_TABLE exists and    */
/*  has 5 character type columns                                 */
/*                                                          */
"RXSQL PREP stmt1 INSERT INTO A_TABLE",
      "VALUES('HEATHER','L','DOBSON','T01','WRITER')"
'RXSQL CALL   stmt1'
'RXSQL COMMIT'
'RXSQL PURGE  stmt1'
```

The previous example will insert only one row into a table. If *variable_names* are coded in the INSERT statement, many rows can be inserted into the database using the same statement. The following example illustrates this:

```
/*                                                          */
/*  Assume that there is a file containing all the input called */
/*  DEPT FILE                                                */
Do forever
  'EXECIO 1 DISKR DEPT FILE * (LIFO '
  parse upper pull fname mid lname department job .
  If fname = '' then leave
  fname = "'"fname"'"  /* This ensures RXSQL will know that the */
  mid   = "'"mid"'"    /* data type is character               */
  lname = "'"lname"'"
  department = "'"department"'"
  job   = "'"job"'"
  'RXSQL PREP stmt1 INSERT INTO A_TABLE',
      'VALUES(:fname, :mid, :lname, :department, :job )'
  'RXSQL CALL   stmt1'
  'RXSQL COMMIT'
  'RXSQL PURGE  stmt1'
End
  'FINIS DEPT FILE *'
```

The previous example will work, but the performance will not be very good because a PREP statement is executed with each iteration of the loop. Performance would be much better if the PREP statement was executed only once, and host variables were used to substitute values into the table using the CALL statement. The following example illustrates this:

```
/*                                                                   */
/*   Assume that there is a file containing all the input called */
/*   DEPT FILE                                                       */
/*                                                                   */
insert_data= 'INSERT INTO A_TABLE VALUES (',
  ':fname,:mid,:lname,:department,:job )'
'RXSQL PREP stmt1' insert_data
Do forever
  'EXECIO 1 DISKR DEPT FILE * (LIFO '
  parse upper pull fname mid lname department job .
  If fname = '' then leave
  fname = "'"fname"'"   /* this ensure RXSQL will know that the */
  mid = "'"mid"'"         /* data type is character             */
  lname = "'"lname"'"
  department = "'"department"'"
  job = "'"job"'"
  'RXSQL CALL stmt1'
End
'FINIS DEPT FILE *'
'RXSQL COMMIT'
'RXSQL PURGE stmt1'
```

With each iteration of the loop new values are retrieved from the input file DEPARTMENT FILE and passed to the database manager in the CALL statement. The PREP and COMMIT statements do not have to to be executed with each iteration of the loop making the program run much more efficiently.

Another point to note is that the PURGE statement is executed the same number of times the PREP statement is executed in all the examples to ensure that the RXSQL temporary storage area does not get filled.

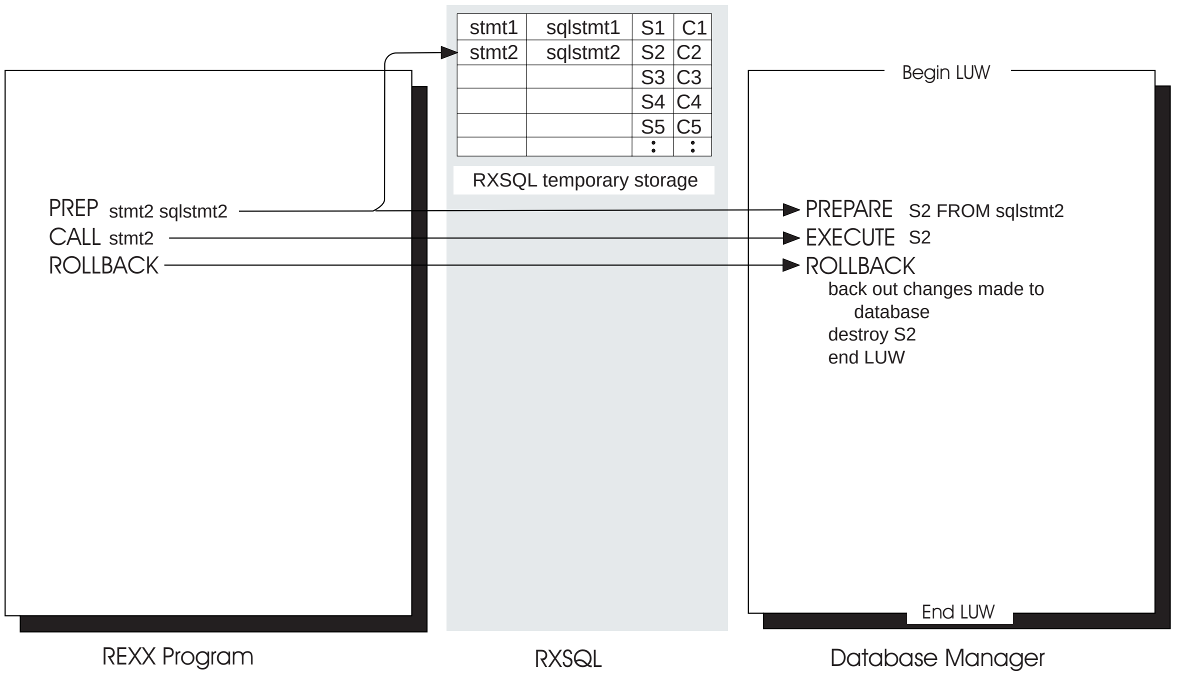


Figure 3. ROLLBACK

A ROLLBACK statement will back out all uncommitted changes. Note that this is opposite to a COMMIT statement which commits all changes. Also note that the prepared statement remains in RXSQL temporary storage even though the work done in the LUW has been rolled back.

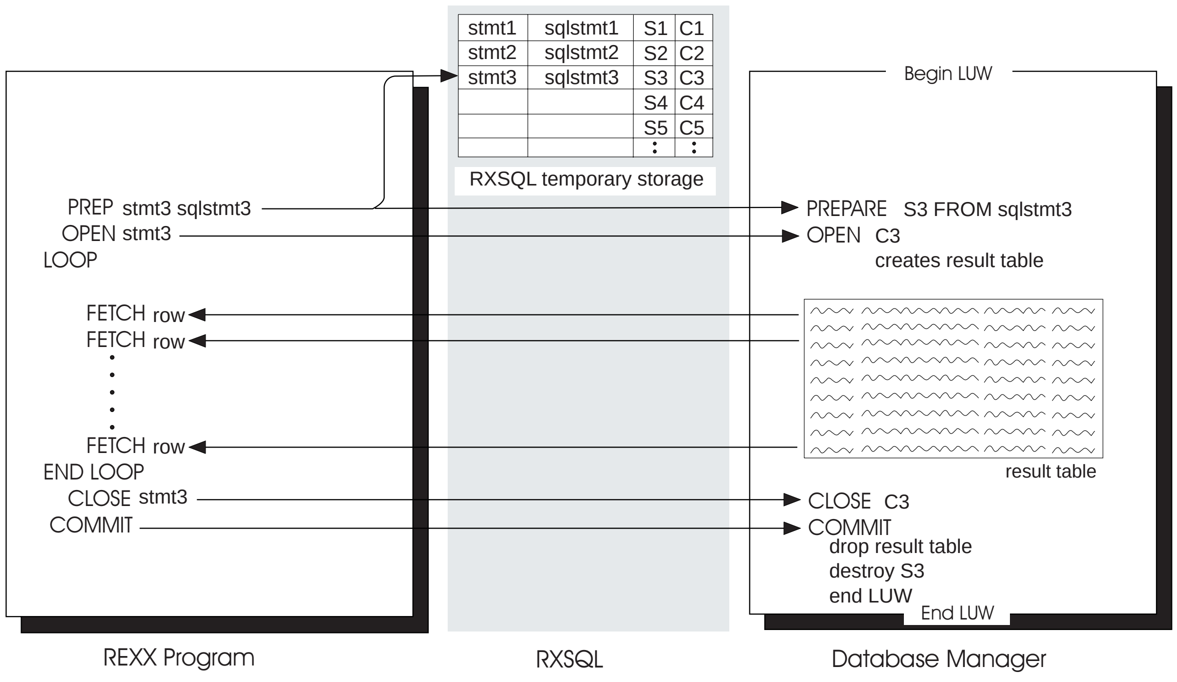


Figure 4. FETCH

In this example, `sqlstmt3` is a SELECT statement. RXSQL issues an OPEN statement to the database manager when an OPEN statement is issued in a program. RXSQL associates the cursor `C3` with the statement named `stmt3`.

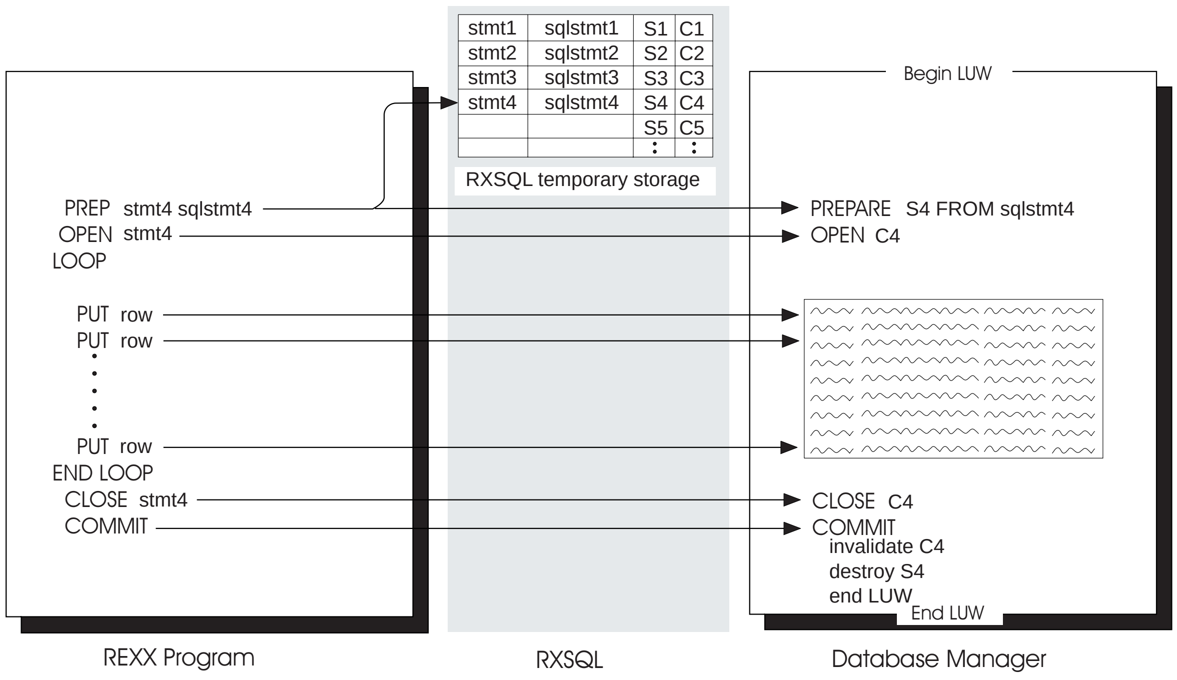


Figure 5. PUT

The prepared statement is an INSERT statement. When the program issues an OPEN statement, an insert-cursor is prepared for block input. Even though the program passes one row at a time to the database manager using the PUT statement, rows are inserted into the table in blocks. This is more efficient than inserting one row at a time into a table.

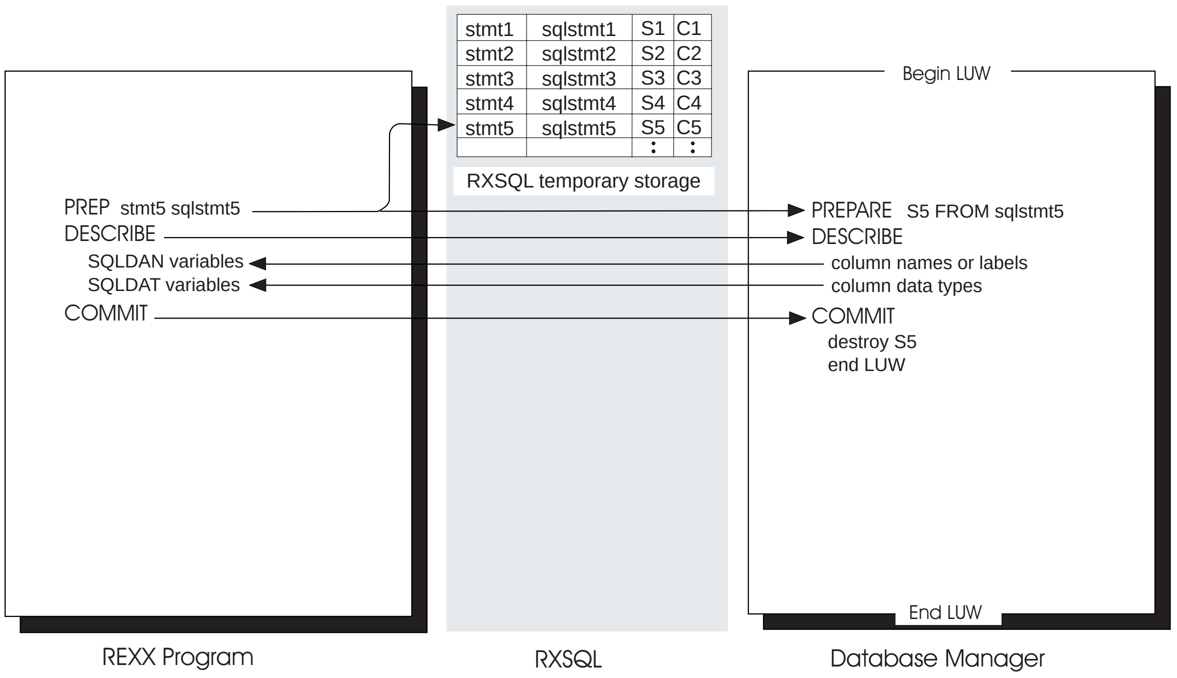


Figure 6. DESCRIBE

The prepared statement is a SELECT statement. When the program issues a DESCRIBE request, DB2 RXSQL returns information about the columns to be fetched into REXX stem variables. This information includes column names and data types.

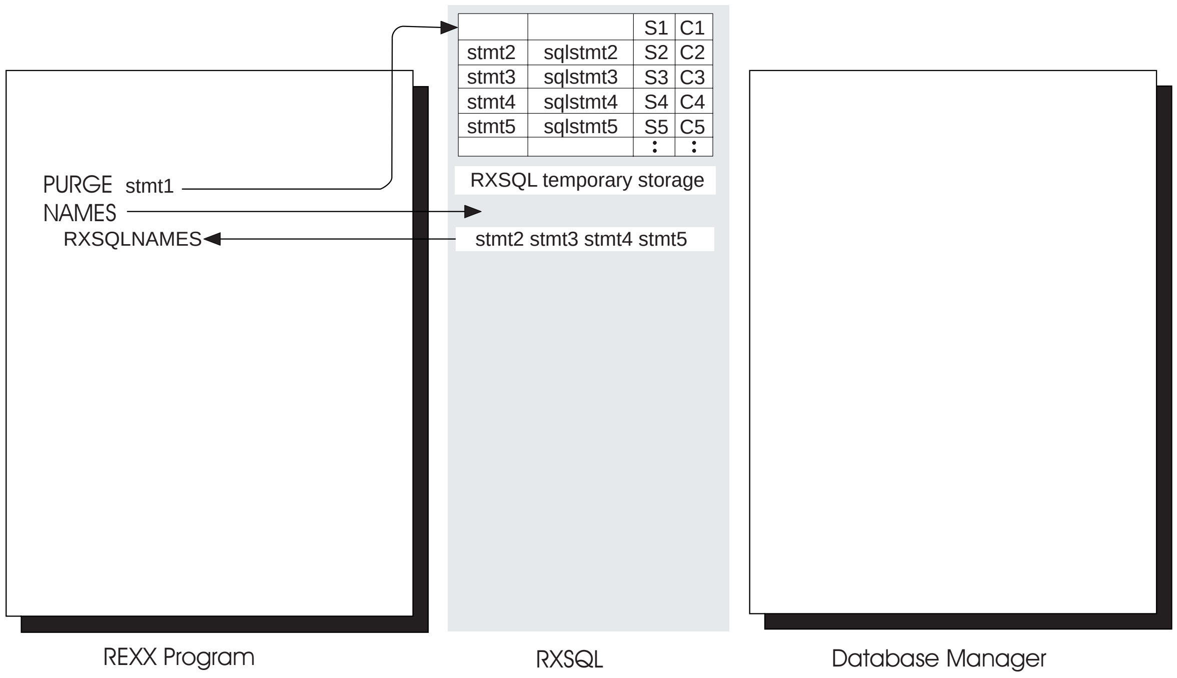


Figure 7. PURGE

Between the time that RXSQL is invoked and control is returned to CMS, RXSQL allows 40 statements to be prepared at one time. If your program tries to prepare more than 40 statements, DB2 RXSQL will return an error indicating that you have tried to prepare more than the allowed number of statements. For this reason, you may want to use the RXSQL PURGE command to maintain your prepared statements. PURGE does not pass a COMMIT statement to the database manager but, if there is an open associated cursor, DB2 RXSQL issues a CLOSE statement to the database manager to close it.

See Appendix B, “Sample Programs with Examples of RXSQL Requests” on page 127 for a detailed illustration of Dynamic SQL. See Chapter 6, “RXSQL Request Descriptions” on page 61 for a detailed description of RXSQL statements and commands.

Illustrations of Extended Dynamic SQL in RXSQL

The following diagrams illustrate how to use Extended Dynamic statements. It is assumed that the examples are executed in the sequence that they are illustrated and Figure 9 on page 24 to Figure 13 on page 26 are executed within one program.

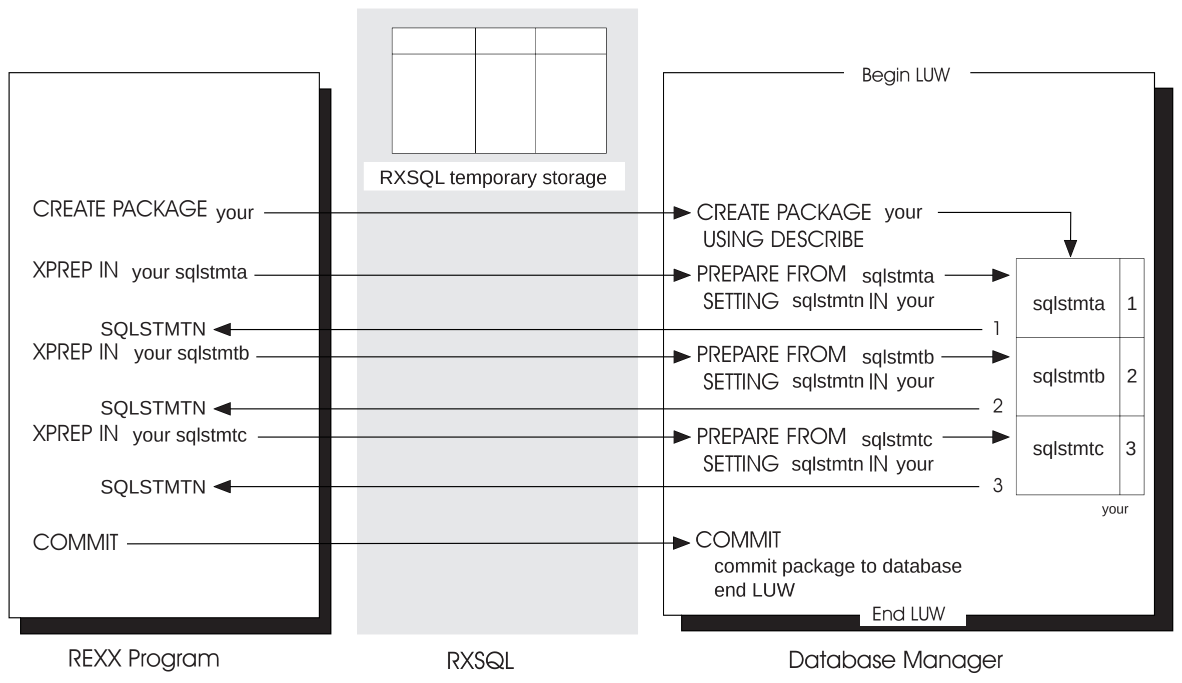


Figure 8. XPREP

Once the COMMIT is issued, the package `your` is stored by the database manger with three statements.

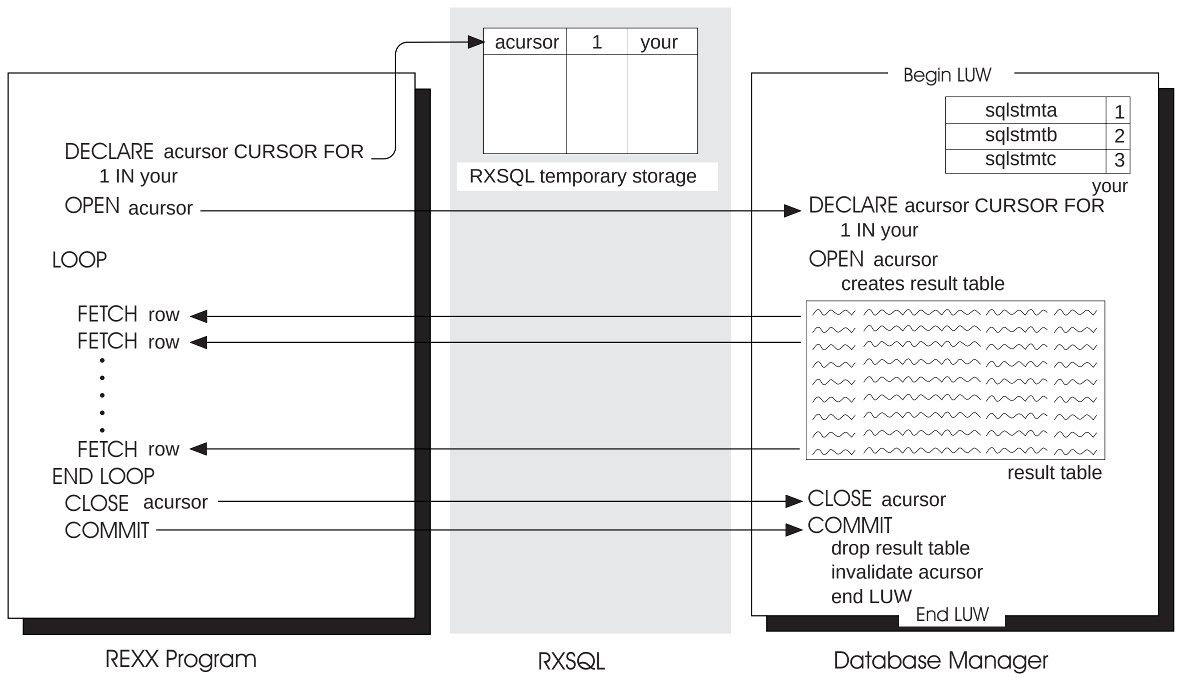


Figure 9. FETCH

The sequence of statements used in Extended Dynamic SQL is very similar to the sequence used in Dynamic SQL. However, note that Extended Dynamic statements have a different syntax to reference sections in a package.

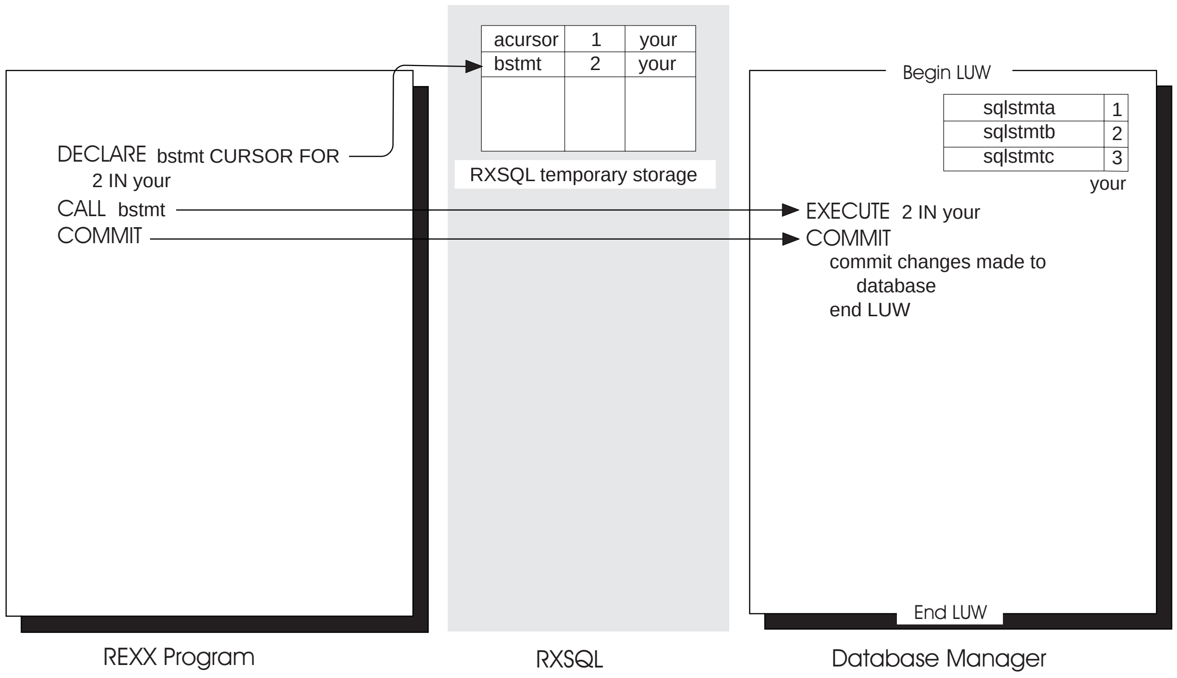


Figure 10. CALL

In this example, the Extended DECLARE defines a statement name `bstmt` for section 2 in `your` package. This statement name is subsequently referenced in the Extended CALL statement to execute the statement.

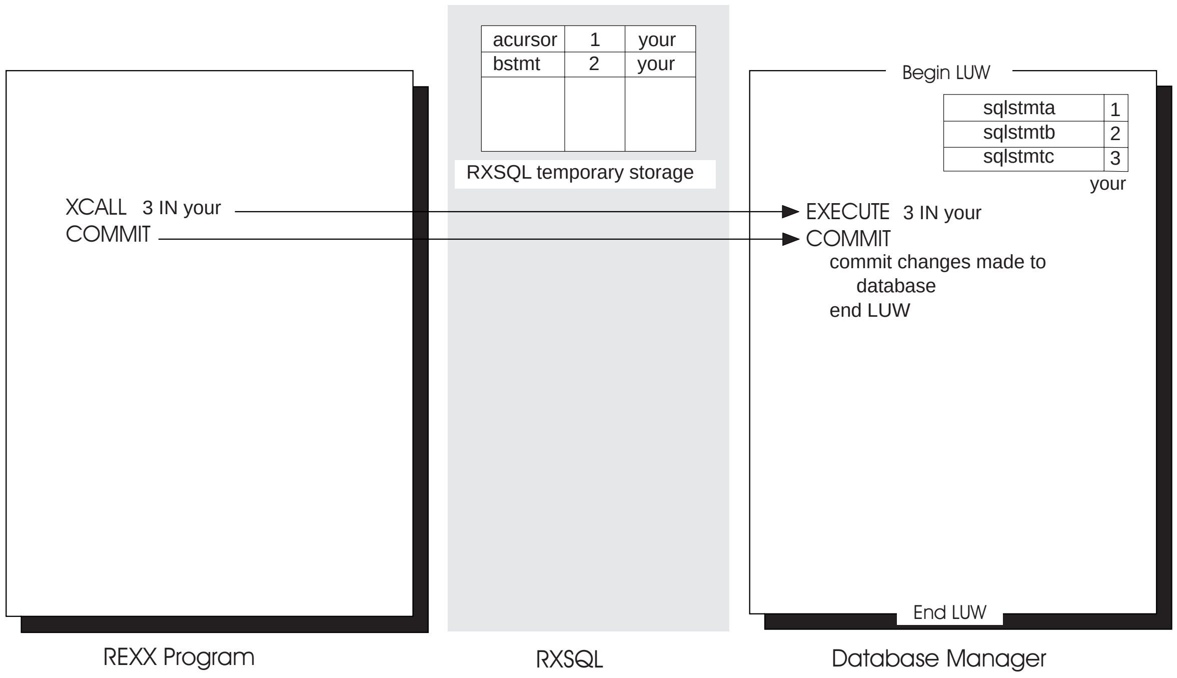


Figure 11. XCALL

Extended EXECUTE and XCALL invoke a statement in a package directly. However, you can not use *rexx_host_variables* in your SQL statement when using the XCALL statement. Use the Extended EXECUTE statement, or the Extended

DECLARE and Extended CALL statements if you want to use *rexx_host_variables* in Extended Dynamic SQL.

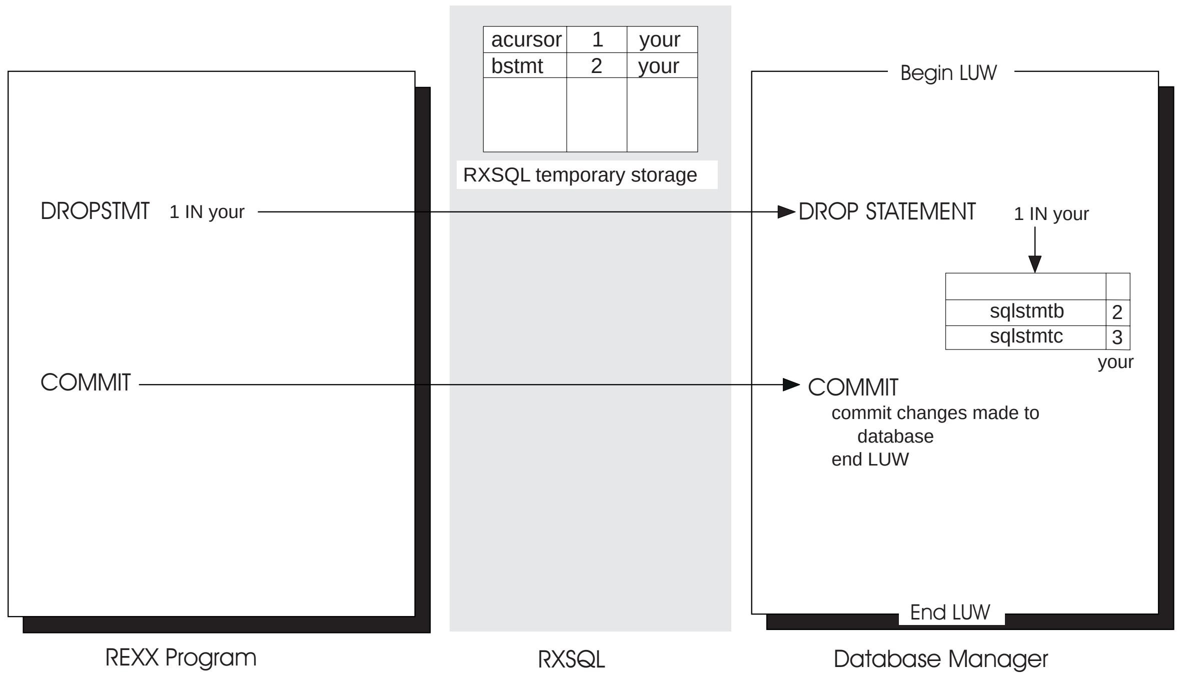


Figure 12. DROPSTMT

The DROPSTMT statement deletes statements from a package, but it does not remove a package from the database. The statement DROP PACKAGE removes a package from the database.

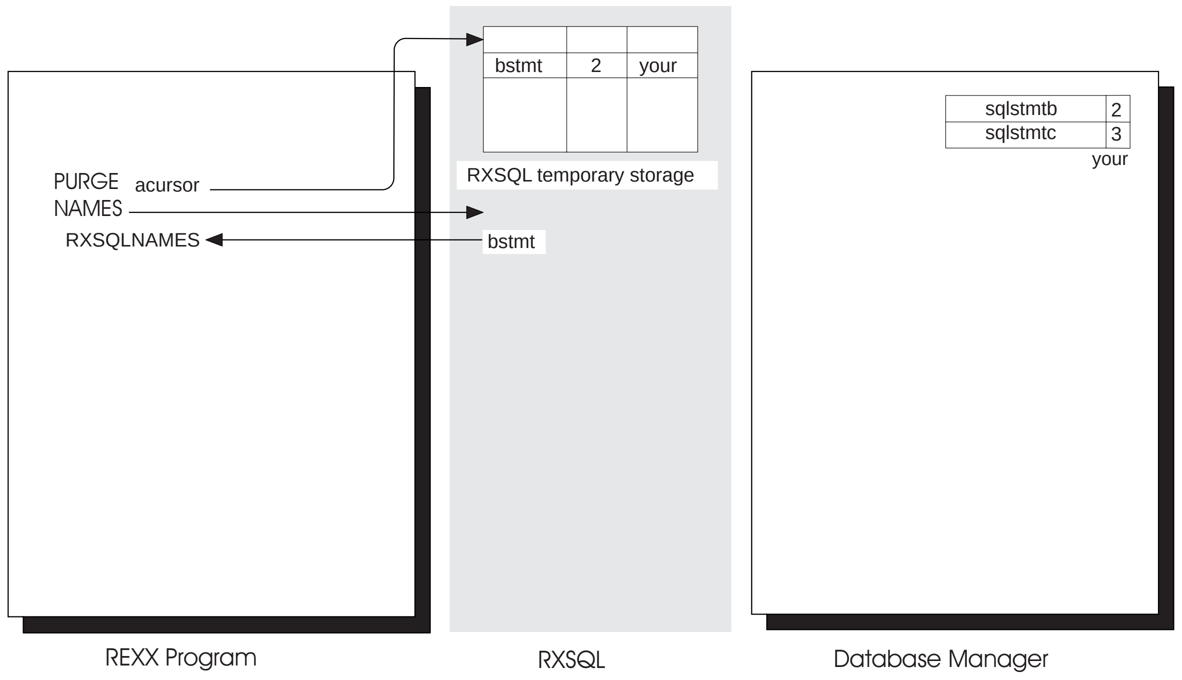


Figure 13. PURGE

RXSQL keeps track of the statements you have declared until control is returned to CMS, or until a PURGE command is issued. Unlike the limit of 40 prepared statements when using Dynamic SQL in RXSQL, the limit in Extended Dynamic SQL is much greater and is determined by the database manager.

Chapter 4. Using Dynamic and Extended Dynamic SQL Statements in RXSQL

Data stored by the database manager can be accessed by Static, Dynamic or Extended Dynamic SQL statements. RXSQL supports only Dynamic and Extended Dynamic SQL. *Dynamic* SQL statements are prepared and executed when your program is run and the operational form of the prepared statements does not persist beyond the logical unit of work. *Extended dynamic* statements support both static and dynamic access to data and are used for the direct creation and maintenance of packages.

Programs that use Dynamic SQL are easier to program and maintain than programs which use Extended Dynamic SQL; however there are situations where Extended Dynamic SQL has definite advantages over Dynamic SQL. The most prominent advantage that Extended Dynamic SQL has over Dynamic SQL is the ability to access data using Static SQL statements.

With Dynamic SQL, the database manager checks the statements being prepared for three things; existence, usage and authority. The database manager checks whether the objects referenced in the prepared statement exist and are being used correctly. The database manager also checks whether the authorization ID of the person executing the program has the authority and privileges required by the prepared statements. The database manager does this checking by looking up information in the catalog tables. This can cause many locks on these tables and slow the execution of the program.

With Static SQL, the database manager checks the statements being prepared for existence, usage and authority only once, when the statements are prepared and stored into a package. The privileges on the objects referenced in the prepared statements must be held by the authorization ID of the person who creates the package. These privileges may then be shared by granting the execute privilege on the package to others.

There are many situations where Static SQL is advantageous. They include the following:

- With Dynamic SQL, the user directs the program to prepare the SQL statement each time the program is invoked; with Static SQL the statement is prepared only once thus, in most cases, improving the performance of the application.
- Preparing a statement requires shared locks to be put on the catalog tables. Using Static SQL reduces the concurrency problems caused by locking because the PREPARE statement is issued only once, when the package is created.
- With Dynamic SQL, users must have the appropriate authority on a table before the statements can be executed. If the table contains sensitive data, the owner may need to manage this authority carefully. With Static SQL, the package owner can prepare statements which will restrict the user to only the necessary data. If the owner of the package has the appropriate authority, she can then grant execute so that a user is limited to issuing only the statements that are in the package.

- Users are not limited to the number of prepared statements in Extended Dynamic SQL. You can use dynamic or static Extended Dynamic SQL to prepare more than the current RXSQL limit of 40 Dynamic statements.

Within Dynamic, static Extended Dynamic and dynamic Extended Dynamic SQL, some of the statements are used to manipulate cursors while others are used to execute statements which do not require a cursor. The structure of the following sections on Dynamic, static Extended Dynamic and dynamic Extended Dynamic SQL matches this division of SQL statements into those which require cursors and those which do not require cursors.

Using Dynamic Statements in DB2 RXSQL

You can write queries, manipulate, control and define data using Dynamic SQL in RXSQL. The sequence of RXSQL statements you use depends on whether or not you require a cursor.

Using Dynamic Statements Which Require a Cursor

Cursors can be used to retrieve, insert, update or delete data. To retrieve data your program defines a *query_cursor* and retrieves rows using the FETCH statement. To insert data your program defines an *insert_cursor* and inserts rows using the PUT statement. You can also use a cursor to update or delete selected rows of an active set or result table by using the Positioned UPDATE or Positioned DELETE statement.

Using Dynamic Query or INSERT Statements

To execute *query_cursor* or *insert_cursor* statements in Dynamic SQL your program must issue the following RXSQL statements:

```
PREPARE sql_statement
DECLARE cursor for sql_statement    (optional)
OPEN cursor
    begin loop
    FETCH or PUT row for cursor
    end loop
CLOSE cursor
```

Note: The DECLARE *cursor* is optional, and may be executed before or after the PREPARE statement. If your program does not DECLARE a cursor for the INSERT or SELECT statement, then RXSQL will require *sql_statement* on the OPEN, FETCH, PUT and CLOSE statements. Other host languages supported by the database manager do not allow statement names to be used on OPEN or CLOSE statements.

Using Dynamic Positioned UPDATE or Positioned DELETE Statements

To execute Positioned UPDATE or Positioned DELETE statements in Dynamic SQL you have a choice of two sequences, illustrated as follows:

Sequence 1: Use the EXECUTE IMMEDIATE statement with the imbedded *select_statement* to update or delete the row retrieved by the *query_cursor*.

```
PREPARE select_statement
DECLARE query_cursor for select_statement
```

```
OPEN query_cursor
    begin loop
    FETCH row using query_cursor
    if the row satisfies the Positioned UPDATE or Positioned DELETE condition
        EXECUTE IMMEDIATE Positioned UPDATE or DELETE statement
    end loop
CLOSE query_cursor
```

The following example illustrates this sequence for a Positioned UPDATE.

```
/*** UPDATE the third column when the value in the second column = 300 ***/

/* The paired outside quotes are stripped off by REXX and the imbedded */
/* single quotes are passed to the database manager.                   */

"EXECSQL PREP stmt1 SELECT col2 FROM table1 WHERE col1='WRITER'",
     "FOR UPDATE OF col3"

'EXECSQL DECLARE query_cursor CURSOR FOR stmt1'

/* open the cursor and position it before first row */
'EXECSQL OPEN query_cursor'

 Do forever
  /* position cursor on a row and fetch data */
  'EXECSQL FETCH query_cursor col2_value'

   /* leave the loop when the cursor reaches the end of the result table */
   if SQLCODE=100 then leave

   if col2_value = 300 then
    "EXECSQL EXECUTE IMMEDIATE UPDATE table1 SET col3= 'S01' ",
      "WHERE CURRENT OF query_cursor"

 End  /* Do forever */

'EXECSQL CLOSE query_cursor'

'EXECSQL COMMIT'
```

Sequence 2: Prepare the *select_statement* and then issue the CALL or EXECUTE statement to update or delete the row retrieved by the cursor.

```
PREPARE select_statement
DECLARE query_cursor for select_statement
PREPARE Positioned UPDATE or Positioned DELETE sql_statement
OPEN query_cursor
    begin loop
    FETCH row using query_cursor
    if the row satisfies the Positioned UPDATE or Positioned DELETE
    condition
        EXECUTE or CALL prepared UPDATE or DELETE sql_statement
    end loop
CLOSE query_cursor
```

The UPDATE or DELETE *sql_statement* has a WHERE CURRENT OF *cursor_name* clause. If the DECLARE statement is not issued, the *select_statement* must be used on the OPEN, FETCH and CLOSE statements. The following example illustrates this sequence for a Positioned UPDATE.

```
/***** UPDATE the third column when the value in the second column = 300 ****/


"RXSQL PREP select_stmt SELECT col2 FROM table1 WHERE col1='WRITER'
     "FOR UPDATE of col3"

'RXSQL PREP stmt2 UPDATE table1 SET col3=:value',
                'WHERE CURRENT OF select_stmt'

/* open the cursor and position it before first row */
'RXSQL OPEN select_stmt'

 Do forever

  /* Position cursor on a row and fetch data */
  'RXSQL FETCH select_stmt INTO col2_value '

  /* leave the loop when the cursor reaches the end of the result table */
   if SQLCODE=100 then leave
   if col2_value = 300 then
   Do
     /* get the value that you want the column to be updated to  */
     say 'Type in the value you want and press enter'
     parse pull value .
     value="'"value"'"
    'RXSQL CALL stmt2 '
   End

 End

'RXSQL CLOSE select_stmt'

'X' COMMIT'
```

The DECLARE statement was not used in the preceding example to illustrate that RXSQL requires the *prepare_name* on the OPEN, FETCH and CLOSE statements, and in the WHERE CURRENT OF clause, when a cursor has not been declared.

Using Dynamic Statements Which Do Not Require a Cursor

To execute SQL statements which do not require a cursor, you have a choice of two different sequences. They are all illustrated as follows:

Sequence 1

Imbed the statement in an EXECUTE IMMEDIATE statement.

EXEC or **EXECUTE IMMEDIATE** *sql_statement*

Sequence 2

Prepare and then execute the statement.

PREPARE *sql_statement*
CALL or **EXECUTE** *sql_statement*

Sequence 1 supports previous versions of DB2 RXSQL.

Sequence 2 provides for preparing a non-cursor statement once and invoking it as often as required while the program is active. This sequence may execute more efficiently if the statement is to be invoked many times.

Using Extended Dynamic Statements in DB2 RXSQL

There are a few basic differences between using Dynamic SQL and Extended Dynamic SQL in RXSQL. The most obvious one is that you have to create a package when using Extended Dynamic SQL. This is **generally** done in a separate REXX program from the program that executes the prepared statements. This minimizes the number of times a statement is prepared.

For introductory illustrations see "Illustrations of Extended Dynamic SQL in RXSQL" on page 23.

Creating a Package

A package is created when the LUW that contains the CREATE PACKAGE statement, is ended by execution of the COMMIT statement. Packages are composed of sections which are added by the Extended PREPARE statement. There are two kinds of sections: those which are permanently filled with an SQL statement, and those which are empty. Those sections which are permanently filled with an SQL statement contain static SQL statements while those which are empty are ready for dynamic SQL statements to be temporarily prepared into them.

The type of section added to a package depends on the format of the Extended PREPARE statement used. Of the four formats of the Extended PREPARE statement, two are used to prepare Static SQL statements, while the other two are used for Dynamic SQL statements.

Static Sections in a Package

- The two formats used to prepare Static SQL are Basic and Single Row.
- The Basic Extended PREPARE statement is used to add a section to a package, and prepare an SQL statement into it.
- The Single Row Extended PREPARE statement is used to add a section to a package, and prepare a SELECT statement that returns only a single row.

Dynamic Sections in a Package

- The remaining two formats of the Extended PREPARE, Empty and Temporary, are used for Dynamic SQL.
- The Empty Extended PREPARE statement adds an empty section to a package.
- The Temporary Extended PREPARE statement fills the empty sections added to the package by the Empty Extended PREPARE.

The Empty Extended PREPARE must be issued in a LUW prior to the LUW that issues the Temporary Extended PREPARE. The package must exist, along with the empty section, before an SQL statement can be prepared into it. The empty section is filled for the duration of the LUW in which the Temporary Extended PREPARE statement is executed. When the LUW ends, the database manager resets the section to its original, empty condition.

There are restrictions on the use of the four formats of the Extended PREPARE statement depending on the type of package you create. You can create a non-modifiable package or a modifiable package.

Non-modifiable Package

A non-modifiable package supports all four formats of the Extended PREPARE statement and thus supports both static and dynamic SQL statements. The Basic, Single Row and Empty Extended PREPARE statements must be executed in the same LUW as the one in which the CREATE PACKAGE was issued. A non-modifiable package cannot be changed after the LUW in which it was created ends. Temporary Extended PREPARE statements must be executed in a later LUW. Static statements that are prepared into a non-modifiable package cannot be executed until package creation is complete, when the LUW ends.

Modifiable Package

In contrast, a modifiable package can be modified after the LUW in which it was created ends. Further, any committed modifications made to the package remain in force for subsequent LUWs. A modifiable package supports only the Basic Extended PREPARE and Single Row Extended PREPARE statement, and thus supports only Static SQL. A modifiable package can exist without any statements in it. Statements can be added to or dropped from a modifiable package at any time. They can also be executed in the same LUW as the one in which they were prepared without the necessity of first completing the creation of the package by committing the LUW.

Using Static Extended Dynamic Statements

When your program issues a Basic or Single Row Extended PREPARE statement, RXSQL returns a section number to your program in a REXX variable to indicate which section of the package your statement is in. Your program must record this number because it is used to refer to the statement when declaring or executing it.

Adding Static Sections to a Package

Static sections containing static statements are added to a package in the following sequence:

CREATE PACKAGE
Basic Extended **PREPARE** or Single Row Extended **PREPARE** *sql_statement*

- The section number is returned in a REXX variable

Write the section number to a file to keep a record of it
COMMIT

There can be many Extended PREPARE statements before the COMMIT WORK statement ends the LUW.

Adding Static Sections for Positioned UPDATE or Positioned DELETE Statements: There must be two statements prepared to execute a Positioned UPDATE or Positioned DELETE statement. The first one is a SELECT statement and the second one is an UPDATE or DELETE statement with a WHERE CURRENT OF *cursor_name* clause. Both the SELECT and the UPDATE or DELETE statements must be prepared into the same package. The following example illustrates this for a Positioned DELETE.

```
'RXSQL XPREP IN pkg1 SELECT col1, col2 FROM table1'
    'FOR UPDATE of col1,col2,col3'
/******* RXSQL returns section number 1 in SQLSTMTN; save it ****/

'RXSQL XPREP IN pkg1 DELETE FROM table1 WHERE CURRENT OF cursor1'
/******* RXSQL returns section number 2 in SQLSTMTN; save it ****/
```

The program that declares the cursor and executes the statements references the same *cursor_name* (`cursor1`) as the second XPREP statement. The cursor declared to retrieve rows must reference the section number (1) returned by the first XPREP statement. To delete a row satisfying the program requirements, the DELETE statement is invoked by referencing the section number (2) returned from the second XPREP statement. For example,

```
'RXSQL DECLARE cursor1 CURSOR FOR 1 IN pkg1'
/* cursor1 is the name of the query_cursor                    */

'RXSQL DECLARE delstmt FOR 2 IN pkg1'
/* delstmt will be referenced in an Extended CALL request     */
```

The SELECT statement must have a FOR UPDATE OF clause when

1. the package was created with the BLOCK option and
2. your program is preparing a Positioned UPDATE statement or, as in the example above, a Positioned DELETE statement.

The FOR UPDATE OF clause is necessary to turn off blocking.

Executing Static Extended Dynamic Statements in a Package

After a package is created, a second REXX program defines the cursor or executes the prepared statements by referring to the saved section number. The sequences of statements you can choose from for the second REXX program are similar to the sequences for cursor and non-cursor statements illustrated earlier.

Executing Static Extended Dynamic Query or INSERT Statements

To use *query_cursors* or *insert_cursors* in static Extended Dynamic SQL your program must issue the following RXSQL statements:

```
Extended DECLARE cursor_1 FOR section-1 IN pkg4
OPEN cursor_1
    FETCH or PUT row using cursor_1
    end loop
CLOSE cursor_1
```

Executing Static Extended Dynamic Positioned UPDATE or Positioned DELETE Statements

Two statements must be prepared to execute a Positioned UPDATE or Positioned DELETE statement. One is a prepared SELECT statement while the other is an UPDATE or DELETE statement with a WHERE CURRENT OF clause. The *cursor_name* in the WHERE CURRENT OF clause must have the same name as the *cursor_name* referenced in the DECLARE, OPEN, FETCH and CLOSE statements.

To execute a Positioned UPDATE or Positioned DELETE statement in static Extended Dynamic SQL, you have a choice of two sequences depending on whether a *statement_name* is declared for the Positioned UPDATE or Positioned DELETE statement.

If declared, the *statement_name* may be used in subsequent Extended CALL requests for the statement. Otherwise, the *section_number* and *package_name* must be used in subsequent Extended EXECUTE or XCALL requests for the statement.

The sequences are illustrated as follows:

Sequence 1:

```
Extended DECLARE cursor_1 FOR section-1 IN pkg1
Extended DECLARE stmt2 FOR section-2 IN pkg1
OPEN cursor_1
    begin loop
    FETCH row using cursor_1
    if the row satisfies the Positioned UPDATE or Positioned DELETE

        Extended CALL stmt2
    end loop
CLOSE cursor_1
```

Sequence 2:

```
Extended DECLARE cursor_1 FOR section-1 IN pkg1
OPEN cursor_1
    begin loop
    FETCH row using cursor_1
    if the row satisfies the Positioned UPDATE or Positioned DELETE
        Extended EXECUTE or XCALL section-2 IN pkg1
    end loop
CLOSE cursor_1
```

Executing Static Extended Dynamic Statements Not Requiring a Cursor

To execute static Extended Dynamic statements which do not require a cursor, you have a choice of two sequences.

Sequence 1: DECLARE and execute the statement.

Note: This sequence is unique to RXSQL.

```
Extended DECLARE stmt_1 FOR section-1 IN pkg3
Extended CALL stmt_1
```

The Extended DECLARE statement gives a name to a prepared statement which is used on the Extended CALL.

Sequence 2: Execute the statement directly.

```
Extended EXECUTE or XCALL FOR section-1 IN pkg3
```

Using Dynamic Extended Dynamic Statements

The Empty Extended PREPARE and Temporary Extended PREPARE support dynamic access to data. The Empty Extended PREPARE statement is used to add empty sections to your package. Then the Temporary Extended PREPARE statement is used to temporarily fill these empty sections. Empty sections are added to your package in the same LUW in which the package is created, while the Temporary Extended PREPARE statement is issued in a separate LUW.

When your program issues an Empty Extended PREPARE statement, RXSQL returns a section number to your program indicating which section of the package your statement is in. The section number must be saved to be used by a subsequent Temporary Extended PREPARE statement.

Adding Dynamic Sections to a Package

Dynamic sections are empty when they are added to a package so that statements can be dynamically prepared into them when the program is executing. The following sequence illustrates how empty sections are added to a package.

CREATE PACKAGE
Empty Extended **PREPARE**

- The section number is returned in a REXX variable

Write the section number to a file to keep a record of it
COMMIT

There can be many Static or Dynamic sections added to a package before the COMMIT statement ends the LUW. Two Dynamic sections must be available to execute a Dynamic Positioned UPDATE or a Positioned DELETE statement.

Executing Dynamic Extended Dynamic Statements in a Package

By preparing an Empty Extended PREPARE statement into your package, you have an empty section that is dynamically filled in a separate LUW by a Temporary Extended PREPARE statement.

Executing Dynamic Extended Dynamic Query or INSERT Statements

To use *query_cursors* or *insert_cursors* in dynamic Extended Dynamic SQL, your program must issue the following RXSQL statements:

```
Temporary Extended PREPARE FOR section-5 IN pkg1 sql_statement
Extended DECLARE cursor_5 FOR section-5 IN pkg1
OPEN cursor_5
    begin loop
    FETCH or PUT row using cursor_5
    end loop
CLOSE cursor_5
```

Executing Dynamic Extended Dynamic Positioned UPDATE or Positioned DELETE Statements

To execute a Positioned UPDATE or Positioned DELETE statement in dynamic Extended Dynamic SQL you have a choice of two sequences depending on whether a *statement_name* is declared for the Positioned UPDATE or Positioned DELETE statement.

If declared, the *statement_name* may be used in subsequent Extended CALL requests for the statement. Otherwise, the *section_number* and *package_name* must be used in subsequent Extended EXECUTE or XCALL requests for the statement.

The sequences are illustrated as follows:

Sequence 1:

Temporary Extended **PREPARE** `sql_stmt_1` into `section-11` **IN** `pkg1`

- `sql_stmt_1` is a SELECT statement with a FOR UPDATE OF clause.

Temporary Extended **PREPARE** `sql_stmt_2` into `section-12` **IN** `pkg1`

- `sql_stmt_2` is an UPDATE statement with a WHERE CURRENT OF `cursor_sel` clause.

```
Extended DECLARE cursor_sel FOR section-11 IN pkg1
Extended DECLARE upd_stmt FOR section-12 IN pkg1
OPEN cursor_sel
    begin loop
    FETCH row using cursor_sel
    if the row satisfies the conditions for Positioned UPDATE
        CALL upd_stmt
    end loop
CLOSE cursor_sel
```

Sequence 2:

Temporary Extended **PREPARE** `sql_stmt_1` into `section-11` **IN** `pkg1`

- `sql_stmt_1` is a SELECT statement with a FOR UPDATE OF clause.

Temporary Extended **PREPARE** `sql_stmt_2` into `section-12` **IN** `pkg1`

- `sql_stmt_2` is an UPDATE statement with a WHERE CURRENT OF `cursor_sel` clause.

```
Extended DECLARE cursor_sel FOR section-11 IN pkg1
OPEN cursor_sel
    begin loop
    FETCH row using cursor_sel
    if the row satisfies the conditions for Positioned UPDATE
        Extended EXECUTE or XCALL section-12 IN pkg1
    end loop
CLOSE cursor_sel
```

Executing Dynamic Extended Dynamic Statements Not Requiring a Cursor

When executing statements which do not require a cursor you have a choice of two sequences depending on whether a *statement_name* is declared for the statement.

If declared, the *statement_name* may be used in subsequent Extended CALL requests for the statement. Otherwise, the *section_number* and *package_name* must be used in subsequent Extended EXECUTE or XCALL requests for the statement.

The sequences are illustrated as follows:

Sequence 1: Prepare the statement, declare the statement to name it, then invoke it.

Note: This sequence is unique to RXSQL.

Temporary Extended **PREPARE** *sql_statement* into `section-3` **IN** `pkg6`
Extended **DECLARE** `stmtx` **FOR** `section-3` **IN** `pkg6`
Extended **CALL** `stmtx`

Sequence 2: Prepare the statement and invoke it.

Temporary Extended **PREPARE** *sql_statement* into `section-3` **IN** `pkg6`
Extended **EXECUTE** or **XCALL** `section-3` **IN** `pkg6`

Chapter 5. Coding DB2 RXSQL Requests

Delimiting and Continuing DB2 RXSQL Requests

You code your DB2 RXSQL requests as part of your REXX program. When DB2 RXSQL receives the first RXSQL or EXECSQL request, it loads RXSQL as a nucleus extension.

Delimiting RXSQL Requests

You should surround your RXSQL requests with paired single or double quotes so the REXX interpreter, recognizing a literal string, does not treat any of the words as REXX variables and resolve them before passing the string on to RXSQL. The REXX interpreter strips off the paired quotes before passing the statement to RXSQL.

This means that your program can have two different types of quotes included as part of your RXSQL request. For example:

```
"RXSQL PREPARE stmt UPDATE table SET col2=100 WHERE col1='salary' "
```

would be passed to RXSQL as:

```
RXSQL PREPARE stmt UPDATE table SET col2=100 WHERE col1='salary'
```

Continuing RXSQL Requests

If your RXSQL request is too long for one line, you can end the line with a comma and start the remainder of the request on a new line. The REXX interpreter recognizes the comma as a continuation character and replaces it with a blank after concatenating the two strings. The comma must be outside a paired set of quotes as illustrated in the following diagram.

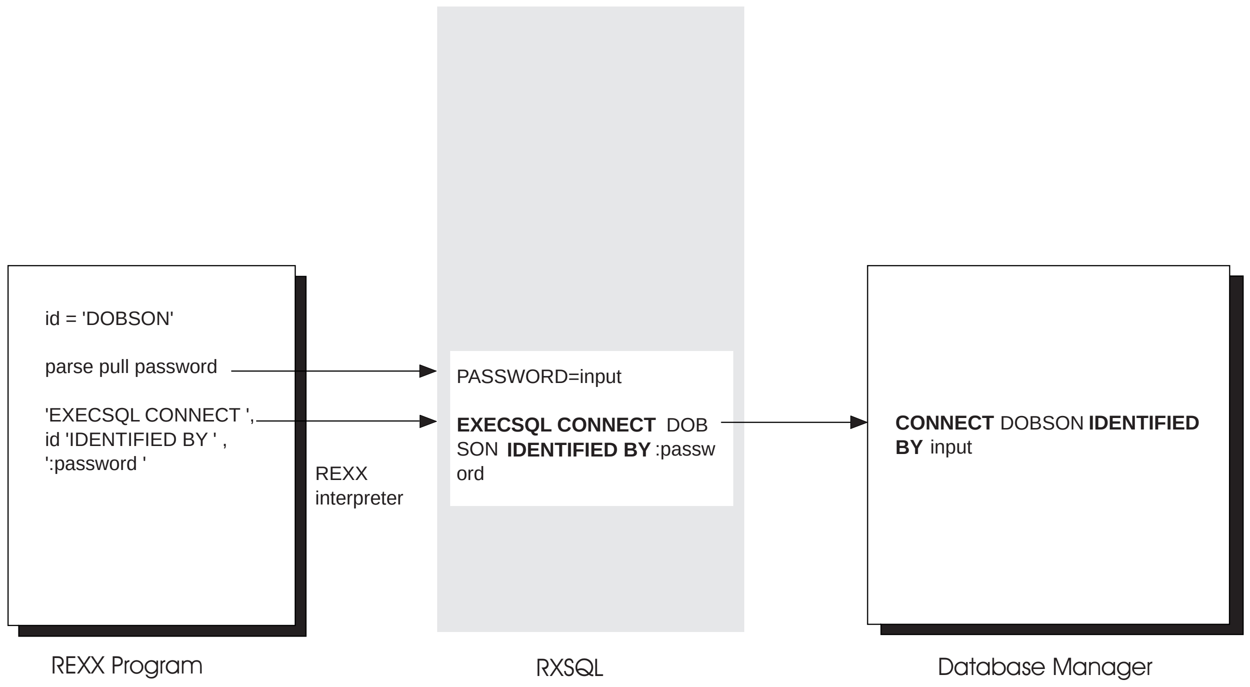

id = 'DOBSON'
parse pull password
'EXECSQL CONNECT ', id 'IDENTIFIED BY ' , ':password '
REXX interpreter
PASSWORD=input
EXECSQL CONNECT DOBSON IDENTIFIED BY :password
CONNECT DOBSON IDENTIFIED BY input
REXX Program
RXSQL
Database Manager

Figure 14. RXSQL Request Processing

This illustrates what happens to a RXSQL request as it gets passed from your program to RXSQL, and from RXSQL to the database manager. When you want the REXX interpreter to resolve a variable you leave it outside the quotes like `id`, and when you want RXSQL to resolve the variable you code it inside quotes and, in most cases, precede it with a colon like `:password`.

Elements of RXSQL Requests

Each RXSQL request is composed of many elements.

When the REXX interpreter is finished with the RXSQL request, it is passed to DB2 RXSQL. DB2 RXSQL then validates the elements and either executes the request or passes it to the database manager.

Keywords

DB2 RXSQL

1. translates your request to upper case to prepare for a keyword search,
2. checks that your keywords are correctly typed and are in the correct positions of your DB2 RXSQL request.

Note: For keyword scanning, DB2 RXSQL uses rules consistent with folding lowercase characters to uppercase using code page 037. See the *DB2 Server for VM System Administration* manual.

String of Characters

A string of characters is a sequence of bytes that RXSQL passes directly to the database manager. RXSQL will not change any part of the string of characters before passing it to the database manager.

Ordinary Identifiers

An *ordinary identifier* is an uppercase letter followed by zero or more characters, each of which is an uppercase letter, a digit, or the underscore character. RXSQL passes ordinary identifiers directly to the database manager for validation. A complete description can be found in the *DB2 Server for VSE & VM SQL Reference* manual.

Parameter Markers

A parameter marker is a question mark (?) that is used wherever a *variable_name* could be used in an SQL statement being processed by a PREPARE statement. RXSQL passes values for the parameter markers when the prepared statement is executed by a CALL, EXECUTE, OPEN or PUT statement.

Placeholders

Placeholders are periods (.) used in place of *rexx_host_variables* when retrieving data from the database manager. These are easy to code, but if performance is a concern you should restrict your select list to the columns of data that your REXX program needs. Placeholders may not have associated *indicator_variables* or *variable_qualifiers*.

REXX Variable Name

The term REXX variable name is used to distinguish between the name of a variable and its value. This distinction is important to RXSQL users because the REXX interpreter processes each RXSQL request before it is passed to RXSQL, and RXSQL often expects a REXX variable name, not a REXX variable value as input. To ensure that the REXX interpreter does not resolve a variable before it is passed to RXSQL, it must be enclosed in paired single or double quotes. The REXX interpreter then removes the quotes and passes the REXX variable name to RXSQL without resolving it. RXSQL retrieves the value of the REXX variable before passing the request to the database manager.

The valid status of a REXX variable name, as checked by the REXX SYMBOL function, can be LIT or VAR. It will be LIT if the REXX variable name has never been assigned a value, or if the REXX variable name has been dropped by the REXX DROP instruction. The variable status may be used to determine if a column has a value or is NULL. Note that there is a difference between a variable having a value consisting of an empty string and the variable representing a NULL value.

RXSQL and EXECSQL Invocation

There are two ways that the REXX interpreter will recognize a DB2 RXSQL request. One is when a command begins with the keyword **RXSQL**, and the other is when a command begins with the keyword **EXECSQL**. Error handling and the syntax rules vary with the way that a DB2 RXSQL request was invoked. This allows DB2 RXSQL to support two different rules of syntax; one to support programs developed for previous versions of RXSQL (RXSQL invocation) and the other to support enhanced error handling (EXECSQL invocation). The terms RXSQL invocation and EXECSQL invocation are used to distinguish the different processing rules. The different processing rules for the two invocations are discussed throughout this chapter.

You are strongly encouraged to use EXECSQL invocation when writing new applications to take advantage of the enhanced error handling.

When you are using RXSQL in an RXSQL or EXECSQL subcommand environment, the keyword EXECSQL or RXSQL always overrides the current RXSQL or EXECSQL subcommand environment. For more information see Appendix E, “RXSQL Subcommand Environment” on page 157.

RXSQL Request Syntax

You invoke the DB2 RXSQL interface by issuing DB2 RXSQL requests. Each request begins with either keyword `EXECSQL` or `RXSQL`, and is followed by a command or statement and then one or more clauses. The clauses may contain metavariables which represent variable values specified when the request is coded.

The following syntax diagram illustrates this:

```
                                    ┌─.──────────────────────────────┐
►►──┬─EXECSQL─┬─(1)──┬─Command───┬──▼─┬─────────┬──┬──────────────┬─┴──────────────────►◄
    └─RXSQL───┘      └─Statement─┘    └─KEYWORD─┘  └─metavariable─┘
```

Note:
1 RXSQL and EXECSQL must be in uppercase when you are using RXSQL in the address command environment.

Command
Statement
An action for the database manager, or a request that is specific to RXSQL. A DB2 RXSQL request may be either a statement or a command. These are illustrated in Figure 18 on page 61

KEYWORD
Part of the RXSQL request. The keywords and metavariables together are called clauses.

metavariables
The following list describes the possible metavariables that are used by DB2 RXSQL. Other metavariables used by the database manager, are passed directly to the database manager without being examined by DB2 RXSQL.

attributes_variable
A REXX variable name with an optional preceding colon whose value is a list of one of more data types, lengths and CCSIDs describing the parameter markers used in the SQL statement. For a description of the valid values allowed in the *attributes_variable*, refer to Figure 20 on page 92.

authorization_name
: A string of 1 to 8 characters designating an SQL authorization ID. In DB2 RXSQL any character is allowed except the `X'00'`. See the *DB2 Server for VSE & VM SQL Reference* manual for details on authorization names.

  Note: `X'00'` is allowed by other host languages supported by the database manager.

cursor_name
statement_name
: A *cursor_name* is a string of characters that represents a cursor for an SQL SELECT or INSERT statement.

  A *statement_name* is a string of characters that represents a prepared SQL statement.

  Names can have a maximum length of 18 characters. They must begin with a letter (A–Z or a–z) or a symbol ($, #, or @). The remaining characters can be letters (A–Z or a–z), symbols ($, #, or @), numbers (0–9), or underscores (_). Leading and trailing blanks are removed. DB2 RXSQL does not allow DBCS characters. DB2 RXSQL folds lowercase characters into uppercase. This means that you cannot have two cursors with names `abc` and `ABC`, for example.

  Notes:

  1. The $, #, and @ characters are all from code page 037. If you are not using code page 037 then you can substitute equivalent values. See the *DB2 Server for VM System Administration* manual.
  2. DB2 RXSQL folds lowercase characters to uppercase characters using different rules from those used by ISQL, DBSU and the DB2 Server for VM relational database manager.

  The *cursor_name* or *statement_name* cannot be the same as any other *cursor_name* or *statement_name* defined in the program.

  Note: This is more restrictive than for other host languages supported by the database manager which allow *statement_names* and *cursor_names* to have the same value.

function_number
trace_level
: The function number and trace level that are specified when using the TRACE command. See “TRACE” on page 104 for a list of the valid values.

operator_command
: A string of characters which form an DB2 Server for VM operator command that DB2 RXSQL passes directly to the database manager. The string does not have to be enclosed in quotations. See the *DB2 Server for VSE & VM Operation* manual for a complete list of operator commands.

option
: An option used in the CREATE PACKAGE statement as listed in the *DB2 Server for VSE & VM SQL Reference* manual. DB2 RXSQL passes this value directly to the database manager.

package_name
: The name of a package in which SQL statements are stored. Its structure is as follows:

package_name

```
►──authorization_name──.──package_id──────────────────────────►
```

package_id
A short ordinary identifier.

password
A string of 1 to 8 characters designating an SQL password. In DB2 RXSQL, any character is allowed except the X'00'.

Note: X'00' is allowed by other host languages supported by the database manager.

rexx_host_stem_name
A *rexx_host_stem_name* is composed of a *main_stem* and an optional *indicator_stem*. Its structure is as follows:

rexx_host_stem_name on EXECSQL invocation

```
►──┬──┬───┬─main_stem.─┬──────────────────────────────────┬─┬──────────────►
   │  └─:─┘            └─┬─────────────┬─ :indicator_stem.─┘ │
   │                     └─INDICATOR───┘                     │
   └──┬───┬─main_stem.:indicator_stem.────────────────────────┘
      └─:─┘
```

rexx_host_stem_name on RXSQL invocation

```
►──┬──┬───┬─main_stem.──────────────────┬────────────────────────►
   │  └─:─┘                             │
   └──┬───┬─main_stem.:indicator_stem.──┘
      └─:─┘
```

main_stem
A REXX stem variable name that has an optional preceding colon and a trailing period.

indicator_stem
A REXX stem variable name that has a mandatory preceding colon and a trailing period.

DB2 RXSQL assigns to the *main_stem* variables the values of the row in the result table, and to the *indicator_stem* variables the indicator values passed from the database manager. DB2 RXSQL sets the first element of the *main_stem* and the *indicator_stem* variables (stem.0) to the number of columns returned from the database manager, and the remaining variables (stem.n) to the *n*th column value and *n*th indicator value.

For example, suppose you have a REXX variable where:

- The stem is `abc.ln`.
- The variable `ln` in the program has the value D
- Four columns are in the result.

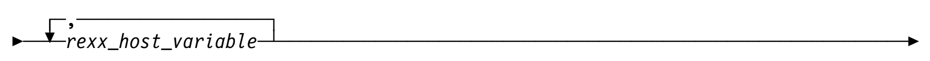
DB2 RXSQL sets the variables as follows:

```
ABC.D.0 = 4 (number of columns returned)
ABC.D.1 = first column value
ABC.D.2 = second column value
ABC.D.3 = third column value
ABC.D.4 = fourth column value
```

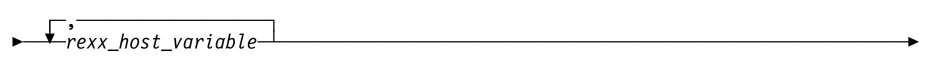
rexx_host_variable_list

A *rexx_host_variable_list* is a list of *rexx_host_variables*, structured as follows:

rexx_host_variable_list on EXECSQL invocation



rexx_host_variable_list on RXSQL invocation

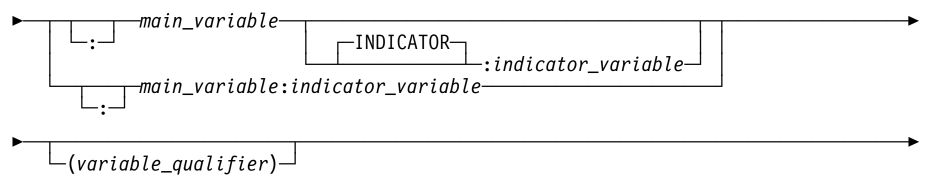


rexx_host_variable

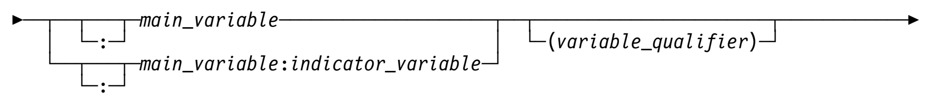
Rexx_host_variables are used to pass and receive data between the program and the database. There are three parts to a *rexx_host_variable*: the *main_variable*, the *indicator_variable*, and the *variable_qualifier*.

A *rexx_host_variable* is structured as follows:

rexx_host_variable on EXECSQL invocation



rexx_host_variable on RXSQL invocation



main_variable

A REXX variable name. The value of this variable is either used as input to the database manager or it is assigned with output from the database manager.

indicator_variable

A REXX variable name. The value of this variable contains the input or output indicator value.

variable_qualifier

A valid data type with an optional CCSID for input or output data. The data types are listed under "Data Type" on page 51.

See the next section for a complete discussion of REXX Host Variables.

section_number
The integer by which a section in a package is referenced when using Extended Dynamic SQL.

server_name
A long ordinary identifier that designates an application server.

sql_statement
A string of characters that forms an SQL statement. DB2 RXSQL passes this string to the database manager.

variable_name
A REXX variable name with a mandatory preceding colon. DB2 RXSQL fetches the value of the *variable_name* and passes the value to the database manager when the statement containing this metavariable is executed. The value of the *variable_name* must conform to the coding rules of the metavariable for which it is being substituted.

REXX Host Variable

Your program uses *rexx_host_variables* for passing data between the database and your program. The *n*th *rexx_host_variable* corresponds to the *n*th column referenced.

There are differences in the way *rexx_host_variables* are coded that depend on the way the RXSQL request is invoked.

EXECSQL Invocation

On EXECSQL invocation,

- *rexx_host_variables* must be separated by commas in a *rexx_host_variable_list*.
- The indicator variable may be separated from its associated variable by blanks, the keyword INDICATOR, or it may directly follow the associated variable.
- Colons are optional for main variables, but their use is strongly recommended.

EXECSQL Invocation Examples:

```
'EXECSQL FETCH cursor INTO :name:indicator1, :department:indicator2'

'EXECSQL FETCH cursor INTO :name :indicator1, :department :indicator2'

'EXECSQL FETCH cursor INTO :name:indicator1,:department:indicator2(SMALLINT)'

'EXECSQL FETCH cursor INTO :name',
    'INDICATOR :indicator1(CHAR(25)),:department:indicator2(SMALLINT)'
```

RXSQL Invocation

On RXSQL invocation,

- *rexx_host_variables* may be separated by commas or blanks in a *rexx_host_variable_list*
- the indicator variable must follow the associated *main_variable* with no intervening blanks

RXSQL Invocation Examples

```
'RXSQL FETCH cursor INTO :name:indicator1 :department:indicator2'

'RXSQL FETCH cursor INTO :name:indicator1(CHAR(25))',
                        ':department:indicator2(SMALLINT)'

'RXSQL FETCH cursor INTO :name:indicator1(CHAR(25)),',
                        ':department:indicator2(SMALLINT)'
```

Main_variable

A *main_variable* is a fully qualified REXX variable name with an optional preceding colon. RXSQL fetches the value of this variable on input or assigns a value to this variable on output.

Input: DB2 RXSQL must infer the data type of your input values, when inserting data and variable qualifiers have not been specified, because REXX variables do not have assigned data types. The rules that RXSQL follows to determine the data type of your input values are outlined under “Inferring the Data Type” on page 53. Variable qualifiers are discussed under “Variable_qualifier” on page 51.

Output: *Rexx_host_variables* in a list are set in order starting with the first column result. You should be careful in your naming of the *main_variables* and *indicator_variables* in the list of *rexx_host_variables*. If a *main_variable* is used within a compound variable name elsewhere in another *main_variable*, unexpected value assignments will result. The following example illustrates this.

```
n=5
'RXSQL FETCH row INTO :first.col.n,:col,:third.col.n '
```

The values returned from the database are respectively: fun, for, all.

REXX Variable Name	REXX Variable Name after resolved by the REXX interpreter	Variable Value
first.col.n	FIRST.COL.5	fun
col	COL	for
third.col.n	THIRD.FOR.5	all

Now, if the program tries to reference `first.col.n`, REXX translates all references to it as `FIRST.for.5`. This variable is undefined and will return a literal value equal to the variable name. The value `fun` that was assigned to the first column is not accessible unless the variable `COL` is dropped.

Indicator_variable

An *indicator_variable* is a REXX variable name that is preceded by a colon. Your program assigns a value to the indicator on input, and RXSQL assigns the value on output. The value of the *indicator_variable* must be zero or a positive or negative integer.

Input: When used on input, an *indicator_variable* is used to indicate whether the *main_variable* is null. *Indicator_variables* are used as described in the following table.

Figure 15. Indicator Variable Values On Input

Invocation	**Main_variable Status**	**Indicator_variable Value**	**Main_variable Value Passed to the Database Manager**
EXECSQL	VAR	>= 0	not null
EXECSQL	VAR	< 0	null
EXECSQL	VAR	indicator_variable not used	not null
EXECSQL	LIT	>= 0	generates an RXSQL error condition
EXECSQL	LIT	< 0	null
EXECSQL	LIT	indicator_variable not used	generates an RXSQL error condition
RXSQL	VAR	>= 0	not null
RXSQL	VAR	< 0	null
RXSQL	VAR	indicator_variable not used	not null
RXSQL	LIT	>= 0	generates an RXSQL error condition
RXSQL	LIT	< 0	null
RXSQL	LIT	indicator_variable not used	null

Note: *Main_variable* status is available using REXX function SYMBOL('*main_variable*')

Output: On output, RXSQL assigns to the *indicator_variable* the value that the database manager passes to DB2 RXSQL. Refer to the *DB2 Server for VM Application Programming* manual for information on possible *indicator_variable* values on output.

If the column value is NULL, then *indicator_variable* is assigned a value according to the following table.

Figure 16. Indicator Variable Value On NULL Output

Invocation	**Main variable**	**Indicator variable**	**Value of Indicator variable after FETCH**	**Value of Main variable value after FETCH**
EXECSQL	present	present	< 0	unchanged
EXECSQL	present	absent	generates an RXSQL error condition	unchanged
RXSQL	present	present	< 0	unchanged
RXSQL	present	absent	not applicable	undefined

As the table illustrates, you must provide an *indicator_variable* when using the EXECSQL invocation if

- there are NULL values in the columns of the table referenced in the SELECT statement or

- a derived result column can develop a numeric conversion error resulting in a NULL value.

Variable_qualifier

You can use *variable_qualifiers* on input and output to specify how you want the data stored in the *main_variables* to be sent to the database manager or how you want to retrieve data into the *main_variables*. When the *main_variable* has a data type supporting a Coded Character Set Identifier (CCSID), the CCSID can be specified following the data type. There must be at least one blank between the data type and the CCSID keyword.

If no data type is given for a *main_variable* then RXSQL will infer the data type according to the rules in "Inferring the Data Type" on page 53. If no CCSID is given for a *main_variable*, then the CCSID defined on the application server is used to perform the operation.

Data Type

The data type is specified first, and can be any of the following:

INTEGER
SMALLINT
REAL - for single precision float
FLOAT - for double precision float
DECIMAL(*m*,*n*) - where *m* = precision, *n* = scale
CHAR(*n*) - where *n* is an integer
VARCHAR(*n*)
LVARCHAR(*n*)
GRAPHIC(*n*)
VARGRAPHIC(*n*)
LVARGRAPHIC(*n*)

Note: ZDECIMAL(*m*,*n*) is also allowed when using the DRDA protocol.

Your program inserts DATETIME data by specifying a CHAR(*n*) data type, where *n* is the length of the DATETIME data string.

CCSID

The CCSID is specified after the data type as `CCSID n`. To specify FOR BIT DATA, set the CCSID to 65535.

Input: On input, the *variable_qualifier* specifies the data type to which you want DB2 RXSQL to convert the input data before passing it to the database manager. Input data is stored in the *main_variable*.

The following is an example of a statement that uses an input *variable_qualifier*:

```
OPEN cursor1 USING :v1:i1(VARCHAR(10) CCSID 500), :v3(INTEGER)
```

For numeric data, when the value of the *main_variable* is too large for, or cannot be converted to the data type specified in the *variable_qualifier*, DB2 RXSQL returns an error indicating that the data would be corrupted if it was passed to the database manager. However, if low order truncation for decimal data occurs because data after the decimal point was lost, then DB2 RXSQL passes the truncated data to the database manager and returns a warning condition.

For character and graphic data, either fixed or variable length, if the variable's data length is larger than specified by the variable qualifier, the data will be truncated on the right. For fixed length character or graphic data, if the variable's data length is shorter than specified by the variable qualifier, then the data is padded with blanks on the right.

Output: On output you can specify the data type and CCSID to which you want a column value being retrieved from the database to be converted. In this case, the database manager does the conversion before the data is passed to RXSQL. For example, you may want data from a column defined as `VARCHAR(20) CCSID 290` to be returned to your program as `CHAR(26) CCSID 500`. If the value returned to this program is less than 26 characters, the database manager will pad the end of the string with blanks and convert it to CCSID 500.

The following is an example showing how to code a *variable_qualifier* on output:

```
'EXECSQL FETCH cursor2 INTO:v1:i1(CHAR(26) CCSID 500),:v2(CCSID 500),:v3'
```

Passing Data from Your REXX Program to RXSQL

When your program passes input data from REXX to the database manager in a CALL, EXECUTE, OPEN or PUT statement, RXSQL must assign a data type and determine whether the input value is NULL. RXSQL assigns a data type to your input data according to the information you provide. If you use *variable_qualifiers*, then RXSQL will assign the data type you specified to the input data before passing it to the database manager. If you do not use *variable_qualifiers*, then RXSQL must infer the data type. Using *variable_qualifiers* is strongly recommended.

When you are using Extended Dynamic SQL you also have the option to provide attributes of input data when you are preparing the SQL statements. The attributes are used by the database manager to help build the access path which the optimizer uses when executing the prepared statements.

All of the ways to specify input data are discussed in the following section.

Using REXX Host Variables to Pass Input Data

Recognizing NULL Values

RXSQL recognizes NULL input values in two ways: a negative *indicator_variable*, or a *main_variable* with an unassigned value. The REXX DROP instruction makes a variable unassigned.

When your DB2 RXSQL request begins with the keyword `EXECSQL`, you must use *indicator_variables* to insert NULL values.

Using Variable Qualifiers to Assign a Data Type

You can use *variable_qualifiers* to specify the data type and CCSID of your input data. See “REXX Host Variable” on page 48. When you use *variable_qualifiers*, DB2 RXSQL will change your input data to the format requested before passing it to the database manager.

Specifying the CCSID: You can override the CCSID of any column for which CCSIDs are allowed by specifying the CCSID after the data type in the *variable_qualifier*.

You can determine the CCSID of a column using a DESCRIBE request for a prepared SELECT statement. The CCSID of each column is returned in a DB2 RXSQL variable.

Inferring the Data Type

After RXSQL retrieves the value of the *main_variable* from REXX, the following rules are used when assigning a data type if a *variable_qualifier* is not used:

Character Data: A string of characters enclosed in paired single quotation marks is a character string. RXSQL removes the leading and trailing quotation marks before passing the character string to the database. A string of characters enclosed in paired double quotation marks is passed with the quotation marks. DB2 RXSQL assumes an empty string is a character string, not a NULL value. A string which does not follow the rules for numeric or graphic data will be assigned a character data type.

To guarantee that RXSQL converts a string to a character data type, ensure that the first and last characters of the string value are single quotation marks. Simply assigning a value in single or double quotes does not work. For example,

```
stringvar='100'
```

will cause REXX to set the REXX variable `stringvar` to the string of characters 100 (without the single quotes) before passing it to RXSQL. RXSQL will assign a data type of integer to the string, and pass it to the database manager without the single quotes. On the other hand, if `stringvar=100`, then,

```
stringvar="'"stringvar"'"
```

will cause REXX to set the REXX variable `stringvar` to the string of characters '100' (with single quotes). RXSQL assumes that the string has a data type of character because it is enclosed with leading and trailing single quotes, and passes the string to the database manager without the single quotes.

Your input values will be assigned the current default CCSID. You must use *variable_qualifiers* if you want to specify the CCSID of your input data.

Graphic Data: Graphic data is recognized by RXSQL in two different ways, depending on how the request was invoked.

Invocation	Graphic data attributes
EXECSQL	A string that has a leading `G` followed by a single quote, a shift-out character (`'0E'X`), an even number of bytes, a shift-in character (`'0F'X`), and ends with a single quote is assigned a data type of VARGRAPHIC. RXSQL strips off the leading `G`, single quote and shift-out along with the trailing shift-in and single quote before passing the string to the database as VARGRAPHIC.
RXSQL	A string that has a leading shift-out character, a trailing shift-in character, and contains an even number of bytes is assigned a graphic data type. RXSQL strips off the leading shift-out and the trailing shift-in characters before passing the string to the database as VARGRAPHIC. To ensure that MIXED data is passed to the database manager as VARCHAR, it must be enclosed in single quotes.

Figure 17 on page 54 expands the above descriptions. Shift-out and shift-in characters are represented by the symbols < and >, respectively.

Figure 17. Input Data Inferred as Character or Graphic Data Types

Data	Invoked by	Data modification	Data sent to the database manager
<...even...>	EXECSQL		VARCHAR
	RXSQL	strip <>	VARGRAPHIC
G'<...even...>'	EXECSQL	strip G'< >'	VARGRAPHIC
	RXSQL		VARCHAR
'.......'	EXECSQL	strip ' '	VARCHAR
	RXSQL	strip ' '	VARCHAR
<...odd....>	EXECSQL		VARCHAR
	RXSQL		VARCHAR
G'<...odd>'	EXECSQL		VARCHAR
	RXSQL		VARCHAR
<...>...	EXECSQL		VARCHAR
	RXSQL		VARCHAR
G'<...>'...	EXECSQL		VARCHAR
	RXSQL		VARCHAR
...<...&gt	EXECSQL		VARCHAR
	RXSQL		VARCHAR
...G'<...>'	EXECSQL		VARCHAR
	RXSQL		VARCHAR

Note: The symbols ... represent any characters including < or >

Numeric Data: Numeric data can be passed to the database manager as one of the three following data types:

FLOAT A number in scientific or engineering notation. For example, a number followed immediately by an E or an e, an optional plus or minus sign, and a series of digits.

DECIMAL A number with a decimal point is passed as a DECIMAL(*m*,*n*), where *m* indicates precision and *n* indicates scale. It can have a leading plus or minus sign.

INTEGER A number with neither decimal point nor exponent. It can have a leading plus or minus sign.

If a number is not within the allowed numeric SQL limits supported by the database manager, then RXSQL will pass it to the database as a character string. See the *DB2 Server for VSE & VM SQL Reference* manual for information on SQL limits.

Datetime Data: When your program is inserting data into a DATE, TIME or TIMESTAMP column, then RXSQL passes your data directly as a character string.

Specifying Attributes of Input Data Using Extended PREPARE

When your program issues an Extended PREPARE statement, it can use *attributes_variables* to provide information about parameter markers in the prepared statement if the data types of the values that will replace the parameter markers are known. However, this does not prevent DB2 RXSQL from assigning a data type when the prepared statements are executed by the Extended CALL, Extended EXECUTE, OPEN and PUT statements as discussed in the previous sections.

Passing Data from RXSQL to Your REXX Program

DB2 RXSQL returns the results of requests in three ways:

- Setting REXX variables your program has defined
- Setting predefined RXSQL variables
- Setting the return code (rc).

Passing Output Data from the Database Manager to REXX Variables

When your program issues a FETCH, Extended CALL, or Extended EXECUTE statement to retrieve data from the database, DB2 RXSQL passes output to the program as requested, using either *rexx_host_variables* or a *rexx_host_stem_name*.

Using REXX Host Variables for Output Data

When your program uses *rexx_host_variables* to receive output data, DB2 RXSQL sets the *main_variable* and the *indicator_variable*, if supplied, for each column of data requested. The *main_variable* is set to the value of the output data according to the *variable_qualifier* setting, if provided, and the *indicator_variable* is set to the value that the database manager passes to RXSQL.

Handling Null Values

For EXECSQL invocation, if your program does not provide *indicator_variables*, DB2 RXSQL may return an error condition for FETCH, Extended CALL or Extended EXECUTE.

You can test for a NULL value returned by checking the *indicator_variable* as follows:

```
If column4ind < 0 then column4 = '?'
```

This example sets the `column4` variable to `?` if the result is NULL as indicated by its corresponding *indicator_variable* `column4ind`.

For more information, see “Indicator_variable” on page 49.

For RXSQL invocation, if your program does not provide *indicator_variables* for FETCH, Extended CALL or Extended EXECUTE, then RXSQL will drop the *main_variable* when the column value is NULL.

Do not test for `''` (empty string) as if it were a NULL. `'` is a valid non-NULL value. You can use the REXX function SYMBOL to test whether a variable is undefined as illustrated in the following example:

```
If SYMBOL('column5') <> 'VAR'
then column5 = '?'
```

This example sets the `column5` variable to `?` if the result is NULL. DB2 RXSQL has done the equivalent of the following REXX instruction:

```
drop column5
```

Using REXX Host Stem Names for Output Data

Your program can also use a *rexx_host_stem_name* to receive the output data and associated indicator values. The same considerations apply to *rexx_host_stem_variables* as for *rexx_host_variables* with respect to NULL values returned.

Handling Null Values

For EXECSQL invocation, if you do not provide an *indicator_stem* and a NULL value is returned, DB2 RXSQL returns an error condition.

For RXSQL invocation, if you do not provide an *indicator_stem* and a NULL value is returned for column n, the corresponding *rexx_host_stem.n* variable is dropped.

RXSQL Variables

RXSQL also returns data to your program in predefined REXX variables. Your program should not assign values to these variables, as RXSQL will reset what you assigned. The following table outlines the variables that RXSQL sets, and which RXSQL request they are assigned on.

Request	Variable	Explanation
All	RXSQLREQUEST	Your program's request as interpreted by DB2 RXSQL. It begins with either `RXSQL` or `EXECSQL`, and can be used to help you resolve errors. In a subcommand environment, DB2 RXSQL inserts either RXSQL or EXECSQL in front of the request if neither was specified. You may use it to display the request being processed at the time of the error.
Any	RXSQLMSG	An empty string if no DB2 RXSQL errors or warnings occurred, otherwise a text message. Message set for warnings only if EXECSQL invocation. `RXSQLMSG` is usually set with the message that corresponds to the DB2 RXSQL return code. If a specific message in the repository cannot be found, or the repository itself is unavailable, then `RXSQLMSG` is set with the CMS message generated at the time of the message repository error. In this case, the message will not correspond to the return code that identifies the RXSQL error or warning. If the DB2 RXSQL environment cannot be initialized, `RXSQLMSG` is not set.

Request	Variable	Explanation
DESCRIBE	SQLDAN.i SQLDAT.i SQLDAC.i SQLDAL.i	Column names, data types, CCSIDs, and labels where: SQLDAN.i = column names or label names SQLDAT.i = column data types SQLDAC.i = column CCSID values SQLDAL.i = column labels i = number of the column For each column i, you have SQLDAN.i, SQLDAT.i, SQLDAC.i, and optionally SQLDAL.i. Before the results are set, SQLDAN., SQLDAT., SQLDAC., and SQLDAL. stems are initialized to empty strings. DB2 RXSQL sets SQLDAN.0, SQLDAT.0, SQLDAC.0, and SQLDAL.0 to the number of columns in the *select_list*. The rest of the variables in these structures are set to column names, data types, CCSID values, and labels corresponding to the SELECT statement.
NAMES	RXSQLNAMES	A list of *cursor_names* and *statement_names* of all the SQL statements that RXSQL has prepared or declared.
OP	SQLOP.i	Each line of the result of the OP command is set into SQLOP.i and SQLOP.0 has the number of lines returned.
SQLDATE with no arguments	SQLDATE	The current date-formatting option character. SQLDATE is set with the first character of the argument specified on the last SQLDATE statement or RESET.
SQLISL with no arguments	SQLISL	The current isolation level. This setting is either *RR*(repeatable read), *CS*(cursor stability), or *UR* (uncommitted read).
SQLTIME with no arguments	SQLTIME	The current time-formatting option character. SQLTIME is set with the first character of the argument specified on the last SQLTIME statement or RESET.
STATE	RXSQLSTATE	Two words that describe the type and state of a particular statement.
STMT	RXSQLSTMT	The SQL statement given on a Dynamic PREPARE statement, or the section number and package name given on an Extended DECLARE statement.
XPREP, PREPARE	SQLSTMTN	The section number assigned by the database manager after an Extended PREPARE statement is executed. This number is used on a later DROPSTMT, Extended CALL, Extended EXECUTE, XCALL, Extended DECLARE, or another Extended PREPARE statement. The section number associates a statement name with a particular section of a package.

SQLCA Variables

The SQLCA variables are set after every RXSQL statement which results in a database manager call. DB2 RXSQL passes the value of these variables directly from the database manager. See the *DB2 Server for VSE & VM SQL Reference* and the *DB2 Server for VM Messages and Codes* manuals for a more detailed explanation of these variables.

SQLCODE	The DB2 Server for VM primary error code. In general, zero denotes successful completion. Codes greater than zero denote successful execution, but with an exception condition. Negative codes represent error conditions.
SQLSTATE	SQLSTATE contains error codes for errors common to the DB2 Server for VM relational database manager and other IBM* distributed relational database managers. SQLSTATE is a variable containing codes for warnings and errors returned as a result of processing SQL statements. The errors and warnings reported include all errors from each of the relational database products. SQLSTATE does not replace SQLCODE. The SQLCA contains both an SQLSTATE value and an SQLCODE value.
SQLERRM	Error and warning message tokens. Adjacent tokens are separated by a byte containing `X'FF'`.
SQLERRMC	Set to the same value as SQLERRM. This was added to conform with SAA SQL.
SQLERRP	The product code and, if there is an error, the name of the module that returned the error.
SQLWARN.*n*	Each of the `SQLWARN.`*n* variables, where *n* ranges from zero to ten, is set to a single-character warning flag. The combination of eleven `SQLWARN.`*n* variables is equivalent to the `SQLWARN` variable. This was added to conform with SAA SQL.
SQLWARN	Eleven single-character warning flags. SQLWARN is a single string of characters that can have embedded blanks. Each flag has a distinct meaning based on its position. Check the first character; then, based on its value you may want to check the remaining fields. Descriptions of the characters are given in the *DB2 Server for VSE & VM SQL Reference* manual.
SQLERRD.*n*	There are six secondary error codes and indicators. *n* can have the value of 1 to 6. These are described in the *DB2 Server for VSE & VM SQL Reference* manual.

Error Handling

The way that RXSQL reports error conditions to your program depends on how the RXSQL request was invoked. The following sections summarize the meaning of the return codes for each type of invocation.

Return Code on EXECSQL Invocation

Each request invoked with EXECSQL sets a return code that can be tested in the REXX EXEC as variable rc. The following settings can occur:

rc = 0	Normal return (no errors).
rc = 10	An SQL warning occurred. At least one of the following conditions is applicable: 1. the first character of *SQLSTATE* equals 0 and the second character is greater than 0 2. the first character of SQLWARN is non-blank. For example, after a FETCH statement, a normal warning is SQLCODE=100 SQLSTATE=02000, which signifies that there are no more rows in the result table.
rc = -10	An SQL error occurred. If the first character of SQLSTATE is greater than 0, the database manager returned an error.
rc > **1000**	A RXSQL warning occurred. Check the RXSQLMSG variable for an explanation.
rc < **-100**	A DB2 RXSQL error occurred. Check the RXSQLMSG variable for an explanation.
rc = -3, 28, 41	A CMS error occurred when the program was invoked.

Return Code on RXSQL Invocation

Each request invoked with RXSQL sets a return code that can be tested in the REXX EXEC as variable rc. The following settings can occur:

rc = 0	Normal return (no errors).
rc = 4	An SQL warning occurred. At least one of the following conditions is applicable: 1. the first character of *SQLSTATE* equals 0 and the second character is greater than 0 2. the first character of SQLWARN is non-blank. For example, after a FETCH statement, a normal warning is SQLCODE=100 SQLSTATE=02000, which signifies that there are no more rows in the result table.
rc = 8	An SQL error occurred. If the first character of SQLSTATE is greater than 0, the database manager returned an error.
rc > **100**	A DB2 RXSQL error occurred. Check the RXSQLMSG variable for an explanation.
rc = -3, 28, 41	A CMS error occurred when the program was invoked.

The RXSQL invocation does not support the detection of DB2 RXSQL warnings.

See Appendix A, “RXSQL Return Codes and Messages” on page 109 for more information on CMS, DB2 Server for VM, and DB2 RXSQL return codes.

Chapter 6. RXSQL Request Descriptions

This chapter contains detailed information on using RXSQL requests. The following table lists where the syntax diagrams and detailed descriptions can be found.

RXSQL handles some requests by passing them directly to the database manager in an EXECUTE IMMEDIATE statement. The description, syntax diagram and usage rules of these requests are described in the *DB2 Server for VSE & VM SQL Reference* manual. These direct requests are translated to uppercase using code page 037. To avoid translation problems across different environments, you should code these commands in uppercase.

Note: The DB2 Server for VM language restrictions on EXECUTE IMMEDIATE prevent the use of host variables or parameter markers for DELETE, DROP, INSERT and UPDATE requests as immediate commands. If you wish to use host variables or parameter markers for these requests, you must code these as dynamic statements and prepare them to be invoked via CALL or EXECUTE.

Figure 18 (Page 1 of 4). RXSQL Requests

Request Type	RXSQL Request	Function	Refer To
Dynamic Statement	ACQUIRE DBSPACE	Obtains and names a dbspace.	*DB2 Server for VSE & VM SQL Reference*
Dynamic Statement	ALTER DBSPACE	Alters the percentage of free space. Also alters the lock size of a PUBLIC dbspace.	*DB2 Server for VSE & VM SQL Reference*
Dynamic Statement	ALTER TABLE	Adds a column to a table or manages referential constraints.	*DB2 Server for VSE & VM SQL Reference*
Dynamic Statement	Dynamic CALL	Executes an SQL statement defined by a previous Dynamic PREPARE or PREP.	page 64
Extended Statement	Extended CALL	Executes an SQL statement defined and stored in a package by a previous Extended PREPARE or XPREP statement and declared by an Extended DECLARE statement.	page 65
Dynamic or Extended Statement	CLOSE	Closes an open cursor.	page 66
Dynamic Statement	COMMENT ON	Replaces or adds a comment to the description of a table, view, or column.	*DB2 Server for VSE & VM SQL Reference*
Dynamic or Extended Statement	COMMIT	Terminates a logical unit of work and commits the database changes made by that logical unit of work.	page 67

Figure 18 (Page 2 of 4). RXSQL Requests

Request Type	**RXSQL Request**	**Function**	**Refer To**
Dynamic Statement	CONNECT	Switches application servers, passes a new authorization ID to the database manager or queries the current authorization id and application server.	page 68
Dynamic Statement	CREATE INDEX	Defines an index on a table.	*DB2 Server for VSE & VM SQL Reference*
Extended Statement	CREATE PACKAGE	Creates an empty package into which SQL statements can be prepared.	page 69
Dynamic Statement	CREATE SYNONYM	Defines an alternate name for a table or view.	*DB2 Server for VSE & VM SQL Reference*
Dynamic Statement	CREATE TABLE	Defines a table.	*DB2 Server for VSE & VM SQL Reference*
Dynamic Statement	CREATE VIEW	Defines a view of one or more tables or views.	*DB2 Server for VSE & VM SQL Reference*
Dynamic Statement	Dynamic DECLARE	Gives a *cursor_name* to a previously dynamically prepared INSERT or SELECT statement.	page 72
Extended Statement	Extended DECLARE	Gives a name to a statement in a package.	page 73
Dynamic Statement	DELETE	Deletes zero or more rows from a table.	*DB2 Server for VSE & VM SQL Reference*
Dynamic Statement	Dynamic DESCRIBE	Returns the names, data types, CCSIDs, and labels of the columns referenced in a prepared SELECT statement.	page 74
Extended Statement	Extended DESCRIBE	Returns the names, data types, CCSIDs, and labels of the columns referenced in a SELECT statement that has been prepared into a package.	page 76
Dynamic Statement	DROP	Deletes a table, index, view, synonym, dbspace, or package.	*DB2 Server for VSE & VM SQL Reference*
Extended Statement	DROP STATEMENT or DROPSTMT	Deletes a statement from a package.	page 78
Dynamic Statement	Dynamic EXECUTE	Executes an SQL statement that was defined by a previous Dynamic PREPARE statement.	page 79
Extended Statement	Extended EXECUTE	Executes an SQL statement in a package.	page 80
Dynamic Statement	EXECUTE IMMEDIATE or EXEC	Submits an SQL statement to be dynamically prepared and executed by the database manager.	page 81

Figure 18 (Page 3 of 4). RXSQL Requests

Request Type	**RXSQL Request**	**Function**	**Refer To**
Dynamic Statement	EXPLAIN	Obtains information about the expected structure and execution performance of a DELETE, INSERT, SELECT or UPDATE statement.	*DB2 Server for VSE & VM SQL Reference*
Dynamic or Extended Statement	FETCH	Assigns values of a row of a result table to variables provided by your program.	page 82
Dynamic Statement	GRANT	Grants system authorities as well as privileges on packages, tables, or views.	*DB2 Server for VSE & VM SQL Reference*
Dynamic Statement	INSERT	Inserts one or more rows into a table.	*DB2 Server for VSE & VM SQL Reference*
Dynamic Statement	LABEL ON	Replaces or adds a label on the description of a table, view, or column.	*DB2 Server for VSE & VM SQL Reference*
Dynamic Statement	LOCK DBSPACE	Either prevents concurrent processes from changing a dbspace or prevents concurrent processes from using a dbspace.	*DB2 Server for VSE & VM SQL Reference*
Dynamic Statement	LOCK TABLE	Either prevents concurrent processes from changing a table or prevents concurrent processes from using a table.	*DB2 Server for VSE & VM SQL Reference*
RXSQL Command	NAMES	Returns all the *cursor_names* and *statement_names* known to RXSQL from previous Dynamic PREPARE, Dynamic DECLARE, and Extended DECLARE statements.	page 84
RXSQL Command	OP	Issues an operator command and returns the result in REXX variables.	page 85
Dynamic or Extended Statement	OPEN	Opens a cursor associated with a SELECT or an INSERT statement.	page 86
Dynamic Statement	Dynamic PREPARE or PREP	Prepares an SQL statement for a subsequent Dynamic CALL, Dynamic DECLARE, Dynamic EXECUTE, or OPEN statement.	page 88
Extended Statement	Extended PREPARE	Prepares an SQL statement into a package.	page 90
RXSQL Command	PURGE	Clears specified statements from RXSQL.	page 94
Dynamic or Extended Statement	PUT	Inserts a row of data into a table.	page 95
Dynamic Statement	REVOKE	Revokes system authorities as well as privileges on packages, tables or views.	*DB2 Server for VSE & VM SQL Reference*
Dynamic or Extended Statement	ROLLBACK	Terminates a logical unit of work and backs out the database changes made by that unit of work.	page 98

Figure 18 (Page 4 of 4). RXSQL Requests

Request Type	RXSQL Request	Function	Refer To
RXSQL Command	SQLDATE	Sets or returns the date format.	page 99
RXSQL Command	SQLISL	Sets or returns the isolation level.	page 100
RXSQL Command	SQLTIME	Sets or returns the time format.	page 101
RXSQL Command	STATE	Returns the type and state values of the SQL statement or cursor you specify.	page 102
RXSQL Command	STMT	Returns the statement associated with a given *statement_name* or *cursor_name*.	page 103
RXSQL Command	TRACE	Sets the trace level for selected functions in RXSQL.	page 104
Dynamic Statement	UPDATE	Updates the values of one or more columns in zero or more rows of a table.	*DB2 Server for VSE & VM SQL Reference*
Dynamic Statement	UPDATE STATISTICS	Updates statistics on tables and indexes in system catalogs.	*DB2 Server for VSE & VM SQL Reference*
Extended Statement	XCALL	Executes an SQL statement in a package that does not reference REXX host variables.	page 106
Extended Statement	XPREP	Prepares an SQL statement in a package.	page 107

ACQUIRE DBSPACE

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

ALTER DBSPACE

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

ALTER TABLE

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

Dynamic CALL

```
►►──CALL──statement_name──┬──────────────────────────────────────────┬──────────────►◄
                          │ ┌─USING─┐                                 │
                          └─┴───────┴──input_rexx_host_variable_list─┘
```

The CALL statement is provided for compatibility with previous versions of DB2 RXSQL. It is functionally interchangeable with the Dynamic EXECUTE statement, although its syntax is slightly different. For a full description of this command, refer to “Dynamic EXECUTE” on page 79.

Examples

Example 1

Execute the prepared statement UPDATE_JOB.

```
'RXSQL CALL UPDATE_JOB'
```

Example 2

Execute the statement RAISE_JOB using a REXX host variable with a variable qualifier.

```
raise_stmt = 'UPDATE EMPVIEW SET SALARY = SALARY * 1.08',
   'WHERE JOB = ?'

'EXECSQL PREPARE RAISE_JOB FROM' raise_stmt

'RXSQL CALL RAISE_JOB USING :job(CHAR(8))'
```

The REXX interpreter resolves *raise_stmt* before the request is passed to RXSQL.

Extended CALL

```
►►──CALL──statement_name──┬─────────────────────────────────────────┬──────────────►
                          │ ┌─USING─┐                               │
                          └─┴───────┴──input_rexx_host_variable_list─┘

►──┬─────────────────────────────────────────────┬──────────────────────────────►◄
   └─INTO───┬─output_rexx_host_variable_list─┬───┘
            └─rexx_host_stem_name────────────┘
```

The Extended CALL statement executes an SQL statement previously defined by an Extended PREPARE or XPREP statement and declared by an Extended DECLARE statement. The Extended DECLARE statement must be issued before the Extended CALL statement to associate a *statement_name* with the prepared statement.

statement_name
: The name given by an Extended DECLARE statement to the SQL statement to be issued.

input_rexx_host_variable_list
: One or more *rexx_host_variables* that supply the input values for the SQL statement.

output_rexx_host_variable_list
: One or more *rexx_host_variables* that receive the values of a row of the result table formed by a Single Row Extended PREPARE statement.

  Place holders may be used to discard unwanted values.

rexx_host_stem_name
: A REXX stem variable that receives all of the values of the row of the result table.

States

Required Initial State	Resulting State
EXTENDED-DYNAMIC UNDECLARED	EXTENDED-DYNAMIC UNDECLARED
EXTENDED-DYNAMIC DECLARED	EXTENDED-DYNAMIC DECLARED

Input

If there are parameter markers in the prepared SELECT statement, then the *input_rexx_host_variable_list* after the USING clause is used to resolve the parameter markers.

Example

Declare and execute the UPDATE_SALARY statement. It is assumed that the UPDATE statement issued by this program has been prepared into the package in a previously run program.

```
'EXECSQL DECLARE UPDATE_SALARY FOR :stmt_number IN RAISE_SALARY'

'EXECSQL CALL UPDATE_SALARY'
```

CLOSE

```
►►──CLOSE──┬─cursor_name──┬──────────────────────────────────────────────────►◄
           └─prepare_name─┘
```

The CLOSE statement closes a cursor previously opened by an OPEN statement. CLOSE leaves the cursor ready to be opened.

cursor_name
The name of the declared cursor to be closed.

If a Dynamic DECLARE has identified a *cursor_name* for a dynamic prepared SELECT or INSERT statement, then CLOSE must use the *cursor_name* defined in the Dynamic DECLARE.

prepare_name
The name given to a statement in a Dynamic PREPARE.

If a Dynamic DECLARE statement has not been executed for a dynamic prepared SELECT or INSERT statement, then CLOSE must use the *prepare_name* defined in the Dynamic PREPARE.

States

Required Initial State	Resulting State
DYNAMIC OPEN	DYNAMIC PREPARED
EXTENDED-DYNAMIC OPEN	EXTENDED-DYNAMIC DECLARED

Example

Close the cursor SELECT_EMPLOYEE.

```
'EXECSQL CLOSE SELECT_EMPLOYEE'
```

COMMENT ON

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

COMMIT

```
             ┌─WORK─┐
►►──COMMIT───┴──────┴───┬─────────┬──────────────────────────────────────────►◄
                        └─RELEASE─┘
```

The COMMIT statement commits all changes made to the database since the beginning of the LUW or since the last COMMIT or ROLLBACK statement. All locks in the database acquired during the LUW are released. COMMIT severs the database connection if you specify the RELEASE option.

States

Required Initial State	Resulting State
DYNAMIC DECLARED-ONLY	DYNAMIC DECLARED-ONLY
DYNAMIC UNPREPARED	DYNAMIC UNPREPARED
DYNAMIC PREPARED	DYNAMIC UNPREPARED
DYNAMIC OPEN	DYNAMIC UNPREPARED
EXTENDED-DYNAMIC UNDECLARED	EXTENDED-DYNAMIC UNDECLARED
EXTENDED-DYNAMIC DECLARED	EXTENDED-DYNAMIC UNDECLARED
EXTENDED-DYNAMIC OPEN	EXTENDED-DYNAMIC UNDECLARED

Notes

1. DB2 RXSQL does not explicitly close any open cursors before issuing the COMMIT in the database manager.
2. All *statement_names* and *cursor_names* are retained by DB2 RXSQL after the COMMIT.
3. Do not execute any RXSQL statements between a COMMIT RELEASE statement and the invocation of an SQLINIT.

   DB2 RXSQL retrieves DATE and TIME specifications from the LASTING GLOBALV file the first time DB2 RXSQL is invoked or at the first invocation after a COMMIT RELEASE or ROLLBACK RELEASE. If your program issues a COMMIT RELEASE or ROLLBACK RELEASE and you wish to change DATE or TIME specifications using SQLINIT, you must do so before the next RXSQL statement is executed.

   For example, if your execution sequence is

```
COMMIT RELEASE

EXECSQL ...

SQLINIT DATE(JIS)
```

DB2 RXSQL will not retrieve the new date format, JIS, but will continue using the same specification you had before.

The correct sequence in this case is as follows:

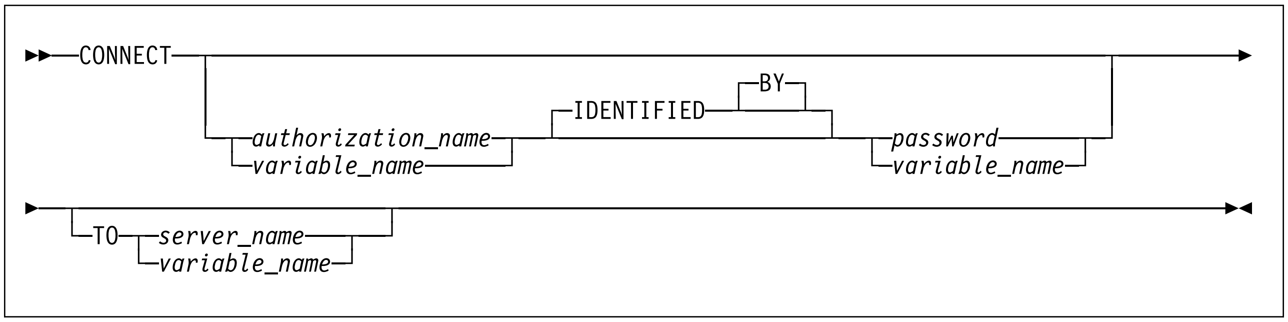
```
COMMIT RELEASE

SQLINIT DATE(JIS)

EXECSQL ...
```

Example

Commit all changes made to the database.

```
'EXECSQL COMMIT'
```

CONNECT



The CONNECT statement lets you

- pass a new authorization ID with a password to the database manager to change the identity of the user,
- switch application servers or
- query the database manager for the current identity of the user and the server to which the user is connected.

For more information about this statement refer to the *DB2 Server for VSE & VM SQL Reference* manual.

authorization_name
An SQL authorization ID.

password
The password that SQL uses to validate the user ID.

server_name
The name of the new application server to which you are connecting.

variable_name
A REXX variable name with a mandatory preceding colon. RXSQL fetches the value of *variable_name* and passes the value to the database manager when the CONNECT statement is executed. The value of *variable_name* must conform to the coding rules of the metavariable for which it is being substituted.

Output

After you issue the CONNECT statement with no parameters, the SQLCA variables `SQLERRM`, `SQLERRMC` and `SQLERRP` are set. Both `SQLERRM` and `SQLERRMC` contain two message tokens separated by the hex character `'FF'`x. The first token is the current SQL authorization ID, and the second one is the application server name. `SQLERRP` contains the product code. The REXX statement for extracting the values from `SQLERRM` looks like:

```
Parse Var sqlerrm sqlid 'FF'x server .
```

Notes

1. If you use the CONNECT statement to switch between application servers, the RXSQL package must be installed on each application server. See the *DB2 REXX SQL for VM/ESA Installation* manual for instructions on installing the RXSQL package.
2. If you are switching between application servers while using RXSQL and the server connection is severed by a severe database error, you must issue another CONNECT statement to re-connect you to the application server to which you were connected before the connection was severed.

Examples

The following examples produce the same result.

Example 1

Connect as SQLDBA.

```
'RXSQL CONNECT SQLDBA IDENTIFIED BY SQLDBAPW'
```

Example 2

Issue a CONNECT using the REXX variables *userid* and *password*.

```
 userid   = 'SQLDBA'
 password = 'SQLDBAPW'
'EXECSQL CONNECT :userid IDENTIFIED BY :password'
```

CREATE INDEX

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

CREATE PACKAGE

```
         ┌─PACKAGE─┐
►►──CREATE──┤         ├──┬─package_name──┬──────────────────────────────►
         └─PROGRAM─┘  └─variable_name─┘

►──┬─────────────────────────────────────────────┬────────────────────►◄
   │         ┌─OPTIONs─┐            ┌─,──────┐     │
   └─┬─USING─┴─────────┴─┬──┬─▼─option─┴──┬─┘
     └───────────────────┘  └─variable_name─┘
```

The CREATE PACKAGE statement creates an empty package. The package is stored in the database after you issue a COMMIT statement.

CREATE PROGRAM is equivalent to CREATE PACKAGE and is provided for compatibility with prior versions of RXSQL.

package_name
The name of the package to be created for storing prepared SQL statements.

option
Refer to the *DB2 Server for VSE & VM SQL Reference* manual for an explanation of all the options for the CREATE PACKAGE statement.

Do not code either the DESCRIBE or the NODESCRIBE option on the RXSQL CREATE PACKAGE statement. RXSQL always adds the DESCRIBE option to the SQL CREATE PACKAGE statement to allow DB2 RXSQL to fetch data correctly. If you code either DESCRIBE or NODESCRIBE, the DESCRIBE which RXSQL adds will result in duplicate or conflicting options. In this case, the database manager will return an error condition.

variable_name
A REXX variable name with a mandatory preceding colon. RXSQL fetches the value of *variable_name* and passes the value to the database manager when the CREATE PACKAGE statement is executed. The value of *variable_name* must conform to the coding rules of the metavariable for which it is being substituted.

Notes

1. A new package is created in the database after the COMMIT statement has executed successfully. Issuing a ROLLBACK statement will prevent this.
2. You can issue COMMIT for a package created with the MODIFY option even if the package contains no statements.
3. The Extended PREPARE and XPREP statement will prepare an SQL statement into a section and add it to a package.
4. The DROP STATEMENT statement will delete a section from a package created with the MODIFY option.
5. The way to use the CREATE PACKAGE statement along with other Extended Dynamic statements is described in “Using Extended Dynamic Statements in DB2 RXSQL” on page 33.

Examples

The packages in the three following examples are created using the ISOL(CS) and BLOCK options to minimize locking and to improve performance by inserting and retrieving rows in groups.

Example 1: Basic non-modifiable package

Create a non-modifiable package. Statements must be added using the Basic Extended PREPARE or XPREP within the same LUW in which the package is created.

```
'RXSQL CREATE PACKAGE CHANGE_STAFF USING BLOCK ISOL(CS)'
'RXSQL XPREP IN CHANGE_STAFF :update_view'
'RXSQL COMMIT'
```

A number is returned to the REXX program in the variable `SQLSTMTN` after the XPREP indicating which section the statement was prepared into. This number can be referenced on subsequent RXSQL requests.

Example 2: Non-modifiable package with empty section

Create a non-modifiable package, and add an empty section. Later, SQL statements can be temporarily prepared into this empty section to dynamically execute an SQL statement.

```
'EXECSQL CREATE PACKAGE NEW_EMPLOYEE USING BLOCK ISOL(CS) '
'EXECSQL PREPARE FROM NULL SETTING :section-number IN NEW_EMPLOYEE '
'EXECSQL COMMIT '
```

A number is returned to the REXX program in the variable `section_number` after the PREPARE statement indicating which section is the empty one. This number is then referenced in subsequent Temporary Extended PREPARE statements.

Example 3: Modifiable package

Create a modifiable package. Statements may be added to the package and deleted from the package even after the LUW in which the package was created ends.

```
'EXECSQL CREATE PACKAGE RAISE_SALARY USING OPTIONS BLOCK MODIFY ISOL(CS)'
'EXECSQL COMMIT '
```

CREATE SYNONYM

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

CREATE TABLE

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

CREATE VIEW

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

Dynamic DECLARE

```
                           ┌CURSOR┐  ┌FOR┐
►►──DECLARE──cursor_name───┴──────┴──┴───┴──prepare_name──────────────────────────►◄
```

The Dynamic DECLARE statement gives a *cursor_name* to an SQL INSERT or SELECT statement. Once you declare a cursor, you must use the *cursor_name* in subsequent OPEN, FETCH, PUT and CLOSE statements.

The use of the Dynamic DECLARE statement is optional. If you do not declare a cursor, then you must use the *prepare_name* on the OPEN, FETCH, PUT and CLOSE statements. The Dynamic DECLARE statement can be placed before or after the Dynamic PREPARE statement in your program.

cursor_name
The name which identifies the SQL cursor.

prepare_name
The name of the SQL INSERT or SELECT statement for which you are declaring a cursor.

States

Required Initial State	**Resulting State**
none	DYNAMIC DECLARED-ONLY
DYNAMIC UNPREPARED	DYNAMIC UNPREPARED
DYNAMIC PREPARED	DYNAMIC PREPARED
DYNAMIC OPEN	DYNAMIC OPEN

Note

1. The *cursor_name* on a dynamic DECLARE statement cannot be the same as another prepared or declared *prepare_name*, *statement_name*, or *cursor_name*. You must issue an explicit PURGE statement before reusing the name. For example, the third statement will generate an error:

```
'PREPARE JACK FROM :jack_string '

'PREPARE JILL FROM :jill_string '

'DECLARE JILL CURSOR FOR JACK '
```

Example

Prepare a SELECT statement, then DECLARE a cursor for the prepared statement.

```
select_ed = "SELECT * FROM EMPVIEW WHERE EDLEVEL >= 16"

'EXECSQL PREPARE SELECT_EDLEVEL FROM :select_ed'

'EXECSQL DECLARE EDLEVEL_16 CURSOR FOR SELECT_EDLEVEL'
```

Extended DECLARE

```
                                      ┌─CURSOR─┐  ┌─FOR─┐
►►──DECLARE──┬─cursor_name────┬───────┴────────┴──┴─────┴──────────────────────►
             └─statement_name─┘

                      ┌─IN─┐
►──┬─section_number─┬─┴────┴──┬─package_name──┬───────────────────────────────►◄
   └─variable_name──┘         └─variable_name─┘
```

The Extended DECLARE statement has a dual role.

- It is primarily used to declare a cursor for an SQL statement that was prepared into a package. The *cursor_name* is referenced in subsequent OPEN, FETCH, PUT or CLOSE statements. In this case, the SQL statement was an INSERT or SELECT statement.
- It can also be used to give a *statement_name* to a prepared SQL statement for subsequent reference by an Extended CALL statement. This second use of Extended DECLARE is unique to RXSQL, and is not supported by other host languages supported by SQL.

cursor_name
The name which identifies the cursor.

statement_name
The name which identifies the SQL statement.

section_number
An integer value representing the statement number of the SQL statement that was assigned by the database manager when an Extended PREPARE or XPREP statement was executed.

package_name
The name of the package in which the SQL statement is stored.

variable_name
A REXX variable name with a mandatory preceding colon. RXSQL fetches the value of *variable_name* and passes the value to the database manager when the Extended DECLARE statement is executed. The value of *variable_name* must conform to the coding rules of the metavariable for which it is being substituted.

States

Required Initial State	Resulting State
none	EXTENDED-DYNAMIC UNDECLARED
EXTENDED-DYNAMIC UNDECLARED	EXTENDED-DYNAMIC UNDECLARED
EXTENDED-DYNAMIC DECLARED	EXTENDED-DYNAMIC DECLARED
EXTENDED-DYNAMIC OPEN	EXTENDED-DYNAMIC OPEN

Notes

1. An Extended DECLARE statement cannot use the same name as that of another prepared or declared statement unless
   - the declaration matches the previous declaration exactly, or
   - the state of the previously declared statement is EXTENDED-DYNAMIC UNDECLARED.

   You must end the LUW and issue an explicit PURGE statement before reusing the name.
2. When your program issues an Extended DECLARE statement for a cursor operation, DB2 RXSQL does not issue a DECLARE CURSOR statement to the database manager until your program issues an OPEN statement for the cursor.
3. The limit to the number of declared names that can exist at any one time is much larger than the limit of 40 dynamic prepared statements and is determined by the database manager.
4. RXSQL does not support the DECLARE CURSOR WITH HOLD statement.

Example

Declare the VIEW_EMPLOYEE cursor for the statement in the package RAISE_SALARY identified by the *variable_name stmt_number*.

```
'EXECSQL DECLARE VIEW_EMPLOYEE CURSOR FOR :stmt_number IN RAISE_SALARY'
```

DELETE

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

Dynamic DESCRIBE

```
                                        ┌─USING─┐   ┌─NAMES──┐
►►──DESCRIBE──┬─cursor_name──┬─────────┴───────┴───┼────────┼──────────────────────────►◄
              └─prepare_name─┘                      ├─LABELS─┤
                                                    ├─ANY────┤
                                                    └─BOTH───┘
```

The Dynamic DESCRIBE statement returns the names, labels, CCSIDs, and data types of the columns of a dynamically prepared statement, or a prepared and declared SELECT statement. The returned values are put into the REXX stem variables `SQLDAN.`, `SQLDAT.`, `SQLDAC.`, and `SQLDAL.`.

cursor_name
: Name given by a Dynamic DECLARE to a prepared SELECT statement.

prepare_name
: Name given by a Dynamic PREPARE to a SELECT statement.

  The *prepare_name* may be used only if a *cursor_name* has not been declared for it.

States

Required Initial State	Resulting State
DYNAMIC UNPREPARED	DYNAMIC PREPARED
DYNAMIC PREPARED	DYNAMIC PREPARED
DYNAMIC OPEN	DYNAMIC OPEN

Output

All REXX variables with the prefix `SQLDAN.`, `SQLDAT.`, `SQLDAC.`, and `SQLDAL.` are reset to empty strings before the result is set. RXSQL sets the `SQLDAN.0`, `SQLDAT.0`, `SQLDAC.0`, and `SQLDAL.0` variables to the number of columns in the *select_list*. This is also the number of result columns that would be returned if the prepared or declared SELECT statement is executed.

For each result column i, the following group of variables is set:

- `SQLDAN.`i is set with the column NAMES or LABELS, depending on what is specified in the USING clause.
- `SQLDAT.`i is set with the SQL data type attributes for the corresponding column. There are two components to the data type attributes; the SQL data type, including the length or scale and precision where appropriate, and an indication of whether the column will accept NULL values.
- `SQLDAC.`i is set with the CCSID number of the column. If the datatype is not eligible for a CCSID value, its corresponding `SQLDAC.`i variable is an empty string.
- `SQLDAL.`i is set with the column label only if the BOTH option was specified.

For example, a DESCRIBE statement on a SELECT statement that has two columns in the *select_list* would cause the variables to be set similar to the following:

```
sqldan.0  = 2
sqldat.0  = 2
sqldac.0  = 2
sqldal.0  = 2
sqldan.1  = LASTNAME
sqldat.1  = V 15
sqldac.1  = 500
sqldan.2  = SALARY
sqldat.2  = D 9 2 N
sqldac.2  = ''
```

If BOTH option is specified:

```
sqldal.1  = Last_Name
sqldal.2  = Salary
```

In this example, you can see that column `LASTNAME` does not accept NULL values because there is no trailing N in `SQLDAT.1` while the column `SALARY` does allow NULL values.

The abbreviations in the data type attributes are listed in Figure 19 on page 76.

SQL Data Type Abbreviations	SQL Data Type	Description
I	INTEGER	31 bit binary integer
S	SMALLINT	15-bit binary integer
R	FLOAT	Single precision, floating-point number
F	FLOAT	Double precision, floating-point number
D *m n*	DECIMAL	*m* is the precision; *n* is the scale
C *n*	CHAR	Fixed length character; *n* is the length
V *n*	VARCHAR	Variable length character; *n* is the maximum length
L *n*	LONG VARCHAR	Variable length character; *n* is the maximum length
G *n*	GRAPHIC	Fixed length graphic; *n* is the number of 2-byte characters
VG *n*	VARGRAPHIC	Variable length graphic; *n* is the maximum length
LG *n*	LONG VARGRAPHIC	Variable length graphic; *n* is the maximum length
DT *n*	DATE	*n* is the length
TM *n*	TIME	*n* is the length
TS *n*	TIMESTAMP	*n* is the length
ZD *m n*	ZONED DECIMAL	*m* is the precision; *n* is the scale. This data type is not valid on DB2 Server for VM application servers.
Notes: 1. For data limits, please refer to the *IBM SQL Reference, Version 2, Volume 1* manual. 2. An `N` following any of the above abbreviations means that the column allows NULL values.		

Figure 19. DB2 RXSQL Abbreviations for Data Type Attributes on DESCRIBE

Note

1. If the state of the statement is DYNAMIC UNPREPARED because you issued a COMMIT or ROLLBACK after you issued the original Dynamic PREPARE statement, DB2 RXSQL prepares the SQL statement again before the DESCRIBE statement is passed to the database manager.

Example

Return information about the prepared SELECT statement.

```
select_stmt = 'SELECT * FROM' table_name

'EXECSQL PREPARE SELECT_TABLE FROM :select_stmt'

'EXECSQL DESCRIBE SELECT_TABLE USING ANY '
```

Extended DESCRIBE

```
►►──DESCRIBE───┬─cursor_name──────────────────────────────────────┬──────────────────►
               ├─statement_name───────────────────────────────────┤
               │                         ┌─IN─┐                   │
               └──┬─section_number─┬─────┴────┴───┬─package_name──┤
                  └─variable_name──┘              └─variable_name─┘

          ┌─USING─┐   ┌─NAMES───┐
►─────────┴───────┴───┼─────────┼──────────────────────────────────────────────────►◄
                      ├─LABELS──┤
                      ├─ANY─────┤
                      └─BOTH────┘
```

The Extended DESCRIBE statement returns the names, labels, CCSIDs, and data types of the columns of a prepared SELECT statement in a package. The REXX stem variables `SQLDAN.`, `SQLDAT.`, `SQLDAC.`, and `SQLDAL.` are set with the descriptor information in the same manner described under the Dynamic DESCRIBE statement.

cursor_name
: Name given by an Extended DECLARE statement to the SELECT statement to be described.

statement_name
: Name given by an Extended DECLARE statement to the Single Row Extended PREPARE statement to be described.

section_number
: An integer value representing the statement number in the package where the SELECT statement to be described is stored. It was assigned by the database manager when an Extended PREPARE statement was executed.

package_name
: The name of the package in which the SELECT statement to be described is stored.

variable_name
: A REXX variable name with a mandatory preceding colon. RXSQL fetches the value of *variable_name* and passes the value to the database manager when the Extended DESCRIBE statement is executed. The value of *variable_name* must conform to the coding rules of the metavariable for which it is being substituted.

States

Required Initial State	**Resulting State**
none	none
EXTENDED-DYNAMIC UNDECLARED	EXTENDED-DYNAMIC UNDECLARED
EXTENDED-DYNAMIC DECLARED	EXTENDED-DYNAMIC DECLARED
EXTENDED-DYNAMIC OPEN	EXTENDED-DYNAMIC OPEN

Output

Same as **Output** for the “Dynamic DESCRIBE” on page 74.

Example

Return information about the declared statement into DB2 RXSQL variables.

```
'EXECSQL DECLARE SELECT_EMPLOYEE CURSOR FOR 3 IN NEW_STAFF'

'EXECSQL DESCRIBE SELECT_EMPLOYEE'
```

DROP

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

DROP STATEMENT or DROPSTMT

```
                                                  ┌─IN─┐
►►──┬─DROP STATEMENT─┬───┬─section_number─┬───────────┬─package_name──┬──────────────────────►◄
    └─DROPSTMT───────┘   └─variable_name──┘           └─variable_name─┘
```

The DROP STATEMENT statement selectively deletes a section and its associated statement from a package. DROP STATEMENT applies only to packages created by a CREATE PACKAGE statement with the MODIFY option.

DROPSTMT is provided for compatibility with previous versions of RXSQL and is functionally equivalent to DROP STATEMENT.

section_number
: An integer value representing the section to be dropped. This number was assigned by the database manager when an Extended PREPARE statement was executed.

package_name
: The name of the package in which the prepared SQL statement to be dropped is stored.

variable_name
: A REXX variable name with a mandatory preceding colon. RXSQL fetches the value of *variable_name* and passes the value to the database manager when the DROP STATEMENT or DROPSTMT statement is executed. The value of *variable_name* must conform to the coding rules of the metavariable for which it is being substituted.

Examples

Example 1

Drop the statement identified by the value of the REXX variable *stmt_num_1* from the package UPDATE_SALARY.

```
'RXSQL DROPSTMT' stmt_num_1 'IN UPDATE_SALARY'
```

Example 2

Drop the statement identified by the section variable *stmt_number* from the package REVIEW_DEPT.

```
'EXECSQL DROP STATEMENT :stmt_number IN REVIEW_DEPT'
```

Dynamic EXECUTE

```
►►──EXECUTE──prepare_name──┬──────────────────────────────────────┬──────────────►◄
                           └─USING──input_rexx_host_variable_list─┘
```

The Dynamic EXECUTE statement executes a prepared SQL statement.

input_rexx_host_variable_list
: A list of one or more *rexx_host_variables* that supply the input values for the SQL statement.

prepare_name
: The name given by a Dynamic PREPARE statement to the SQL statement to be issued. It must not represent a SELECT statement.

States

Required Initial State	Resulting State
DYNAMIC UNPREPARED	DYNAMIC PREPARED
DYNAMIC PREPARED	DYNAMIC PREPARED

Input

If you include an *input_rexx_host_variable_list* on the EXECUTE statement, RXSQL retrieves the values of the input variables. These values are used to replace the parameter markers in the SQL statement specified by the Dynamic PREPARE statement. If the SQL statement contained variable names rather than parameter markers when it was prepared, the *input_rexx_host_variable_list* will override the variable names specified in the prepared SQL statement.

Notes

1. If the initial state of the statement is DYNAMIC UNPREPARED because a COMMIT or ROLLBACK was issued after the statement was prepared, DB2 RXSQL prepares the SQL statement again before issuing the EXECUTE statement.
2. *Prepare_name* must not be a *cursor_name* defined by a Dynamic DECLARE.

Examples

Example 1

Execute the prepared statement.

```
'EXECSQL EXECUTE INSERT_EMPLOYEE'
```

Example 2

Execute the statement DELETE_EMPLOYEE using a REXX variable with a variable qualifier.

```
delete_stmt = 'DELETE FROM RXEMP WHERE EMPNO = ?'

'EXECSQL PREPARE DELETE_EMPLOYEE FROM :delete_stmt'

'EXECSQL EXECUTE DELETE_EMPLOYEE USING :emp_number(CHAR(6))'
```

Extended EXECUTE

```
                                    ┌─IN─┐
►►──EXECUTE──┬─section_number─┬─────┴────┴──┬─package_name──┬──────────────────────────►
             └─variable_name──┘              └─variable_name─┘

►──┬─────────────────────────────────────────┬────────────────────────────────────────►
   └─USING──input_rexx_host_variable_list──┘

►──┬──────────────────────────────────────────────┬──────────────────────────────────►◄
   └─INTO──┬─output_rexx_host_variable_list─┬──┘
           └─rexx_host_stem_name──────────────┘
```

The Extended EXECUTE statement executes an SQL statement that was previously inserted into a package by an Extended PREPARE statement. It allows you to specify both input and output variables.

section_number
: An integer value representing the statement number of the SQL statement to be issued. It was assigned by the database manager when an Extended PREPARE statement was executed.

package_name
: The name of the package in which the prepared SQL statement to be issued is stored.

input_rexx_host_variable_list
: One or more *rexx_host_variables* that supply the input values for the SQL statement.

output_rexx_host_variable
: One or more *rexx_host_variables* that receive the result values of a Single Row Extended PREPARE statement.

: Place holders may be used to discard unwanted values.

rexx_host_stem_name
: A REXX stem variable name that receives all of the values of the result table row.

variable_name
A REXX variable name with a mandatory preceding colon. RXSQL fetches the value of *variable_name* and passes the value to the database manager when the Extended EXECUTE statement is executed. The value of *variable_name* must conform to the coding rules of the metavariable for which it is being substituted.

States

Required Initial State	Resulting State
none	none
EXTENDED-DYNAMIC DECLARED	EXTENDED-DYNAMIC DECLARED
EXTENDED-DYNAMIC UNDECLARED	EXTENDED-DYNAMIC UNDECLARED

Input

If there are parameter markers in the prepared SELECT statement, then the USING *input_rexx_host_variable* clause is used to resolve the parameter markers.

Example

Issue statement 3 in the package NEW_EMPLOYEE.

```
'EXECSQL EXECUTE 3 IN NEW_EMPLOYEE',
             'USING :emp_data INDICATOR :emp_data_ind'
```

EXECUTE IMMEDIATE or EXEC

```
►►──┬─EXECUTE IMMEDIATE─┬──┬─sql_statement─┬──────────────────────────────────────►◄
    └─EXEC──────────────┘  └─variable_name─┘
```

The EXECUTE IMMEDIATE statement submits an SQL statement to be dynamically prepared and executed by the database manager.

sql_statement
Any SQL statement that is allowed by the SQL statement EXECUTE IMMEDIATE. Refer to the *DB2 Server for VSE & VM SQL Reference* manual for a complete list of allowable SQL statements.

Since RXSQL passes the SQL statement to the database manager with no argument substitution and no data transfer, the statement must not contain *variable_names* or parameter markers.

variable_name
A REXX variable name with a mandatory preceding colon. RXSQL fetches the value of *variable_name* and passes the value to the database manager when the EXECUTE IMMEDIATE statement is executed. The value of *variable_name* must conform to the coding rules of the metavariable for which it is being substituted.

Example

Submit the SQL statement CREATE VIEW to the database manager to be executed.

```
create_view = 'CREATE VIEW EMP_VIEW',
   '(EMPNO, FIRSTNME, MIDINIT, LASTNAME,',
    'WORKDEPT, PHONENO, JOB, EDLEVEL)',
   'AS SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT,',
            'PHONENO, JOB, EDLEVEL',
      'FROM RXEMP'

'EXECSQL EXECUTE IMMEDIATE :create_view'
```

EXPLAIN

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

FETCH

```
                                    ┌─INTO─┐
►►──FETCH───┬─cursor_name──┬────────┴──────┴────┬─output_rexx_host_variable_list─┬──────────────►◄
            └─prepare_name─┘                    └─rexx_host_stem_name────────────┘
```

The FETCH statement obtains the next row from the result table of the SELECT statement that was processed by an OPEN statement.

cursor_name
: The name of the open cursor representing a SELECT statement.

  If a Dynamic DECLARE has been issued, your program must use the *cursor_name* on the Dynamic FETCH statement.

prepare_name
: The name given to an SQL SELECT statement by a Dynamic PREPARE.

  The *prepare_name* can be used only if a Dynamic PREPARE has given a *prepare_name* to the SELECT statement and a Dynamic DECLARE has not associated a cursor with the SELECT statement before the statement was opened.

output_rexx_host_variable_list
: One or more *rexx_host_variables* that receive the values of the columns of a result table row.

  If the number of columns in the active set or result table is

  1. greater than the number of *rexx_host_variables* you have provided,
     - all the variables will be set
     - the extra columns will be discarded by RXSQL
     - the EXECSQL invocation will return a warning indication
  2. less than the number of *rexx_host_variables* you have provided, the extra variables are not set.

Note: In other languages supported by the database manager the number of output variables must match the number of columns in the result table.

Placeholders may be used to discard unwanted columns from the active set.

rexx_host_stem_name
A REXX stem variable name that receives all of the values of the result table row.

States

Required Initial State	Resulting State
DYNAMIC OPEN	DYNAMIC OPEN
EXTENDED-DYNAMIC OPEN	EXTENDED-DYNAMIC OPEN

Output

The variables provided on the FETCH statement are set with the retrieved data.

Note

1. When the SQLCODE variable is equal to 100 (SQLSTATE 02000), the last FETCH executed tried to return a row beyond the last one in the active set and there are no more result rows.

Examples

The following examples illustrate a variety of ways in which data can be fetched using a cursor.

Example 1

Get data from selected columns of the result table into REXX variables. Indicator variables are used to see if the *job_title*, *phone_number*, and *sal* variables contain nulls. A variable qualifier is used on the *job_title* variable to ensure that it is returned using CCSID 037. Placeholders are used to avoid setting variables with unwanted data.

```
'EXECSQL FETCH SELECT_EMPLOYEE INTO :job_title :job_indicator(CCSID 037),',
   ':phone_number :phone_indicator,., :lname ,.,.,:sal :sal_indicator'
```

Example 2

Get data from selected columns of the result table into REXX variables.

```
'RXSQL FETCH EMPLOYEE_INFO :emp :fname :mid :lname :job'
```

Example 3

Get data from all columns of the result table into the `emp_stats.` stem variable.

```
'EXECSQL FETCH EMPLOYEE_INFO INTO :emp_stats.'
```

GRANT

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

INSERT

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

LABEL ON

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

LOCK DBSPACE

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

LOCK TABLE

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

NAMES

```
►►──NAMES──────────────────────────────────────────────────────────────►◄
```

The NAMES command returns all the *statement_names*, *prepare_names* and *cursor_names* known to DB2 RXSQL from Dynamic PREPARE statements and Dynamic and Extended DECLARE statements. This statement sets the REXX variable `RXSQLNAMES` to a string of names separated by blanks. You may then use each *cursor_name* or *statement_name* in a STATE request to determine the state of each statement or, in a STMT command, to obtain the assigned SQL statement.

Output

The `RXSQLNAMES` variable is set. When there are no names known to RXSQL, `RXSQLNAMES` will be set to a string of length zero (a null string).

Note

1. For dynamic declared and prepared statements which have a statement and cursor name, `RXSQLNAMES` is set with the statement name followed by the cursor name in parentheses.

Example

Display the names of all prepared and declared statements that have not been purged.

```
'EXECSQL NAMES'
Say rxsqlnames
```

Output similar to the following is displayed.

```
STATEMENT_1 STATEMENT_2(CURSOR_2) STATEMENT_3(CURSOR_3)
```

OP

```
►►──OP operator_command──────────────────────────────────────────────►◄
```

The OP command issues an operator command and returns the result in REXX variables. For example, the SHOW and COUNTER operator commands monitor the DB2 Server for VM system operation.

operator_command
The DB2 Server for VM operator command to be issued.

RXSQL passes the string of characters that follows the keyword OP, directly to the database manager. Refer to the *DB2 Server for VSE & VM Operation* manual for an explanation of the valid operator commands.

Output

`SQLOP.0` is set with the number of lines returned. `SQLOP.1` contains the first line, `SQLOP.2` contains the second line, and so on.

Notes

1. All command keywords must be presented from REXX to DB2 RXSQL in uppercase.
2. The OP command cannot be issued during an active LUW.
3. The OP command cannot be issued when using the DRDA protocol.
4. The OP command cannot issue an operator command that must be issued from the operator console.

Example

Issue the operator command SHOW USERS and display the contents of the stem variable `sqlop.`.

```
'EXECSQL OP SHOW USERS'
Do i=1 to sqlop.0
   Say sqlop.i
End
```

Output similar to the following is displayed.

```
Status of connected users:
  2  users are connected to application server.
  1  Users are active.
     User ID: MERCIER   SQL ID: MERCIER
  0  Users are waiting.
  1  Users are inactive.
     User ID: HDOBSON  SQL ID: HDOBSON
  4  Agents are available.
  3  User connections are available.
ARI0065I Operator command processing is complete.
```

OPEN

```
►►──OPEN──┬─cursor_name──┬──┬──────────────────────────────────────────┬─────────────►◄
          └─prepare_name─┘  │ ┌─USING─┐                                │
                            └─┴───────┴──input_rexx_host_variable_list─┘
```

The OPEN statement opens a cursor associated with either a declared or prepared SELECT statement (*query_cursor*) or a declared or prepared INSERT statement (*insert_cursor*).

For a *query_cursor*, when the OPEN statement is executed, the cursor is positioned before the first row of the result table defined by the SELECT statement. Each execution of a FETCH statement moves the cursor forward and retrieves the next row of the result table.

For an *insert_cursor*, when the OPEN statement is executed, block input is established on the tables. Each execution of a PUT statement moves the cursor forward and inserts a row into the input block.

cursor_name
: The name of the declared cursor to be opened.

  If a Dynamic DECLARE has identified a *cursor_name* for a dynamic prepared SELECT or INSERT statement, then OPEN must use the *cursor_name* defined in the Dynamic DECLARE.

prepare_name
: The name given to a statement in a Dynamic PREPARE.

  If a Dynamic DECLARE statement has not been executed for a dynamic prepared SELECT or INSERT statement, then OPEN must use the *prepare_name* defined in the Dynamic PREPARE.

input_rexx_host_variable_list
: One or more *rexx_host_variables* that supply input values for the SQL SELECT statement.

  If the prepared SQL statement is a SELECT statement, any parameter markers or *variable_names* in the SQL statement are resolved by RXSQL with values in the *input_rexx_host_variable_list* on the OPEN statement. If there is no *rexx_host_variable_list* supplied and *variable_names* are included in the dynamic prepared SQL statement, RXSQL retrieves the values of the variable names from REXX and these values are used when OPEN is executed.

If the prepared SQL statement is an INSERT statement, the USING clause is ignored. Any parameter markers or *variable_names* in the SQL statement will be resolved by RXSQL with values in the USING clause on the PUT statement.

States

Required Initial State	Resulting State
DYNAMIC UNPREPARED	DYNAMIC OPEN
DYNAMIC PREPARED	DYNAMIC OPEN
DYNAMIC OPEN	DYNAMIC OPEN
EXTENDED-DYNAMIC UNDECLARED	EXTENDED-DYNAMIC OPEN
EXTENDED-DYNAMIC DECLARED	EXTENDED-DYNAMIC OPEN
EXTENDED-DYNAMIC OPEN	EXTENDED-DYNAMIC OPEN

Input

RXSQL fetches the values of *input_rexx_host_variables* or the values of *variable_names* in the prepared SQL statement and passes them to the database manager as input values for SELECT.

Notes

1. If the state of a *cursor_name* for a cursor operation is EXTENDED-DYNAMIC UNDECLARED, DB2 RXSQL issues a DECLARE CURSOR statement to the database manager when your program issues an OPEN for the statement.
2. When using Extended Dynamic SQL, if you DECLARE a cursor, OPEN the cursor, PURGE the cursor, then DECLARE and OPEN the same cursor, the database manager will return an error condition. When DB2 RXSQL executes PURGE, the *cursor_name* is not invalidated by the database manager. The cursor is still active in the database manager when the second DECLARE and OPEN are executed, resulting in the error.

   Once the cursor is purged from DB2 RXSQL, you must end the LUW before issuing the second DECLARE and OPEN by issuing a COMMIT or ROLLBACK statement.
3. If the state of the statement is DYNAMIC UNPREPARED because you issued a COMMIT or ROLLBACK statement after you issued the original Dynamic PREPARE statement, the SQL statement is prepared again by RXSQL before the OPEN statement is executed.
4. If the state of the statement is DYNAMIC OPEN or EXTENDED-DYNAMIC OPEN and your program issues another OPEN statement, then DB2 RXSQL will close the statement before it executes the OPEN statement.

Example

In this part of a REXX program, data is read from the file EMPLOYEE INPUT and inserted into the table RXEMP. The INSERT statement is prepared with parameter markers which are resolved on the PUT statement. Variable qualifiers are used so the program does not have to surround the input values with: "'". The DECLARE statement is not used in this example.

```
insert_row = 'INSERT INTO RXEMP VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?)'

'EXECSQL PREPARE INSERT_EMPLOYEE FROM :insert_row'

'EXECSQL OPEN INSERT_EMPLOYEE'

/* Begin loop */

   /* Read data from file */
   'EXECIO 1 DISKR EMPLOYEE INPUT * (LIFO'
   /* Get the data into REXX variables */
   Parse Upper Pull emp fname mid lname wdpt wdpt_ind,
                    ph ph_ind hire hire_ind job job_ind,
                    ed sex sex_ind birth birth_ind,
                    sal sal_ind bon bon_ind comm comm_ind .

   'EXECSQL PUT INSERT_EMPLOYEE USING',
      ':emp(CHAR(6)), :fname(VARCHAR(12)),',
      ':mid(CHAR(1)), :lname(VARCHAR(15)),',
      ':wdpt :wdpt_ind (CHAR(3)), :ph :ph_ind (CHAR(4)),',
      ':hire :hire_ind (CHAR(8)), :job :job_ind (CHAR(8)),',
      ':ed(SMALLINT), :sex :sex_ind (CHAR(1)),',
      ':birth :birth_ind (CHAR(8)), :sal :sal_ind (DECIMAL(9,2)),',
      ':bon :bon_ind (DECIMAL(9,2)), :comm :comm_ind (DECIMAL(9,2))'

/* End loop */

'EXECSQL CLOSE INSERT_EMPLOYEE'
```

Dynamic PREPARE or PREP

```
                                    ┌─FROM─┐
►►──┬─PREPARE─┬──prepare_name───────┴──────┴───┬─sql_statement─┬─────────────────────►◄
    └─PREP────┘                                └─variable_name─┘
```

The Dynamic PREPARE statement prepares an SQL statement for a subsequent Dynamic CALL, Dynamic DECLARE, Dynamic DESCRIBE, Dynamic EXECUTE, OPEN, or PURGE request. PREP is provided as a synonym to support programs written for previous versions of RXSQL.

prepare_name
: The name you wish to give to the SQL statement. The *prepare_name* links the RXSQL Dynamic PREPARE statement with subsequent RXSQL requests.

  A Dynamic PREPARE statement cannot use the same *prepare_name*, *cursor_name*, or *statement_name* as that of another prepared or declared statement unless the state of the previously prepared statement is DYNAMIC UNPREPARED or DYNAMIC DECLARED-ONLY. If it is any other state, you must issue a PURGE command before reusing the name.

sql_statement
The SQL statement to be prepared.

If the SQL statement is a SELECT, INSERT, UPDATE or DELETE statement, it can contain *variable_names* in the WHERE, VALUES or SET clause. RXSQL fetches the values of these variables before a Dynamic CALL, Dynamic EXECUTE, OPEN, or PUT statement is passed to the database manager to be executed. If you provide an *input_rexx_host_variable_list* on a Dynamic CALL, Dynamic EXECUTE, OPEN, or PUT statement, the *input_rexx_host_variables* override any variables named in the SQL statement on the Dynamic PREPARE statement.

You can also use parameter markers (?) to denote the locations of REXX host variables within the SQL statement. You must then provide an *input_rexx_host_variable_list* on the Dynamic CALL, Dynamic EXECUTE, OPEN (for FETCH), or PUT statement. These *rexx_host_variables* replace the parameter markers when the CALL, EXECUTE, OPEN, or PUT statement is executed. The rules for using parameter markers are documented in the *DB2 Server for VSE & VM SQL Reference* manual.

variable_name
A REXX variable name with a mandatory preceding colon. RXSQL fetches the value of *variable_name* and passes the value to the database manager when the Dynamic PREPARE statement is executed. The value of *variable_name* must conform to the coding rules of the metavariable for which it is being substituted.

States

Required Initial State	**Resulting State**
none	DYNAMIC PREPARED
DYNAMIC UNPREPARED	DYNAMIC PREPARED
DYNAMIC DECLARED_ONLY	DYNAMIC PREPARED

Notes

1. If the SQL statement contains both parameter markers and *variable_names*, the *variable_names* in the SQL statement are replaced with parameter markers (?).
2. RXSQL has a limit of 40 active statements at any one time, i.e. whose state is DYNAMIC UNPREPARED, DYNAMIC PREPARED, or DYNAMIC OPEN. Use the PURGE command to discard prepared statements that are no longer needed.

Example

RXSQL retrieves the variable *phone_number* when the PREPARE statement is invoked and retrieves the value of the variable *emp* when the OPEN statement is invoked.

```
phone_number = "SELECT EMPNO, PHONENO FROM RXEMP WHERE EMPNO = :emp"

'EXECSQL PREPARE FIND_PHONENO FROM :phone_number'

'EXECSQL OPEN FIND_PHONENO'
```

OPEN was invoked here without a DECLARE to demonstrate that DECLARE is optional.

Extended PREPARE

Basic Extended PREPARE

```
                                         ┌─SETTING :SQLSTMTN──────┐  ┌─IN─┐
►►──PREPARE FROM statement_variable──────┼────────────────────────┼──┴────┴──────────────────►
                                         └─SETTING variable_name──┘

►──┬─package_name──┬─┬─────────────────────────────────┬─────────────────────────────────►◄
   └─variable_name─┘ └─USING───attributes_variable─────┘
```

Single Row Extended PREPARE

```
                       ┌─ROW─┐                             ┌─SETTING :SQLSTMTN──────┐  ┌─IN─┐
►►──PREPARE SINGLE─────┴─────┴──FROM statement_variable────┼────────────────────────┼──┴────┴───►
                                                           └─SETTING variable_name──┘

►──┬─package_name──┬─┬─────────────────────────────────┬─────────────────────────────────►◄
   └─variable_name─┘ └─USING───attributes_variable─────┘
```

Empty Extended PREPARE

```
                          ┌─SETTING :SQLSTMTN──────┐  ┌─IN─┐
►►──PREPARE FROM NULL─────┼────────────────────────┼──┴────┴───┬─package_name──┬────────────►◄
                          └─SETTING variable_name──┘           └─variable_name─┘
```

Temporary Extended PREPARE

```
                                                                  ┌─IN─┐
►►──PREPARE FROM statement_variable FOR──┬─section_number─┬───────┴────┴───┬─package_name──┬───►◄
                                         └─variable_name──┘                └─variable_name─┘
```

The **Basic** Extended PREPARE and the **Single Row** Extended PREPARE statements permit an SQL statement to be prepared and stored in a package for later execution. The **Empty** Extended PREPARE statement is used to provide support for Dynamic SQL statements. It is used in conjunction with the **Temporary** Extended PREPARE statement.

The Basic, Single Row and Empty Extended PREPARE statements return a number to a REXX variable that designates in which section the statement is stored. This number is then used in the Extended DECLARE statement, the XCALL statement, the Extended EXECUTE statement, the DROP STATEMENT statement or the Temporary Extended PREPARE statement.

You can use the Single Row Extended PREPARE statement on any SELECT statement that will return only one row. The Extended CALL or Extended EXECUTE statement with the INTO clause can be used to fetch the single row without using the OPEN, FETCH, CLOSE sequence of statements.

The Empty Extended PREPARE statement creates an indefinite section in the package. You can temporarily fill this section with an SQL statement in a subsequent logical unit of work by issuing a Temporary Extended PREPARE statement.

statement_variable
A REXX variable name with a mandatory preceding colon that holds the SQL statement to be prepared. RXSQL fetches the value of the *statement_variable* and passes the value to the database manager when the Extended PREPARE statement is executed.

If the statement is a SELECT, INSERT, UPDATE, or DELETE statement, parameter markers (?) can be used to denote the locations of variable_names within the SQL statement. You must provide an *input_rexx_host_variable_list* on the Extended CALL, Extended EXECUTE, OPEN, or PUT statement to give a value to all parameter markers. *Variable_names* are not allowed in the SQL statement.

SETTING :SQLSTMTN
SETTING *variable_name*
The REXX variable, either `SQLSTMTN` or one you specify with a mandatory preceding colon, is set with the section number into which the SQL statement is prepared. Your program must retain this number to refer to the statement on a later Extended DECLARE, Extended DESCRIBE, Extended EXECUTE, Extended XCALL, Extended DROP STATEMENT, or Temporary Extended PREPARE statement.

package_name
Indicates the name of the package in which the prepared SQL statement will be stored.

attributes_variable
A REXX variable with an optional preceding colon that contains a list which specifies the data type and length attributes for all parameter markers (?) in the SQL statement.

Do not use the DT (date), TM (time), or TS (timestamp) types that are returned by the DESCRIBE statement. Use the C n or V n types instead. An integer must be substituted for the m and the n in the following abbreviation list.

SQL Data Type Abbreviations	SQL Data Type	Description
I	INTEGER	31 bit binary integer
S	SMALLINT	15 bit binary integer
R	FLOAT	Single precision, floating-point number
F	FLOAT	Double precision, floating-point number
D *m n*	DECIMAL	*m* is the precision; *n* is the scale.
C *n*	CHAR	Fixed length character; *n* is the length
V *n*	VARCHAR	Variable length character, *n* is the maximum length
L *n*	LONG VARCHAR	Variable length character, *n* is the maximum length
G *n*	GRAPHIC	Fixed length graphic; *n* is the number of 2-byte characters
VG *n*	VARGRAPHIC	Variable length graphic, *n* is the maximum length
LG *n*	LONG VARGRAPHIC	Variable length graphic, *n* is the maximum length
ZD *m n*	ZONED DECIMAL	*m* is the precision; *n* is the scale. This data type is not valid on DB2 Server for VM application servers.

Notes:

1. For data limits, please refer to *IBM SQL Reference, Version 2, Volume 1*, SC26-8416.
2. An N following any of the above abbreviations means that the input data value may be null.

Figure 20. Extended PREPARE Attributes Abbreviations

Specifying Attributes of Input Data that May Be Null

To specify that the input data corresponding to a parameter marker may be null, place an N after the SQL data type abbreviation. If you do not place an N after the SQL data type abbreviation, the default SQL data type will be such that null input data values are not allowed.

Note: If you specify an N, the application server may incur some extra processing overhead.

Keywords and numbers must be separated by blanks. For example:

```
attribute = 'C 3 N C 4 N CCSID 500 D 6 2'
```

Specifying the CCSID in Attributes Variables

The CCSID may be specified in the USING *attributes_variable* clause in the Extended PREPARE or XPREP statement. To specify the CCSID for the input data corresponding to a parameter marker, the keyword CCSID followed by the CCSID number n must be placed in the string following the appropriate SQL data type abbreviation.

Keywords and numbers must be separated by blanks. For example:

```
attribute = 'C 3 CCSID 500 S V 12 N CCSID 65535 V 20 D 6 2 C 1'
```

For more information on CCSID or coded character sets, refer to the *DB2 Server for VSE & VM SQL Reference* manual.

variable_name
A REXX variable name with a mandatory preceding colon. RXSQL fetches the value of *variable_name* (except for SETTING *variable_name*) and passes the value to the database manager when the Extended PREPARE statement is

executed. The value of *variable_name* must conform to the coding rules of the metavariable for which it is being substituted.

section_number
An integer value that specifies the statement number of the indefinite section of a package that was created by the Empty Extended PREPARE statement. The indefinite section of the package is used to prepare the statement for processing, but the package is not permanently modified. When the LUW ends, the statement is removed from the indefinite section.

Output

`SQLSTMTN` is set to the integer value of the statement number assigned by the database manager unless another REXX variable was specified in the SETTING *variable_name* clause. The Extended DECLARE, DROP STATEMENT, XCALL, Extended EXECUTE and temporary Extended PREPARE statements use the integer value to specify the corresponding prepared statement or indefinite section of the package.

Notes

1. When you are using a Positioned UPDATE or Positioned DELETE, the *cursor_name* in the WHERE CURRENT OF clause must belong to a SELECT statement in the same package.
2. When using *attributes_variables*, RXSQL builds an input SQLDA structure.
3. Although the *cursor_name* in a WHERE CURRENT OF clause may be delimited with double quotes (") when the SELECT statement is prepared using extended PREPARE, all other DB2 RXSQL references to the cursor (DECLARE, OPEN, FETCH, PUT, CLOSE) must not delimit the *cursor_name* with double quotes.

Note: A *cursor_name* that is a reserved word when used in an DB2 Server for VM statement must be delimited with double quotes.

Examples

Example 1: Basic PREPARE

Prepare a SELECT statement into the package DEPT_VIEW. RXSQL puts the section number into the REXX variable `dept_stmt_num`.

```
select_dept = "SELECT * FROM RXEMP WHERE WORKDEPT = ?"

'EXECSQL PREPARE FROM :select_dept',
    'SETTING :dept_stmt_num IN DEPT_VIEW'
```

Example 2: PREPARE Adding Empty Section

Prepare an empty section into the package identified by the package variable `emp_package`. DB2 RXSQL puts the section number in the REXX variable `emp_stmt_num`.

```
'EXECSQL PREPARE FROM NULL SETTING :emp_stmt_num IN :emp_package'
```

Example 3: PREPARE Filling Empty Section

Prepare a DELETE statement, filling in an empty section in the package CHANGE_STAFF.

```
remove_employee = "DELETE FROM RXEMP WHERE EMPNO = ?"

'EXECSQL PREPARE FROM :remove_employee',
        'FOR :stmt_num_1 IN CHANGE_STAFF'
```

Example 4: Single Row PREPARE

Prepare a SELECT statement that will return only one row.

```
count_ed = "SELECT COUNT(*) FROM EMPVIEW WHERE EDLEVEL = ?"

attributes = "S"

'EXECSQL PREPARE SINGLE ROW FROM :count_ed IN EMPLOYEE_STATS',
   'USING :attributes'
```

PURGE

```
            ┌─,──────────────────┐
►►──PURGE──┬──▼─┬─statement_name─┬─┴──┬──────────────────────────────────────────────────►◄
           │    ├─prepare_name───┤    │
           │    └─cursor_name────┘    │
           └─── * ────────────────────┘
```

The PURGE command clears one or more statements from RXSQL temporary storage allowing them to be reused.

In Dynamic SQL, if an SQL statement has both a *prepare_name* and a corresponding *cursor_name*, the purging of either name will clear the SQL statement and both names from RXSQL temporary storage immediately.

statement_name
: The name given by an Extended DECLARE statement to a non-cursor SQL statement.

prepare_name
: The name given by a Dynamic PREPARE statement to an SQL statement.

cursor_name
: The name given to a dynamically prepared *prepare_name* by a Dynamic DECLARE statement or the name given to a statement in a package by an Extended DECLARE statement.

***** An asterisk (*) requests clearing *all* active statement and cursor names stored by RXSQL.

States

Required Initial State	Resulting State
DYNAMIC DECLARED-ONLY	none
DYNAMIC UNPREPARED	none
DYNAMIC PREPARED	none
DYNAMIC OPEN	none
EXTENDED-DYNAMIC UNDECLARED	none
EXTENDED-DYNAMIC DECLARED	none
EXTENDED-DYNAMIC OPEN	none

Notes

1. If the name belongs to a cursor that is open, the cursor is closed before purging the name. If PURGE was invoked with EXECSQL, an RXSQL warning will be generated.
2. If you do not purge a statement or cursor name, RXSQL will store it from the time it is prepared or declared until control is returned to CMS. Invoking NUCXDROP RXSQL will also purge all statements and cursors but this is not recommended as a general technique.
3. There is a limit of 40 dynamic prepared statements at any one time.
4. If the *cursor_name* belongs to a statement that an Extended DECLARE and OPEN statement created, the database retains the *cursor_name* when the PURGE is executed. The database removes the *cursor_name* when the LUW ends. The *cursor_name* cannot be used again on an Extended DECLARE and OPEN statement sequence until the next logical unit of work is established.

Example

Clear the statements SELECT_EMPLOYEE and INSERT_EMPLOYEE from RXSQL temporary storage.

```
'EXECSQL PURGE SELECT_EMPLOYEE, INSERT_EMPLOYEE'
```

PUT

```
►►──PUT──┬─cursor_name──┬──┬──────────────────────────────────────────┬──────────►◄
         └─prepare_name─┘  │  ┌─USING─┐                               │
                           └──┼───────┼──input_rexx_host_variable_list─┘
                              └─FROM──┘
```

The PUT statement performs an SQL INSERT. It is valid only for a block input statement processed by an OPEN statement.

cursor_name
The declared name of the *insert_cursor*.

If a Dynamic DECLARE has been issued, your program must use the *cursor_name* on the Dynamic PUT statement.

prepare_name
The name specified in the PREPARE request for the INSERT statement.

The *prepare_name* must be used if a Dynamic PREPARE has given a *prepare_name* to the INSERT statement and a Dynamic DECLARE has not associated a cursor with the INSERT statement before the statement was opened.

FROM/USING
FROM is equivalent to *USING* and is provided for compatibility with prior versions of RXSQL.

input_rexx_host_variable_list
One or more *rexx_host_variables* that supply the input values for the SQL statement.

States

Required Initial State	Resulting State
DYNAMIC OPEN	DYNAMIC OPEN
EXTENDED-DYNAMIC OPEN	EXTENDED-DYNAMIC OPEN

Input

RXSQL fetches the values of *input_rexx_host_variables* or the values of *variable_names* in the prepared SQL statement and passes them to the database manager as input values.

Notes

1. If you specify an *input_rexx_host_variable_list*, then any variables named in the SQL statement are overridden.
2. If you are inserting values without using variable qualifiers, you must make sure that the length and type of each input field is the same on each PUT statement when blocking is in effect. Since DB2 RXSQL determines the data type of each variable from its value, it may not use the same data type for each PUT statement.

   For example, if the first PUT statement has a variable NUM with 4 as its value, DB2 RXSQL passes the value to the database manager with a datatype of INTEGER. On a subsequent PUT statement, if NUM has 4.1 as its value. DB2 RXSQL passes the value to the database manager with a datatype of DECIMAL(2,1). When blocking is on, the database manager returns an error indication to RXSQL indicating that the data type has changed. When this occurs, DB2 RXSQL CLOSEs the *insert_cursor*, OPENs the *insert_cursor* again, and retries the PUT command. If PUT was invoked using EXECSQL invocation, DB2 RXSQL returns a warning indication. While this implementation bypasses the error, it may affect performance.
3. If blocking performance is critical to an application, do one of the following to ensure correct data types and lengths:
   a. Use variable qualifiers in your *input_rexx_host_variable* list to indicate to DB2 RXSQL what type of data is being inserted into the table.
   b. Take the following measures:

Strings
: Pad all strings with blanks on the right to make them of equal maximum length.

Decimal numbers
: Pad all numbers with zeros both left and right of the decimal (being careful of the sign) to indicate the appropriate precision and scale to DB2 RXSQL.

Numbers
: Do not mix decimal, integer, and scientific notation numbers for the same column.

Example

In this part of a REXX program, data is read from the file EMPLOYEE INPUT and inserted into the table RXEMP. The INSERT statement is prepared with parameter markers which are resolved on the PUT statement. Variable qualifiers are used to indicate what type of data is being inserted.

```
insert_row = 'INSERT INTO RXEMP VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?)'

'EXECSQL PREPARE INSERT_EMPLOYEE FROM :insert_row'

'EXECSQL OPEN INSERT_EMPLOYEE'

/* Begin loop */

/* Read data from file */
'EXECIO 1 DISKR EMPLOYEE INPUT * (LIFO'
/* Get the data into REXX variables */
Parse Upper Pull emp fname mid lname wdpt ph hire job,
                 ed sex birth sal bon comm .

'EXECSQL PUT INSERT_EMPLOYEE USING',
   ':emp(CHAR(6)), :fname(VARCHAR(12)),',
   ':mid(CHAR(1)), :lname(VARCHAR(15)),',
   ':wdpt(CHAR(3)), :ph(CHAR(4)),',
   ':hire(CHAR(8)), :job(CHAR(8)),',
   ':ed(SMALLINT), :sex(CHAR(1)),',
   ':birth(CHAR(8)), :sal(DECIMAL(9,2)),',
   ':bon(DECIMAL(9,2)), :comm(DECIMAL(9,2))'

/* End loop */

'EXECSQL CLOSE INSERT_EMPLOYEE'
```

REVOKE

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

ROLLBACK

```
              ┌─WORK─┐
►►──ROLLBACK──┴──────┴──┬─────────┬──────────────────────────────────────────────►◄
                        └─RELEASE─┘
```

The ROLLBACK statement backs out all changes made to the database since the LUW began or since the last COMMIT or ROLLBACK statement. All locks in the database are removed. If you specify RELEASE, the database connection is severed.

States

Required Initial State	Resulting State
DYNAMIC DECLARED-ONLY	DYNAMIC DECLARED-ONLY
DYNAMIC UNPREPARED	DYNAMIC UNPREPARED
DYNAMIC PREPARED	DYNAMIC UNPREPARED
DYNAMIC OPEN	DYNAMIC UNPREPARED
EXTENDED-DYNAMIC UNDECLARED	EXTENDED-DYNAMIC UNDECLARED
EXTENDED-DYNAMIC DECLARED	EXTENDED-DYNAMIC UNDECLARED
EXTENDED-DYNAMIC OPEN	EXTENDED-DYNAMIC UNDECLARED

Notes

1. DB2 RXSQL does not explicitly close any open cursors before issuing the ROLLBACK statement.
2. All *prepare_names* and *cursor_names* are retained by DB2 RXSQL after the ROLLBACK.
3. Do not execute any RXSQL statements between a ROLLBACK RELEASE statement and the invocation of an SQLINIT.

   DB2 RXSQL retrieves DATE and TIME specifications from the LASTING GLOBALV file the first time DB2 RXSQL is invoked or at the first invocation after a COMMIT RELEASE or ROLLBACK RELEASE. If your program issues a COMMIT RELEASE or ROLLBACK RELEASE and you wish to change DATE or TIME specifications using SQLINIT, you must do so before the next RXSQL statement is executed.

   For example, if your execution sequence is

   ```
   ROLLBACK RELEASE

   EXECSQL ...

   SQLINIT DATE(JIS)
   ```

   DB2 RXSQL will not retrieve the new date format, JIS, but will continue using the same specification you had before.

   The correct sequence in this case is as follows:

```
ROLLBACK RELEASE

SQLINIT DATE(JIS)

EXECSQL ...
```

Example

Cancel the changes made to the database since the LUW began and release the database connection.

```
'EXECSQL ROLLBACK RELEASE'
```

SQLDATE

```
►►──SQLDATE──┬───────┬──────────────────────────────────────────────────────►◄
             ├─Eur───┤
             ├─Iso───┤
             ├─Jis───┤
             ├─Local─┤
             ├─Reset─┤
             ├─Usa───┤
             └─*─────┘
```

The SQLDATE command sets the date format for the database manager or returns the current setting. If you specify a format, DB2 RXSQL sets a flag so that any date output columns will be converted into the specified format. If you do not specify a format, the current format is returned in the REXX variable `SQLDATE`.

The valid formats are:

Eur
European standard is *dd.mm.yyyy*

Iso
International Standards Organization standard is *yyyy-mm-dd*

Jis
Japanese industrial standard is *yyyy-mm-dd*

Local
Installation defined format

Reset
Initial setting.

For dynamic statements, the format is taken from the LASTING GLOBALV file. If DB2 RXSQL fails to retrieve the format from this file, the date format is set to ISO.

For extended dynamic statements, the date format is set to be the same as the format in effect when the package was created.

Usa
USA standard is *mm/dd/yyyy*

***** Default form taken from the LASTING GLOBALV file. If this information is not available, the date format is set to ISO.

Output

If no option is provided, the REXX variable `SQLDATE` is set to a single character indicating the current date format, with one exception. If the current format is Reset, `SQLDATE` is set to the word `RESET`.

Note

1. After an OPEN, the format in effect for the first FETCH statement that fetches date data is the format used until the cursor is closed.

Example

Set the date format to the Japanese industrial standard.

```
'EXECSQL SQLDATE J'
```

SQLISL

```
►►──SQLISL──┬────┬──────────────────────────────────────────────────────────►◄
            ├─Rr─┤
            ├─Cs─┤
            └─Ur─┘
```

The SQLISL command sets the isolation level for the database manager or returns the current setting. If you specify an option, the isolation level is set to that option; otherwise the current two-character isolation level value is returned in the REXX variable `SQLISL` in uppercase.

Once you specify an isolation level, any new cursor operations will be performed at that level.

Rr Repeatable read

Cs Cursor stability

Ur Uncommitted read

Output

The REXX variable `SQLISL` is set to the current isolation level if no input is provided.

Notes

1. Repeatable read is the default isolation level.
2. You can change the isolation level at any time so that subsequent SQL statements are processed at the new isolation level. However, if the level is changed while a cursor is open, all operations on that cursor (until the cursor is closed) are processed at the isolation level in effect when the cursor was

opened. Note that the changed isolation level is used (without error) for SQL statements that do not reference the opened cursor.

Example

Set the isolation level to cursor stability.

```
'EXECSQL SQLISL C'
```

SQLTIME

```
►►──SQLTIME──┬───────┬──────────────────────────────────────────────────────►◄
             ├─Eur───┤
             ├─Iso───┤
             ├─Jis───┤
             ├─Local─┤
             ├─Reset─┤
             ├─Usa───┤
             └─*─────┘
```

The SQLTIME command sets or returns the time format. If you specify a format, an internal variable is set. RXSQL converts any time output columns into the desired format. If you do not specify a format, the current format is returned in the REXX variable `SQLTIME`.

The valid formats are:

Eur
European standard is *hh.mm*[.*ss*]

Iso
International Standards Organization standard is *hh.mm*[.*ss*]

Jis
Japanese industrial standard is *hh*:*mm*[:*ss*]

Local
Installation defined form

Reset
Initial setting.

For dynamic statements, the format is taken from the LASTING GLOBALV file. If DB2 RXSQL fails to retrieve the format from this file, the time format is set to ISO.

For extended dynamic statements, the time format is set to be the same as the format in effect when the package was created.

Usa
USA standard is hh:mm AM or PM

***** Default form taken from the LASTING GLOBALV file. If this information is not available, the time format is set to ISO.

Output

If no option is provided, the REXX variable `SQLTIME` is set to a single character indicating the current time format, with one exception. If the current format is Reset, `SQLTIME` is set to the word `RESET`.

Notes

1. After an OPEN, the format in effect for the first FETCH statement that fetches time data is the format used until the cursor is closed.

Example

Set the time format to the European standard.

```
'EXECSQL SQLTIME E'
'EXECSQL SQLTIME'
 Say 'SQLTIME is' SQLTIME

/* produces    SQLTIME is E   */
```

STATE

```
►►──STATE──┬─statement_name─┬──────────────────────────────────────────────────►◄
           ├─cursor_name────┤
           └─prepare_name───┘
```

The STATE command returns the type and state values of the SQL statement you specify.

statement_name
: The name given by an Extended DECLARE to a statement in a package for a non-cursor operation.

cursor_name
: The name given by a Dynamic DECLARE to a previously prepared *statement_name* or the name given by an Extended DECLARE to a statement in a package for a cursor operation.

prepare_name
: The name given by a Dynamic PREPARE to an SQL statement.

Output

DB2 RXSQL sets the REXX variable `RXSQLSTATE` with a character string consisting of the statement type and state separated by a blank. The possible type and state combinations are as follows.

Type	**State**	
DYNAMIC	DECLARED-ONLY	This dynamic statement has been declared, but not prepared
DYNAMIC	UNPREPARED	This dynamic statement has become unprepared after COMMIT or ROLLBACK
DYNAMIC	PREPARED	This dynamic statement has been prepared
DYNAMIC	OPEN	This dynamic cursor statement has been opened
EXTENDED-DYNAMIC	UNDECLARED	This Extended Dynamic statement has been declared with DB2 RXSQL Extended DECLARE, but either 1. the declare has not been passed to the database manager or 2. COMMIT or ROLLBACK was issued
EXTENDED-DYNAMIC	DECLARED	This Extended Dynamic statement has been declared and opened, but its state changed to DECLARED after a CLOSE
EXTENDED-DYNAMIC	OPEN	This Extended Dynamic cursor statement has been opened

When an error in processing occurs, RXSQLSTATE is set to an empty string.

Example

Display the type and the state of the statement VIEW_DEPT.

```
'EXECSQL STATE VIEW_DEPT'
Say rxsqlstate
```

Output similar to the following is displayed.

```
DYNAMIC PREPARED
```

STMT

```
►►──STMT──┬─statement_name─┬──────────────────────────────────────────────►◄
          ├─cursor_name────┤
          └─prepare_name───┘
```

The STMT command returns the SQL statement associated with a given *statement_name*, *cursor_name*, or *prepare_name*.

statement_name
: The name given by an Extended DECLARE to a statement in a package for a non-cursor operation.

cursor_name
: The name given by a Dynamic DECLARE to a previously prepared *statement_name* or the name given by an Extended DECLARE to a statement in a package for a cursor operation.

prepare_name
: The name given by a Dynamic PREPARE to an SQL statement.

Output

The statement is put into the variable `RXSQLSTMT`.

If the statement is dynamic, the variable `RXSQLSTMT` is set with the statement string value with one exception; if the state is DYNAMIC DECLARED-ONLY, `RXSQLSTMT` is set with an empty string.

If the statement is extended dynamic, `RXSQLSTMT` is set with the section number and package name associated with the statement.

When an error in processing occurs `RXSQLSTMT` is set to a null string.

Note

1. If the given name is for a declared statement in a package, the returned string is in the format used on an Extended DECLARE statement, as follows:

   ```
   nn IN package_name
   ```

   where *nn* is the section number.

Example

Display the dynamic statement associated with the cursor SELECT_EMPLOYEE.

```
'EXECSQL STMT SELECT_EMPLOYEE'
Say rxsqlstmt
```

Output similar to the following is displayed.

```
SELECT * FROM EMPVIEW WHERE SALARY < :salary AND JOB = :job
```

TRACE

```
          ┌─,─────────────────────────────┐
►►──TRACE──▼─function_number trace_level─┴──────────────────────────►◄
```

The TRACE command sets the trace level (0 to 3) for one or all the functions in DB2 RXSQL. It is provided to aid IBM support personnel in problem determination.

function_number
: Specifies either the DB2 RXSQL function number, or a 0 for all functions. The following list outlines which functions you can specify.

The function numbers correspond to the following functions:

0	All RXSQL components
1	All functions
7	Database interface component
8	Environment setup component
9	Parsing component
10	CLOSE
11	COMMIT
12	CONNECT
13	CREATE PACKAGE
14	DECLARE
15	DESCRIBE
16	DROP STATEMENT, DROPSTMT
17	EXECUTE, CALL, XCALL
18	EXECUTE IMMEDIATE, EXEC
19	FETCH
20	NAMES
21	OP
22	OPEN
23	PREPARE - Dynamic
24	PREPARE - Extended Dynamic, XPREP
25	PURGE
26	PUT
27	ROLLBACK
28	SQLDATE
29	SQLISL
30	SQLTIME
31	STATE
32	STMT
33	TRACE

trace_level
Specifies the trace level. The following list outlines the trace levels you can specify:

The trace levels correspond to the following events:

0	No tracing
1	Function entry and exit points
2	Data at function entry and exit points
3	All internal trace points

You can specify multiple pairs of function numbers and trace levels on one statement.

Output

The TRACE command displays the trace information to the terminal unless redirected by the CP SPOOL command.

Specifying a level will trace all points for levels with lower numbers. For example, if you specify trace level 2, all level 2 and level 1 trace points will be displayed.

Example

Set the trace level for all components at trace level 2.

```
'EXECSQL TRACE 0 2'
```

UPDATE

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

UPDATE STATISTICS

See the *DB2 Server for VSE & VM SQL Reference* manual for details.

XCALL

```
                              ┌─IN─┐
►►──XCALL──┬─section_number─┬─┴────┴──┬─package_name──┬────────────────────────────────►◄
           └─variable_name──┘         └─variable_name─┘
```

The XCALL statement processes an SQL statement that was previously inserted into a package by an Extended PREPARE statement.

XCALL is provided for compatibility with previous versions of RXSQL and it does not support input or output *rexx_host_variables*. If you wish to use input or output *rexx_host_variables* with an extended dynamic statement, you may use the Extended EXECUTE statement, or the Extended DECLARE statement in combination with the Extended CALL statement.

section_number
An integer value representing the section number of the SQL statement to be issued. It was assigned by the database manager when an Extended PREPARE statement was executed.

package_name
The name of the package in which the prepared SQL statement to be issued is stored.

variable_name
A REXX variable name with a mandatory preceding colon. RXSQL fetches the value of *variable_name* and passes the value to the database manager when the XCALL statement is executed. The value of *variable_name* must conform to the coding rules of the metavariable for which it is being substituted.

Notes

1. You cannot use the XCALL statement if your SQL statement is a SELECT statement.

Examples

Example 1

Issue the statement in the package SALARY_REVIEW identified by the REXX variable `stmt_number`.

```
'EXECSQL XCALL :stmt_number IN SALARY_REVIEW'
```

Example 2

Issue the statement identified by the value of the variable `section_no` in the package RAISE_SALARY.

```
 section_no = 1
'RXSQL XCALL' section_no 'IN RAISE_SALARY'
/* REXX resolves this to 'RXSQL XCALL 1 IN RAISE_SALARY' */
```

XPREP

Basic XPREP

```
          ┌SETTING :SQLSTMTN──────┐    ┌IN┐
►►──XPREP─┼───────────────────────┼────┴──┴───┬package_name──┬──────────────────────────►
          └SETTING variable_name──┘           └variable_name─┘

►──┬──────────────────────────────┬──┬sql_statement─────┬───────────────────────────►◄
   └USING──────attributes_variable┘  └statement_variable┘
```

Single Row XPREP

```
                   ┌ROW┐   ┌SETTING :SQLSTMTN──────┐    ┌IN┐
►►──XPREP SINGLE───┴───┴───┼───────────────────────┼────┴──┴───┬package_name──┬──────────►
                           └SETTING variable_name──┘           └variable_name─┘

►──┬──────────────────────────────┬──┬sql_statement─────┬───────────────────────────►◄
   └USING──────attributes_variable┘  └statement_variable┘
```

Empty XPREP

```
          ┌SETTING :SQLSTMTN──────┐    ┌IN┐                        ┌NULL┐
►►──XPREP─┼───────────────────────┼────┴──┴───┬package_name──┬─────┴────┴─────────────►◄
          └SETTING variable_name──┘           └variable_name─┘
```

Temporary XPREP

```
                                      ┌IN┐
►►──XPREP FOR──┬section_number─┬──────┴──┴───┬package_name──┬───┬sql_statement─────┬─────►◄
               └variable_name──┘             └variable_name─┘   └statement_variable┘
```

XPREP is provided for compatibility with previous versions of DB2 RXSQL. It is functionally interchangeable with the Extended PREPARE statement, although its syntax is somewhat different.

The descriptions on how to use XPREP, and all metavariables except the following, can be found under "Extended PREPARE" on page 90.

sql_statement
The actual SQL statement to be prepared into the package.

If the statement is a SELECT, INSERT, UPDATE, or DELETE statement, parameter markers (?) can be used to denote the locations of variables within the SQL statement. You must provide a list of variable names on the Extended CALL, Extended EXECUTE, OPEN, or PUT statement to replace the parameter markers. Do not use *variable_names* in the SQL statement.

Note

1. For Temporary XPREP, the USING *attributes_variable* is not included in the syntax diagram, but is still permitted for compatibility with previous versions of RXSQL. It is ignored when the statement is passed to the database manager with RXSQL invocation, but DB2 RXSQL will return a warning with EXECSQL invocation.

Examples

Example 1: BASIC XPREP

Prepare the UPDATE statement identified by the statement variable `salary_stmt` into the package UPDATE_SALARY using the variable `attributes_list` to specify the data types and length attributes for each parameter marker.

```
salary_stmt = "UPDATE RXEMP SET SALARY = SALARY * ? WHERE EMPNO = ?",
          " AND JOB <> 'MANAGER'"

attributes_list = "D 9 2 C 6"

'EXECSQL XPREP UPDATE_SALARY USING :attributes_list :salary_stmt'
```

Example 2: Single Row XPREP

Prepare into the package SALARY_REVIEW a SELECT statement that returns only one row of data. RXSQL returns the statement number into the variable `salary_statement`.

```
average_salary = 'SELECT AVG(SALARY) FROM EMPVIEW WHERE JOB = ?'

'EXECSQL XPREP SINGLE ROW SETTING :salary_statement IN SALARY_REVIEW',
   ':average_salary'
```

Example 3: Empty XPREP

Prepare an empty section into the package.

```
'EXECSQL XPREP :package_name NULL'
```

Example 4: Temporary XPREP

Fill in the empty section identified by the section variable `stmt_number` in the package CHANGE_STAFF.

```
delete_employee = 'DELETE FROM RXEMP WHERE EMPNO = ?'

'EXECSQL XPREP FOR :stmt_number IN CHANGE_STAFF :delete_employee'
```

Appendix A. RXSQL Return Codes and Messages

This appendix contains a list of DB2 RXSQL return codes and error messages. The return codes are returned in the REXX variable RC.

CMS-Related Return Codes

The following are the CMS return codes issued when DB2 RXSQL is not found or could not get through initialization.

-3

Explanation: DB2 RXSQL was not found by CMS or RXSQL was called through an unsupported interface (for example, call RXSQL ...).

System Action: The program ends and the user is returned to CMS.

User Response: Link to the correct DB2 RXSQL production library.

28 DMSSOP036E OPEN ERROR CODE '04' ON 'RXSQL '.

Explanation: RXSQL LOADLIB was not found by RXSQL MODULE.

System Action: The program ends and the user is returned to CMS.

User Response: Ensure that the correct DB2 RXSQL production library is linked.

28 DMSSOP036E OPEN ERROR CODE '04' ON 'ARISQLLD'.

Explanation: ARISQLLD LOADLIB was not found by the DB2 Server for VM Resource Adapter.

System Action: The program ends and the user is returned to CMS.

User Response: Ensure that the DB2 Server for VM production library is linked.

41 DMSFRE159T INSUFFICIENT STORAGE AVAILABLE TO SATISFY DMSFREE REQUEST FROM *address*.

Explanation: There was not enough free storage to load the DB2 RXSQL code from the RXSQL LOADLIB. A minimum of approximately 500K bytes of free storage is needed.

System Action: The program ends and the user is returned to CMS.

User Response: Obtain more storage for your virtual machine and rerun the program.

DB2 Server for VM-Related Return Codes

The following are the return codes from a request where DB2 RXSQL had no error, but the DB2 Server for VM system gave an error or warning.

-10

Explanation: DB2 Server for VM has issued a serious error as a result of the last database request. See the SQLCA variables.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check the returned SQLCODE variable and other SQLCA variables in your program and refer to *DB2 Server for VM Messages and Codes* manual for an explanation.

4

Explanation: DB2 Server for VM has issued a warning as a result of the last database request. See the SQLCA variables.

System Action: The DB2 RXSQL request was completed with warnings. Control is returned to the user's REXX program.

User Response: Check the returned SQLCODE variable and other SQLCA variables in your program and refer to *DB2 Server for VM Messages and Codes* manual for an explanation.

8

Explanation: DB2 Server for VM has issued a serious error as a result of the last database request. See the SQLCA variables.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check the returned SQLCODE variable and other SQLCA variables in your program and refer to *DB2 Server for VM Messages and Codes* manual for an explanation.

10

Explanation: DB2 Server for VM has issued a warning as a result of the last database request. See the SQLCA variables.

System Action: The DB2 RXSQL request was completed with warnings. Control is returned to the user's REXX program.

User Response: Check the returned SQLCODE variable and other SQLCA variables in your program and refer to *DB2 Server for VM Messages and Codes* manual for an explanation.

Message Format and Associated Text

In the following sections, the conventions for showing messages are:

- All the letters, numbers, and punctuation marks (such as periods, commas, underscores, and dashes) following the message identifier in **bold** show the actual message text.
- Letters, words, and numbers in the text of the message that are in *italics* represent variables. RXSQL substitutes specific values for the variables when it displays the message.

  In the explanation following the message text, the variables are in *italics*.

All RXSQL messages begin with a message identifier of the form **ELOnnnnt**, where:

ELO	Shows that the message is from RXSQL.
nnnn	Identifies the particular message number.
t	Is the action indicator:

E	Error
I	Warning or Information

Explanation contains information about the message, such as:

- Definitions for variable fields in the message text
- Possible reasons why the message occurred
- Additional descriptive information about the condition that caused the message to occur.

System Action describes the resulting action that was or will be taken by RXSQL.

User Response, **Operator Response**,

RXSQL Error Messages

The following messages are issued when there is an RXSQL error. The return code will be less than -100 on EXECSQL invocation, and greater than 100 on RXSQL invocation.

ELO0101E Insufficient storage available to start DB2 RXSQL.

Explanation: The attempt to start DB2 RXSQL failed because not enough storage could be obtained.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Obtain more storage for your virtual machine and rerun the program

ELO0103E Insufficient storage available for DB2 RXSQL to process the request.

Explanation: The amount of storage DB2 RXSQL needs to process the request is not available.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Obtain more storage for your virtual machine and rerun the program.

ELO0104E Error trying to fetch or set a REXX variable. Return code from EXECOMM = -1.

Explanation: An unexpected error was returned from EXECCOMM while DB2 RXSQL was trying to fetch or set a REXX variable.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: See the *VM/ESA REXX/VM Reference* manual for a description of the -1 return code from EXECCOMM and apply corrective measures.

ELO0105E The data type *data_type_number* **of column** *col_number* **is not supported by DB2 RXSQL.**

Explanation: DB2 RXSQL did not recognize the data type of the column in position *col_number*. The number found in the SQLTYPE field of the SQLDA was *data_type_number*. DB2 RXSQL did not recognize the SQLTYPE supplied by the database manager in the SQLDA as one of the supported SQLTYPEs.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Make sure you are running compatible release levels of the database manager and DB2 RXSQL. Ensure that DB2 RXSQL supports all data types in the tables being accessed.

ELO0106E DB2 RXSQL Error code = *code***: invalid internal code used.**

Explanation: The DB2 RXSQL interface module to the database manager has received an invalid internal code.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Inform your DB2 RXSQL system programmer of the error.

System Programmer Response: Check that there are no inconsistencies with release levels of DB2 RXSQL. If there are none, contact IBM support with the problem.

ELO0107E DB2 RXSQL was not invoked from a REXX program.

Explanation: The return code from EXECCOMM was -3. There was no EXECCOMM environment available. DB2 RXSQL was not called from a REXX program.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Refer to the *VM/ESA REXX/VM Reference* manual for a description of the -3 return code from EXECCOMM. Ensure that your program is a REXX program.

ELO0108E Invalid variable name *name***.**

Explanation: REXX rejected the request to set or fetch a variable because the name did not conform to REXX standards. Only the first 20 characters of your REXX variable name are shown. You may have had more main variables on the DB2 RXSQL request than DB2 RXSQL expected.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check the variable names on the request and ensure that they are valid REXX variables. Rerun the program.

ELO0109E Unexpected error trying to fetch or set a REXX variable. RC=*hexadecimal_code***.**

Explanation: An unexpected error was returned from EXECCOMM as a result of trying to fetch or set REXX variables. *hexadecimal_code* is the SHVRET value for an EXECCOMM call.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Refer to the *VM/ESA REXX/VM Reference* manual for a description of the return codes from EXECCOMM (SHVRET).

ELO0110E An error has occurred while trying to read GLOBALV variable *name*.

Explanation: An error occurred when DB2 RXSQL tried to access the LASTING GLOBALV file. The LASTING GLOBALV file may be updated by the SQLINIT program and the CMS GLOBALV command. You did not invoke the SQLINIT program before using DB2 RXSQL, or you deleted a value from the LASTING GLOBALV file, or you placed an incorrect parameter in the LASTING GLOBALV file. The group name for values used by DB2 RXSQL is SQL/DS. DB2 RXSQL tried to access the global variable *name*.

If *name* was one of 'DATEFORMAT', 'TIMEFORMAT', 'LDATELEN' or 'LTIMELEN' then any date or time data fetched in subsequent requests will be returned in the ISO format. It is assumed that there are no local date and time exits.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Issue the SQLINIT program with the correct parameters. If the error persists, notify your system programmer.

System Programmer Response: Verify the parameters that were specified in the SQLGLOB program during initialization.

ELO0111E Unexpected error from EXECCOMM. Return code is negative.

Explanation: An unexpected error was returned from EXECCOMM. DB2 RXSQL was trying to fetch or set a REXX variable. The EXECCOMM return code was negative but not recognized by DB2 RXSQL. (The return code is not equal to -1, -2 or -3)

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Refer to the *VM/ESA REXX/VM Reference* manual for a description of the negative return codes from EXECCOMM and apply corrective measures.

ELO0115E Insufficient storage was available to set REXX variables.

Explanation: DB2 RXSQL was attempting to set a REXX variable value and EXECCOMM could not acquire enough storage to set the value. The return code from EXECCOMM was -2.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Obtain more storage for your virtual machine and rerun the program. For more information refer to the *VM/ESA REXX/VM Reference* manual for a description of the -2 return code from EXECCOMM.

ELO0116E *name* does not represent a SELECT statement. *request* request cannot be executed.

Explanation: The statement name or prepare name specified in the *request* request does not refer to a prepared or declared SELECT statement.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check your spelling of the statement name and rerun the program.

ELO0117E *name* is not open. FETCH request cannot be executed.

Explanation: The cursor name, prepare name or statement name, *name*, as an argument of the FETCH request has not been previously OPENed.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's program.

User Response: Issue an OPEN request using the cursor name or statement name, *name*, and then issue the FETCH.

ELO0118E *name* represents a SELECT statement. A PUT request cannot be executed.

Explanation: The statement name or prepare name specified in a PUT request represents a SELECT statement.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check your spelling of the statement name and rerun the program.

ELO0120E An attempt was made to prepare more than the allowed limit of *n* statements.

Explanation: No more than *n* SQL statements can be processed by the PREP request in a single program.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Issue a PURGE request for one or more of the existing statements, reuse a statement name in a PREPARE request, or use Extended Dynamic statements.

ELO0129E *metavariable* **is an invalid function number.**

Explanation: The function metavariable following the TRACE command has specified a function number that is either not numeric or is not one of the valid function numbers.

System Action: The DB2 RXSQL request was executed successfully. Control is returned to the user's REXX program.

User Response: Check the function metavariable following the TRACE command, change it and rerun the program.

ELO0130E *metavariable* **is an undefined trace level.**

Explanation: The trace metavariable following a TRACE command has specified a trace level that was not one of the valid levels.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check the trace metavariable following the TRACE command, change it and rerun the program.

ELO0134E **Too many input rexx-host-variables were coded on the statement.**

Explanation: The statement contained too many rexx-host-variables in the input-rexx-host-variable-list to pass on to the database. The maximum number of input rexx-host-variables allowed is 32767.

System Action: The request has not executed successfully. The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Reduce the number of input rexx-host-variables and rerun your program. Note that the database manager may have a smaller limit than DB2 RXSQL and a further error could be produced by the database manager if its maximum is exceeded.

ELO0135E **Section** *section* **of package** *package* **owned by** *creator* **is not a SELECT. Request was** *request***.**

Explanation: The statement represented by the section number, creator name and package name on a *request* request is not a SELECT statement. The creator name may not be present if one was not given on the request.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Make sure that you are referencing the correct section in your package. Change the request and rerun your program.

ELO0138E *name* **has a cursor-name** *cursor-name***.** *request* **must use the cursor name.**

Explanation: A cursor for the statement associated with *name* exists. As a result, the DB2 RXSQL request must be issued with the *cursor-name*.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Change the request to use the cursor name, and rerun the program.

ELO0139E *name* **is not OPEN - unable to** *operation***.**

Explanation: A CLOSE or PUT request has been issued for an SQL statement that has not been previously processed by an OPEN request.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: An OPEN request must be issued before a CLOSE or PUT request can be performed. Change the program accordingly and rerun the program.

ELO0142E **Name** *name* **has no associated dynamically prepared statement.**

Explanation: An operation was issued without providing a dynamically prepared SQL statement for the name *name*.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: You must issue a Dynamic PREPARE statement to assign a statement to the statement name you used in your request.

ELO0143E **The name** *name* **is too long.**

Explanation: The cursor name, prepare name or statement name given is more than 18 characters long.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Shorten the statement name or cursor name shown to 18 characters or less. Rerun the program.

ELO0144E *name* **is not valid.**

Explanation: The name specified for an SQL statement on a Dynamic PREPARE, Dynamic DECLARE or Extended DECLARE request contains an invalid character.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Refer to the *DB2 REXX SQL for VM/ESA Reference* manual for a description of valid cursor names, prepare names or statement names. Change the statement name to the allowed format and rerun the program.

ELO0145E **The name** *name* **does not exist.**

Explanation: *name* has not been provided on any previous PREPARE or DECLARE statements. There is no SQL statement associated with *name*.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: A PREPARE or DECLARE request must be issued to define a statement name or cursor name and assign it to an SQL statement.

ELO0149E **SQL statement of length** *len* **is too long.**

Explanation: The maximum length for an SQL statement is 32767 characters. If your cursor or statement name in the WHERE CURRENT OF clause is 1 or 2 characters long, DB2 RXSQL will expand your statement by up to 2 characters when it substitutes its own values for cursor or statement name before the request is passed to the database manager.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Reduce the size of your SQL statement. Rerun the program.

ELO0150E **Value passed in position** *pos* **of length** *len* **is too long.**

Explanation: The maximum length for an SQL character string is 32767 characters.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Reduce the size of the character string. Rerun the program.

ELO0151E **Operator command of length** *length* **is too long.**

Explanation: The maximum length for an operator command accepted by RXSQL is 80 characters.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Reduce the size of your operator command. Rerun the program.

ELO0152E **Error in operator command processing.**

Explanation: The attempt to issue an operator command failed because the database returned unexpected information to DB2 RXSQL. The REXX SQLCA variables have been set.

System Action: The request has not executed successfully. The DB2 RXSQL request was not completed. Control is returned to the user's REXX program.

User Response: Check the SQLCA variables, correct the problem and rerun the program.

ELO0154E **The name** *name* **in a Dynamic EXECUTE statement is associated with an Extended Dynamic statement.**

Explanation: The name *name* found in a Dynamic EXECUTE statement has been declared as a statement name for an extended dynamic statement. A statement name cannot be used in a Dynamic EXECUTE statement when it is already being used in an Extended Dynamic statement.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Correct the statement and rerun the request.

ELO0156E **Work unit information could not be obtained. DMSCSL return code** *rc***.**

Explanation: System information for the CMS work unit could not be obtained. No further processing can be done in this CMS work unit. If CMS work units were being used for the first time, no further processing can be done by the program.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Refer to the *VM/ESA: CMS Application Development Reference* manual for more information on the return code for the Callable Services Library Routines: DMSQWUID and DMSERP.

ELO0163E **The *parameter* value *value* on the CONNECT request is too long.**

Explanation: The value for the specified *parameter* is too long.

System Action: The request has not executed successfully. The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Correct the input and rerun the program. See the *DB2 REXX SQL for VM/ESA Reference* manual for the details on the CONNECT statement.

ELO0168E **The authorization name *name* is too long.**

Explanation: The authorization name (creator) must be a string of 1 to 8 characters.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Shorten the name and rerun the program.

ELO0169E **Section number *number* is not in the valid integer range.**

Explanation: The section number provided was either negative, zero or greater than 2,147,483,647. A valid section number ranges from 1 to 2,147,483,647.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Ensure that your section number is in the valid range and rerun the program.

ELO0170E **Name *name* matches the cursor name.**

Explanation: A cursor name and prepare name on a Dynamic DECLARE statement were identical.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Change either the cursor name or prepare name so that they are different. Make sure that all other requests using the old name are also changed. Rerun the program.

ELO0171E **Cursor name *cursor_name* is already in use.**

Explanation: The cursor name *cursor_name* provided on a DECLARE statement is already defined for another statement. It matches either a cursor name or a prepare name.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: To reuse the name, issue an explicit PURGE request for the name. Otherwise, change the name and rerun the program.

ELO0172E **The package id *id* is too long.**

Explanation: The package identifier *id* must be a string of 1 to 8 characters.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Shorten the name and rerun the program.

ELO0173E **Non-numeric section number *value*.**

Explanation: The section number must be numeric.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Ensure that the value for the section number is numeric and rerun your program.

ELO0174E **The name *name* is already in use.**

Explanation: The name *name* provided is already in use. It matches either a previously declared cursor name or a statement name.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: To reuse the name, issue an explicit PURGE request for the name. Otherwise, change the name and rerun the program.

ELO0184E **Output is not supported on a dynamic *request*.**

Explanation: The INTO clause is not supported for the Extended Dynamic *request* that you issued. It is only supported for the Extended *request* for a SELECT statement you prepared into a package with the SINGLE ROW format.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Change your *request* by removing the INTO clause and rerun the program.

ELO0187E The syntax of the section number variable named *section-var* is invalid.

Explanation: The section number variable name must start with a colon.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Correct the input and rerun the program.

ELO0189E The syntax of the statement variable named *statement-var* is invalid.

Explanation: The statement variable name must start with a colon.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Correct the input and rerun the program.

ELO0192E Invalid *date-time* length value retrieved. Value was *value*.

Explanation: DB2 RXSQL retrieved the local date or time length value (LDATELEN or LTIMELEN) from your LASTING GLOBALV file. Either the length is an invalid length (too large or too small) or an illegal character was encountered during conversion.

Any date or time data fetched in subsequent requests will be returned in the ISO format. It is assumed that there are no local date and time exits.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Issue the SQLINIT program with correct parameters. If the error persists, notify your system programmer.

System Programmer Response: Verify the parameters that were specified in the SQLGLOB EXEC during initialization.

ELO0194E *date-time* format is invalid. Value retrieved was *value*.

Explanation: DB2 RXSQL fetched the DATEFORMAT or TIMEFORMAT from your LASTING GLOBALV file. An unrecognized value *value* was retrieved. Only the first 20 characters of the value are displayed.

Any date or time data fetched in subsequent requests will be returned in the ISO format. It is assumed that there are no local date and time exits.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Issue the SQLINIT program with correct parameters. If the error persists, notify your system programmer.

System Programmer Response: Verify the parameters that were specified in the SQLGLOB EXEC during initialization.

ELO0195E Unrecognized *date-time* data received from the database manager. Data was *data*.

Explanation: DB2 RXSQL has received date or time data (*data*) from the database manager in a format that cannot be recognized. The data is not in any of the database manager date or time formats and the local date or time exit does not recognize the data. Only the first 20 characters of the actual data is given in the message.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Inform your DB2 RXSQL system programmer of the error.

System Programmer Response: Check that there are no inconsistencies with the local date or time exit. For example, the exit should recognize the same date or time formats on both input and output. Also, make sure that the same version of the local exit has been linked with both DB2 Server for VM and DB2 RXSQL modules. Correct the problem, link-edit DB2 RXSQL, and rerun the program.

ELO0196E Supplied local *date-time* exit not replaced by user's local exit.

Explanation: DB2 RXSQL has tried to format a date or time value into the local format. However, the DB2 Server for VM local date or time exit was not replaced by a local exit when DB2 RXSQL was installed.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Inform your DB2 RXSQL system programmer of the error.

System Programmer Response: Make sure that the DB2 RXSQL link-edit was successful and that the correct local date or time exit is being linked with DB2 RXSQL. Re-link if necessary and rerun the program.

ELO0197E **Invalid** ***date-time*** **value, function number =** ***number*****, local exit RC =** ***rc*****.**

Explanation: DB2 RXSQL has called the local date or time exit to format a value retrieved from the database manager. The local exit indicated that the date or time value was invalid or that the date or time value was not in a valid format (check the return code *rc*).

The function number *number* indicated the direction in which the conversion was being performed by the local exit. If the number was '4' then the data was being converted from the LOCAL format to the ISO format. If the number was '8' then the data was being converted from the ISO format to the LOCAL format.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Inform your DB2 RXSQL system programmer of the error.

System Programmer Response: Find out why the local date or time exit did not recognize the data format that was passed to it. Correct the problems with the local exit, link-edit DB2 RXSQL again, and rerun the program.

ELO0198E **Error returned from local** ***date-time*** **exit. Function =** ***func*****, rc =** ***rc*****.**

Explanation: DB2 RXSQL has called the local date or time exit to perform the specified function, as indicated by *func*. The local exit returned the return code *rc* when trying to perform the operation.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Inform your DB2 RXSQL system programmer of the error.

System Programmer Response: Find out why an error occurred in the local date or time exit using the information returned in the message. Look at the function being performed and the return code from the routine. Correct the problems with the local exit, link-edit DB2 RXSQL again, and rerun the program.

ELO0202E **Variable** ***var*** **is undefined or dropped.**

Explanation: The variable *var* was undefined. This variable must contain a value.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Ensure that the variable's value has been set. Change your program and rerun it.

ELO0204E ***main_variable*** **is undefined and the value of** ***indicator_variable*** **was non-negative.**

Explanation: The main variable was undefined, yet the indicator variable indicated that the main variable would be set with a value.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check that the main variable or the indicator variable was set correctly or that the main variable was not inadvertently dropped. Change your program and rerun your program.

ELO0209E **NULL data was retrieved for main variable** ***main-var*** **but no indicator variable was provided.**

Explanation: DB2 RXSQL was invoked using EXECSQL invocation. NULL data was retrieved from the database, yet no indicator variable was provided to indicate this. Indicator variables are compulsory when retrieving NULL data using EXECSQL invocation.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Ensure that indicator variables are provided for all columns which can contain NULL values in your select list.

ELO0211E **The indicator variable** ***ind_var*** **is undefined.**

Explanation: The indicator variable did not have a value. DB2 RXSQL requires that an indicator variable, if provided, must have a value.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Assign a value to the indicator variable or remove the indicator variable from the statement. Rerun the program.

ELO0213E **An indicator variable is required for main variable** ***main-var*** **to input NULL data.**

Explanation: DB2 RXSQL was invoked using EXECSQL invocation. The main variable provided was undefined and no indicator variable was provided. This is not allowed using EXECSQL invocation.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: If you wish to input NULL data to the database manager, you must provide an indicator variable and assign a negative value to the variable.

ELO0214E **GRAPHIC data *data* in variable *var* has an odd number of bytes.**

Explanation: GRAPHIC data must have an even number of bytes.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check your GRAPHIC data to ensure that you have an even number of bytes. Rerun your program.

ELO0215E **The value *number* in variable *variable* is out of the valid range for a floating point number.**

Explanation: The number was too small or too large to be represented as a floating point number for the target operating system.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Ensure that your data is within the range for your operating system and rerun the program.

ELO0216E **An invalid floating point number *number* was in variable *variable*.**

Explanation: DB2 RXSQL did not recognize the number as a valid floating point number.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Ensure that your data is correct and rerun the program.

ELO0217E **The value *number* you provided in *variable* is out of the range for SMALLINT.**

Explanation: The value is out of the valid range for a small integer for the target operating system.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Ensure that your column data type definition should be SMALLINT. If you really wish to insert this number, your column data type definition should be changed to INTEGER. Otherwise, ensure that your data is correct. Rerun your program.

ELO0218E **The value *number* you provided in *variable* is out of the range for INTEGER.**

Explanation: The value is out of the valid range for an integer for the target operating system.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Ensure that your column data type definition should be INTEGER. If you really wish to insert this number, your column data type definition should be changed. Otherwise, ensure that your data is correct. Rerun your program.

ELO0219E **The value *number* you provided for *variable* is an invalid integer.**

Explanation: DB2 RXSQL did not recognize the value *number* as a valid integer number.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Ensure that your data is correct and rerun the program.

ELO0220E **The value *number* for variable *variable* is an invalid decimal number.**

Explanation: DB2 RXSQL did not recognize the value as a valid decimal number.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Ensure that your data is correct and rerun the program.

ELO0221E **The value *value* for variable *variable* will result in high order truncation.**

Explanation: Significant high order digits will be truncated from your number with the precision and scale given.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Ensure that your data is correct or ensure that the precision and scale are correct for the column. Rerun the program.

ELO0230E **The value of variable *variable* has an invalid character *hex_value* at position *pos*.**

Explanation: The variable value is invalid because it contains an invalid character.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Correct the request and rerun the program.

ELO0232E **The value for the variable *variable* has a length of zero.**

Explanation: The REXX variable *variable* was an empty string. For the request you are executing, the variable must have a length greater than zero.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Assign a valid value to the variable and rerun the program.

ELO0239E ***name* must be used instead of *cursor-name* for the *request*.**

Explanation: A cursor was declared for the statement associated with *name*, but the statement name or prepare name must be used for the request *request*.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Change the request to use the statement name, and rerun your program.

ELO0241E **Initial state for *name* is invalid for the *request*.**

Explanation: The statement must be in one of the initial states listed for the *request* in the *DB2 REXX SQL for VM/ESA Reference* manual.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Make sure that the state of the statement is allowed for the request and rerun the program.

ELO0242E **The variable *var* is an invalid stem variable.**

Explanation: A column and an indicator variable pair were provided, and only one of them is a valid stem variable.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Change the invalid variable name. Rerun your program.

ELO0243E **Variable qualifiers are not allowed for stem variables.**

Explanation: A variable qualifier was provided for a stem variable. DB2 RXSQL does not allow this.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Remove the variable qualifier or change the variable name or names so that they don't look like stem variables. Rerun your program.

ELO0250E **The CREATE PACKAGE options of length *len* are too long.**

Explanation: The maximum length for the options on a CREATE PACKAGE statement is 32767 characters. DB2 RXSQL appends the DESCRIBE option to the string that you specified, and this total length must not exceed the maximum.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Reduce the size of the options and rerun the program. Note that the database manager may have a smaller limit than DB2 RXSQL and a further error could be produced by the database manager if its maximum is exceeded.

ELO0330E **The SQL statement has an invalid character, *hex-value*, at position *pos*.**

Explanation: The character found at the indicated position in the SQL statement is not a valid character.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Replace the character found with a valid character and rerun the request.

ELO0340E **The request has an invalid character, *hex-value*, at position *pos*.**

Explanation: The character found at the indicated position in the request is not a valid character.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Replace the character found with a valid character and rerun the request.

ELO0344E **The request begins properly but is incomplete.**

Explanation: The DB2 RXSQL request was correct up to the point where no more input was found.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check that the request is complete and rerun the program.

ELO0345E **Syntax error detected at word *word*, position *position* in the request.**

Explanation: The word *word* is the first word that DB2 RXSQL does not recognize as being part of a valid request as shown in the DB2 RXSQL variable rxsqlrequest. This word may not be the cause of the error, it may be prior to this word.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check the value of rxsqlrequest, and correct the syntax of the request. Ensure that all keywords are spelled correctly. Rerun the program.

ELO0350E **Invalid syntax at position *position* of the input string for *request*.**

Explanation: The syntax for the request was incorrect. The error occurred at or near position *position* relative to the first non-blank character of the input string. Look for misspelled keywords.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check the syntax of your request. Change the statement and rerun your program.

ELO0444E **The variable list begins properly but is incomplete.**

Explanation: The variable list was correct up to the point where no more input was found.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check that the variable list is complete and rerun the program.

ELO0445E **Syntax error detected at word *word*, position *position* in the variable list.**

Explanation: A syntax error has occurred in the variable list at word *word*. This word is *position* characters from the beginning of the variable list.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check the syntax of the variable list and rerun the program.

ELO0450E **The *field* provided, *value*, is not valid.**

Explanation: On a variable qualifier, the value supplied was not correct for a *field*.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Change the value of the variable qualifier and rerun the request.

ELO0500E **Unexpected C function error. File: *file-name* on Line: *line-num* detected the error.**

Explanation: A call to a C function returned with an error.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Contact your system programmer with all the accompanying information.

System Programmer Response: Contact your IBM Service representative for corrective measures. Be prepared to have the necessary information for the error.

ELO0544E **The value of the attributes variable begins properly but is incomplete.**

Explanation: The value of the attributes variable was correct up to the point where no more input was found.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check that the value of the attributes variable is complete and rerun the program.

ELO0545E **Syntax error detected at word *word*, position *position* in the attributes variable value.**

Explanation: A syntax error has occurred in the value of the attributes variable at word *word*. This word is *position* characters from the beginning of the value of the attributes variable.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Check the syntax of the value of the attributes variable and rerun the program.

ELO0901E **A severe error occurred. Extra info:** *text* **File:** *file-name* **Line:** *line-num*

Explanation: DB2 RXSQL has discovered an internal inconsistency. *file-name* and *line-num* identify the DB2 RXSQL module that discovered the error. *text* may be provided to indicate more information about the error.

System Action: The DB2 RXSQL request was not executed successfully. Control is returned to the user's REXX program.

User Response: Contact your system programmer with all the accompanying information.

System Programmer Response: Contact your IBM Service representative for corrective measures. Be prepared to have the necessary information for the error.

ELO0960E **Error trying to initialize the environment for DB2 RXSQL.**

Explanation: The attempt to start DB2 RXSQL failed in subcommand processing.

System Action: The DB2 RXSQL request has not executed successfully. Control is returned to the user's REXX program.

User Response: This could be due to shortage of storage. Obtain more storage for your virtual machine and rerun the program. If the error persists, contact your system programmer with all the accompanying information.

System Programmer Response: Contact your IBM Service representative for corrective measures. Be prepared to have the necessary information for the error.

ELO0961E **Error trying to initialize the environment for DB2 RXSQL.**

Explanation: The attempt to start DB2 RXSQL failed in inter-language setup.

System Action: The DB2 RXSQL request has not executed successfully. Control is returned to the user's REXX program.

User Response: This could be due to shortage of storage. Obtain more storage for your virtual machine and rerun the program. If the error persists, contact your system programmer with all the accompanying information.

System Programmer Response: Contact your IBM Service representative for corrective measures. Be prepared to have the necessary information for the error.

ELO0962E **Error trying to initialize the environment for DB2 RXSQL.**

Explanation: The attempt to start DB2 RXSQL failed when trying to determine the operating environment.

System Action: The DB2 RXSQL request has not executed successfully. Control is returned to the user's REXX program.

User Response: Ensure that DB2 RXSQL is running on an operating system that is supported for DB2 RXSQL. If it is, contact your system programmer with all the accompanying information.

System Programmer Response: Contact your IBM Service representative for corrective measures. Be prepared to have the necessary information for the error.

ELO0963E **Error trying to initialize the environment for DB2 RXSQL.**

Explanation: The attempt to start DB2 RXSQL failed when trying setup the environment for DB2 RXSQL.

System Action: The DB2 RXSQL request has not executed successfully. Control is returned to the user's REXX program.

User Response: Contact your system programmer with all the accompanying information.

System Programmer Response: Contact your IBM Service representative for corrective measures. Be prepared to have the necessary information for the error.

RXSQL Warning Messages

The following messages are only issued on EXECSQL invocation, and indicate that RXSQL has issued a warning. The return code is always greater than 1000.

ELO1136I **The statement for *name* was forced into a prepared or declared state during *request* processing.**

Explanation: The statement corresponding to *name* was not in a prepared or declared state. One of the following SQL statements was executed to force a prepared or declared state: CLOSE, Dynamic PREPARE or Extended DECLARE.

System Action: A warning has been issued. The DB2 RXSQL request was completed. Control is returned to the user's REXX program.

User Response: You may continue processing. To avoid this warning, issue the appropriate request in order to put the statement associated with *name* in a Dynamic PREPARED or Extended Dynamic DECLARED state.

ELO1138I **The input data on the OPEN request for *name* was ignored.**

Explanation: An OPEN request for the cursor or statement named *name* has a variable list, however the statement corresponding to *name* is not a statement for which input is valid. As a result, the input has been ignored.

System Action: A warning has been issued. The DB2 RXSQL request was completed. Control is returned to the user's REXX program.

User Response: You may continue processing. To avoid this warning, remove the extra parameters on the OPEN request.

ELO1140I **A Dynamic DECLARE has already been issued.**

Explanation: A Dynamic DECLARE was issued when a Dynamic DECLARE had already been done for the cursor name and statement name. A Dynamic DECLARE is not necessary in this instance and should be removed from your program to improve the performance of your program.

System Action: A warning has been issued. The DB2 RXSQL request was completed. Control is returned to the user's REXX program.

User Response: You may continue processing. To avoid this warning, remove the redundant DECLARE request from your program.

ELO1142I ***number* variables were given, but there are *select_number* select-list columns.**

Explanation: The number of main variables in the REXX host variable list does not match the number of select-list expressions for the select statement.

System Action: A warning has been issued. The DB2 RXSQL request was completed. The number of REXX variables set with the data retrieved from the database is the minimum of *number* and *select_number*. Control is returned to the user's REXX program.

User Response: You may continue processing. To avoid this warning, the number of REXX variables to receive the fetched data must match the number of select-list columns.

ELO1170I **An Extended Dynamic DECLARE has already been issued.**

Explanation: An Extended Dynamic DECLARE was issued when an Extended Dynamic DECLARE had already been done for the cursor name, section number and package name supplied. An Extended Dynamic DECLARE is not necessary in this instance and should be removed from your program to improve the performance of your program.

System Action: A warning has been issued. The DB2 RXSQL request was completed. Control is returned to the user's REXX program.

User Response: You may continue processing. To avoid this warning, remove the redundant DECLARE request from your program.

ELO1175I **The *attributes-variable* specified on the Temporary Extended PREPARE was ignored.**

Explanation: A Temporary Extended PREPARE with an attributes variable was issued to fill an empty section. This attributes variable was ignored because it is ignored by the database manager.

System Action: A warning has been issued. The DB2 RXSQL request was completed. Control is returned to the user's REXX program.

User Response: You may continue processing. To avoid this warning, remove the USING clause from the Extended Dynamic PREPARE request.

ELO1221I **The value for variable *variable* will result in low order truncation.**

Explanation: Low order digits will be truncated from your number with the precision and scale given for the column.

System Action: A warning has been issued. The DB2 RXSQL request was completed. Control is returned to the user's REXX program.

User Response: You may continue processing.

ELO1299I **PUT was reissued after database error. SQLSTATE=*sqlstate*, SQLERRD1=*sqlerrd1*, SQLERRD2=*sqlerrd2*.**

Explanation: A PUT request was rejected by the database manager because the data type or length has changed for a main variable whose ordinal position was *sqlerrd2* in the REXX host variable list. DB2 RXSQL has intercepted this error from the database manager, has issued an SQL CLOSE and OPEN, and has reissued the PUT.

System Action: A warning has been issued. The DB2 RXSQL request was completed. Control is returned to the user's REXX program.

User Response: You may continue processing. To avoid this warning, ensure that all the data you are inserting into column number *sqlerrd2* looks like the same data type and has the same length. You may alternately provide variable qualifiers on your PUT statement that defines the target column's data type.

RXSQL-Supplied Program Messages

The following messages are issued only from the DB2 RXSQL-supplied programs.

ELO2100E **XMITMSG RC= *rc*.**

Explanation: An error occurred when a message from the DB2 RXSQL message repository was being retrieved. XMITMSG ended with a return code of *rc*.

System Action: No further processing is done. Control is returned to the user. The system waits for the user.

User Response: Refer to any other system messages that may have occurred. If the return code was 16 and a message was displayed saying the ELO repository was not found, then enter 'SET LANGUAGE (ADD ELO USER' to load the DB2 RXSQL message repository. For other errors refer to the *VM/ESA: System Messages and Codes* manual for the proper user response.

ELO2101E **Invalid or conflicting *fn ft* parameter(s).**

Explanation: The parameters used for the *fn ft* EXEC were not recognized or resulted in conflicting actions.

System Action: The operation was not executed successfully. Control is returned to CMS.

User Response: Refer to the section in the *VM/ESA REXX/VM Reference* manual describing the *fn ft* parameters allowed on the invocation.

ELO2102I ***** DB2 RXSQL Version *num1* Release *num2* Modification *num3* *****

Explanation: Identifies the DB2 RXSQL version and release modification level the user is invoking.

System Action: Normal completion. The user is returned to CMS.

ELO2111E ***type* text is not installed or database is not available.**

Explanation: The HELP text was not available at the time of invocation. Either the appropriate tables were not installed, not available, or the database was not available.

System Action: The operation was not executed successfully. Control is returned to CMS.

User Response: Refer to the SQLCA variables for more information. Ensure that the database is available and that the HELP table(s) indicated in the SQLCA variables have been created. If the HELP table(s) have been installed, the table(s) may have been temporarily unavailable. Try again later.

ELO2112I *type* **text is not available for topic** *topic* **for language** *langkey***.**

Explanation: There was no DB2 RXSQL or DB2 Server for VM HELP text for the topic *topic*, in the language indicated by *langkey*.

System Action: The operation was not executed successfully. Control is returned to CMS.

User Response: If the topic was an ELO or ARI message ID, ensure that you entered all of the numbers in that message ID. If the topic was an unsigned number, a positive number is assumed. If you wanted HELP for a negative number, place a minus (-) sign before the number.

ELO2121I ********** End-of-Data ************

Explanation: This is an informational message put at the end of the file being XEDITed. It indicates that all data from the query has been retrieved.

System Action: You are at the bottom of the file in XEDIT. The system waits for you to enter the next command.

ELO2122I **Enter MORE to display more rows of data.**

Explanation: Informational message indicating that there is more data to be retrieved and written to the file.

System Action: The system waits for you to enter the next command.

User Response: Enter MORE in the command line to get more rows of data into the file being XEDITed.

ELO2123I **Enter RXSQLHLP** *topic* **to get more information.**

Explanation: More information on the *topic* given may be retrieved by entering RXSQLHLP *topic* on the command line.

System Action: The system waits for you to enter the next command.

User Response: Enter 'RXSQLHLP topic' on the command line, if you wish to get more information on the topic.

ELO2124I **Line truncated** *num* **characters by EXECIO.**

Explanation: The row retrieved was too long to put in the file. *Num* indicates the number of characters that were truncated while writing to the file.

System Action: The truncated row was inserted in the file. The system continues writing rows into the file.

ELO2131I **Current setting is** *case***.**

Explanation: The current case setting used by the RXSELECT EXEC is *case*. If *case* is UPPER, the SQL statement will be converted to upper case before being sent to the database manager. If the *case* is STRING, the statement will be sent to the database manager as it was entered.

System Action: The command was completed and the user is returned to CMS. The system waits for the next command.

ELO2141I **No 'WHERE' clause was used on** *cmd* **statement.**

Explanation: No 'WHERE' clause was specified on your *cmd* statement. More rows may have been affected than was intended.

System Action: The command was completed with warnings. The system waits for the next command.

User Response: You should verify that the change was intended unconditionally on the entire table. Refer to other messages issued and take action as indicated by those messages.

ELO2142I **Row count =** *num***.**

Explanation: *Num* represents the number of rows processed by the command.

System Action: The command was completed with warnings. The system waits for the next command.

User Response: Check the row count against what you expected the number of affected rows to be. Refer to other messages displayed for more information and for an action to be taken. Also refer to SQLERRD.3 description in *DB2 Server for VM Application Programming* manual.

ELO2143I **Row count =** *num1* **, Dependent row count =** *num2***.**

Explanation: *Num1* represents the number of rows processed by the command. *Num2* represents the number of dependent rows affected when the object table is a part of a referential structure.

System Action: The command was completed with warnings. The system waits for the next command.

User Response: Check the row counts against what you expected the number of affected rows to be. Refer to other messages displayed for more information and for an action to be taken. Also refer to SQLERRD.3 and SQLERRD.5 descriptions in *DB2 Server for VM Application Programming* manual.

ELO2144I Enter ROLLBACK or CANCEL to cancel the changes done.

Explanation: The system is waiting for a ROLLBACK or CANCEL response. If neither is entered, a COMMIT will be performed.

System Action: The system waits for a response from the user.

User Response: If you wish to rollback the changes made in this LUW, then enter ROLLBACK or CANCEL. Otherwise, press enter to commit all changes made so far.

ELO2151E Operator commands may not be issued while in a logical unit of work.

Explanation: You are in a logical unit of work. Operator commands are not allowed while you are in a LUW.

System Action: The operator command was not executed. The system waits for the user to enter a command.

User Response: Refer to other messages issued and take action as indicated by those messages.

ELO2152I Do a COMMIT or ROLLBACK and try the command again.

Explanation: Enter COMMIT or ROLLBACK on the command line. If you wish to keep all changes made thus far, enter COMMIT; otherwise, enter ROLLBACK.

System Action: The system waits for the user to enter a command.

User Response: Issue a COMMIT or ROLLBACK to end the current LUW. Rerun the command.

ELO2153E Operator commands may not be issued while in single-user-mode.

Explanation: You are in single-user-mode. Operator commands are not allowed while you are in single-user-mode.

System Action: The operator command was not executed. The system waits for the user to enter a command.

User Response: Do not issue operator commands while in single-user-mode. Restart the database in multiple user mode if you wish to enter operator commands.

ELO2154E Error processing DB2 Server for VM operator commands.

Explanation: An error was encountered while executing the DB2 Server for VM operator command.

System Action: The operator command was not executed. The system waits for the user to enter a command.

User Response: Refer to other messages issued and take action as indicated by those messages.

ELO2155E Invalid SHOW command.

Explanation: The SHOW command you entered was invalid.

System Action: The operator command was not executed. The system waits for the user to enter a command.

User Response: Refer to the *DB2 Server for VSE & VM Operation* manual for more information on the correct usage of the SHOW command. Change the command and rerun the program. Refer to any other messages issued and take action as suggested.

ELO2156E Invalid DB2 Server for VM operator command.

Explanation: The operator command entered was not a valid DB2 Server for VM operator command.

System Action: The operator command was not executed. The system waits for the user to enter a command.

User Response: Refer to other messages issued and take action as indicated by those messages.

ELO2157E No DB2 Server for VM operator command entered.

Explanation: There was no DB2 Server for VM operator command entered.

System Action: The operator command was not executed. The system waits for the user to enter a command.

User Response: Refer to other messages issued and take action as indicated by those messages.

ELO2158E DB2 Server for VM operator command value too long.

Explanation: A value entered with the operator command was too long.

System Action: The operator command was not executed. The system waits for the user to enter a command.

User Response: Refer to other messages issued and take action as indicated by those messages.

ELO2159E Invalid operator command.

Explanation: The operator command entered was not a valid operator command.

System Action: The operator command was not executed. The system waits for the user to enter a command.

User Response: Refer to the *DB2 Server for VSE & VM Operation* manual for more information on operator commands. Change the command as necessary and rerun the program.

ELO2160E The CMS GLOBALV command failed. The return code from GLOBALV was *rc*.

Explanation: An error occurred when an attempt was made to update the LASTING GLOBALV file. GLOBALV ended with a return code of *rc*.

System Action: No further processing is done. Control is returned to the user. The system waits for the user.

User Response: Refer to the *VM/ESA: CMS Command Reference* manual for the proper user response.

ELO2161E The language *language* was not found in the *table* table.

Explanation: *language* was not found in the LANGUAGE or LANGID column of *table*. This national language is not installed.

System Action: No further processing is done. Control is returned to the user. The system waits for the user.

User Response: Query *table* to determine which languages are installed.

Appendix B. Sample Programs with Examples of RXSQL Requests

The example programs in this section describe how to write a set of REXX programs that create a table, insert data into it, and make queries on the table.

Examples Using RXSQL Requests

The first example program shows a REXX program that can be used to create the table named RXEMP. Figure 21 shows a sample input file that can be used to create the RXEMP table. The RXEMP table consists of columns with the headings EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE, SALARY, BONUS, and COMMISSION

```
002130 GARY    M  SAMS      B12  5643  1969-10-01  MANAGER   17  M  1956-11-21  41700  900  4130
002300 JANET   L  HEDGLEY   B09  2345  1972-12-15  ANALYST   16  F  1963-06-01  37900  800  3178
001010 RON     A  LOWRY     D14  2313  1978-01-15  ANALYST   20  M  1959-09-17  38240  600  3000
000990 RANDY   M  SCHENKER  A07  1430  1983-03-22  OPERATOR  15  M  1960-12-17  30190  700  2660
002020 TERRY   A  RAINEY    D11  3243  1989-09-05  DESIGNER  20  M  1967-09-13  32560  500  2408
001840 PAUL    P  CORDON    B09  7070  1985-07-21  FILEREP   18  M  1965-03-05  28090  600  3090
002330 LES     H  FABER     A10  2119  1977-03-18  CLERK     14  M  1952-02-25  27800  400  1777
009236 HEATHER B  DOBSON    D08  3467  1979-04-03  WRITER    16  F  1964-05-31  37600  800  2900
002574 JAY     Q  MERCIER   A11  2946  1991-05-06  WRITER    15  M  1971-09-22  33400  600  2650
003567 DICK    E  SCHMIDT   C04  3847  1972-11-17  CLERK     14  M  1960-12-03  25790  500  2540
002419 HARRY   P  ATWALA    A07  9127  1980-10-28  OPERATOR  16  M  1962-10-30  37940  800  3105
003326 MARY    K  GOODBAR   B09  3943  1974-07-13  MANAGER   18  F  1959-02-25  40360  900  3980
003589 STEVE   S  GOULD     D07  3565  1976-06-12  WRITER    17  M  1956-04-25  39250  350  3050
```

Figure 21. Example of Input File EMPLOYEE INPUT

Creating Tables and Inserting Data

Figure 22 on page 128 is an example of how to use DB2 RXSQL to create a table and how to insert data rows in it.

```
/* EMPCRE */
/*   5648-A70 (C) Copyright IBM Corp. 1973, 1998.                     */
/*   All rights reserved.                                             */
/*   US Government Users Restricted Rights -                          */
/*   Use, duplication or disclosure restricted by                     */
/*   GSA ADP Schedule Contract with IBM Corp.                         */
/*                                                                    */
/*   Licensed Materials - Property of IBM                             */

/* An exec to create a table and a view of the table in SQL to control */
/* employee information.                                              */
/* Data from a file "EMPLOYEE INPUT" is loaded into the table         */
Address 'COMMAND' 1

/* Define the data structure of the table to create                   */

creat_emp = 'CREATE TABLE RXEMP (', 2
                    'EMPNO       CHAR(6) NOT NULL,',
                    'FIRSTNME    VARCHAR(12) NOT NULL,',
                    'MIDINIT     CHAR(1) NOT NULL,',
                    'LASTNAME    VARCHAR(15) NOT NULL,',
                    'WORKDEPT    CHAR(3),',
                    'PHONENO     CHAR(4),',
                    'HIREDATE    DATE,',
                    'JOB         CHAR(8),',
                    'EDLEVEL     SMALLINT NOT NULL,',
                    'SEX         CHAR(1),',
                    'BIRTHDATE   DATE,',
                    'SALARY      DECIMAL(9,2),',
                    'BONUS       DECIMAL(9,2),',
                    'COMM        DECIMAL(9,2) )' 3

'RXSQL EXEC'  creat_emp 4            /* Create the table  */
If rc <> 0 then Signal 'ERROR' 5

view = 'CREATE VIEW EMPVIEW (EMPNO, FIRSTNME, MIDINIT, LASTNAME, JOB,',
       'EDLEVEL, SALARY)',
       'AS SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, JOB, EDLEVEL,',
       'SALARY FROM RXEMP'

'RXSQL EXEC'  view                           /* Create the view  */
If rc <> 0 then Signal 'ERROR'

/* Define the insert statement to fill in rows of the table */

ins_emp = 'INSERT INTO RXEMP VALUES (', 6
              ':emp,:fname,:mid,:lname,:wdpt,:ph,:hire,:job,',
              ':ed,:sex,:birth,:sal,:bon,:comm )'

'RXSQL PREP IEMP'  ins_emp 7         /* Prepare the INSERT statement */
If rc <> 0 then Signal 'ERROR'
```

Figure 22 (Part 1 of 2). Program to Create a Table and Insert Data

```
Do Forever                                    /* Input data to the table */
   'EXECIO 1 DISKR EMPLOYEE INPUT * (LIFO'  8   /* Read from file */
   If rc <> 0 then Leave  9
   /* Get the data into REXX variables */
   Parse Upper Pull emp fname mid lname wdpt ph hire job,
                    ed sex birth sal bon comm . 10
   If emp = '' then Iterate  11        /* Check for Blank record           */
   emp = "'"emp"'"  12                 /* Make sure it is a char string so */
   fname = "'"fname"'"                 /* RXSQL does not change numeric chars */
   mid = "'"mid"'"                     /* to numeric type input type.      */
   lname = "'"lname"'"
   wdpt = "'"wdpt"'"
   ph = "'"ph"'"
   hire= "'"hire"'"
   job = "'"job"'"
   sex = "'"sex"'"
   birth = "'"birth"'"

   'RXSQL CALL IEMP'  13               /* Call the prepared INSERT statement */
   If rc <> 0 then Signal 'ERROR'
End
'FINIS EMPLOYEE INPUT *'               /* Close input file         */
'RXSQL COMMIT'  14                     /* Commit the inserted data */
'RXSQL PURGE IEMP'  15                 /* Purge prepared statement */
If rc = 0 then Exit


ERROR:                          /* A severe error occurred  */
                             /* Forget all inserted data so far */
errrc = rc
If errrc >= 100 then Do  16
   Say '  RXSQL Error:' errrc rxsqlmsg
   'RXSQL ROLLBACK'
End
Else If errrc = 4 | errrc = 8 then Do  17
   Say '  Sqlcode:' sqlcode  18
   Say ' Sqlstate:' sqlstate
   Do errd = 1 to 6
      If sqlerrd.errd <> 0 then Say '  Sqlerrd.'errd':' sqlerrd.errd
   End
   If sqlerrp <> '' then Say '  Sqlerrp:' sqlerrp
   If sqlerrm <> '' then Say '  Sqlerrm:' sqlerrm
   If sqlwarn <> '' then Say '  Sqlwarn:' '"'sqlwarn'"'
   If sqlcode < 0 & "INDEX"('WS',"SUBSTR"(sqlwarn,7,1)) = 0 then  19
      'RXSQL ROLLBACK'
End
Exit errrc  20
```

Figure 22 (Part 2 of 2). Program to Create a Table and Insert Data

Notes for EMPCRE Program

The following notes refer to the lines in the EMPCRE program identified by reverse video numbers in Figure 22 on page 128:

1 `Address COMMAND` causes REXX to pass command lines directly to a module without searching for a program of the same name. You should include `Address COMMAND` in your programs for performance reasons.

2 The comma at the end of the line causes REXX to concatenate the current line and the one that follows, replacing the comma with a blank. Using a comma for concatenation allows you to build long SQL statements.

3 This CREATE statement does not include an `IN dbspace_name` phrase. As a result, SQL chooses a PRIVATE DBSPACE for the table. To explicitly select a particular dbspace, use the `IN dbspace_name` phrase.

4 REXX expands the `creat_emp` variable and passes it to DB2 RXSQL as part of the command. DB2 RXSQL passes the statement to SQL for processing.

5 The REXX variable `rc` is set with the return code from DB2 RXSQL. If REXX, DB2 RXSQL, or the database manager indicates an error, this program (at label `ERROR`) displays all the error variables set by DB2 RXSQL, and then ends processing.

6 The names of the REXX variables are preceded by colons. They hold the values to be inserted into the fields of the RXEMP table.

7 This RXSQL PREP statement prepares the INSERT statement and gives it the name IEMP. DB2 RXSQL stores all the input variable names until the CALL request is made.

8 EXECIO is a CMS command to read a line from the EMPLOYEE INPUT file and put it on the program stack in last-in-first-out (LIFO) order.

9 The loop ends when the EXECIO command has an error (assumed to be end-of-file).

10 `Parse Upper Pull` gets the line read by EXECIO from the program stack. Each token in the line is assigned to the variables `emp`, `fname`, `mid`, `lname`, `wdpt`, `ph`, `hire`, `job`, `ed`, `sex`, `birth`, `sal`, `bon`, and `comm` respectively. The period after `comm` causes REXX to ignore any extra tokens on the line. Without the period, `comm` is set with all the characters remaining in the record, including trailing blanks.

11 If the record is all blank, the loop iterates and the program ignores the record.

12 The values `emp`, `fname`, `mid`, `lname`, `wdpt`, `ph`, `hire`, `job`, `sex`, and `birth` are put into quotation marks to ensure that DB2 RXSQL does not interpret the values as numeric. DB2 RXSQL removes the quotation marks before sending the values to the database manager on the CALL request.

13 The CALL request executes the IEMP statement prepared earlier. The input variables, held since the PREP statement (see line **7**), are converted from REXX variable values to SQL input data.

14 Processing reaches this point only if no errors occurred. The RXSQL COMMIT statement commits all data inserted in the database.

15 This RXSQL PURGE statement removes the prepared SQL statement named IEMP.

16 Processing reaches this point only if the program encounters a non-zero return code. This line checks for a DB2 RXSQL error (a number greater than 99). If the return code is greater than 99, the program displays the `rxsqlmsg` variable (a textual description of the error that occurred). The RXSQL ROLLBACK statement cancels any changes made to the database.

17 If the return code is not an RXSQL error, the program checks whether the return code is an SQL return code of 4 or 8. A return code of 4 indicates an SQL warning. A return code of 8 indicates an SQL error.

18 This line begins processing the SQL error indicator variables that display the following information:

`sqlcode`	The primary SQL error code. You can issue the `EXEC RXSQLHLP sqlcode` command to determine the meaning of the displayed code.
`sqlstate`	Error codes common to all distributed relational databases.
`sqlerrd.1 - sqlerrd.6`	The secondary error codes and other information.
`sqlerrp`	The SQL module that detected the error.
`sqlerrm`	The values to be substituted into the error message text. See the *DB2 Server for VM Messages and Codes* manual for full error text.
`sqlwarn`	Eleven characters of error indicators. For a more detailed explanation of `sqlwarn`, see page 58.

19 When the database manager cancels a transaction, it sets the seventh character of `sqlwarn` to W or S. This line checks whether the database manager has canceled the transaction or not by checking the value of the seventh character of `sqlwarn`. If the value is not W or S, the program issues an explicit DB2 RXSQL ROLLBACK command to cancel any changes to the database.

20 If the program encounters an error, it finishes processing and exits to VM.

Note: You can create a more sophisticated program to handle both recoverable situations and unrecoverable situations.

Retrieving Data from a Table

Figure 23 on page 132 shows a REXX program that you can use to retrieve data from the table created by the EMPCRE program.

```
/* EMPSEL */
/*   5648-A70 (C) COPYRIGHT IBM CORP. 1990, 1998.   */
/*   Licensed material - Program Property of IBM    */
/*   Refer to copyright instructions form G120-2083 */

/* An exec to display employees with a salary less then some amount */
/* Defaults: ANALYSTs with a salary less than $38,000              */
Address 'COMMAND'

Parse Upper Arg job salary . 1
If job = '' then job = 'ANALYST' 2
Else job = "'"job"'"  3               /* Make sure it is char type */
If salary = '' then salary = 38000  4

/* Define the select statement to pick part of the data from DB2 Server for VM */

sel_emp = 'SELECT * FROM EMPVIEW WHERE SALARY < :salary AND JOB = :job' 5

'RXSQL PREP SELEMP' sel_emp  6        /* Prepare the SELECT statement */
if rc<>0 then signal 'ERROR'
'RXSQL OPEN SELEMP'  7                /* Open the SELECT statement */
if rc<>0 then signal 'ERROR'

If rc = 0 then Do                     /* If no errors occurred then */
   Do Forever                         /*   Get data from the table */

      /* Get the data into REXX variables from DB2 Server for VM */
      'RXSQL FETCH SELEMP emp fname mid lname job ed sal'  8
      If (rc = 4 & SQLCODE = 100)  then Leave  9  /* If there is no more */
      Else If rc <> 0 then Signal 'ERROR'     /* If there is an error */

      /* Check for Nulls */
      If "SYMBOL"('job') <> 'VAR' then job = '?'  10
      If "SYMBOL"('sal') <> 'VAR' then sal = '?'

      /* Type the data on the users terminal */
      Say 'Employee: 'emp '    ' fname mid lname
      Say 'Job:'job'    Education:'ed'     Salary:'sal
      Say ''  11                      /* Space down one line */
   End
   'RXSQL CLOSE SELEMP' 12            /* Close the SELECT statement */
   if rc<>0 then signal 'ERROR'
End
'RXSQL COMMIT'                        /* Commit the transaction */
'RXSQL PURGE SELEMP' 13              /* Purge PREPed statement */
If rc=0 then Exit
```

Figure 23 (Part 1 of 2). Program to Select Data from an SQL Table

```
ERROR:                               /* A fatal error occurred  */
                                     /* Forget all inserted data so far */
errrc = rc
If errrc >= 100 then Do
   Say '  RXSQL Error:' errrc rxsqlmsg
   'RXSQL ROLLBACK'
End
Else If errrc = 4 | errrc = 8 then Do
   Say '  Sqlcode:' sqlcode
   Say ' Sqlstate:' sqlstate
   Do errd = 1 to 6
      If sqlerrd.errd <> 0 then Say '  Sqlerrd.'errd':' sqlerrd.errd
   End
   If sqlerrp <> '' then Say '  Sqlerrp:' sqlerrp
   If sqlerrm <> '' then Say '  Sqlerrm:' sqlerrm
   If sqlwarn <> '' then Say '  Sqlwarn:' "'"sqlwarn"'"
   If sqlcode < 0 & "INDEX"('WS',"SUBSTR"(sqlwarn,7,1)) = 0 then
      'RXSQL ROLLBACK'
End
Exit errrc
```

Figure 23 (Part 2 of 2). Program to Select Data from an SQL Table

Notes for EMPSEL Program

The following notes refer to the lines in the EMPSEL program identified by reverse video numbers:

1 This program allows two arguments to be passed to it: `salary` and `job`.

2 If `job` is null, this line causes DB2 RXSQL to set `job` to a value of ANALYST.

3 This line ensures that any value specified for `job` is identified to DB2 RXSQL as a character type.

4 If `salary` is null, this line causes DB2 RXSQL to set `salary` to a value of 38000.

5 This SQL SELECT statement queries the RXEMP table for for employees with a certain job who have less than a certain salary. Colons in the variable names indicate to the database manager that the REXX variables `job` and `salary` are to supply these values.

6 This RXSQL PREP statement prepares the SELECT statement into the database. The statement is SELEMP for DB2 RXSQL. DB2 RXSQL stores the input variables `job` and `salary` for a subsequent OPEN request.

7 This RXSQL OPEN statement opens a cursor (for prepared statement SELEMP) on the query, using the values of the input variables `job` and `salary`.

8 This RXSQL FETCH statement reads one row from the result table each time through the loop. DB2 RXSQL uses the fields of the row as values for the variables listed on the FETCH request.

9 This line causes the loop to end if no more rows satisfy the query. This is not a true error situation. When there is a true error situation, the error routine takes control.

10 The columns of the table might contain null values. The `SYMBOL` function tests the REXX variable to see if it is defined or not. If the column value is null, DB2 RXSQL makes the variable that is to receive the column value undefined, and the SYMBOL function returns `LIT`. This means that the variable does not have a value. If the column has a value, DB2 RXSQL sets the variable that is to receive the column value, and the SYMBOL function returns `VAR`, meaning that the variable has a value.

11 These three lines format the results from one row of the query and display it for the user.

12 This RXSQL CLOSE statement closes the cursor (for prepared statement SELEMP) on the result table. A cursor for SELEMP can be opened again with another set of input `job` and `salary` values.

13 This RXSQL PURGE statement deletes the prepared statement named SELEMP. After the program processes this statement, SELEMP no longer exists. A cursor cannot be opened for it with another set of input values.

Examples Using Extended Dynamic RXSQL Requests

You can use an XPREP (or Extended PREPARE) request to accomplish the same results as the PREP request in the EMPSEL program. With an XPREP request, two programs replace the EMPSEL program. One program creates and prepares a package in the database. The other allows general users to access the data.

When the XPREP statement is processed, the database manager returns a section number identifying the position in the package that the prepared statement occupies. To use this prepared statement in any run-time program, you must use the same section number returned by the XPREP, in an extended DECLARE statement.

Since the XPREP needs to be done only once and the DECLARE every time you wish to use the prepared statement, two separate programs are typically used. One contains the XPREP statement and the other, a run-time program, contains the extended DECLARE statement. The following example programs describe a method of passing the section number from the first program to the run-time program.

Creating a Package

The following programs show one method of saving the section number. The EMPPRP program (shown in Figure 24 on page 135), contains the XPREP statement and generates the intermediate EMPDCL program (shown in Figure 25 on page 137). The EMPSELX program (shown in Figure 26 on page 138) is a run-time program that invokes the EMPDCL program to declare statement names for the appropriate SQL statement.

Note: The example generates a package with only one statement, but you can put more than one statement into a single package.

```
/* EMPPRP */
/*   5648-A70 (C) Copyright IBM Corp. 1973, 1998.                      */
/*   All rights reserved.                                              */
/*   US Government Users Restricted Rights -                           */
/*   Use, duplication or disclosure restricted by                      */
/*   GSA ADP Schedule Contract with IBM Corp.                          */
/*                                                                     */
/*   Licensed Materials - Property of IBM                              */

/* An exec to create and prepare the SQL statements available */
/* to the users                                               */
Address 'COMMAND'

/* Define the select statement to pick part of the data from DB2 Server for VM */
/* Note: Use parameter markers not variable names */
/*       Variable names given on OPEN */
sel_emp = 'SELECT * FROM EMPVIEW WHERE SALARY < ? AND JOB = ?' 1

/* Find out who I am */
'IDENTIFY (LIFO'  2
Parse Upper Pull myid . 3

/* Prepare to add DECLAREs to a generated exec called EMPDCL */
'ERASE EMPDCL EXEC'  4
/* Put in comment so exec will be a REXX exec */
'EXECIO 1 DISKW EMPDCL EXEC A (STRING /* EMPDCL */'  5
'EXECIO 1 DISKW EMPDCL EXEC A (STRING Trace "Err"' 6

Trace 'Err'
Signal ON ERROR  7                    /* Any error aborts */

/* Create package */
'RXSQL CREATE PACKAGE' myid'.EMPPROG USING BLOCK'  8

/* Prepare the SELECT statement */
'RXSQL XPREP' myid'.EMPPROG' sel_emp  9

/* Generate DECLARE with result of XPREP for EMPSELX to use */
decl ="'RXSQL DECLARE SELEMP CURSOR FOR" sqlstmtn "IN" myid".EMPPROG'"  10
'EXECIO 1 DISKW EMPDCL EXEC A (STRING' decl  11

'RXSQL COMMIT WORK'  12               /* Commit the transaction */

'RXSQL EXEC GRANT RUN ON EMPPROG TO PUBLIC'  13

'RXSQL COMMIT WORK'                          /* Commit the transaction */
Exit
```

Figure 24 (Part 1 of 2). Program to Define Extended Prepared Package

```
ERROR:
Signal OFF ERROR 14
errrc = rc
If errrc >= 100 then Do
   Say '  RXSQL Error:' errrc rxsqlmsg
   'RXSQL ROLLBACK'
End
Else If errrc = 4 | errrc = 8 then Do
   Say '  Sqlcode:' sqlcode
   Say ' Sqlstate:' sqlstate
   Do errd = 1 to 6
      If sqlerrd.errd <> 0 then Say '  Sqlerrd.'errd':' sqlerrd.errd
   End
   If sqlerrp <> '' then Say '  Sqlerrp:' sqlerrp
   If sqlerrm <> '' then Say '  Sqlerrm:' sqlerrm
   If sqlwarn <> '' then Say '  Sqlwarn:' "'"sqlwarn"'"
   If sqlcode < 0 & "INDEX"('WS',"SUBSTR"(sqlwarn,7,1)) = 0 then
      'RXSQL ROLLBACK'
End
Exit errrc
```

Figure 24 (Part 2 of 2). Program to Define Extended Prepared Package

Notes for EMPPRP Program

The following notes refer to the lines in the EMPPRP program identified by reverse video numbers:

1 This SELECT statement is the same as the SELECT statement used in the EMPSEL program, except that the variable names are replaced by parameter markers. The OPEN request in the EMPSELX program provides the variable names to replace the parameter markers.

2 IDENTIFY is a CMS command that returns the current VM user ID, along with other information. The LIFO option causes IDENTIFY to stack the result in last-in-first-out order.

3 This statement reads the stacked result of IDENTIFY and sets the `myid` variable with the current user ID. This value is used as the owner ID of the package. Some other value could be used if the user has appropriate database privileges.

4 This line erases any existing copies of the program to be generated.

5 This line of the EMPPRP program writes the first line of the generated EMPDCL program. The first line of the EMPDCL program must be a REXX comment to identify the program as a REXX program.

6 This line causes a trace of any errors that occur when the EMPDCL program runs.

7 If any command after this point sets a non-zero return code, REXX immediately branches to the label ERROR.

8 This RXSQL CREATE statement creates the package into which the SELECT statement is prepared. The BLOCK option allows the performance improvements achieved by blocking multiple rows from the database on the DB2 RXSQL FETCH statement. If you choose not to use the BLOCK option, you do not have to change any programs that use the package.

The default options REPLACE, NOMODIFY, and KEEP are not specified on this CREATE statement. These options replace the package if it already exists, keeping the authorizations that were granted on the previous package.

9 This RXSQL XPREP statement prepares the SELECT statement into the package. DB2 RXSQL sets the SQLSTMTN variable with the SQL statement number of the statement in the package.

10 This line sets the variable `decl` to a RXSQL DECLARE statement. The DECLARE statement declares the name SELEMP for the statement in the package identified by SQLSTMTN

11 This line writes the DECLARE statement to the generated program.

12 COMMIT causes the database manager to save the package in the database.

13 This line processes an SQL GRANT statement to allow the public to use the package called EMPPROG. Without this GRANT statement, only the creator of the package can use the package. You do not necessarily have to authorize the public to use the package; you can use a GRANT statement that authorizes only specific users.

14 This line turns off the automatic signal on ERROR. Turning off the signal prevents error loops.

The Generated EMPDCL program

Figure 25 shows the program generated by the EMPPRP Program. The DB2 RXSQL DECLARE statement declares the cursor `SELEMP` and associates it with section `1` in the package `xxx.EMPPROG`. The characters `xxx` are replaced with the name of the user running the EMPPRP program.

```
/* EMPDCL */
Trace "Err"
'RXSQL DECLARE SELEMP CURSOR FOR 1 IN xxx.EMPPROG'
```

Figure 25. Exec Generated by EMPPRP

Selecting Data

The EMPSELX Program in Figure 26 on page 138 is a slight modification of the EMPSEL program shown in Figure 23 on page 132. Instead of a PREP request, the EMPSELX program has a DECLARE request that defines a statement name for a prepared statement in a package.

```
/* EMPSELX version using extended dynamic SQL */
/*   5648-A70 (C) COPYRIGHT IBM CORP. 1990, 1998.   */
/*   Licensed material - Program Property of IBM    */
/*   Refer to copyright instructions form G120-2083 */

/* An exec to display employees with a salary less than some amount */
/* Defaults: ANALYSTs with a salary less than $38,000 */
Address 'COMMAND'

Parse Upper Arg job salary .
If job = '' then job = 'ANALYST'
Else job = "'"job"'"                           /* Make sure it is char type */
If salary = '' then salary = 38000

/* Declare name for SELECT statement prepared in EMPPRP */
'EXEC EMPDCL'  1                        /* Exec generated by EMPPRP */

/* Open the SELECT statement and give the input parms */
'RXSQL OPEN SELEMP salary job'  2
if rc<>0 then signal 'ERROR'

If rc = 0 then Do                              /* If no errors occurred then */
   Do Forever                                  /* Get data from the table */

      /* Get the data into REXX variables from DB2 Server for VM */
      'RXSQL FETCH SELEMP emp fname mid lname job ed sal'
      If (rc = 4 & SQLCODE = 100)  then Leave  /* If there is no more */
      Else If rc <> 0 then Signal 'ERROR'      /* If there is an error */

      /* Check for Nulls */
      If "SYMBOL"('job') <> 'VAR' then job = '?'
      If "SYMBOL"('sal') <> 'VAR' then sal = '?'

      /* Type the data on the users terminal */
      Say 'Employee: 'emp '    ' fname mid lname
      Say 'Job:'job'    Education:'ed'     Salary:'sal
      Say ''       /* Space down one line */

   End
   'RXSQL CLOSE SELEMP'                        /* Close the SELECT statement */
   if rc<>0 then signal 'ERROR'
End
```

Figure 26 (Part 1 of 2). Program to Select Data from an SQL Table after Extended Prepare

```
'RXSQL COMMIT'                               /* Commit the transaction */
'RXSQL PURGE SELEMP'                         /* Purge Declared name */
If rc=0 then Exit

ERROR:                                       /* A fatal error occurred  */
                                             /* Forget all inserted data so far */
errrc = rc
If errrc >= 100 then Do
   Say '  RXSQL Error:' errrc rxsqlmsg
   'RXSQL ROLLBACK'
End
Else If errrc = 4 | errrc = 8 then Do
   Say '  Sqlcode:' sqlcode
   Say ' Sqlstate:' sqlstate
   Do errd = 1 to 6
      If sqlerrd.errd <> 0 then Say '  Sqlerrd.'errd':' sqlerrd.errd
   End
   If sqlerrp <> '' then Say '  Sqlerrp:' sqlerrp
   If sqlerrm <> '' then Say '  Sqlerrm:' sqlerrm
   If sqlwarn <> '' then Say '  Sqlwarn:' "'"sqlwarn"'"
   If sqlcode < 0 & "INDEX"('WS',"SUBSTR"(sqlwarn,7,1)) = 0 then
      'RXSQL ROLLBACK'
End
Exit errrc
```

Figure 26 (Part 2 of 2). Program to Select Data from an SQL Table after Extended Prepare

Notes for EMPSELX Program

The following notes refer to the lines in the EMPSELX program identified by reverse video numbers:

1 This program statement invokes the generated EMPDCL program, that contains the DECLARE request.

An alternative to executing the program is to include it in the EMPSELX program after running EMPPRP. Including the EMPDCL program in the EMPSELX program requires that only one program be made available for general use instead of two.

2 This OPEN statement names the input variables. In the EMPSEL program, DB2 RXSQL stored the variable names given in the PREP statement until subsequent processing of the OPEN statement. This method worked because DB2 RXSQL recalled the variables in the same transaction. In the program in Figure 26, variables are not used in the same transaction. DB2 RXSQL cannot store variables between transactions.

More Examples Using Extended Dynamic RXSQL Requests

The following programs use extended dynamic SQL to interactively update entries in the table and to insert new records into the database.

Defining a Package

The EMPPRPM program shown in Figure 27 defines the package for the statements that the EMPUPD program shown in Figure 29 on page 143 uses for updating, inserting, and scaling salaries. EMPPRPM program prepares the SQL statements into a different package than the one created by EMPPRP program because the statements are authorized for use by managers only. The EMPPRP program authorizes the general public to use its statements.

```
/* EMPPRPM */
/*   5648-A70 (C) Copyright IBM Corp. 1973, 1998.                    */
/*   All rights reserved.                                            */
/*   US Government Users Restricted Rights -                         */
/*   Use, duplication or disclosure restricted by                    */
/*   GSA ADP Schedule Contract with IBM Corp.                        */
/*                                                                   */
/*   Licensed Materials - Property of IBM                            */

/* An exec to create and prepare the SQL statements available to */
/* managers to update the tables */
Address 'COMMAND'

/* Define the insert statement to fill in rows of the table */
ins_emp = 'INSERT INTO EMPVIEW VALUES(?,?,?,?,?,?,?)' 1

/* Define the update to change a emp salary */
set_salary = 'UPDATE EMPVIEW SET SALARY = ?',  2
             'WHERE EMPNO = ? AND LASTNAME = ?'

/* Define the update to mark down/up all the salaries */
scale_salary = 'UPDATE EMPVIEW SET SALARY = SALARY * ?'

/* Find out who I am */
'IDENTIFY (LIFO'
Parse Upper Pull myid .

/* Prepare to add DECLAREs to a generated exec called EMPDCLM */
'ERASE EMPDCLM EXEC'
/* Put in comment so exec will be a REXX exec */
'EXECIO 1 DISKW EMPDCLM EXEC A (STRING /* EMPDCLM */'
'EXECIO 1 DISKW EMPDCLM EXEC A (STRING Trace "Err"'

Trace 'Err'
Signal ON ERROR                               /* Any error aborts */

/* Create package */
'RXSQL CREATE PACKAGE' myid'.EMPUPD'  3

/* Prepare the INSERT statement */
'RXSQL XPREP' myid'.EMPUPD' ins_emp
```

Figure 27 (Part 1 of 2). Program to Define Extended Prepare Package for Updating

```
/* Generate DECLARE with result of XPREP for EMPUPD to use */
decl = "'RXSQL DECLARE INSEMP CURSOR FOR" sqlstmtn "IN" myid".EMPUPD'"
'EXECIO 1 DISKW EMPDCLM EXEC A (STRING' decl

/* Prepare the first UPDATE statement */
'RXSQL XPREP' myid'.EMPUPD' set_salary
decl = "'RXSQL DECLARE SETEMP CURSOR FOR" sqlstmtn "IN" myid".EMPUPD'"
'EXECIO 1 DISKW EMPDCLM EXEC A (STRING' decl

Trace 'Off'
Signal OFF ERROR  4                     /* Don't signal on error */

/* Prepare the second UPDATE statement */
dcllist = 'F'  5                        /* Describe one input variable, Float type */
'RXSQL XPREP' myid'.EMPUPD USING dcllist' scale_salary  6
If rc > 4 then Signal 'ERROR'  7        /* Ignore expected warning */

Trace 'Err'
Signal ON ERROR  8                      /* Any error aborts */

decl ="'RXSQL DECLARE SCALEEMP CURSOR FOR" sqlstmtn "IN" myid".EMPUPD'"
'EXECIO 1 DISKW EMPDCLM EXEC A (STRING' decl

'RXSQL COMMIT WORK'                            /* Commit the transaction */

'RXSQL EXEC GRANT RUN ON EMPUPD TO MANAGER'  9

'RXSQL COMMIT WORK'                            /* Commit the transaction */
Exit

ERROR:
Signal OFF ERROR
errrc = rc
If errrc >= 100 then Do
   Say '  RXSQL Error:' errrc rxsqlmsg
   'RXSQL ROLLBACK'
End
Else If errrc = 4 | errrc = 8 then Do
   Say '  Sqlcode:' sqlcode
   Say ' Sqlstate:' sqlstate
   Do errd = 1 to 6
      If sqlerrd.errd <> 0 then Say '  Sqlerrd.'errd':' sqlerrd.errd
   End
   If sqlerrp <> '' then Say '  Sqlerrp:' sqlerrp
   If sqlerrm <> '' then Say '  Sqlerrm:' sqlerrm
   If sqlwarn <> '' then Say '  Sqlwarn:' "'"sqlwarn"'"
   If sqlcode < 0 & "INDEX"('WS',"SUBSTR"(sqlwarn,7,1)) = 0 then
      'RXSQL ROLLBACK'
End
Exit errrc
```

Figure 27 (Part 2 of 2). Program to Define Extended Prepare Package for Updating

Notes for EMPPRPM

The following notes refer to the lines in the EMPPRPM program identified by reverse video numbers:

1 A parameter marker represents each input value to be provided when the program calls the INSERT statement.

2 The expression used to scale SALARY is not a simple variable and requires a USING clause on a subsequent XPREP statement (see line **6**).

3 This CREATE PACKAGE statement does not specify the BLOCK option. Blocking is not appropriate here, because the EMPPRPM program is an interactive application.

4 This line turns off the REXX automatic branch-on-error action, because the XPREP request on the following line always generates an SQL warning, indicating that the SQL statement does not have a WHERE clause. This warning sets the fifth character of the SQLWARN variable to W.

5 This UPDATE statement has one input variable. The variable type is floating point. With numeric input, you should specify a data type that encompasses all possible input values so that you do not lose data when SQL translates the input value to the type you specify.

6 This RXSQL XPREP statement uses the name of the variable that contains the definition list `dcllist`. DB2 RXSQL obtains the value before preparing the statement.

7 If the program encounters an error more serious than the anticipated warning, this line signals the error exit routine.

8 This line turns on automatic error branching for the remainder of processing.

9 This DB2 RXSQL program statement processes the SQL statement to grant run authority to a specific SQL ID named MANAGER.

The Generated EMPDCLM program

Figure 28 shows the program generated by the EMPPRP program.

```
/* EMPDCLM */
Trace "Err"
'RXSQL DECLARE INSEMP CURSOR FOR 1 IN xxx.EMPUPD'
'RXSQL DECLARE SETEMP CURSOR FOR 2 IN xxx.EMPUPD'
'RXSQL DECLARE SCALEEMP CURSOR FOR 3 IN xxx.EMPUPD'
```

Figure 28. Program Generated by EMPPRPM

Interactive Updating

The EMPUPD program shown in Figure 29 on page 143 is an example of an interactive application that waits for commands from a user who wants to update the RXEMP table. The program can insert new rows into the table, set a new salary for an existing employee in the table, or make a percentage change to all the salaries in the table.

```
/* EMPUPD */
/*   5648-A70 (C) Copyright IBM Corp. 1973, 1998.        */
/*   All rights reserved.                                */
/*   US Government Users Restricted Rights -             */
/*   Use, duplication or disclosure restricted by        */
/*   GSA ADP Schedule Contract with IBM Corp.            */
/*                                                       */
/*   Licensed Materials - Property of IBM                */
/* An exec to interactively insert, set a salary, or     */
/* mark up/down all salaries                             */
trace 'Off' /* 1 */
Address 'COMMAND'

/* Declare INSEMP, SETEMP, SCALEEMP as cursor and   */
/* statement names by executing the exec generated  */
/* by EMPPRPM                                       */
'EXEC EMPDCLM' /* 2 */

Do Forever /* 3 */
   Say 'Enter command: Insert, Set, Update, COMMit, ROLLback, or Quit' /* 4 */
   Parse Upper Pull cmd .  /* 5 */
   Select /* 6 */
      When cmd = '' then Nop /* 7 */
      When ABBREV('QUIT',cmd,1) then Leave /* 8 */ /* terminate loop */
      When ABBREV('INSERT',cmd,1) then Do
         Do Forever /* 9 */
           Say 'Enter Employee number, First name, Middle initial,', /* 10 */
               'Last name, Job,'
           Say 'Education level and Salary'
           Say ' (Empty line to quit)'

           Parse Upper Pull employee_number first_name,
                            middle_initial last_name job ed_level salary .

           If employee_number = '' then Leave /* 11 */
           /* Got all input? */
           If salary <> '' then Do /* 12 */
          /* Use variable qualifiers to ensure */
          /* proper types                      */
             'EXECSQL CALL INSEMP USING',
                  ':employee_number (CHAR(6)),',
                  ':first_name   (VARCHAR(12)),',
                  ':middle_initial  (CHAR(1)),',
                  ':last_name    (VARCHAR(15)),',
                  ':job             (CHAR(8)),',
                  ':ed_level     (SMALLINT),',
                  ':salary       (DECIMAL(9,2))' /* 13 */
               If rc <> 0 then Call DSCERROR /* 14 */
               Else Say 'Inserted.'
            End  /* salary <> ''     */
         End     /* inner Do forever */
      End        /* When INSERT      */
```

Figure 29 (Part 1 of 3). Example Interactive Program to Update the EMPLOYEE Table

```
      When ABBREV('UPDATE',cmd,1) then Do
         Say 'Enter percent salary change. (Empty line to quit)'
         Say '  5 means add 5 percent to all current salaries,'
         Say ' -4 means subtract 4 percent from all current salaries.'
         Parse Pull factor .
         If factor <> '' then Do
            factor = 1 + factor/100 /* 15 */
            /* Scale all salaries */
            'EXECSQL CALL SCALEEMP USING :factor (DECIMAL(6,3))'
            If rc >= 0, /* 16 */
               Then Say sqlerrd.3 'row(s) updated.'
            If rc <> 0 then call DSCERROR
         End /* factor <> '' */
      End    /* When UPDATE  */
      When ABBREV('SET',cmd,1) then Do
         Do Forever
            Say 'Enter Emp#, Last name, SALARY. (Empty line to quit)'
            Parse Upper Pull,
                  employee_number last_name salary .
            /* Terminate inner loop */
            If employee_number = '' then Leave
            If last_name <> '' & salary <> '' then Do
               /* Reset salary*/
               'EXECSQL CALL SETEMP',
                   ':salary     (DECIMAL(9,2)),',
                   ':employee_number (CHAR(6)),',
                   ':last_name  (VARCHAR(15))'
               If rc >= 0,
                  then Say sqlerrd.3 'row(s) updated.'
               If rc <> 0 Then Call DSCERROR
            End
         End  /* Inner Do Forever */
      End     /* When SET        */
      When ABBREV('COMMIT',cmd,4) then Do /* 17 */
         'EXECSQL COMMIT'
         If rc <> 0 then Call DSCERROR
      End
      When ABBREV('ROLLBACK',cmd,4) then Do
         'EXECSQL ROLLBACK'
         If rc <> 0 then Call DSCERROR
      End
      Otherwise Say 'Unknown command' /* 18 */
   End  /* outer Select     */
End     /* outer Do Forever */

/* Commit the transaction */
'EXECSQL COMMIT RELEASE' /* 19 */
If rc <> 0 then Call DSCERROR
/* Purge Declared names */
'EXECSQL PURGE INSEMP, SETEMP, SCALEEMP'
Exit
```

Figure 29 (Part 2 of 3). Example Interactive Program to Update the EMPLOYEE Table

```
DSCERROR:
errrc = rc

/* Display the RXSQL request */
say 'RXSQL Request   :' rxsqlrequest

Select
  When errrc >=1000  then msgpart='RXSQL warning:'
  When errrc  =  10,
       then msgpart='Application Server warning:'
  When errrc  = -10,
       then msgpart='Application Server error:'
  When errrc <=-100  then msgpart='RXSQL error:'
  Otherwise msgpart='Unexpected return code from RXSQL:'
end

Say msgpart errrc rxsqlmsg
If errrc = 10 | errrc = -10 Then do
   Say '  Sqlcode:' sqlcode
   Say ' Sqlstate:' sqlstate
   Do errd = 1 to 6
      If sqlerrd.errd <> 0 then Say '  Sqlerrd.'errd':' sqlerrd.errd

   End
   If sqlerrp    <> '' then Say ' Sqlerrp :' sqlerrp

   If sqlerrmc   <> '' then Say ' Sqlerrmc:' sqlerrmc
   If sqlwarn.0  <> '' then Do
      warnflgs=sqlwarn.0||sqlwarn.1||sqlwarn.2||sqlwarn.3
      warnflgs= warnflgs||sqlwarn.4||sqlwarn.5||sqlwarn.6
      warnflgs= warnflgs||sqlwarn.7||sqlwarn.8||sqlwarn.9
      warnflgs= warnflgs||sqlwarn.10
      Say ' Sqlwarn : '"'"warnflgs"'"
   end
   If INDEX('WS',sqlwarn.6) <> 0, /* 20 */
      Then Exit errrc /* Terminate exec */

End
/* return back to interactive questions */
Return /* 21 */
```

Figure 29 (Part 3 of 3). Example Interactive Program to Update the EMPLOYEE Table

Notes for EMPUPD

The following notes refer to the lines in the EMPUPD program identified by reverse video numbers:

1 REXX tracing is set to `Off` from the default of `Normal`. Changing the default prevents REXX from displaying the EXECSQL statement whenever DB2 RXSQL returns a negative return code to signal an error condition.

2 The generated program declares all three statement names. Declaring all the names at once, even if they are not used, does not affect performance.

3 This Do Forever loop asks for input until a REXX Leave statement breaks the loop.

4 This line prompts for input. The capital letters indicate the minimum abbreviation.

5 This line reads the command typed from the terminal. The first token is assigned to the variable `cmd`. Any other tokens are discarded.

6 This REXX SELECT statement processes the first When clause that evaluates as true. If no When clause evaluates as true, the Select statement processes the Otherwise clause in line **18**.

7 This line tests for null input. If the input is null, the program prompts the user to type another command.

8 The ABBREV function tests if `cmd` matches the character string `QUIT`. The number `1` indicates the minimum number of characters needed for `cmd` to match the character string. If `cmd` is Q, QU, QUI, or QUIT, it matches. This line also contains the Leave statement which ends the Do Forever loop.

9 This line is another example of a Do Forever loop. This loop ends with the Leave statement in line **11**.

10 These lines prompt the user for all the input variables needed.

11 This line tests for null input.

12 If the `salary` variable has a value, then the program has all the input.

13 When writing programs that use DB2 RXSQL statements, you must make sure that the variables named match the columns of the table. DB2 RXSQL retrieves the values for the host variables from REXX as needed.

All variables in the list have *variable-qualifiers* to ensure that DB2 RXSQL passes appropriate data type values to the database manager.

14 If an error occurs, the DSCERROR subroutine displays the error variables. The DSCERROR subroutine does not cause the program to end abnormally. The program resumes processing at the point where DSCERROR was called (see line **21**).

15 The program converts the user's input percentage value to a scaling factor.

16 If the update was successful (indicated by a return code greater than or equal to zero) the program displays the number of rows that were updated. The program uses the number returned in SQLERRD.3. Note that DB2 RXSQL may still return a warning even though the update was successful.

17 Note that the minimum abbreviation required for COMMIT is 4, not 1.

18 If none of the When clauses are true, then this program does not recognize the `cmd` variable as valid input. The program informs the user that it does recognize the input by issuing the message **Unknown command**.

19 The program explicitly commits any as yet uncommitted changes to the database manager and drops the connection to free up resources.

20 If this condition is true, a serious database error occurred and the program ends abnormally.

21 Return causes the program to resume processing at the line of the program that called DSCERROR.

Appendix C. Performance and Diagnosis

The first section of this appendix covers some of the common errors and performance concerns that users have. The second section is a list of hints and tips provided to help you diagnose a problem.

This is not intended to be a comprehensive discussion on either performance or diagnosis, because RXSQL is merely an interface between REXX and the database manager. There are many factors that can contribute to the performance of your programs, or to errors in your programs. This discussion just highlights some of the more common pitfalls.

If you have a hint or a tip that you feel should be in this appendix please send it in to IBM on the **Readers' Comment Form** at the back of the manual. IBM is committed to providing you with quality documentation, and welcomes any suggestions you have. **IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.**

Performance

Performance Improvement After Testing

You can turn off the setting and resetting of a group of RXSQL variables to improve an application's performance, after you finish testing it. Before setting the LASTING GLOBALV variable (GLOBALV) described here, make sure that the application contains no programmed dependencies that will become incorrect with the changed settings of the RXSQL variables described in this section.

Use a GLOBALV variable to improve an application’s performance. The GLOBALV variable, $$RXSQL32$$ in group SQL/DS, is retrieved at the first invocation of a RXSQL command in an EXEC. If the variable exists and the first character of its value is `Y` or `y`, several RXSQL variables will not be set or reset at each RXSQL or EXECSQL invocation. Changing the value of $$RXSQL32$$ after the first RXSQL request in an EXEC has no effect until control is returned to CMS.

The GLOBALV variable can be set with the following CMS command:

```
GLOBALV SELECT SQL/DS SETP $$RXSQL32$$ Y
```

When the above command is issued before the first RXSQL request in the EXEC, the affected RXSQL variables are:

- The `SQLERRMC`, `SQLWARN.`*n*, and `RXSQLREQUEST` variables are not set after any RXSQL request.
- The SQLCA variables `SQLCODE`, `SQLSTATE`, `SQLERRM`, `SQLERRP`, `SQLWARN`, and `SQLERRD.`*n* are not reset after each RXSQL request. If the request results in a database manager call, these variables are set with the corresponding values returned from the database manager.
- The `RXSQLMSG` variable is not reset after each RXSQL request. If a RXSQL interface error occurs, or if an EXECSQL interface error or warning occurs, this variable is set with a text message.

Setting $$RXSQL32$$ as a GLOBALV variable continues to affect the execution of RXSQL applications because the variable remains in the logon session after the application terminates, unless it is explicitly cleared. To stop the effect of this variable on an EXEC, clear the GLOBALV variable using an appropriate GLOBALV CMS command before the first RXSQL request in the EXEC.

Hints and Tips

There are a few basic rules to follow when coding your programs for optimum performance.

1. The performance of a program which uses RXSQL depends extensively on the design of the database and the efficiency of your SQL statements used to access and manipulate data in the database. This includes:

   - maintaining objects in the database so the optimizer is using current statistics
   - using static and dynamic SQL when appropriate to minimize locks put on catalog tables when preparing statements
   - ending LUWs by issuing COMMITs to free up any locks on objects as soon as possible.

   As an application designer you often have to weigh the pros and cons of many design strategies and many factors including system considerations, database considerations, and object design. Subjects that will help you make these decisions include: Locking, normalization, DASD balancing, indexes, and access authorities. Both the *DB2 Server for VM Application Programming* manual and the *DB2 Server for VM Database Administration* manual will provide you with more information on these and other topics.

2. Minimize the number of calls DB2 RXSQL must make to the database manager by minimizing the number of DB2 RXSQL requests executed in your programs.

3. Minimize the number of calls DB2 RXSQL must make to CMS by minimizing the number of *variable-names* within your RXSQL statements. Each *variable-name* that needs resolution by DB2 RXSQL requires a call to a CMS interface. Where possible, let REXX resolve the values of *variable-names* before DB2 RXSQL is passed the request.

4. Use *variable-qualifiers* so that RXSQL does not have to infer the data types.

5. Ensure that data types are consistent from one insert to the next. If DB2 RXSQL infers a different data type from the execution of one PUT statement to the next, then DB2 RXSQL traps the error returned from the database manager, closes the cursor and re-opens it. This will affect the performance of your application. For example, if you insert the value 12 on one PUT statement, DB2 RXSQL will infer that the data type is integer. If the value on the next PUT statement is 12.07, then DB2 RXSQL will infer that the data type is decimal.

6. Shorten the CMS command search path by using either `Address COMMAND` or the DB2 RXSQL subcommand environment. See Appendix E, “RXSQL Subcommand Environment” on page 157.

Diagnosis

Hints and Tips

In the following discussion, all SQLCODES and return codes mentioned are for DB2 Server for VM.

Common Error conditions:

- **SQLCODE error -301 (SQLSTATE 22507) on a** CALL, EXECUTE, OPEN, or PUT statement

  This DB2 Server for VM error condition means that the data type of a *rexx-host-variable* and the target data column are not compatible. It usually occurs when the database expects a character value, but DB2 RXSQL infers that the data type of the *rexx-host-variable* is numeric.

  To ensure that DB2 RXSQL passes a character value to the database, use *variable-qualifiers*, or enclose the value of the *rexx-host-variable* in single quotation marks ('). DB2 RXSQL removes the leading and trailing quotation marks and passes the value to the database as a character string.

  For example, typing:

  ```
     charvar = "'"charvar"'"
  ```

  before a CALL, EXECUTE, OPEN or PUT statement ensures that `charvar` will be interpreted as a character string.

  If you are using *variable-qualifiers*, do not surround your input values with quotes.

- **Unpredictable results on FETCH, CALL, EXECUTE, PUT, OPEN**

  When specifying an *output-rexx-host-variable-list* on FETCH or an *input-rexx-host-variable-list* on CALL, EXECUTE, OPEN or PUT, ensure that the *rexx-host-variable-list* is enclosed within quotation marks so that REXX does not resolve the variables before the statement is passed to DB2 RXSQL.

- **Unexpected ROLLBACK when an EXEC completes**

  Don't rely on an implicit COMMIT to occur when an EXEC finishes processing. There are VM timing dependencies when the IUCV communications connection to the database is terminated. Either a COMMIT or a ROLLBACK can occur. Make sure you use the COMMIT statement explicitly to save the changes you made to the database before the EXEC finishes processing.

- **RXSQL errors ELO0163E, ELO0168E, ELO0172E**

  These DB2 RXSQL error messages tell you that a host variable value is too long. If the variable value does not appear to be too long, check that it does not have any leading or trailing blanks as part of its value. DB2 RXSQL does not strip leading or trailing blanks from the value in host variables when retrieved from REXX. For example:

  ```
  package_id = '   USERAAA.MYPACK'
  ```

  In this example, the *authorization-name* appears to DB2 RXSQL as if it is 10 characters long, but *authorization-name* is limited to 8 characters in length.

- **SQL statements folding to uppercase**

Enclose each DB2 RXSQL statement in single quotes to prevent CMS from folding it to uppercase.

- **Truncation of long SQL statements on PREP, EXEC, or XPREP statements**

  If you use DB2 RXSQL in the XEDIT environment, as for example from XEDIT macros, XEDIT truncates all DB2 RXSQL commands that it finds (even if implicitly passed to CMS when IMPCMSCP is ON).

  When you use `Address COMMAND` or the DB2 RXSQL subcommand environment, the DB2 RXSQL statement bypasses XEDIT handling of the command.

  ```
  Address COMMAND 'RXSQL ... '
  ```

- **RXSQLOP EXEC or DB2 RXSQL OP statement fails to return the results of an operator command**.

  This is usually the result of

  1. OP being called in an EXEC that has not ended the current logical unit of work or
  2. DB2 RXSQL not receiving the operator command in uppercase.

Miscellaneous Tips

- **Consider the following programming techniques:**

  – When using EXECSQL invocation, specify `trace 'Off'`. Overriding the default of `normal` prevents REXX from displaying the EXECSQL statement whenever DB2 RXSQL returns a negative return code to signal an error condition.

  – PURGE all names previously used in PREP or DECLARE statements.

  – Put all DECLAREs at the beginning of your program.

  – If you use DB2 RXSQL in a program that invokes other application packages that use IUCV, make sure all application packages access IUCV through the CMSIUCV interface, instead of the CP IUCV interface. The DB2 Server for VM resource manager loses track of its path to the database if the CP IUCV interface is used.

  – When you are writing a non-terminating program, such as in a service machine, ensure that you include the RELEASE option on the COMMIT or ROLLBACK statement when the database is not needed between service requests. This allows the database to shut down, if necessary, without being locked up by the service machine program.

- **Stopping SQL processing of a database statement**

  Type:

  ```
  SQLHX
  ```

  A return code of -914 (user cancel) is issued from the database manager, and a ROLLBACK is issued for your current logical unit of work. You receive a return code of 8 from DB2 RXSQL, and an SQLCODE of -914 is returned to your EXEC.

- **Avoiding program checks**.

  – When a program that has been using DB2 RXSQL issues an EXEC SQLRMEND or an EXEC SQLINIT, the database manager becomes independent of DB2 RXSQL. This may result in a program check on the next DB2 RXSQL statement. Before calling either SQLRMEND or SQLINIT, issue one of:

```
COMMIT WORK RELEASE
ROLLBACK WORK RELEASE
```

The next DB2 RXSQL statement reconnects to the database manager and continues processing.

- **Recovering from a severed database connection**

  After a severe error has terminated your connection to the application server, any request other than a CONNECT request results in an SQLCODE of -900 and an SQLSTATE of 51018 in the SQLCA. You must issue a CONNECT request to reconnect to the application server when the error has been fixed.

  On DB2 Server for VM releases prior to Version 3 Release 4, after a severe error had occurred on a previous request, the DB2 Server for VM system would accept only the CONNECT request, and would abend on any other request. To avoid abends from the DB2 Server for VM system, any RXSQL release prior to Version 3 Release 4 did the following after it detected a severe error:

  – It submitted only CONNECT requests.
  – For non-CONNECT requests, it did not submit the request and instead returned the SQLCA of the original severe error.

  Starting with RXSQL Version 3 Release 4, RXSQL submits SQL requests to the application server without checking for severe errors. If a severe error had occurred previously, the RXSQL user will get an SQLCODE of -900 and an SQLSTATE of 51018, issued by SQL/DS system Version 3 Release 4 or later.

  Note: If RXSQL Version 3 Release 4 or later submits requests to an DB2 Server for VM system on a release prior to Version 3 Release 4, users might encounter abends after a severe error has occurred.

- **Supporting different versions of DB2 RXSQL**

  If you are switching between application servers that have different versions of the RXSQL package, you must:

  1. COMMIT or ROLLBACK RELEASE
  2. NUCXDROP RXSQL
  3. access the appropriate DB2 RXSQL production minidisk or SFS directory

  before you issue the CONNECT statement.

Appendix D. Support for CMS Work Units

The database manager and DB2 RXSQL will use CMS work units unless you specify WORKUNIT(NO) when you invoke the SQLINIT EXEC. CMS work unit support is available in VM/ESA.

CMS work unit support allows a virtual machine to have multiple independent paths into one or more DB2 Server for VM databases. You can have multiple active DB2 Server for VM logical units of work (LUW), each accessing the same or separate databases. The application can switch between work units while CMS, the DB2 Server for VM Resource Adapter, and DB2 RXSQL maintain all the necessary information.

For example, an application can copy data from one database to another without using a temporary file. Previously, you could not switch databases until your LUW ended. With CMS work unit support, you can copy data from one database to another using the following steps:

1. Establish a work unit, WU1.
2. Connect to database SQL1 and open a cursor on a SELECT statement.
3. Establish a work unit, WU2.
4. Connect to database SQL2 and open a cursor on an INSERT statement.
5. Make WU1 the current work unit.
6. Fetch into an array as many rows as feasible.
7. Make WU2 the current work unit.
8. Put all rows from array into SQL2.
9. Repeat steps 5-8 until all rows are read and put in the database.
10. Commit work on each database.

DB2 Server for VM will prevent an application from switching databases within the same LUW unless all work is committed or rolled back.

CMS work units also allow an EXEC to shield itself from other applications that may be accessing a database at the same time in the user's virtual machine.

For example, suppose a currently running EXEC with an active LUW in a database calls a second application EXEC that accesses the same or another database. In this situation, the called EXEC creates a new CMS work unit. Within the second CMS work unit, the called EXEC can access and modify the database, commit the changes, and return to the calling application without disturbing the LUW of the calling application. With two CMS work units, there are no DB2 RXSQL name or status conflicts between the applications.

To control the CMS work units, the application must use the VM/ESA CMS callable services library (CSL) interface for creating and managing work units. The REXX function, CSL, invokes the CMS callable services library. Figure 30 on page 154 shows how to Get, Push, and Pop CMS work units in REXX.

```
/* Acquire a new work unit ID */

Call CSL('DMSGETWU retcode reason workunit')
If retcode <> 0 then Say "Get WU rc="retcode "reason="reason

/* Note: the work unit is not active yet */
  ⋮
'RXSQL ...'

/* RXSQL creates environment for default work unit */
  ⋮
/* Make work unit the active work unit */

Call CSL('DMSPUSWU retcode reason workunit')
If retcode <> 0 then Say "Push WU rc="retcode "reason="reason
  ⋮
'RXSQL ...'

/* RXSQL creates new environment for new work unit */
  ⋮
Call CSL('DMSPOPWU retcode reason')
If retcode <> 0 then Say "Pop WU rc="retcode "reason="reason
  ⋮
'RXSQL ...'

/* RXSQL uses default work unit environment */
  ⋮
Exit
```

Figure 30. Example Using the REXX Function CSL

DB2 RXSQL uses the DMS CSL routines to determine the current work unit identifier. If the DMSQWUID routine fails, DB2 RXSQL uses the DMSERP routine to determine the current work unit identifier. DB2 RXSQL determines which DB2 RXSQL control blocks are used to handle the request based on the current work unit identifier.

When DB2 RXSQL finds a new work unit identifier, it creates a complete set of DB2 RXSQL control blocks for that work unit. Work units do not have to share data. Even settings such as isolation level (SQLISL), date and time format settings (SQLDATE and SQLTIME), and tracing levels (TRACE) are not shared, and you must set them independently for each work unit.

When DB2 RXSQL detects end-of-command processing, it frees storage and removes all of the environments created by DB2 RXSQL.

Setting CMS Work Unit Support Off

If CMS work unit support is not required in your program, and you suspect it is causing your CPU time to be high, you may turn this feature off. CMS work unit support is turned off by specifying WORKUNIT(NO) when invoking the SQLINIT EXEC. If you specify this, both the database manager and DB2 RXSQL will not use CMS work unit support until you execute another SQLINIT with the WORKUNIT(YES) parameter.

The SQLINIT EXEC and DB2 RXSQL use the LASTING GLOBALV file to check whether CMS work unit support is on or off. If this file is corrupted, DB2 RXSQL will issue an error message indicating that the SQLINIT EXEC must be invoked again.

Switching Work Unit Support

Do not switch work unit support by using the SQLINIT EXEC from within your program. Unexpected results will occur if you do. The value for the WORKUNIT parameter is retrieved from the LASTING GLOBALV file when your program is started. It may not be changed until control is returned to CMS at the completion of your program.

Changing Database Parameters When Using CMS Work Units

Do not use the SQLINIT EXEC to change parameters like Date or Time formats after the first RXSQL or EXECSQL statement has executed. DB2 RXSQL retrieves these parameters from the LASTING GLOBALV file at the first invocation of RXSQL or EXECSQL when the application starts, or at first invocation of RXSQL or EXECSQL after a COMMIT RELEASE or ROLLBACK RELEASE request.

For predictable results with these parameters, use the DB2 RXSQL requests SQLDATE and SQLTIME after starting a new work unit before the first request to fetch or insert data. For a cursor operation, the parameters must be changed before opening the cursor.

For SQLDATE, SQLISL, and SQLTIME, the changed environment is applicable only within the CMS Work Unit in which these commands were issued.

Appendix E. RXSQL Subcommand Environment

DB2 RXSQL commands may be executed in either the EXECSQL or RXSQL subcommand environment. The default environment may be altered (between various subcommand environments or the host environment) using the **ADDRESS** command. If `ADDRESS EXECSQL` or `ADDRESS RXSQL` has been invoked, you may code requests without the prefix 'EXECSQL' or 'RXSQL' respectively.

The command is passed directly to DB2 RXSQL without the search for EXECs or CP commands. Judicious use of either subcommand environment will improve the performance of a DB2 RXSQL application.

Do the following to invoke the DB2 RXSQL subcommand environment:

1. 'ADDRESS COMMAND' may be issued first to eliminate the CMS search for EXECs and CP commands.
2. DB2 RXSQL must first be invoked with a regular DB2 RXSQL command to load DB2 RXSQL as a nucleus extension and set up the subcommand environment. This command may request a harmless process such as setting tracing off for all modules or getting the names of all cursors used. Alternatively, the first command may do something relevant to your EXEC processing.
3. The 'ADDRESS EXECSQL' or 'ADDRESS RXSQL' command may be issued after the first DB2 RXSQL command. All non-REXX commands following this ADDRESS command are assumed to be EXECSQL or RXSQL subcommands and will be passed on to DB2 RXSQL.
4. It is assumed that the invocation of your DB2 RXSQL request is the same as the subcommand environment you are in unless you override it. For example, if your program issues 'ADDRESS EXECSQL',

   - any command that begins with the keyword `EXECSQL`, or does not have a keyword RXSQL, is processed using EXECSQL invocation rules.
   - any command that begins with the keyword `RXSQL` is processed using RXSQL invocation rules.

   If you wish to execute CMS or other non-RXSQL commands, another ADDRESS command must be issued. For more information on the use of ADDRESS, see the *VM/ESA REXX/VM Reference* manual.

Appendix F. RXSQL Runtime Environment

The RXSQL MODULE is a transient module that initiates NUCXLOAD to load the DB2 RXSQL nucleus extension from the RXSQL LOADLIB module. After it loads the nucleus extension, the transient module is not invoked again, because the CMS command search order looks for the nucleus extension first.

The nucleus extension is loaded in an area obtained by CMSSTOR so any CMS abnormal end of task (abend) such as HX causes an implicit NUCXDROP of the extension. The next invocation of DB2 RXSQL locates the transient module again and reloads the nucleus extension.

The RXSQL MODULE loads the nucleus extension with the SERVICE and ENDCMD options so that DB2 RXSQL is notified when CMS ends abnormally or returns control to the user after the typed command finishes processing successfully. When CMS notifies DB2 RXSQL of these actions, DB2 RXSQL stops communications with the database, purges all prepared SQL statements, and de-allocates or releases dynamic storage.

There is also an EXECSQL MODULE whose sole purpose is to pass control to the RXSQL MODULE or nucleus extension.

DB2 RXSQL communicates with the DB2 Server for VM database manager through the DB2 Server for VM Resource Adapter interface code. The Resource Adapter does end-of-command processing to end existing communications links with the database. When this happens, the Resource Adapter implicitly commits or rolls back changes to the database depending on how the user's command ends. If a CMS abnormal end-of-task (abend) occurs, changes are rolled back. Otherwise, an implicit COMMIT is done. Changes are committed even if the command has an error return code.

Note: End-of-command processing occurs only after a command is entered from the virtual console. It does not occur when a command is issued from an EXEC, a user program, or CMS SUBSET mode.

Tailoring the RXSQL Transient Module

RXSQL ASSEMBLE is the source file for the DB2 RXSQL transient module. The source file is distributed to allow local tailoring of the module. The RXSQL ASSEMBLE source file issues the following commands:

```
FILEDEF RXSQL DISK RXSQL LOADLIB * (NOCHANGE
NUCXLOAD RXSQL RXSQL RXSQL (ENDCMD SERVICE
CMSCALL RXSQL
```

The command `CMSCALL RXSQL` calls the loaded code.

If you plan to use the EXECSQL interface, you must not change the CALLTYPE option on the CMSCALL statement. DB2 RXSQL will not execute correctly if you do so.

To generate a new RXSQL MODULE type the following commands:

```
GLOBAL MACLIB DMSSP CMSLIB
HASM RXSQL
LOAD RXSQL (RLDSAVE
OR
GENMOD RXSQL (SYSTEM AMODE=31 RMODE=ANY    for VM/ESA
```

Appendix G. Single User Mode Environment

To run DB2 RXSQL in single-user mode, you require SQLEXEC as an interface between DB2 Server for VM and the EXEC. When you start DB2 Server for VM in single user mode, DB2 Server for VM can only invoke modules. SQLEXEC provides the module interface and puts the user parameter list into standard format for invoking an EXEC. An untokenized parameter list is passed; therefore, the EXEC must be written in REXX. Tokenized parameter lists are used in EXEC1 and EXEC2 EXECs.

SQLEXEC must be in USERLIB LOADLIB for the DB2 Server for VM database start-up EXECs to find it. To create USERLIB or to add SQLEXEC to an existing USERLIB, type:

```
LKED SQLEXEC (LIBE USERLIB
```

To run an EXEC that uses DB2 RXSQL in single user mode, type:

```
SQLSTART DBNAME(dbname) [DCSSID(dcssid)]
PARM(SYSMODE=S,[PARMID=parmid,]
PROGNAME=SQLEXEC/[EXEC] execname execparms)
```

For more information on the SQLSTART EXEC, see the *DB2 Server for VSE & VM Operation* manual.

Appendix H. Considerations and Restrictions When Using the DRDA Protocol

This appendix outlines what you must do to use RXSQL to access a non-DB2 Server for VM application server, and also outlines the restrictions that are specific to using RXSQL in the DRDA protocol. You should also read the appendix in the *DB2 Server for VSE & VM SQL Reference* manual that discusses the DB2 Server for VM restrictions when using the DRDA protocol.

Publications that contain additional information on Distributed Relational Database and Distributed Data Library can be found in "Bibliography" on page 171.

Using RXSQL in a Distributed Environment

With RXSQL, you can exploit the ability of the DB2 Server for VM application requester to communicate with other application servers that implement the DRDA architecture. The non-DB2 Server for VM application server will receive and process your RXSQL requests and return data to your program.

Prerequisites to Accessing Distributed Data

Software Requirements

DRDA support is available only in the following environments:

- VM/ESA Version 2 Release 1 or later, Licensed Program Number 5684-112

Setup Steps

There are a few setup steps which you must follow before you can use RXSQL to access distributed data.

1. ***Install RXSQL in Your Distributed Environment***

   Before you can use RXSQL to access a non-DB2 Server for VM application server, you should make sure that RXSQL has been installed in your distributed environment. See the *DB2 REXX SQL for VM/ESA Installation* manual for more details.

2. ***Setup COMDIRs***

   A communication directory must be set up specifying the name and address of the application server. The communication directory describes the information pathway between the application requester and the target application server. See the *DB2 Server for VM System Administration* manual for more details.

3. ***Issue SQLINIT With Appropriate Protocols***

   When invoking SQLINIT, the protocol parameters should be either DRDA or AUTO. When DRDA is specified, the Isolation Level must be CS. Any RXSQL SQLISL commands are ignored and the Isolation Level is fixed as CS. See the *DB2 Server for VM Database Administration* manual for more details.

Accessing Distributed Data

To access data in a non-DB2 Server for VM application server you must be connected to that server. You connect to the non-DB2 Server for VM application server when you issue the SQLINIT EXEC, or you can switch application servers while using RXSQL by issuing the CONNECT statement. For the syntax and recommended usage of this statement and other SQL statements used to access data in non-DB2 Server for VM application servers, see the *IBM SQL Reference, Version 2, Volume 1* manual. When you are using Extended Dynamic SQL to access a non-DB2 Server for VM application server which does not support Extended Dynamics, such as DB2, then each prepared statement is mapped to either a Static or a Dynamic statement.

Restrictions When Using RXSQL in the DRDA Protocol

DB2 RXSQL has some restrictions in addition to those listed in the appendix *"Restrictions with Distributed Relational Database Architecture (DRDA) Protocol"* of the *DB2 Server for VSE & VM SQL Reference* manual.

- Operator commands are not supported when connected to a non-DB2 Server for VM application server. An error condition is returned if you attempt to use them.
- When using Extended Dynamics while accessing a non-DB2 Server for VM application server which maps these statements to Dynamic or Static statements, RXSQL issues an Extended DESCRIBE statements whenever your program issues an OPEN statement. If the prepared insert or select statement is a Static Extended Dynamic statement (for example, it was prepared using a Basic Extended PREPARE or a Single Row Extended PREPARE), then DB2 RXSQL will return an error condition.

  In summary, only Dynamic SQL (regular or Extended Dynamic) is supported when accessing a non-DB2 Server for VM application server that does not support Extended Dynamic SQL.
- When using the Basic Extended PREPARE statement, if the SQL statement being prepared includes parameter markers, you must include the USING *attributes_variable* clause.
- When using an AS/400* application server and inserting a NULL value for a column, you must use both an *indicator_variable* and a *variable_qualifier* for that column. Further, the *variable_qualifier* must have the correct data type attribute for the column, even when it is a NULL value, to prevent an error condition being returned.

Enhancements When Using RXSQL in DRDA Protocol

You may specify a data type of zoned decimal in a *variable_qualifier* on data input or output when connected to an application server that supports this data type. Zoned decimal is not valid when connected to an DB2 Server for VM application server.

Appendix I. DB2 RXSQL Incompatibilities by Release

Version 3 Release 3 Modification 0

Authorization ID

RXSQL does not allow the hexadecimal value `X'00'` in the *authorization-name* or *password* in the CONNECT or CREATE PACKAGE statement.

STATE Command

When the STATE command is issued, the variable `RXSQLSTATE` is now set with two words indicating the type and state of the statement. In previous releases this variable was set with integers indicating the type and state of the statement.

NAMES Command

The NAMES command has been enhanced to support a new feature, *cursor-names* declared for dynamic prepared statements. When the NAMES command is issued, if a statement has a *prepare-name* and a *cursor-name* associated with it, `RXSQLNAMES` will include both names. However, the *cursor-name* will be enclosed in brackets and immediately follow the *prepare-name*. Previously, the NAMES command returned all names prepared or declared as a string of names.

Fetching DATETIME Values

The DATETIME defaults are retrieved from the LASTING GLOBALV file instead of the SYSTEM.SYSOPTIONS table in response to SQL/DS Version 3 Release 3 Modification 0 changes.

New and Changed Error Messages

Many of the error messages have been changed to support the extensions to RXSQL. Different RXSQL error codes and messages may be returned to your programs when run in this release of RXSQL.

Version 3 Release 2

DB2 RXSQL was modified to correctly retrieve DATE and TIME defaults in Version 3 Release 2; however no DB2 RXSQL statements may be executed between a COMMIT RELEASE (or ROLLBACK RELEASE) statement and the invocation of the SQLINIT EXEC.

For additional information on this restriction, see:

- " COMMIT" beginning on page 67
- " ROLLBACK" beginning on page 98
- " Appendix D, Support for CMS Work Units" beginning on page 153

Version 3 Release 1

Some of the REXX Variables that RXSQL sets are renamed as outlined in following table.

Variable Name in Version 3 Release 1	Variable Name in Program Offering
RXSQLSTATE	SQLSTATE
RXSQLNAMES	SQLNAMES
RXSQLSTMT	SQLSTMT

Glossary

This glossary includes terms and definitions from:

- The *American National Standard Dictionary for Information Systems* ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018. Definitions are identified by the symbol (A) after the definition.
- The *Information Technology Vocabulary* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

access. The manner in which files or data sets are referred to by a computer.

access module. See *package*.

application server. The database manager component that receives and processes database requests issued by the application requester.

authority. See *CONNECT authority, RESOURCE authority,* and *DBA authority*.

authorization. The right granted to a user to communicate with or make use of a computer system.

authorization ID. (1) A character string that designates a set of privileges. The authorization ID of a statement is used by the database manager for authorization checking and as an implicit qualifier for the names of tables, views, and indexes. (2) A user or a group of users. Contrast with *user ID*.

byte. A unit of storage consisting of eight adjacent binary digits operated on as a unit and constituting the smallest addressable unit in the system.

catalog. A set of tables that contain descriptions of objects such as tables and views, and other entities such as conversion tables and authorization IDs maintained by the database manager.

catalog table. A table maintained dynamically by the DB2 Server for VSE & VM database manager containing information about objects such as tables and views, and other entities such as conversion tables and authorization IDs defined to the database manager.

character. (1) A letter, digit, or other symbol in a data character set that is part of the data. (2) A data type in SQL.

CMS. See *conversational monitor system.*

column. The vertical component of a table. A column has a name and a particular data type (for example, character, decimal, or integer).

command. A request for system action. ISQL and DBS Utility commands can be used with the DB2 Server for VSE & VM database manager. Contrast with *statement*. DB2 RXSQL commands do not have SQL equivalents.

commit. (1) The operation that terminates a *unit of work* by releasing locks so that the database changes made by that unit of work can be perceived by other processes. (2) The process that allows data changes to be made permanent. When a commit occurs, other applications can reference the just-committed data.

CONNECT authority. The authority to use database functions, such as SQL statements, on a database.

constant. Data that has an unchanging, predefined value to be used in processing. Constants are classified as string constants or numeric constants. Synonymous with *literal*. Contrast with *variable*.

control program (CP). A component of a VM system that manages the resources of a single computer so multiple computing systems appear to exist. Each virtual machine is the functional equivalent of an IBM System/370™ or System/390® system.

conversational monitor system (CMS). A virtual machine operating system that provides general interactive time sharing, problem solving, and program development capabilities, and operates only under control of the VM *control program*.

CP. See *control program*.

DASD. See *direct access storage device*.

database machine. A virtual machine that has access to SQL/DS™ code and a database. The database machine controls access to the database.

database management system (DBMS). A software system that controls the logical and physical resources and facilities of a database.

database manager. A program product that processes SQL statements.

Database Services utility (DBS utility) Services utility (DBS utility). An application program supplied with the DB2 Server for VM and DB2 Server for VSE database managers to provide several utility functions,including the loading and unloading of data and packages,and the execution of SQL statements previously saved in a command file.

date. A three-part value that designates a day, month, and year.

DB2 for VM. Short form of DB2 Server for VM.

DBA authority. The authority to perform all SQL operations on all SQL tables. DBA authority includes both CONNECT authority and RESOURCE authority.

DBMS. See *database management system.*

DBS Utility (Database Services Utility). See *Database Services Utility*.

dbspace. A logical allocation of space in a *storage pool* contained in a database. Contains one or more tables and their associated indexes.

default. Pertaining to an attribute, value, or option that is assumed when none is explicitly specified.

direct access storage device (DASD). A device in which access to data is independent of the location of the data.

directory. A list of identifiers that map corresponding items of data. For example, an SQL/DS directory maps dbspaces to addresses on a physical device.

distributed relational database architecture (DRDA). Pertaining to an IBM connection protocol for the access and use of distributed relational data wherever it resides in an interconnected network of relational database products.

DRDA (distributed relational database architecture). See *distributed relational database architecture*.

EXEC. A CMS file with a file type of EXEC. It contains a series of commands or instructions that are executed when you enter the filename of the EXEC file.

file mode. A 2-character CMS file identifier field comprised of a file mode letter (A through Z) followed by the file mode number (0 through 6). The file mode letter indicates the minidisk or SFS directory on which the file resides. The file mode number indicates the access mode of the file.

file type. A 1- to 8-character alphanumeric field, comprised of A through Z, 0 through 9, and special characters $ # @ + - (hyphen) : (colon) _ (underscore), that is used as a descriptor or as a qualifier of the file name field in the CMS file identifier.

Interactive Structured Query Language (ISQL). A facility of the SQL/DS system that provides online data query and report writing support.

ISQL. See *Interactive Structured Query Language*.

K. Kilobyte

Kanji. A graphic character set consisting of symbols used in Japanese ideographic alphabets.

keyword. In programming languages, a predefined word that has a special meaning or function. For example, in the DB2 Server for VM and DB2 Server for VSE database managers, SELECT is the keyword used to retrieve data from a table.

kilobyte. 1 024 bytes

literal.. A string whose value is given by the content of the string itself. For example, the numeric literal 7 has the value 7. Synonym for *constant.*

load. To transfer data from one medium to another; for example, to transfer data from a sequential file to a database, or to transfer a program from storage to computer memory.

logical unit of work (LUW). A recoverable sequence of operations within an application process. At any time, an application process is a singleunit of work, but the life of an application process can involve many units of work as a result of commit or rollback operations.

LUW. See *logical unit of work (LUW).*

minidisk. Logical division of a physical direct access storage device.

multiple user mode. A mode of operating the database manager in which one or more users or application programs can access the database at the same time. Contrast with *single user mode.*

NLS. National language support.

owner. The authorization ID associated with an SQL object.

package. A control structure produced during program preparation that is used to execute SQL statements. module.

parameter. A variable that is given a constant value for a specified operation.

private dbspace. A logical space in a DB2 Server for VM database owned by one user.

public dbspace. A logical space in a DB2 Server for VM database that is accessible to many users.

rc. See *return code.*

RESOURCE authority. The authority to create tables in a public dbspace and to acquire a private dbspace.

return code. A string, typically a number passed in an implementation-dependent way, that conveys some information about the command that has been executed. Return codes usually indicate the success or failure of the command but can also be used to represent other information.

REXX. A procedural language that allows programs and algorithms to be written in a clear and structured way.

row. A horizontal component of a table. A row consists of a sequence of values, one for each column of the table.

RXSQL. DB2 REXX SQL for VM/ESA.

scroll. To move a display image vertically or horizontally to view data that is not otherwise visible in a display or window.

SFS directory (shared file system directory). See *shared file system directory.*

shared file system directory (SFS directory). A part of CMS that allows users to organize their files into groups known as directories, and to selectively share those files and directories with other users.

single user mode. A mode of operation in which the database manager and one application run in the same virtual machine. No other application programs or userscan access the database at the same time. Contrast with *multiple user mode.*

SQLCA. See *SQL communication area (SQLCA).*

SQLCODE. A code set by the database manager after an SQL statement is executed to indicate the success or failure of the SQL statement. The value returned in the SQLCODE is specific to the DB2 Server for VSE & VM system. Contrast with *SQLSTATE*. See also *SQL communication area*.

SQL communication area (SQLCA). A set of variables that provides an application program with information about the execution of its SQL statements.

SQL/DS. See *Structured Query Language/Data System*.

SQLSTATE. A code set by the database manager after an SQL statement is executed to indicate the success or failure of the SQL statement. Contrast with *SQLCODE*.

statement. An instruction in a program or procedure. A language construct that represents a step in a sequence of actions. Contrast with *command*. RXSQL statements have SQL equivalents.

storage pool. A specific set of available storage areas. These areas are used by the database administrator to control storage of the database. A storage pool contains one or more dbspaces.

Structured Query Language/Data System (SQL/DS). A standardized language for defining and manipulating data in a relational database. This was the previous product name for DB2 Server for VSE & VM.

table. A named data object consisting of a specified number of columns and any number of unordered rows.

time. A three-part value that designates a time of day in hours, minutes, and seconds.

unit of work. See *logical unit of work (LUW).*

user ID. A string of characters that uniquely identifies a user to a system. Contrast with *authorization ID*.

variable. A quantity that can assume any of a given set of values. Contrast with *constant*.

Virtual Machine (VM). A functional simulation of a computer and its associated devices. VM manages the computer's resources in such a way that all workstation users have their own virtual machine. All users can work at their virtual machines as though each is the only person using the real computer.

VM. See *Virtual Machine*.

VM/ESA environment. Virtual Machine/Enterprise Systems Architecture. The 370 feature addresses a maximum of 16 megabytes of virtual storage per virtual machine. The ESA Feature addresses a maximum of 2 gigabytes of virtual storage per virtual machine.

Bibliography

This bibliography lists publications that are referenced in this manual or that may be helpful.

DB2 Server for VM Publications

- *DB2 Server for VM Application Programming*, SC09-2661
- *DB2 Server for VM Database Administration*, SC09-2654
- *DB2 Server for VSE & VM Database Services Utility*, SC09-2663
- *DB2 Server for VM Diagnosis Guide and Reference*, LC09-2672
- *DB2 Server for VSE & VM Overview*, GC09-2806
- *DB2 Server for VSE & VM Interactive SQL Guide and Reference*, SC09-2674
- *DB2 Server for VM Master Index and Glossary*, SC09-2666
- *DB2 Server for VM Messages and Codes*, GC09-2664
- *DB2 Server for VSE & VM Operation*, SC09-2668
- *DB2 Server for VSE & VM Quick Reference*, SC09-2670
- *DB2 Server for VM System Administration*, SC09-2657

DB2 Data Spaces Support Publications

- *DB2 Server Data Spaces Support for VM/ESA*, SC09-2675

Related Publications

- *DB2 Server for VSE & VM Data Restore*, SC09-2677
- *DRDA: Every Manager' s Guide*, GC26-3195
- *IBM SQL Reference, Version 2, Volume 1*, SC26-8416
- *IBM SQL Reference*, SC26-8415

VM/ESA Publications

- *VM/ESA: General Information*, GC24-5745
- *VM/ESA: VMSES/E Introduction and Reference*, GC24-5837
- *VM/ESA: Installation Guide*, GC24-5836
- *VM/ESA: Service Guide*, GC24-5838
- *VM/ESA: Planning and Administration*, SC24-5750
- *VM/ESA: CMS File Pool Planning, Administration, and Operation*, SC24-5751
- *VM/ESA: REXX/EXEC Migration Tool for VM/ESA*, GC24-5752
- *VM/ESA: Conversion Guide and Notebook*, GC24-5839
- *VM/ESA: Running Guest Operating Systems*, SC24-5755
- *VM/ESA: Connectivity Planning, Administration, and Operation*, SC24-5756
- *VM/ESA: Group Control System*, SC24-5757
- *VM/ESA: System Operation*, SC24-5758
- *VM/ESA: Virtual Machine Operation*, SC24-5759
- *VM/ESA: CP Programming Services*, SC24-5760
- *VM/ESA: CMS Application Development Guide*, SC24-5761
- *VM/ESA: CMS Application Development Reference*, SC24-5762
- *VM/ESA: CMS Application Development Guide for Assembler*, SC24-5763
- *VM/ESA: CMS Application Development Reference for Assembler*, SC24-5764
- *VM/ESA: CMS Application Multitasking*, SC24-5766
- *VM/ESA: CP Command and Utility Reference*, SC24-5773
- *VM/ESA: CMS Primer*, SC24-5458
- *VM/ESA: CMS User's Guide*, SC24-5775
- *VM/ESA: CMS Command Reference*, SC24-5776
- *VM/ESA: CMS Pipelines User's Guide*, SC24-5777
- *VM/ESA: CMS Pipelines Reference*, SC24-5778
- *VM/ESA: XEDIT User's Guide*, SC24-5779
- *VM/ESA: XEDIT Command and Macro Reference*, SC24-5780
- *VM/ESA: Master Index and Glossary*, SC09-2398
- *VM/ESA: Quick Reference*, SX24-5290
- *VM/ESA: Performance*, SC24-5782
- *VM/ESA: Dump Viewing Facility*, GC24-5853
- *VM/ESA: System Messages and Codes*, GC24-5841
- *VM/ESA: Diagnosis Guide*, GC24-5854
- *VM/ESA: CP Diagnosis Reference*, SC24-5855
- *VM/ESA: CP Diagnosis Reference Summary*, SX24-5292
- *VM/ESA: CMS Diagnosis Reference*, SC24-5857

- *VM/ESA: CMS Data Areas and Control Blocks*, SC24-5858
- *VM/ESA: CP Data Areas and Control Blocks*, SC24-5856
- *IBM VM/ESA: CP Exit Customization*, SC24-5672
- *VM/ESA REXX/VM User's Guide*, SC24-5465
- *VM/ESA REXX/VM Reference*, SC24-5770

C for VM/ESA Publications

- *IBM C for VM/ESA Diagnosis Guide*, SC09-2149
- *IBM C for VM/ESA Language Reference*, SC09-2153
- *IBM C for VM/ESA Compiler and Run-Time Migration Guide*, SC09-2147
- *IBM C for VM/ESA Programming Guide*, SC09-2151
- *IBM C for VM/ESA User's Guide*, SC09-2152

Other Distributed Data Publications

- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526
- *IBM Distributed Data Management (DDM) Architecture, Implementation Programmer's Guide*, SC21-9529
- *VM/Directory Maintenance Licensed Program Operation and User Guide Release 4*, SC23-0437
- *IBM Distributed Relational Database Architecture Reference*, SC26-4651
- *IBM Systems Network Architecture, Format and Protocol*
- *SNA LU 6.2 Reference: Peer Protocols*
- *Reference Manual: Architecture Logic for LU Type 6.2*
- *IBM Systems Network Architecture, Logical Unit 6.2 Reference: Peer Protocols*
- *Distributed Data Management (DDM) List of Terms*
- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526
- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526
- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526

CCSID Publications

- *Character Data Representation Architecture, Executive Overview*, GC09-2207
- *Character Data Representation Architecture Reference and Registry*, SC09-2190

DB2 Server RXSQL Publications

- *DB2 REXX SQL for VM/ESA Installation*, GC09-2660
- *DB2 REXX SQL for VM/ESA Reference*, SC09-2676

C/370 Publications

- *IBM C/370 Installation and Customization Guide*, GC09-1387
- *IBM C/370 Programming Guide*, SC09-1384

Communication Server for OS/2 Publications

- *Up and Running!*,GC31-8189
- *Network Administration and Subsystem Management Guide* SC31-8181
- *Command Reference*, SC31-8183
- *Message Reference*, SC31-8185
- *Problem Determination Guide*, SC31-8186

Distributed Database Connection Services (DDCS) Publications

- *DDCS User's Guide for Common Servers*, S20H-4793
- *DDCS for OS/2 Installation and Configuration Guide* S20H-4795

VTAM Publications

- *VTAM Messages and Codes*, SC31-6493
- *VTAM Network Implementation Guide*, SC31-6494
- *VTAM Operation*, SC31-6495
- *VTAM Programming*, SC31-6496
- *VTAM Programming for LU 6.2*, SC31-6497
- *VTAM Resource Definition Reference*, SC31-6498
- *VTAM Resource Definition Samples*, SC31-6499

CSP/AD and CSP/AE Publications

- *Developing Applications*, SH20-6435
- *CSP/AD and CSP/AE Installation Planning Guide*, GH20-6764
- *Administering CSP/AD and CSP/AE on VM*, SH20-6766
- *Administering CSP/AD and CSP/AE on VSE*, SH20-6767
- *CSP/AD and CSP/AE Planning*, SH20-6770
- *Cross System Product General Information*, GH23-0500

Query Management Facility (QMF) Publications

- *QMF General Information*, GC26-4713
- *QMF VSE/ESA Setup and Usage Guide*, GG24-4196
- *Managing QMF for VSE/ESA*, SC26-3252
- *Installing QMF on VSE/ESA*, SC26-3254
- *QMF Learner's Guide*, SC26-4714
- *QMF Advanced User's Guide*, SC26-4715
- *QMF Reference*, SC26-4716
- *Installing QMF on VM*, SC26-4718
- *QMF Application Development Guide*, SC26-4722
- *QMF Messages and Codes*, SC26-4834
- *Using QMF*, SC26-8078
- *Managing QMF for VM/ESA*, SC26-8219

DL/I DOS/VS Publications

- *DL/I DOS/VS Application Programming*, SH24-5009

COBOL Publications

- *VS COBOL II Migration Guide for VSE*, GC26-3150
- *VS COBOL II Migration Guide for MVS and CMS*, GC26-3151
- *VS COBOL II General Information*, GC26-4042
- *VS COBOL II Language Reference*, GC26-4047
- *VS COBOL II Application Programming Guide*, SC26-4045
- *VS COBOL II Application Programming Debugging*, SC26-4049
- *VS COBOL II Installation and Customization for CMS* SC26-4213
- *VS COBOL II Installation and Customization for VSE* SC26-4696
- *VS COBOL II Application Programming Guide for VSE* SC26-4697

Data Facility Storage Management Subsystem/VM (DFSMS/VM) Publications

- *DFSMS/VM User's Guide*, SC26-4705

Systems Network Architecture (SNA) Publications

- *SNA Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084
- *SNA Format and Protocol Reference: Architecture Logic for LU Type 6.2*, SC30-3269
- *SNA LU 6.2 Reference: Peer Protocols*, SC31-6808
- *SNA Synch Point Services Architecture Reference* SC31-8134

Miscellaneous Publications

- *IBM 3990 Storage Control Planning, Installation, and Storage Administration Guide*, GA32-0100
- *Dictionary of Computing*, ZC20-1699
- *APL2 Programming: Using Structured Query Language*, SH21-1056
- *ESA/390 Principles of Operation*, SA22-7201

Related Feature Publications

- *Control Center Installation and Operations Guide for VM*, GC09-2678
- *IBM Replication Guide and Reference*, S95H-0999

Index

Communicating Your Comments to IBM

DB2® REXX SQL for VM/ESA®
Reference
Version 6 Release 1

Publication No. SC09-2676-00

If there is something you like—or dislike—about this book, please let us know. You can use one of the methods listed below to send your comments to IBM. If you want a reply, include your name, address, and telephone number. If you are communicating electronically, include the book title, publication number, page number, or topic you are commenting on.

The comments you send should only pertain to the information in this book and its presentation. To request additional publications or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give it to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - United States and Canada: 416-448-6161
  - Other countries: (+1)-416-448-6161
- If you prefer to send comments electronically, use the network ID listed below. Be sure to include your entire network address if you wish a reply.
  - Internet: torrcf@ca.ibm.com
  - IBMLink: toribm(torrcf)
  - IBM/PROFS: torolab4(torrcf)
  - IBMMAIL: ibmmail(caibmwt9)

Readers' Comments — We'd Like to Hear from You

DB2® REXX SQL for VM/ESA®
Reference
Version 6 Release 1

Publication No. SC09-2676-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

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

Name	Address
Company or Organization	
Phone No.	

Readers' Comments — We'd Like to Hear from You
SC09-2676-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 EGLINTON AVENUE EAST
NORTH YORK ONTARIO CANADA M3C 1H7

Fold and Tape | **Please do not staple** | Fold and Tape

SC09-2676-00

Cut or Fold
Along Line

File Number: S370/4300-50
Program Number: 5648-A70-02

SC09-2676-00

Spine information:

DB2® REXX SQL for VM/ESA® Reference *Version 6 Release 1*