

Cincom

## **SUPRA SERVER PDM**

RDM COBOL Programming Guide  
(OS/390 & VSE)

P26-8330-62



---

# SUPRA<sup>®</sup> Server PDM RDM COBOL Programming Guide (OS/390 & VSE)

## Publication Number P26-8330-62

© 1987, 1991, 1993, 1998, 2000, 2002 Cincom Systems, Inc.  
All rights reserved

This document contains unpublished, confidential, and proprietary information of Cincom. No disclosure or use of any portion of the contents of these materials may be made without the express written consent of Cincom.

The following are trademarks, registered trademarks, or service marks of Cincom Systems, Inc.:

AD/Advantage <sup>®</sup>	iD CinDoc <sup>™</sup>	MANTIS <sup>®</sup>
C+A-RE <sup>™</sup>	iD CinDoc Web <sup>™</sup>	Socrates <sup>®</sup>
CINCOM <sup>®</sup>	iD Consulting <sup>™</sup>	Socrates <sup>®</sup> XML
Cincom Encompass <sup>®</sup>	iD Correspondence <sup>™</sup>	SPECTRA <sup>™</sup>
Cincom Smalltalk <sup>™</sup>	iD Correspondence Express <sup>™</sup>	SUPRA <sup>®</sup>
Cincom SupportWeb <sup>®</sup>	iD Environment <sup>™</sup>	SUPRA <sup>®</sup> Server
CINCOM SYSTEMS <sup>®</sup>	iD Solutions <sup>™</sup>	Visual Smalltalk <sup>®</sup>
 gOOj <sup>™</sup>	intelligent Document Solutions <sup>™</sup>	VisualWorks <sup>®</sup>
	Intermax <sup>™</sup>	

UniSQL<sup>™</sup> is a trademark of UniSQL, Inc.  
ObjectStudio<sup>®</sup> is a registered trademark of CinMark Systems, Inc.

All other trademarks are trademarks or registered trademarks of their respective companies.

Cincom Systems, Inc.  
55 Merchant Street  
Cincinnati, Ohio 45246-3732  
U. S. A.

PHONE: (513) 612-2300  
FAX: (513) 612-2000  
WORLD WIDE WEB: <http://www.cincom.com>

---

### Attention:

Some Cincom products, programs, or services referred to in this publication may not be available in all countries in which Cincom does business. Additionally, some Cincom products, programs, or services may not be available for all operating systems or all product releases. Contact your Cincom representative to be certain the items are available to you.

---

---

## Release information for this manual

The *SUPRA Server PDM RDM COBOL Programming Guide (OS/390 & VSE)*, P26-8330-62, is dated January 15, 2002. This document supports Release 2.7 of SUPRA Server PDM in IBM mainframe environments.

### We welcome your comments

We encourage critiques concerning the technical content and organization of this manual. Please take the [survey](#) provided with the online documentation at your convenience.

*Cincom Technical Support for SUPRA Server PDM*

FAX: (513) 612-2000  
Attn: SUPRA Server Support

E-mail: [helpna@cincom.com](mailto:helpna@cincom.com)

Phone: 1-800-727-3525

Mail: Cincom Systems, Inc.  
Attn: SUPRA Server Support  
55 Merchant Street  
Cincinnati, OH 45246-3732  
U. S. A.



# Contents

<b>About this book</b>	<b>ix</b>
Using this document.....	ix
Document organization .....	ix
Revisions to this manual .....	x
Conventions .....	xi
SUPRA Server documentation series .....	xiv
<b>Overview of COBOL application programming with RDM</b>	<b>17</b>
Application programming overview .....	18
Understanding RDM views.....	20
Creating user views.....	22
Understanding columns and keys .....	23
Introduction to the Relational Data Manipulation Language (RDML) .....	25
Introduction to the DBAID utility subset.....	26

<b>Using the DBAID utility subset</b>	<b>27</b>
DBAID commands list.....	28
DBAID formatting guidelines.....	31
DBAID commands .....	32
= command.....	32
BYE command.....	33
BY-LEVEL command.....	34
CAUTIOUS command.....	36
COLUMN-TEXT command.....	37
COMMIT command.....	39
DELETE command.....	40
ERASE command.....	42
FIELD-DEFN command.....	43
FORGET command.....	45
GET command .....	46
GO command .....	50
INSERT command.....	53
KEEP command .....	56
LINESIZE command.....	57
MARK command .....	58
MARKS command.....	59
OPEN command.....	60
PAGESIZE command.....	62
RELEASE command .....	63
RESET command.....	64
SIGN-OFF command.....	65
SIGN-ON command .....	66
SURE command.....	67
UPDATE command .....	68
USER-LIST command.....	70
USERS command .....	71
VIEW-DEFN command .....	72
VIEWS command .....	73
VIEWS-FOR-USER command.....	74

<b>Coding RDM COBOL application programs</b>	<b>75</b>
Using the programmer's report .....	76
Coding the DATA DIVISION .....	77
Specifying views and user views .....	78
Specifying TIS-CONTROL .....	79
RDM status indicators .....	80
Coding the PROCEDURE DIVISION .....	84
Signing on/off .....	85
Maintaining storage .....	86
Using the GET statement to retrieve rows .....	86
Accessing multiple views .....	89
Using the MARK statement to save row position .....	90
Using explicit and automatic record holding .....	91
Handling error conditions .....	92
Modifying rows .....	93
Using the INSERT statement .....	94
Using the COMMIT/RESET statements .....	95
Handling errors requiring a recompile .....	96
<b>RDM COBOL application program statements</b>	<b>97</b>
DATA DIVISION statements .....	98
INCLUDE view-data .....	98
INCLUDE TIS-CONTROL .....	101
PROCEDURE DIVISION statements .....	103
COMMIT .....	104
DELETE .....	105
FORGET .....	107
GET .....	109
INSERT .....	114
MARK .....	116
RELEASE .....	118
RESET .....	119
SIGN-OFF .....	120
SIGN-ON .....	121
UPDATE .....	122
<b>Compiling and linking an RDM COBOL application program</b>	<b>123</b>
Executing the RDML precompiler .....	124
Linking a compiled program .....	124

<b>OS/390 and VSE samples and procedures</b>	<b>125</b>
OS/390 samples and procedures .....	125
VSE samples .....	126
<b>Sample RDM COBOL application program</b>	<b>127</b>
<b>Index</b>	<b>133</b>

---

# About this book

---

---

## Using this document

This manual is for COBOL application programmers who wish to write RDM COBOL applications for SUPRA PDM.

### Document organization

The information in this manual is organized as follows:

#### **Chapter 1—Overview of COBOL application programming with RDM**

Presents an overview of the requirements and considerations that a COBOL programmer needs to be aware of before writing an RDM COBOL program.

#### **Chapter 2—Using the DBAID utility subset**

Describes DBAID utility commands.

#### **Chapter 3—Coding RDM COBOL application programs**

Presents requirements and guidelines for coding RDM COBOL application programs.

#### **Chapter 4—RDM COBOL application program statements**

Contains format descriptions and usage considerations for the two groups of RDM COBOL program statements.

#### **Chapter 5—Compiling and linking an RDM COBOL application program**

Presents information on the RDML Precompiler, including instructions for executing the precompiler and linking considerations for each operating system.

#### **Appendix A—OS/390 and VSE samples and procedures**

Lists samples and procedures for running certain tasks in OS/390 and VSE environments.

#### **Appendix B—Sample RDM COBOL application program**

Displays a sample COBOL application to execute a set of RDML statements.

#### **Index**

## Revisions to this manual

The following changes have been made for this release:

- ◆ A note was added to the introductory text in “Coding RDM COBOL application programs” on page 75 to indicate Cincom’s recommendation for an alternative to use of the REMARKS verb.
- ◆ References to CMS have been removed.
- ◆ References to MVS have been changed to OS/390.

## Conventions

The following table describes the conventions used in this document series:

Convention	Description	Example
Constant width type	Represents screen images and segments of code.	<pre>PUT 'customer.dat' GET 'miller\customer.dat' PUT '\DEV\RMT0'</pre>
Slashed b ( <i>b</i> )	Indicates a space (blank).  The example indicates that four spaces appear between the keywords.	BEGN <b>bbbb</b> SERIAL
Brackets [ ]	Indicate optional selection of parameters. (Do not attempt to enter brackets or to stack parameters.) Brackets indicate one of the following situations:	
	A single item enclosed by brackets indicates that the item is optional and can be omitted.	[WHERE <i>search-condition</i> ]
	The example indicates that you can optionally enter a WHERE clause.	
	Stacked items enclosed by brackets represent optional alternatives, one of which can be selected.	<b>[<u>(WAIT)</u> (NOWAIT)]</b>
	The example indicates that you can optionally enter either WAIT or NOWAIT. (WAIT is underlined to signify that it is the default.)	

Convention	Description	Example
Braces { }	<p>Indicate selection of parameters. (Do not attempt to enter braces or to stack parameters.) Braces surrounding stacked items represent alternatives, one of which you must select.</p> <p>The example indicates that you must enter ON or OFF when using the MONITOR statement.</p>	<pre>MONITOR {ON         }         {OFF}</pre>
<p><u>Underlining</u> (In syntax)</p>	<p>Indicates the default value supplied when you omit a parameter.</p> <p>The example indicates that if you do not choose a parameter, the system defaults to WAIT.</p> <p>Underlining also indicates an allowable abbreviation or the shortest truncation allowed.</p> <p>The example indicates that you can enter either STAT or STATISTICS.</p>	<pre>[<u>WAIT</u>] [<u>NOWAIT</u>]</pre> <hr/> <pre><u>STATISTICS</u></pre>
Ellipsis points...	<p>Indicate that the preceding item can be repeated.</p> <p>The example indicates that you can enter multiple host variables and associated indicator variables.</p>	<pre>INTO :host-variable [:ind- variable],...</pre>

Convention	Description	Example
UPPERCASE lowercase	In most operating environments, keywords are not case-sensitive, and they are represented in uppercase. You can enter them in either uppercase or lowercase.	COPY MY_DATA.SEQ HOLD_DATA.SEQ
<i>Italics</i>	Indicate variables you replace with a value, a column name, a file name, and so on.  The example indicates that you must substitute the name of a table.	FROM <i>table-name</i>
Punctuation marks	Indicate required syntax that you must code exactly as presented.  ( ) parentheses . period , comma : colon ' ' single quotation marks	<i>(user-id, password, db-name)</i> INFILE 'Cust.Memo' CONTROL LEN4
SMALL CAPS	Represent a required keystroke. Multiple keystrokes are hyphenated.	ALT-TAB
<div style="border: 1px solid black; padding: 2px; display: inline-block;">OS/390</div> <div style="border: 1px solid black; padding: 2px; display: inline-block;">VSE</div>	Information specific to a certain operating system is flagged by a symbol in a shadowed box ( <div style="border: 1px solid black; padding: 2px; display: inline-block;">OS/390</div> ) indicating which operating system is being discussed. Skip any information that does not pertain to your environment.	<div style="border: 1px solid black; padding: 2px; display: inline-block;">OS/390</div> See the SUPRA Server procedure library member TIS\$RDM for a list of RDM procedures.  <div style="border: 1px solid black; padding: 2px; display: inline-block;">VSE</div> See the SUPRA Server RDM sublibrary member TXJ\$INDX for a list of JCL.

## SUPRA Server documentation series

SUPRA<sup>®</sup> Server is the advanced relational database management system for high-volume, update-oriented production processing. A number of tools are available with SUPRA Server including DBA Functions, DBAID, precompilers, SPECTRA, and MANTIS. The following list shows the manuals and tools used to fulfill the data management and retrieval requirements for various tasks. Some of these tools are optional. Therefore, you may not have all the manuals listed. For a brief synopsis of each manual, refer to the *SUPRA Server PDM Digest (OS/390 & VSE)*, P26-9062.

### Overview

- ◆ *SUPRA Server PDM Digest (OS/390 & VSE)*, P26-9062

### Getting started

- ◆ *SUPRA Server PDM Migration Guide (OS/390 & VSE)*, P26-0550\*
- ◆ *SUPRA Server PDM CICS Connector Systems Programming Guide (OS/390 & VSE)*, P26-7452

### General use

- ◆ *SUPRA Server PDM Glossary*, P26-0675
- ◆ *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126

## Database administration tasks

- ◆ *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250
- ◆ *SUPRA Server PDM Directory Online User's Guide (OS/390 & VSE)*, P26-1260
- ◆ *SUPRA Server PDM Directory Batch User's Guide (OS/390 & VSE)*, P26-1261
- ◆ *SUPRA Server PDM DBA Utilities User's Guide (OS/390 & VSE)*, P26-6260
- ◆ *SUPRA Server PDM Logging and Recovery (OS/390 & VSE)*, P26-2223
- ◆ *SUPRA Server PDM Tuning Guide (OS/390 & VSE)*, P26-0225
- ◆ *SUPRA Server PDM RDM Administration Guide (OS/390 & VSE)*, P26-8220
- ◆ *SUPRA Server PDM RDM PDM Support Supplement (OS/390 & VSE)*, P26-8221
- ◆ *SUPRA Server PDM RDM VSAM Support Supplement (OS/390 & VSE)*, P26-8222
- ◆ *SUPRA Server PDM Migration Guide (OS/390 & VSE)*, P26-0550\*
- ◆ *SUPRA Server PDM Windows Client Support User's Guide*, P26-7500\*
- ◆ *SPECTRA Administrator's Guide*, P26-9220

## Application programming tasks

- ◆ *SUPRA Server PDM DML Programming Guide (OS/390 & VSE)*, P26-4340
- ◆ *SUPRA Server PDM RDM COBOL Programming Guide (OS/390 & VSE)*, P26-8330
- ◆ *SUPRA Server PDM RDM PL/1 Programming Guide (OS/390 & VSE)*, P26-8331
- ◆ *SUPRA Server PDM Migration Guide (OS/390 & VSE)*, P26-0550\*
- ◆ *SUPRA Server PDM Windows Client Support User's Guide*, P26-7500\*

## Report tasks

- ◆ *SPECTRA User's Guide*, P26-9561



---

Manuals marked with an asterisk (\*) are listed more than once because you use them for multiple tasks.

---



---

Educational material is available from your regional Cincom education department.

---

# 1

## Overview of COBOL application programming with RDM

SUPRA is an advanced relational database management system for high-volume update-oriented processing. The Relational Data Manager (RDM) is the SUPRA component which accepts and processes requests from end users and application programmers. RDM retrieves the data needed for an application program while providing database security and integrity and insulating the application program from changes to the database. RDM allows the application programmer to write COBOL programs without knowing about the physical structure of the database.

RDM supports the OS/390 and VSE (DOS/VSE and VSE Advanced Functions) operating systems. Within these operating systems, it supports the following programming languages:

- ◆ COBOL
- ◆ COBOL II
- ◆ CICS COBOL (Command Level)
- ◆ MANTIS
- ◆ PL/I



The catalogued procedures for running DBAID, the RDML Precompiler and Directory reports in your operating system are supplied on your tailored installation tape.

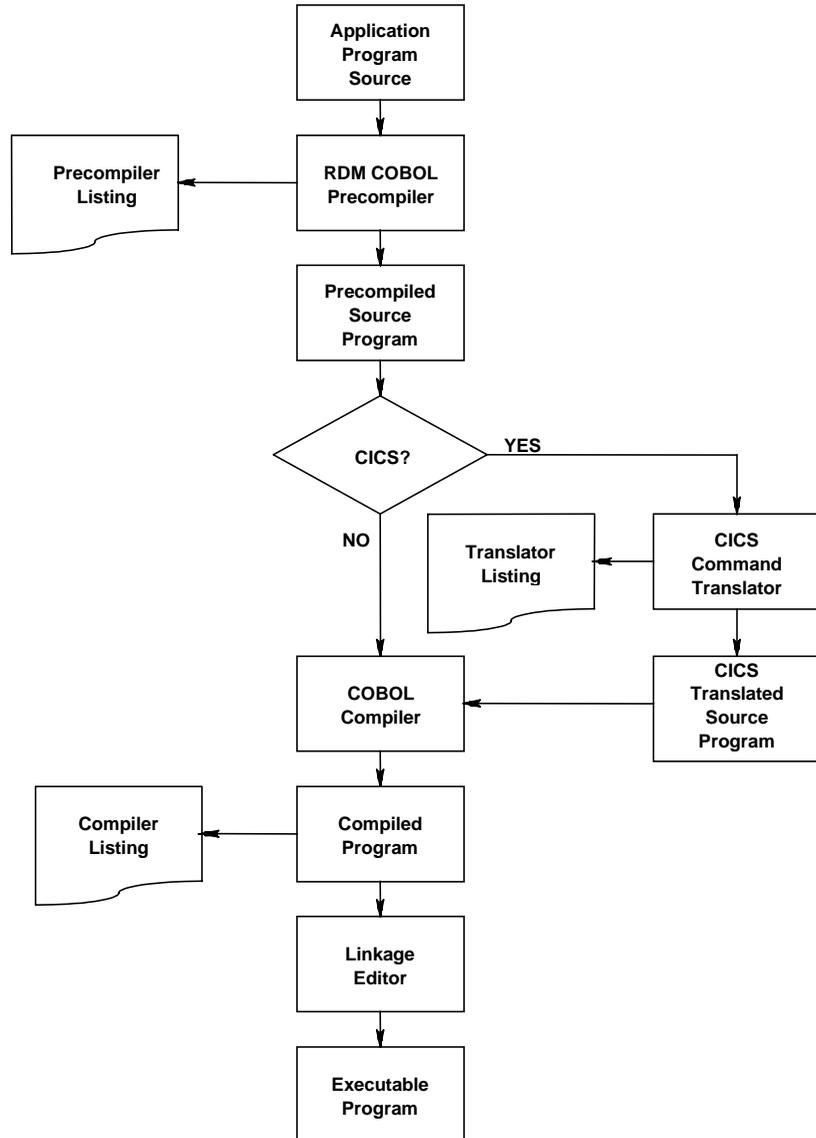
This chapter presents an overview of the requirements and considerations that a COBOL programmer needs to be aware of before writing an RDM COBOL program. This chapter addresses:

- ◆ The steps for writing an application
- ◆ Understanding RDM views
- ◆ Introduction to the Relational Data Manipulation Language (RDML)
- ◆ Introduction to using the DBAID utility subset

## Application programming overview

The figure on the following page illustrates the sequence for COBOL application programming with SUPRA RDM. The process has the following steps:

1. Optionally testing your RDM views using DBAID before testing with applications, following the guidelines given in “[Using the DBAID utility subset](#)” on page 27.
2. Coding your application program. This involves:
  - Addressing the RDM COBOL programming issues covered in this chapter and the coding requirements and guidelines in “[Coding RDM COBOL application programs](#)” on page 75.
  - Embedding RDML statements in your program, following the instructions given in “[RDM COBOL application program statements](#)” on page 97.
3. Running the RDML Precompiler and linking the compiled program following the guidelines in “[Compiling and linking an RDM COBOL application program](#)” on page 123. The RDML Precompiler converts RDML statements into standard COBOL source code. Standard compilers then convert the COBOL source code into object code. When the program executes, the Directory uses the physical data descriptions, and RDM uses the logical data descriptions, to access the database and present the data in the view requested by the application program.



## Understanding RDM views

RDM provides a view of physical field values from one or many files in a flat, two-dimensional format of rows. A view is set of one or more rows describing physical field values within the database. A view is presented in the form of a table which consists of rows and columns. Views are defined by the DBA in the Directory by describing the required fields and by providing access to the file(s) containing these fields. When you write COBOL application programs with RDM, you do so using the information contained in the appropriate predefined views assigned by your DBA.

The DBA maintains complete control over the definition and generation of views of the data, determines data needs, and assists in determining the best method of structuring views and relationships. It is important, however, for the application programmer to understand views in order to write effective application programs.

The following terms are integral to a discussion of views:

- ◆ **Row.** A set of one or more related data items stored in computer memory.
- ◆ **Column.** In a row, a specified area used for a particular category of data.
- ◆ **Value.** A quantity assigned to a constant, a variable, a parameter or a symbol.
- ◆ **Key.** One or more data items, the contents of which identify the type or location of a row, or the ordering of data.
- ◆ **User View.** A subset of a view which may consist of all or part of the view.

In a view, columns are mapped between the externally constructed row and the physical database. During physical navigation of the database, RDM collects certain occurrences of the physical records based on the row definition. RDM then selects the appropriate physical fields and maps them to the constructed row.

The following figure illustrates a view.

VIEW—A table of data		
CUSTOMER Number	CUSTOMER Name	CUSTOMER Class
E40000	DOUG REED	Q1
F80081	TOM LANGDON	B4
H22233	ATHENS INC	J1

↑
↑
↑

COLUMN
COLUMN
COLUMN

← ROW  
← ROW  
← ROW

Picture a set of views stored sequentially as a flat file. You can retrieve records from the file according to relative positioning within the file or by selecting key values. The operations available to RDM: GET, INSERT, UPDATE, DELETE, are those needed to manipulate the records on an occurrence-by-occurrence basis.

Although a view resembles a flat file, there are two important differences:

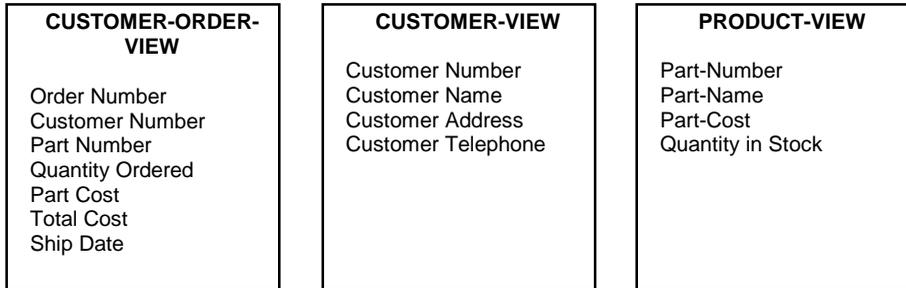
- ◆ The ordering of rows within the file is not always controlled by your maintenance operations.
- ◆ Fields (columns) can have null values.

The DBA sets up on the Directory views available to you. Refer to the Relational Data Manager COBOL Programmer's Report for a list of views available to you. See ["Using the programmer's report"](#) on page 76 for information about this report.

## Creating user views

You can subset the row(s) or reorder the column(s) within a view according to your needs. This is done in the application program through RDM language specification. This subset of data is called a user view. Once the DBA has defined the columns included in a view, you can use all or part of the view as a user view.

The following figure illustrates a view and a user view.



## Understanding columns and keys

Each view contains one or more columns that the DBA can designate as keys to the view. The keys can appear anywhere in the view. The DBA can define four different types of keys:

- ◆ Unique key
- ◆ Compound unique key
- ◆ Simple nonunique key
- ◆ Compound nonunique key

The simplest view has one unique key. This key value allows you to select and retrieve data. A unique key must have a valid, non-null value.

In a compound unique key, several columns are designated as unique logical keys, and the combination of the key values is unique, an "and" connection between the columns is implied. For example, to check customer orders for a certain part number, you would use a view with both customer number and part number as key values. RDM will retrieve the specific customer number and part number combination if it is present.

A nonunique key allows more than one row to contain the same value in a key column. An example is a customer file where you keep a list of notes or comments concerning each customer. You do not date the comments and you do not want more than one key, but for each customer, you want to retrieve a list of comments that may have been posted. In this case, the customer number could be defined as a single nonunique key. When the program does its first GET using a customer number, it will retrieve the first comment for that customer. A subsequent GET will retrieve the second comment; the third GET the third comment, etc. After RDM reaches the last comment for that customer, it will reach a boundary condition and return a NOT FOUND to the program.

A compound nonunique key is an extension of the simple nonunique key. Here, more than one column is defined as a nonunique key. However, all the nonunique keys together still do not completely describe the record occurrence as unique. You can still have more than one record with the same compound nonunique key.

You can access a set of rows by assigning values to the keys of the view (if there are any). The DBA determines which columns are keys and defines them on the Directory. The keys can be used to locate a specific record or to perform a generic read. Both types of reads return all qualifying rows from the views. You can also access a set of rows sequentially by not supplying any values for the keys.

A required column must contain a valid non-null value in order for the record to be included in the view. By default, a column is "not required" and does not need a value.

## Introduction to the Relational Data Manipulation Language (RDML)

While your application programs are written in COBOL, RDM uses the Relational Data Manipulation Language (RDML) to sign on and off the system, to maintain storage, to manipulate data, and to control data recovery. These functions are accomplished by the following RDML statements:

- ◆ **SIGN-ON/SIGN-OFF.** The SIGN-ON statement establishes communication between the programmer and RDM and identifies you as the user of the system. The SIGN-OFF statement informs RDM that you want to terminate your session.
- ◆ **RELEASE/FORGET.** The RELEASE and FORGET statements free internal storage without signing off the system.
- ◆ **DELETE.** Removes a row from the view.
- ◆ **GET.** Retrieves a row from the view.
- ◆ **INSERT.** Inserts a new row into the view.
- ◆ **UPDATE.** Updates column values in an existing row.
- ◆ **COMMIT/RESET.** Control database recovery. These statements function differently depending on the environment and recovery system supported.

Using the Directory data to build division entries, RDML statements are converted into standard COBOL source code by the RDML Precompiler. Standard compilers then convert the COBOL source code into object code. When the program executes, the Directory uses the physical data descriptions, and RDM uses the logical data descriptions, to access the database and present the data in the view requested by the application program.

For information on using the Relational Data Manipulation Language (RDML), see [“Coding RDM COBOL application programs”](#) on page 75; for the syntax of the RDML statements, see [“RDM COBOL application program statements”](#) on page 97.

## Introduction to the DBAID utility subset

The DBAID utility, an online and batch tool, allows the DBA to define a new view, open the view, issue RDML statements and examine the results. The DBA can then change the view, if necessary, reorder for efficiency, or try different access methods. These activities have no impact on the Directory.

Certain DBAID commands are also available to non-DBA users for use when constructing programs that use RDM views. Using the DBAID Utility subset of commands, you can test a view before testing with applications until you are sure that the view is correctly defined. You can also use the DBAID Utility subset as an educational tool for immediate hands-on experience with the views being accessed.

The DBAID subset has three command categories:

- ◆ System commands
- ◆ Built-in view commands
- ◆ RDML commands

System commands display information about the DBAID Utility currently executing including current users and active views. Built-in view commands allow you to inspect a view after it is opened. RDML commands allow you to test data with a defined view to make sure the view has been properly defined.

See [“Using the DBAID utility subset”](#) on page 27 for more information on the DBAID subset of commands available to you.

# 2

## Using the DBAID utility subset

A subset of the DBAID utility commands is available to the application programmer to use for testing a view before testing with applications. To make sure a view fits your specific requirements, DBAID can be run in a batch or online environment. Using DBAID, you can run test cases until you are satisfied that the view is correctly defined.

The DBAID utility subset has three command categories:

- ◆ **System commands.** These commands display information about the currently executing DBAID utility such as current users and active views.
- ◆ **Built-in view commands.** Use these commands to inspect a view after it is opened.
- ◆ **RDML commands.** Use RDML commands to test data with a defined view to make sure the view has been properly defined.

## DBAID commands list

The following table lists all the commands available to you by category with a brief description and a section reference for detailed information.

Some DBAID Utility subset commands have specific underlying file system restrictions. For more information on the restrictions for PDM file systems, refer to the *SUPRA Server PDM RDM PDM Support Supplement (OS/390 & VSE)*, P26-8221. For information on VSAM restrictions, refer to the *SUPRA Server PDM RDM VSAM Support Supplement (OS/390 & VSE)*, P26-8222.

Command	Description	Section
<b>System commands</b>		
LINESIZE	Specifies line width for DBAID output.	"LINESIZE command" on page 57
MARKS	Lists all open MARKs and the views they are marking.	"MARKS command" on page 59
PAGESIZE	Specifies the number of lines on the page/screen for DBAID output.	"PAGESIZE command" on page 62
USER-LIST	Displays column list for the view named.	"USER-LIST command" on page 70
USERS	Displays the current users of the system.	"USERS command" on page 71
VIEWS	Displays all views active in DBAID.	"VIEWS command" on page 73

Command	Description	Section
<b>Built in view commands</b>		
BY-LEVEL	Displays the column names in the view by level of occurrence.	"BY-LEVEL command" on page 34
COLUMN-TEXT	Displays the short and long text for a column in a view. (You can also code this as FIELD-TEXT.)	"COLUMN-TEXT command" on page 37
FIELD-DEFN	Displays the full description of a column in a view.	"FIELD-DEFN command" on page 43
VIEW-DEFN	Displays a condensed description of the view.	"VIEW-DEFN command" on page 72
VIEWS-FOR-USER	Lists the views related to the signed on user and the short text for the view.	"VIEWS-FOR-USER command" on page 74
<b>RDML commands</b>		
=	Reissues previous RDML command.	"= command" on page 32
BYE	Causes the DBAID Utility to exit.	"BYE command" on page 33
CAUTIOUS	Prohibits an automatic COMMIT.	"CAUTIOUS command" on page 36
COMMIT	Makes all updates since last commit permanent in the database.	"COLUMN-TEXT command" on page 37
DELETE	Removes a view record occurrence from database.	"DELETE command" on page 40
ERASE	Issues an RDM RESET if an 'X' FSI is returned.	"ERASE command" on page 42
FORGET	Frees the storage allocated by a previously issued MARK command.	"FORGET command" on page 45
GET	Retrieves and displays the requested row for the view indicated.	"GET command" on page 46
GO	Issues multiple GET commands and displays the rows in tabular format.	"GO command" on page 50
INSERT	Places a view row in the physical database on relative location specified.	"INSERT command" on page 53

Command	Description	Section
<b>RDML commands (cont.)</b>		
KEEP	Prohibits an automatic RESET.	“KEEP command” on page 56
MARK	Marks the current position of the view name established by the previous GET.	“MARK command” on page 58
OPEN	Readies either a virtual or stored view for use by the DBAID Utility.	“OPEN command” on page 60
RELEASE	Closes one or all views that have been opened and releases the occupied storage.	“RELEASE command” on page 63
RESET	Forces task level abend and rolls back any database updates since last commit.	“RESET command” on page 64
SIGN-OFF	Signs off a user from the DBAID Utility.	“SIGN-OFF command” on page 65
SIGN-ON	Identifies a user to the DBAID Utility.	“SIGN-ON command” on page 66
SURE	Causes a COMMIT after each successful insert, update, or delete.	“SURE command” on page 67
UPDATE	Updates data values in the database.	“UPDATE command” on page 68

## DBAID formatting guidelines

DBAID format is a series of commands, with one command per line, and a maximum of 72 characters per command. The guidelines for formatting DBAID in a batch environment or in an online environment are the same except that batch output is to a line printer.

Several DBAID syntax options simplify the use of DBAID:

- ◆ The FOR option used with the GO command (see “GO command” on page 50)
- ◆ The := syntax used with the UPDATE command (see “UPDATE command” on page 68)
- ◆ The MASS option used with the INSERT command (see “INSERT command” on page 53)

You can use an asterisk (\*) in DBAID for two functions. An \* entered in column 1 denotes a comment line. For example:

```
OPEN VIEW
GET VIEW                               Performs GET on VIEW
*THIS IS A TEST VIEW
```

You can also use the \* in a command as a substitute for the last view name used. For example:

```
OPEN VIEW
GET *                                   Performs GET on VIEW
OPEN VIEW2 = * FIELD1, FIELD5          Performs OPEN of user view
                                       VIEW2 where VIEW2 is created
                                       from VIEW1 specifying columns
                                       FIELD1, FIELD5 only.)
GET *                                   Performs GET on VIEW2
```

Using the \* as a substitute for the last view name used is described in each supported command's explanation.

## DBAID commands

The following sections describe the individual DBAID commands. Each section contains a description of the command's format, a list of considerations for using the command, if necessary, and a coding example or the expected output.

### **= command**

The = command reissues the previous RDML command.

---

=

---

### **Example**

In the following example, "=" causes another "GET NEXT CUST-PROD:"

```
GET NEXT CUST-PROD
```

```
=
```

## BYE command

The BYE command causes you to exit the DBAID Utility.

---

### BYE

---

#### General considerations

- ◆ In an online environment, the BYE command returns you to the RDM sign-on screen or other user-installed menu screens.
- ◆ If you entered DBAID with the task already signed-on to RDM, the BYE command does not perform a **SIGN-OFF**. If you entered DBAID with the task signed-off from RDM, the BYE command performs a sign-off.
- ◆ In a batch environment, the BYE command terminates the task.
- ◆ DBAID erases any unsaved virtual views.

## BY-LEVEL command

The BY-LEVEL command displays the column names in a view by level of occurrence, starting with the 0 level, followed by level 1, etc. RDM generates the column number when displaying this data.

---

**BY-LEVEL** [*view-name* [*column-number* ]]

---

---

### *view-name*

**Description** *Optional.* Specifies the name of the view whose column names you want to display.

**Format** Must be a valid, open view.

### Considerations

- ◆ If you omit this parameter, the BY-LEVEL command displays all column names for all of your open views.
  - ◆ You can enter an \* instead of a *view-name*. This substitutes the last view-name used.
- 

### *column-number*

**Description** *Optional.* Specifies the number of the column whose name is to be displayed.

**Format** Numeric characters

### Considerations

- ◆ If you use this parameter, you must have specified a *view-name*.
- ◆ If you omit this parameter, the BY-LEVEL command displays all column names of the specified view.

**Example**

BY-LEVEL			
NUMBER	VIEW NAME	COLUMN NAME	LEVEL
1	CUST-PROD	CUST-NO	0
2	CUST-PROD	PROD-NO	1
3	CUST-PROD	RENT	1
4	CUST-PROD	MAINT	1
5	CUST-PROD	INSTALL-DATE	1
6	CUST-PROD	CANCEL-DATE	1
7	CUST-PROD	PURCHASE-PRICE	1
1	CUSTOMER	CUST-NO	0
2	CUSTOMER	NAME	0
3	CUSTOMER	STATE	0
1	TEST	ZONED5	1
2	TEST	PACKED5	1
3	TEST	KEY2	1

## **CAUTIOUS command**

The CAUTIOUS command prohibits an automatic COMMIT. This command is the opposite of the SURE command. DBAID does not issue a COMMIT when an RDML modifying command returns an "\*" FSI. Instead, you must issue the COMMIT.

---

## **CAUTIOUS**

---

### **General consideration**

DBAID normally issues a COMMIT after every successful RDML modification. The CAUTIOUS command is not required; however, you can use it when you want manual control over COMMIT commands when updating the database.

---

## COLUMN-TEXT command

The COLUMN-TEXT command displays the short and long text for a column in a view.

---

**COLUMN-TEXT** [*view-name* [*column-name* ]]

---

---

### *view-name*

**Description**     *Optional.* Specifies the view to be used.

**Format**             Must be a valid, open view.

### **Considerations**

- ◆ If you omit this parameter, the COLUMN-TEXT command displays all column descriptions for all your open views.
  - ◆ You can enter an \* instead of a *view-name*. This substitutes the last view-name used.
- 

### *column-name*

**Description**     *Optional.* Identifies the column whose text is to be displayed.

**Format**             The column must already be part of the view.

### **Considerations**

- ◆ If you use this parameter, you must have specified a *view-name*.
- ◆ If you omit this parameter, the COLUMN-TEXT command displays the short and long text for all columns.
- ◆ You can substitute FIELD-TEXT as the command for COLUMN-TEXT.

### Example

```
COLUMN-TEXT CUST-PROD PROD-NO
VIEW NAME          COLUMN NAME
-----
CUST-PROD          PROD-NO
-----
                    SHORT TEXT
PRCU-PROD-NUM SHORT TEXT
67890123456789012345678901234567890123456789012
-----
                    LONG TEXT
-----
PRCU-PROD-NUM LONG TEXT 100
PRCU-PROD-NUM LONG TEXT 200
PRCU-PROD-NUM LONG TEXT 300
12345678901234567890123456789012345678901234567890123456789012
```

---

## **COMMIT command**

The COMMIT command makes all updates since the last COMMIT permanent in the database.

---

### **COMMIT**

---

#### **General consideration**

DBAID issues a COMMIT after every successful RDML modification unless you have issued a **CAUTIOUS** command. You can use the COMMIT command if you have issued a CAUTIOUS command.

## DELETE command

The DELETE command removes a view row from the database.

---

**DELETE** [ALL] *view-name*

---

---

### ALL

**Description** *Optional.* Deletes all rows retrieved by automatically generated GET NEXTs using the logical key qualification of the GET command issued before the DELETE.

**Consideration** If a program specifies a GET without a USING phrase, DELETE ALL deletes all rows in a view.

---

### *view-name*

**Description** *Required.* Identifies the name of the view containing the row(s) to be deleted.

**Format** Must be a valid, open view.

### General considerations

- ◆ Before performing the DELETE, you must perform a successful GET command.
- ◆ You can enter an \* instead of a *view-name*. This causes DBAID to substitute the last view-name used.

## Examples

- ◆ This example deletes the one occurrence of SAMPLE-VIEW you obtained based on the value in KEY1:

```
GET SAMPLE-VIEW FOR UPDATE USING KEY1
DELETE SAMPLE-VIEW
```

- ◆ This example deletes all occurrences of rows you obtained based on the value in KEY1:

```
GET SAMPLE-VIEW FOR UPDATE USING KEY1
DELETE ALL SAMPLE-VIEW
```

- ◆ The previous code processes in the manner shown in this code:

```
GET FIRST SAMPLE-VIEW FOR UPDATE USING KEY1.
DELETE SAMPLE-VIEW.
RETURN.
GET NEXT SAMPLE-VIEW FOR UPDATE USING KEY1
    NOT FOUND GO TO CONTINUE.
DELETE SAMPLE-VIEW.
GO TO RETURN.
CONTINUE.
```

## **ERASE command**

The ERASE command causes DBAID to automatically issue an RDM RESET if an RDML command results in an "X" FSI. This command is the opposite of the **KEEP** command and causes RDM to automatically issue a **RESET** if an "X" FSI is returned.

---

### **ERASE**

---

## FIELD-DEFN command

The FIELD-DEFN command displays the full description of columns in a view.

---

**FIELD-DEFN** [ *view-name* [ *column-name* ] ]

---

---

### *view-name*

**Description** *Optional.* Specifies the view to be used.

**Format** Must be a valid, open view.

### **Considerations**

- ◆ If you omit this parameter, the FIELD-DEFN command displays all column descriptions for all your open views.
- ◆ You can enter an \* instead of a *view-name*. This causes DBAID to substitute the last view-name used.

---

### *column-name*

**Description** *Optional.* Identifies the column whose description is to be displayed.

**Format** The column must already be part of the view.

### **Considerations**

- ◆ If you use this parameter, you must specify a *view-name*.
- ◆ If you omit this parameter, the FIELD-DEFN command displays all column descriptions of the specified view.

## Example

```
FIELD-DEFN
VIEW-NAME          (+) CUSTOMER
COLUMN-NAME        (+) CUST-NO
COLUMN-POS         (+) 0
COLUMN-LEN         (+) 5
ASI-POS           (+) 60
COLUMN-DEC         (+) 0
OUTPUT-LEN        (+) 5
MASK-LEN          (+) 15
FORMAT            (+) Z
EDIT-MASK         (+) ZZZZZZZZZZZZZZZ9
READING           (+) CUST;NO
DELETABLE         (+) Y
INSERTABLE        (+) Y
REPLACEABLE       (+) N
COLUMN-LVL        (+) 0
KEY-NUMBER        (+) 1
REQUIRED          (+) Y
UNIQUE            (+) Y
EDIT-TRANS        (+) E
ORDERING          (-)
***MORE***
```

---

## FORGET command

The FORGET command frees the storage allocated by a previously issued MARK command.

---

**FORGET** *mark-name*

---

---

### *mark-name*

**Description** *Required.* Specifies what mark information should be forgotten.

**Format** 1–30 alphanumeric characters

**Consideration** Must be a name you assigned with the **MARK** command.

### **General consideration**

Once you issue a FORGET command, you release the indicated mark and cannot regain it without issuing a new MARK command.

## GET command

The GET command retrieves and displays a row for the indicated view.

---

```
GET [NEXT  
    LAST  
    SAME view-name  
    FIRST  
    PRIOR]  
  
[FOR UPDATE]  
[AT mark-name]  
[USING literal1[literal2...literaln ]]
```

---

```
[NEXT  
LAST  
SAME  
FIRST  
PRIOR]
```

**Description**     *Optional.* Modifies the order of row retrieval.

**Default**         NEXT    If no current position exists, NEXT defaults to FIRST.

## Considerations

- ◆ For a unique key:
  - GET NEXT retrieves either the row immediately after the current row or the first row, if no current position exists.
  - GET LAST retrieves the last row.
  - GET SAME retrieves the latest row if a current position exists.
  - GET FIRST retrieves the first row in the view.
  - GET PRIOR retrieves either the row immediately before the current row or the last row if no current position exists.
  - Use GET PRIOR only in connection with a "USING key" phrase for predictable results.
  - If the underlying file system cannot perform the GET PRIOR and GET LAST functions, an error is returned.
- ◆ For a nonunique key:
  - GET NEXT retrieves the next occurrence of the row within the generic group.
  - GET LAST retrieves the last occurrence of the row.
  - GET SAME retrieves the latest row if a current position exists.
  - GET FIRST retrieves the first occurrence of the row with the indicated key.
  - GET PRIOR will perform a read reverse within the group of nonunique keyed rows.

---

### *view-name*

**Description**    *Required.* Specifies the view to be used.

**Format**        Must be a valid, open view.

**Consideration** You can enter \* instead of a *view-name*. This causes DBAID to substitute the last view-name used.

## FOR UPDATE

**Description** *Optional.* Allows you to lock out other users' modifications to the row you are retrieving.

### Considerations

- ◆ The FOR UPDATE phrase allows you to perform modifications dependent on the current contents of the row.
- ◆ If you do not need to be certain of the content of the row, you can use a GET without the FOR UPDATE phrase. When you use an UPDATE or DELETE function, the "automatic hold" facility of the system performs the lock before modifying the row.
- ◆ FOR UPDATE implies that all physical resources will be locked until you issue another GET or an **INSERT**, **UPDATE**, **DELETE**, **COMMIT**, or **RESET**. This practice may lead to system inefficiency.
- ◆ When you issue a GET, RDM can return data to you that is currently being updated by another tasks. If you subsequently issue a FOR UPDATE, that update may fail in the following ways:
  - The other task has not yet committed. The update will fail with an FSI=U status and message VIEW HELD BY ANOTHER TASK - RETRY LATER.
  - The other task committed between the GET and the update. The update will fail with an FSI=D and the message COLUMN VALUES HAVE BEEN CHANGED.

---

## AT *mark-name*

**Description** *Optional.* Repositions a view previously marked with the **MARK** command.

**Consideration** The USING and AT phrases cannot be used with the same GET command.

---

**USING *literal1*[*literal2* ... *literaln*]**

**Description**     *Optional.* Identifies a value or set of values to be used for a keyed GET.

**Format**            Either character, hexadecimal, or numeric data. Character and hexadecimal data must be enclosed in quotes; numeric data need not be, for example:

```
USING 'ABCD'        - character data
USING 1234          - numeric data
USING X'A10C'      - hexadecimal
USING 123 'ABC'    - combination (two keys)
```

**Considerations**

- ◆ The number of keys specified in the GET statement must be less than or equal to the number of keys in your specified column list. No more than nine keys are allowed in one view.
- ◆ RDM treats any omitted keys as generic keys. The use of generic keys is a convenient feature for allowing both direct access to a view and a sequential scan of many rows. RDM returns all occurrences of a particular unspecified column as long as the other keys are satisfied.
- ◆ The order of specified keys in the USING phrase corresponds to the order of key declarations in your column list.

## GO command

The GO command issues a penetration **GET** request followed by a series of sweeping GET requests and displays the rows in tabular format.

---

GO  $\left[ \begin{array}{l} \text{NEXT} \\ \text{PRIOR} \end{array} \right]$  *view-name*

$\left[ \begin{array}{l} \text{START} \left\{ \begin{array}{l} \text{NEXT} \\ \text{LAST} \\ \text{SAME} \\ \text{FIRST} \\ \text{PRIOR} \\ \text{AT } \textit{mark-name} \end{array} \right\} \end{array} \right]$

FOR *number-of-rows*

$\left[ \left\{ \begin{array}{l} \text{FROM} \\ \text{USING} \end{array} \right\} \textit{literal}_1(\textit{literal}_2 \dots \textit{literal}_n) \right]$

---

$\left[ \begin{array}{l} \text{NEXT} \\ \text{PRIOR} \end{array} \right]$

**Description** *Optional.* Specifies the **GET** command modifier to be used in retrievals after the initial penetration.

**Default** NEXT

---

### *view-name*

**Description** *Required.* Specifies the view to be accessed.

**Format** Must be a valid, open view.

**Consideration** You can enter an \* instead of a *view-name*. This causes DBAID to substitute the last view-name used.

---

START {  
 NEXT  
 LAST  
 SAME  
 FIRST  
 PRIOR  
 AT *mark-name*}

**Description** *Optional.* Specifies the **GET** command modifier to be used for the initial penetration of the database.

**Default** FIRST

---

### FOR *number-of-rows*

**Description** *Optional.* Indicates the number of rows (or number of **GET** NEXTs minus 1) to be performed.

**Format** Numeric characters

**Consideration** GET NEXTs repeat until the count is exhausted or until the last row is retrieved, whichever occurs first.

---

[ {  
 FROM  
 USING } *literal* [ *literal*<sub>2</sub> ... *literal*<sub>n</sub> ] ]

**Description** *Optional.* Identifies a value or set of values to use for a keyed **GET**.

**Format** Either character or numeric data. Character data, if it includes blanks, must be enclosed in quotes; numeric data need not be.

**Options**

FROM	Uses the key values only on the initial penetration; the scan is unqualified.
USING	Uses the key values for both the initial penetration and the subsequent scan.

## Considerations

- ◆ The number of keys specified in the **GET** statement must be less than or equal to the number of keys in your specified column list.
- ◆ RDM treats any omitted keys as generic keys. The use of generic keys allows for both direct access to a view and a sequential scan of many rows. RDM returns all occurrences of a particular unspecified key as long as the other keys are satisfied.
- ◆ The order of specified keys in the **USING** phrase corresponds to the order of key declarations in your column list.

## General considerations

- ◆ RDM displays the output in columnar fashion. If more data is to be displayed than will fit on a screen/page, RDM uses an alternate format.
- ◆ After the **GO** command displays a page of rows (see **PAGESIZE**), the prompt **\*\*MORE\*\*** will be issued. Enter a blank line for each additional page in batch mode, and press "ENTER" for each additional page in online mode.
- ◆ At the end of the series of rows retrieved by **GO**, the prompt **\*\*END\*\*** is issued.
- ◆ Do not use "FOR *number-of-rows*" for online because it does not pause until the last screen.
- ◆ The **GO** command always looks ahead one more row so it can determine whether to display the **\*\*MORE\*\*** or **\*\*END\*\*** message. It can be confusing if you issue a **GET** after the **GO** because a row may appear to have been skipped.

## INSERT command

The INSERT command places a view row in the physical database based on the relative location specified.

---

```
INSERT [ NEXT
        LAST
        FIRST
        PRIOR ] view-name [MASS]
```

---

**NEXT**  
**LAST**  
**FIRST**  
**PRIOR**

**Description** *Optional.* Specifies the relative location of the row to be inserted in relation to existing rows. The Access Set Description (ASD) may override this specification.

**Default** NEXT If not positioned in the view, NEXT defaults to LAST, and PRIOR defaults to FIRST.

**Considerations** For nonuniquely keyed values:

- ◆ If the DBA specified ordering in the view definition, or if the physical data manager does not allow program control of ordering, the specification on the INSERT statement is ignored.
- ◆ INSERT FIRST places a row in the first position in the view.
- ◆ INSERT NEXT places a row after the current row. If no current position exists, the row is placed in the last position in the view.
- ◆ INSERT PRIOR places a row before the current row. If no current position exists, the row is placed in the first position in the view.
- ◆ INSERT LAST places a row in the last position of the view.

**view-name**

- Description** *Required.* Specifies the name of the view in which you want the rows inserted.
- Format** Must be a valid, open view.
- Consideration** You can enter an \* instead of a *view-name*. This causes DBAID to substitute the last view-name used.
- 

**MASS**

**Description** *Optional.* Allows multiple rows to be inserted in the physical database.

**Considerations**

- ◆ The positioning parameter specified (NEXT, LAST, FIRST, or PRIOR) is used on every RDM **INSERT** command issued by MASS insert.
- ◆ Views are entered immediately following this command after the two prompt lines, MASS INSERT PROCESSING INITIATED and ENTER "END." TO EXIT MASS INSERT, are displayed (see the second example, below).
- ◆ Rows are inserted as flat records. Separate the column values with commas. To insert rows longer than one line, terminate the list of values with a comma and continue the input on the next line.
- ◆ Place multiple rows on a single line by leaving a blank between rows.
- ◆ Use a pair of single quote marks to insert columns containing spaces.
- ◆ If you have columns with no values, enter two consecutive commas to indicate their absence. This value is treated as a null value for packed or zoned fields, as a large number (X'40404040' or 67372036 integer) for binary fields, and as blanks for a character field.
- ◆ Specify END. after entering all rows to be inserted into the view. The period after END is required.

**General considerations**

- ◆ After entering column values on a single insert (not using MASS), the row is displayed. The message INSERT (Y/N)? appears. Enter a Y to insert the row. Any other response does not insert the row.
- ◆ Processing stops if ten errors are detected while using the MASS insert; otherwise, enter END. to terminate insert processing.

## Examples

### ◆ Example of a single INSERT:

```

> INSERT *
NUMBER
> 9998
PRODUCT
> AAAA
INSTALLED
> 100883
NUMBER ( ) 9998
PRODUCT ( ) AAAA
INSTALLED ( ) 100883
INSERT (Y/N)?
> Y
FSI: * VSI: + MSG: SUCCESSFUL COMPLETION

```

### ◆ Example of a MASS INSERT (single row):

```

> INSERT * MASS
MASS INSERT PROCESSING INITIATED.
ENTER "END." TO EXIT MASS INSERT.
> 9997,BBBB,100783
FSI: * VSI + MSG: SUCCESSFUL COMPLETION
* Example of MASS insert (using comma to continue to next
line):
> 9996,CCCC,
> 100683
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
* Example of a MASS insert (multiple rows on a single line):
> 9995,DDDD,100583 9994,EEEE,100483 9993,FFFF,100383
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
* Example of ending the MASS insert processing:
> END.
MASS INSERT PROCESSING COMPLETED.

```

## **KEEP command**

The KEEP command prohibits an automatic **RESET**. This command is the opposite of the **ERASE** command. KEEP causes DBAID not to issue a RESET when it receives an "X" FSI from RDM. Instead, DBAID keeps the database as it is and allows the user to decide to RESET or not.

---

### **KEEP**

---

---

## LINESIZE command

The LINESIZE command specifies the number of characters to display in a line.

---

**LINESIZE** [*number-of-characters*]

---

---

### *number-of-characters*

**Default** 79 characters per line.

**Description** *Optional.* Indicates the number of characters to display on a line.

**Format** 2–3 numeric characters

**Options** 10 through 132

### Considerations

- ◆ In an online environment, the line size maximum is restricted to the line capacity of the screen.
- ◆ If you omit the *number-of-characters*, the command displays the current LINESIZE setting.

## MARK command

The MARK command marks the current position of the view row established by the previous GET command.

---

### MARK *view-name* AT *mark-name*

---

---

#### *view-name*

- Description** *Required.* Identifies the view name established by the previous GET command.
- Format** Must be a valid, open view.
- Consideration** You can enter an \* instead of a *view-name*. This substitutes the last view-name used.
- 

#### AT *mark-name*

- Description** *Required.* Assigns a name to the location where the position of the current view will be marked.
- Format** 1–30 alphanumeric characters
- Consideration** The name assigned is the name you use in a later GET AT request to retrieve this same view row.

#### General considerations

- ◆ Use the AT phrase in the GET command to reposition the view at the position set by the MARK command.
- ◆ You can create any number of MARKs for a view, but to save internal memory space, it is best to reuse *mark-name* when possible.
- ◆ A *mark-name* may be reused.
- ◆ The number of MARKs you can create is limited by the amount of internal memory space allocated to your task.

---

## MARKS command

The MARKS command lists all open MARKs and the views they are marking.

---

### MARKS

---

#### Example output

MARKS	MARK NAME	VIEW NAME
MARK6		CUST-PROD
MARK5		CUST-PROD
MARK4		CUST-PROD
MARK3		CUST-PROD

## OPEN command

The OPEN command readies a saved or virtual view for use by DBAID.

---

**OPEN** [*user-view-name*=]*view-name*[*column1*[,...,*columnn*] ]

---

### *user-view-name*=

- Description** *Optional.* Gives an existing view a name for use in DBAID.
- Format** 1–30 alphanumeric characters and the special characters # and \$. The first character must be alphabetic or a special character. If the first character is a special character, the second character must be alphabetic.

### Considerations

- ◆ If *user-view-name* is not specified, it will be the same name as the *view-name*.
- ◆ You can use this method (together with the column parameter) to create many smaller views from one common view.
- ◆ To OPEN a view that has not been listed or defined in the same session of DBAID, the user must be related to the view in the Directory.

---

### *view-name*

- Description** *Required.* Identifies the virtual or stored view to be readied for use.
- Format** Must be a valid view.

### Considerations

- ◆ You can enter an \* instead of a *view-name*. This substitutes the last view-name used.
- ◆ The list of column names can be continued on successive lines by ending the line you are entering with a comma. The command "USER-LIST" displays the list of columns used to open the view after it has been opened.
- ◆ Issuing an OPEN request on a view without first issuing a LIST request causes RDM to directly open the view with the user relations checked but without text available to DBAID.
- ◆ If a virtual view has the same name as a saved view, the virtual view is used.

---

**column1[,...,columnn]**

**Description**     *Optional.* Identifies the column or list of columns to be included in the user view. If omitted, all columns in the view are in the user view.

**Format**             The columns must already be part of the view being opened.

**General consideration**

The OPEN returns information about the storage used in the form of the message:

```
nnnnn BYTES USED IN OPENING VIEW
```

where *nnnnn* is the amount of storage used by the view.

**Example**

```
OPEN CP-ONLY = CUST-PROD CUST-NO, PROD-NO
```



---

Only CUST-NO and PROD-NO are returned when you do GET CP-ONLY, even though CUST-PROD has six columns defined.

---

## PAGESIZE command

The PAGESIZE command specifies the number of lines to display on a screen/page.

---

**PAGESIZE** [*number-of-lines*]

---

---

### *number-of-lines*

**Description** *Optional.* Indicates the number of lines to display on a screen/page.

**Default** 24 lines

**Options** Must be greater than 10, with no maximum limit.

### **Considerations**

- ◆ In an online environment, the PAGESIZE maximum should be no more than the screen capacity.
- ◆ If you omit the number-of-lines, the command displays the current PAGESIZE setting.

---

## RELEASE command

The RELEASE command closes a specific view or all views that are open, and releases the occupied storage within RDM.

---

**RELEASE** [*view-name*]

---

---

### *view-name*

**Description**    *Optional.* Specifies the view to release.

**Format**        Must be a valid, open view.

### **Considerations**

- ◆ You can enter an \* instead of a *view-name*. This substitutes the last *view-name* used.
- ◆ If you omit this parameter, the RELEASE command releases all of your open views.

### **General consideration**

This command does not affect virtual view text of the view(s).

## RESET command

The RESET command rolls back any database updates since the last COMMIT point.

---

### RESET

---

#### General considerations

- ◆ Use RESET only after unsuccessful RDML updates. DBAID does not automatically issue a RESET command when an "X" FSI is returned. See the KEEP and ERASE commands.
- ◆ In CICS, a RESET backs out any database updates since the last COMMIT point but does not restart DBAID.
- ◆ The RESET command restores your database to the last COMMIT point and you lose position on all views. Therefore, the GET SAME, DELETE, or UPDATE commands are not valid after a RESET. A GET NEXT command positions you on the first record while a GET PRIOR command positions you on the last record after a RESET.

## **SIGN-OFF command**

The SIGN-OFF command signs off the user from DBAID.

---

### **SIGN-OFF**

---

#### **General consideration**

Use the SIGN-OFF command to remove yourself as a user without terminating DBAID.

## SIGN-ON command

The SIGN-ON command identifies the user to DBAID.

---

**SIGN-ON** *user-name* [*password*]

---

---

### *user-name*

**Description** *Required.* Indicates the name of the user.

**Format** Must be a valid user name already defined on the Directory.

---

### *password*

**Description** *Optional.* Indicates the user's password.

**Format** Must be a valid password defined on the Directory.

---

### General considerations

- ◆ In an online environment, initializing DBAID completes the SIGN-ON before you enter DBAID and need not be repeated.
- ◆ In a batch environment, DBAID blanks the password field before printing the output.

### Example

```
SIGN-ON  JDOE  PRGMPSWD
```

## **SURE command**

The SURE command causes a **COMMIT** after each successful **INSERT**, **UPDATE** or **DELETE**. The SURE command is the opposite of the **CAUTIOUS** command and causes RDM to automatically issue a **COMMIT** if an "\*" FSI that alters the database is returned by an RDML command.

---

### **SURE**

---

## UPDATE command

The UPDATE command updates data values in the database.

---

### UPDATE *view-name*

**[*column*<sub>1</sub>:=*literal*<sub>1</sub>[,...,*column*<sub>*n*</sub>:=*literal*<sub>*n*</sub>]**

---

---

#### *view-name*

**Description** *Required.* Identifies the view you wish to update.

**Format** Must be a valid, open view.

**Consideration** You can enter an \* instead of a *view-name*. This causes DBAID to substitute the last view-name used.

---

#### ***column*<sub>1</sub>:=*literal*<sub>1</sub>[,...,*column*<sub>*n*</sub>:=*literal*<sub>*n*</sub>]**

**Description** *Optional.* Identifies a column in the view that is to have the value of the literal.

**Format** *column* The column must already be part of the view being updated.

**:=** Must be coded as shown.

*literal* Character or numeric data. Hexadecimal value is not allowed.

## Considerations

- ◆ In an online environment, DBAID displays each updateable column, and accepts replacement values. Entering a null line leaves the column unchanged; entering new data replaces the column value in the row. After all updateable columns are processed, DBAID displays the prompt "UPDATE (Y/N)" and requires a response.
- ◆ In a batch environment, use the "Column, := literal syntax" when updating columns in the row. DBAID updates only the columns you specify; all others remain the same. To update a row, indicate the column you want to update, the :=, and the new value for the column.
- ◆ Do not use single quotes around character or numeric literals.
- ◆ Use single quotes to change the value of a column to blanks. A literal of spaces (keyed in) must be in single quotes. Pressing ENTER does not affect the column's value.
- ◆ You cannot use the UPDATE function to modify key column values.
- ◆ To UPDATE a row, you must first retrieve the row using the GET command.
- ◆ You cannot use UPDATE to change all the values in a defined column to a specific value.



---

You cannot change all PROD-CODES to "T100."

---

## Example

```
UPDATE CUST-PROD  RENT: = 175.00, MAINT = 50.00
```

## USER-LIST command

The USER-LIST command displays the column list for the user view named.

---

**USER-LIST** *view-name*

---

---

### *view-name*

**Description** *Required.* Identifies the view or user view to display.

**Format** Must be a valid view.

**Consideration** You can enter an \* instead of a *view-name*. This substitutes the last view-name used.

### Example output

```
USER-LIST PO-CODE-ONLY
USER VIEW NAME :    PO-CODE-ONLY
VIEW NAME : CUSTOMER-PURCHASE-ORDER
USER VIEW LIST :
CUST-NO , PURCHASE-ORDER-CODE , END.
```

## USERS command

The USERS command displays information about the current users of the system.

---

### USERS

---

#### General considerations

- ◆ The information displayed with this command includes:
  - Station Number—The number of the user's station.
  - User Name—The name of the user for that station.
  - Time of Sign-on—The sign-on time of that user.
  - Processing Time—The total CPU time that user has used.
  - Request Count—The number of requests that user has issued.
  - Duration of Last Request—The duration of the user's last request.
- ◆ This command is operational only in the online environment.

#### Example output

In a non-CICS system, a USERS display looks like this:

STN.#	USER NAME	REQ.#	I/O TIME	SIGN-ON
LAST REQ.				
102	Character Name of User	572	12:05:32	12:06:50
09:02:35				

## VIEW-DEFN command

The VIEW-DEFN command displays a condensed description of a view.

---

**VIEW-DEFN** [*view-name*]

---

---

### *view-name*

**Description**    *Optional.* Specifies the view whose condensed description you want to display.

**Format**        Must be a valid, open view.

### Considerations

- ◆ You can enter an \* instead of a *view-name*. This causes DBAID to substitute the last view-name used.
- ◆ Omitting this parameter displays a condensed description of all your open views.

### Example

```
> VIEW-DEFN
VIEW-NAME           (+) CUSTOMER
INS-ORDER           (+) N
TOTAL-SIZE          (+) 63
TOTAL-COLUMNS      (+) 3
TOTAL-LEVELS        (+) 1
TOTAL-DELETABLE     (+) 3
TOTAL-INSERTABLE    (+) 3
TOTAL-REPLACABLE    (+) 3
TOTAL-REQUIRED      (+) 1
TOTAL-KEYS           (+) 1
TOTAL-NONUNIQUE     (+) 0
***MORE***
```

---

## VIEWS command

The VIEWS command displays all of the views currently active in DBAID.

---

### VIEWS

---

#### General consideration

The information displayed with this command includes:

- ◆ User View—The name of the user view.
- ◆ View—The name of the view of which this user view is a part.
- ◆ Status—Indicates whether the user view is open or released.

#### Example output

USER VIEW	VIEW	STATUS
CUSTOMER-PURCHASE-ORDER	CUSTOMER-PURCHASE-ORDER	OPENED
PO-CODE-ONLY	CUSTOMER-PURCHASE-ORDER	OPENED

## VIEWS-FOR-USER command

The VIEWS-FOR-USER command lists the names and short text for the views related to the signed on user.

---

### VIEWS-FOR-USER

---

#### Example output

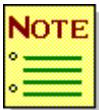
```
VIEWS-FOR-USERS
VIEW NAME          DATE          TIME
          SHORT DESCRIPTION
-----
CUST-QUERY          08/17/99     17:27:32
-----
PROD-QUERY          08/17/99     17:36:22
-----
PRCU-QUERY          08/17/99     17:30:18
-----
EDUC-QUERY          08/17/99     17:33:20
-----
INST-QUERY          03/16/99     17:45:39
-----
SYST-QUERY          10/19/99     13:34:58
.....
***MORE***
```

# 3

## Coding RDM COBOL application programs

This chapter presents the requirements and general guidelines for coding RDM COBOL application programs. The following topics are addressed:

- ◆ Using the programmer's report
- ◆ Coding the DATA DIVISION
- ◆ Coding the PROCEDURE DIVISION



There is one special requirement for IDENTIFICATION DIVISION section and none for the ENVIRONMENT DIVISION section of your programs. If the verb REMARKS is used, it MUST appear before the line containing the verb PROGRAM-ID. It is Cincom's recommendation to use the "\*" comment line instead, as the REMARKS verb is no longer an ANSI standard COBOL verb. Otherwise, code the statements in these two sections of your program as you would in any COBOL program. See "[Sample RDM COBOL application program](#)" on page 127 for a sample COBOL program.

After you code your application programs, the Relational Data Manipulation Language (RDML) Compiler converts the statements into the proper set of assignments and CALL statements to utilize RDM. Each RDML statement must start on a line by itself and must end with a period. Under CICS, you must use command level COBOL; macro level COBOL is not supported. See "[Compiling and linking an RDM COBOL application program](#)" on page 123 for information on using the RDML Compiler and for linking the compiled program.

## Using the programmer's report

RDM provides a programmer's report which describes the layout of a view and the column names in a view. The DBA provides the report which should be used when you are constructing keyed GETs or when you are determining the columns to subset for a user view.

The programmer's report provides information about the column type, column name, and the picture clause generated by the RDML compiler. The column type may be KEY, NONUNIQUE KEY, or REQUIRED. The following example shows a programmer's report:

```
*** RELATIONAL DATA MANAGER COBOL PROGRAMMER'S REPORT FOR SCHEMA QUERYDBM ***  
VIEW:  CUST-PROD
```

COLUMN TYPE	COLUMN NAME	PICTURE
KEY	CUST-NO	9(05) CUST-NO
KEY	PROD-NO	X(0004) PRCU-PROD_NUM
	RENT	9(07)V9(02) USAGE COMP-3
	MAINT	9(07)V9(02) USAGE-COMP-3
	INSTALL-DATE	9(06)
	CANCEL-DATE	9(06)
	PURCHASE-PRICE	9(07)V9(02) USAGE COMP-3

---

## Coding the DATA DIVISION

The DATA DIVISION of a COBOL program describes the information or data to be processed by the program. This information includes the input/output records and their data fields. The DATA DIVISION also defines miscellaneous work areas and the format and characteristics of each. This section presents the requirements and addresses the general guidelines for coding statements in the DATA DIVISION of a RDM COBOL program as follows:

- ◆ Specifying views and user views
- ◆ Specifying TIS-CONTROL
- ◆ Data validation indicators



---

Do not use COBOL reserved words in a view. The COBOL compiler will try to interpret them as COBOL commands, not as part of the view.

---

## Specifying views and user views

To use a specific view in your program, code an **INCLUDE** statement with the name of the view, for example:

```
01 INCLUDE CUST.
```

You can create your own user view by selecting columns from a view. This subset of the view is also specified with the **INCLUDE** statement. Indicate the name of your user view as well as the columns to be included from the view. For example:

```
01 CUST-MAIL INCLUDE CUST (NAME, ADDRESS, CITY, STATE, ZIP).
```

Unless you are accessing data values that are shared between views, each user view can be positioned independently and can act independently. If you create your own user view, you cannot change the effect of required columns in terms of their being available for inserts. However, you can modify the column order in a view. Be aware that rearranging key columns may adversely affect your application's performance, and we do not recommend it.

Each **INCLUDE** statement also has associated row and status data areas generated by RDM. The row data area specifies where data for each included view will be placed in the program.

The status data area has one-to-one mapping to the fields in the row data area and contains one byte of information indicating whether the data is valid, has changed since your last access, or is missing (see "**RDM status indicators**" on page 80 for information on RDM status indicators). For example:

```
*
*01 CUSTOMER INCLUDE CUST (CUST-NO,NAME,CITY).
01 RDM-CUSTOMER.
10 CUSTOMER.
   20 CUST-NO                PIC S9(05).
   20 NAME                   PIC X(040).
   20 CITY                   PIC X(020).
10 ASI-CUSTOMER.
   20 ASI-CUST-NO            PIC X.
   20 ASI-NAME               PIC X.
   20 ASI-CITY               PIC X.
```

## Specifying TIS-CONTROL

You must include the special view TIS-CONTROL in each program issuing an RDML request. TIS-CONTROL is used for passing parameters between the application and RDM and contains operation, status, and other information required to control access to all views. When you specify an **INCLUDE for TIS-CONTROL**, the RDML compiler generates the following:

```
*01  INCLUDE TIS-CONTROL.
      10  TIS-OBJECT-NAME      PIC X(30).
      10  TIS-OPERATION.
           20  TIS-ID          PIC X(2).
           20  TIS-OPCODE     PIC X.
           20  TIS-POSITION   PIC X.
           20  TIS-MODE       PIC X.
           20  TIS-KEYS       PIC X.
      10  TIS-FSI             PIC X.
      10  TIS-VSI             PIC X.
      10  FILLER              PIC X(2).
      10  TIS-MESSAGE        PIC X(40).
      10  TIS-PASSWORD       PIC X(8).
      10  TIS-OPTIONS        PIC X(4).
      10  TIS-CONTEXT        PIC X(4).
      10  TIS-LVCONTEXT      PIC X(4).
```

If you want to pass rows to external modules (subroutines) from your application, code an INCLUDE statement in the LINKAGE SECTION of the subroutine instead of the WORKING-STORAGE SECTION. If a subroutine issues any RDML commands, it must define, or be passed, a TIS-CONTROL area.

Use the TIS-OPTIONS field to specify DEBUG and TRACE. To debug an RDM call, place DBUG in the TIS-OPTIONS field before the call to RDM. To trace an RDM call, place TRAC in the TIS-OPTIONS field before the call to RDM. Debug and trace output is written to the DMLPRINT dataset. To close the DMLPRINT file within an application, place NBUG in the TIS-OPTIONS field before the call to RDM.

## RDM status indicators

RDM returns status indicators to the application program to indicate RDML processing results. Although the status indicators are actually returned by RDM during execution of the program and should be checked in the procedure division logic, we discuss them here because the status indicator fields are located in the RDML Precompiler expanded INCLUDE code in the DATA DIVISION.

The type and validity of the values you can place in the columns of your program statements are determined by the DBA. If you code a value which does not meet established criteria, you will receive a value error in the form of a status indicator. There are three kinds of status indicators:

- ◆ Function Status Indicator (FSI)
- ◆ Attribute (Column) Status Indicator (ASI)
- ◆ Validity Status Indicators (VSI)

## Function Status Indicator (FSI)

A Function Status Indicator (FSI) reflects the success or failure of your RDML request. (The FSIs are obtained from TIS-CONTROL.) An associated message is provided in the TIS-MESSAGE area of TIS-CONTROL. The following code shows an example of the generation of this control region. The RDML compiler changes RDML requests into comments by placing an asterisk in column 7 of each statement. (All other statements are generated by the RDML compiler).

```
*01  INCLUDE TIS-CONTROL.
      10  TIS-OBJECT-NAME      PIC X(30).
      10  TIS-OPERATION.
           20  TIS-ID          PIC X(2).
           20  TIS-OPCODE     PIC X.
           20  TIS-POSITION   PIC X.
           20  TIS-MODE       PIC X.
           20  TIS-KEYS       PIC X.
      10  TIS-FSI             PIC X.
      10  TIS-VSI             PIC X.
      10  FILLER              PIC X(2).
      10  TIS-MESSAGE        PIC X(40).
      10  TIS-PASSWORD       PIC X(8).
      10  TIS-OPTIONS        PIC X(4).
      10  TIS-CONTEXT        PIC X(4).
      10  TIS-LVCONTEXT      PIC X(4).
```

FSIs have the following meanings:

FSI value	Meaning
*	Successful.
D	Data error. The request would have run with valid values in the columns. You need to check the ASIs to find the column(s) that contains the invalid value.
F	Fail. This indicates a major error. Something may be wrong with the database, or you may have attempted to perform an illegal function on the user view.
N	Fail due to occurrence problem. This may be due to a GET not found or an INSERT duplicate found.
S	Security.
U	Unavailable resources.
X	RESET recommended. While processing, RDML function modifications were made to the database before the error condition was detected. Issue a RESET to restore the database. This code overrides D, F, S, or U indicators.

### Attribute (Column) Status Indicator (ASI)

The Attribute (Column) Status Indicator (ASI) reflects the status of each column defined in your view. ASIs have one-to-one mapping to each column in the user view and are placed immediately following the last column in your user view.

You can access ASIs through COBOL-assigned names generated by the RDML compiler. When you code your program, an **INCLUDE** statement for the user view is required. The RDML compiler generates a field for each column included in the user view. The RDML compiler also generates a field for each required ASI column by preceding each column name with the characters ASI-. When you have a data column name containing more than 26 characters, the RDML compiler truncates any trailing characters when forming the column status field. The following is an example of this generation (the asterisk indicates the statement you code; the RDML compiler generates all other statements):

```
*01  PROD1 INCLUDE PRODUCT.
01  LUV-PROD1.
    10  PROD1.
        20  PROD-NO                PIC X(004).
        20  PROD-DESC              PIC X(040).
    10  ASI-PROD1.
        20  ASI-PROD-NO            PIC X.
        20  ASI-PROD-DESC         PIC X.
```

ASIs have the following meanings:

ASI value	Meaning
=	The column exists and has not changed since the last access. (Valid for GET processing only.)
-	The column is missing. It has a null value. (Valid for GET processing only.)
+	The column exists but has changed since the last access. (Valid for GET processing only.)
V	The column contains an invalid value.
C	Column value changed by another view.
N	Used to set a column to its null value. (INSERT and UPDATE processing only. RDM never returns this.)

There are four ways to use ASIs:

- ◆ When you issue a GET command, certain returned columns may not have a value. You can check this status (on unaltered columns) with the ASI.
- ◆ If you receive an FSI indicating a data error, you can use the ASI to find which columns have illegal values.
- ◆ When a view contains packed or zoned values, the ASIs allow you to avoid unintentional abends. You can do this by examining each ASI for such columns before performing arithmetic or move operations. If the ASI for a column is "V," the value is actually placed in the row even though it is not in a valid format. When a "-" ASI is returned, the field value is a valid zero value for packed, zoned and binary fields. Note that +, - and = are only meaningful on GET processing. Your application program ignores these values on INSERT, DELETE and UPDATE processing. INSERT, DELETE, and UPDATE processing returns ASIs of +, C, or V.
- ◆ For INSERT or UPDATE processing, moving N to the ASI for a column before the function is performed indicates the column is null.

### Validity Status Indicator (VSI)

The Validity Status Indicator (VSI) reflects the validity of the user view row returned by your last RDML request. RDM returns the VSI to the program in an area generated as part of the programmer-supplied TIS-CONTROL statement. The VSI helps you determine if any additional processing of ASIs is needed to correct invalid data or to fill in missing values. VSIs help you determine the most significant ASI returned by RDM, according to the hierarchy indicated in the chart below:

VSI value	Meaning
C	Column value changed by another view.
V	At least one invalid ASI was returned.
-	No invalid ASIs were returned, but at least one missing ASI was returned.
+	No invalid or missing ASIs were returned, but at least one new physical occurrence in the database was returned.
=	No invalid, missing or new physical occurrences were returned by this RDM function.

## Coding the PROCEDURE DIVISION

The PROCEDURE DIVISION of a COBOL program identifies the actions necessary to process, solve, or perform a required function or group of functions. This section presents the requirements and addresses the general guidelines for coding statements in the PROCEDURE DIVISION of a RDM COBOL program.

RDM COBOL PROCEDURE DIVISION programming requirements are as follows:

- ◆ Supplying user name when signing on/off (see “[Signing on/off](#)” on page 85)

General RDM COBOL PROCEDURE DIVISION programming guidelines are addressed as follows:

- ◆ Maintaining storage (see “[Maintaining storage](#)” on page 86)
- ◆ Retrieving rows (see “[Using the GET statement to retrieve rows](#)” on page 86)
- ◆ Modifying rows (see “[Modifying rows](#)” on page 93)
- ◆ Controlling database recovery (see “[Using the COMMIT/RESET statements](#)” on page 95)
- ◆ Handling error conditions. (see “[Handling errors requiring a recompile](#)” on page 96)

## Signing on/off

The only operating system-wide RDM requirement for COBOL applications is that you must supply a user name with the SIGN-ON statement. A password is optional, but you must supply it if you have been assigned a password in the Directory. At run time, RDM checks the Directory for the validity of the user name (and password, if necessary).

The **SIGN-ON** statement establishes communication between a task and the RDM. A task can be a batch job or an online task. When an RDM COBOL program is a subroutine to another RDM task, the task as a whole is already signed on, and an additional sign-on is required only if you are changing the user I.D.

Sign off an RDM COBOL subroutine only if the logical unit of work is complete and will do no further RDM processing before the next **SIGN-ON**.

The **SIGN-OFF** statement tells RDM that you want to terminate your session. RDM will release the storage areas acquired. Issue a SIGN-OFF at the end of every application program.

An example of an RDM COBOL subroutine is an interface to MANTIS. MANTIS using RDM takes care of the **SIGN-ON** (when processing a view statement) and the **SIGN-OFF** (when the task is terminating). If the RDM COBOL interface issues a SIGN-OFF, the logical unit of work may be invalid, and the task as a whole will no longer be in communication with RDM.

RDM supports a CICS pseudoconversational transaction sequence by retaining internal context areas for each transaction executed at the terminal. The application used by the initial transaction must issue an RDM **SIGN-ON**. The applications used by each transaction in the sequence must issue an RDM **COMMIT** instead of an RDM **SIGN-OFF** before releasing control to CICS. Only the initial application in the sequence issues an RDM SIGN-ON; subsequent applications do not. The application used by the final transaction in the sequence should issue an RDM SIGN-OFF.

## Maintaining storage

Use the **RELEASE** and **FORGET** statements to free internal storage without signing off the system. Use **FORGET** to release the storage allocated by a **MARK** statement (see “Using the **MARK** statement to save row position” on page 90).

Use **RELEASE** to close a specific view and free the storage allocated for that one view. If you do this, you lose any **MARKs** associated with that view. The **RELEASE** statement is also useful when you are accessing multiple views and want to remove all **MARKs**. You can use **RELEASE** (without specifying a view name) to close all views and free all allocated storage. If you do this, you remove all **MARKs**, and you lose the current position in all views you are using.

## Using the **GET** statement to retrieve rows

You can retrieve three types of rows using the **GET** statement:

- ◆ Rows containing unique keys
- ◆ Rows containing nonunique keys
- ◆ Rows containing no keys

The **USING** phrase in the **GET** statement indicates which key values to use to access the view. The system goes to the indicated view and retrieves the row for that particular customer.

## Retrieving rows containing unique keys

If the row you want to retrieve has a unique key (for example, ACCOUNT-NUMBER) and your program supplies a value for the unique key, the **GET** command retrieves the specific row having that key. For example:

```
MOVE 71560 TO ACCOUNT-NUMBER
GET ACCOUNT-DATA USING ACCOUNT-NUMBER.
```

RDM retrieves the row in the view, ACCOUNT-DATA, for the ACCOUNT-NUMBER indicated.

You can retrieve in sequential order all user rows with unique keys. The statement GET FIRST instructs the system to retrieve the first row in the user view, for example:

```
GET FIRST ACCOUNT-DATA.
```

The statement GET NEXT retrieves the next row, for example:

```
GET NEXT ACCOUNT-DATA.
```

A GET NEXT statement automatically retrieves the first row in a user view if no current position exists (i.e., no other GET statements have been issued). The GET SAME statement retrieves the same row as accessed on the previous GET statement; GET PRIOR retrieves the previous row; and GET LAST retrieves the last row.

After the last user row has been retrieved, a NOT FOUND condition results. Indicate what should be done in your program:

```
GET NEXT ACCOUNT-DATA
NOT FOUND GO TO STOP.
```

## Retrieving rows containing nonunique keys

You can also retrieve a row with a nonunique key in sequential order. Again, the GET FIRST statement retrieves the first row, and the GET NEXT statement retrieves the next row.

GET NEXT will also automatically retrieve the first row in the view if no other commands have been issued. Keep two considerations in mind if you use the GET NEXT statement to retrieve the first row:

- ◆ GET NEXT operates as GET FIRST if no current position exists.
- ◆ The DBA may define some nonuniquely keyed views without a logical key for performing a direct read to the first row. In this case, the USING phrase is invalid and causes an error. The Relational Data Manager Programmer's Report shows you if no columns can be used as keys.

A NOT FOUND condition results when you reach the end of the view. Supply a NOT FOUND clause on the GET request to tell the system what to do.

Another method for retrieving a nonuniquely keyed user row is to include a USING phrase and key value with your GET command.

```
GET ACCOUNT-TRANS USING ACCOUNT-NO.
```

The remaining rows with the same key can be retrieved with a GET NEXT command that contains a USING phrase.

```
GET NEXT ACCOUNT-TRANS USING ACCOUNT-NO.
```

The NOT FOUND condition appears after the last row with the specified key has been retrieved.

The GET PRIOR, GET LAST, and GET SAME commands also operate on nonuniquely keyed user rows. GET PRIOR retrieves the previous row; GET LAST retrieves the last row; and GET SAME retrieves the same row.

## Retrieving rows without keys

You can also retrieve a row that does not contain a key. For example, by repeatedly issuing the request, GET CUST-INFO, you can retrieve in sequential order every row in the view, CUST-INFO. You can also use GET FIRST, GET NEXT, GET PRIOR, GET LAST and GET SAME to retrieve rows without a key.

## Accessing multiple views

You may want to use more than one view in a program. For example, you may want to code a program which will print a customer name and the name of the part that customer ordered. Assume that you have a customer number and order number, and you want to use the views shown below. The columns in each view are listed below the view name.

CUSTOMER-ORDER-VIEW	CUSTOMER-VIEW	PRODUCT-VIEW
Order Number Customer Number Part Number Quantity Ordered Part Cost Total Cost Ship Date	Customer Number Customer Name Customer Address Customer Telephone	Part-Number Part-Name Part-Cost Quantity in Stock

First, retrieve the CUSTOMER-ORDER-VIEW (using the customer number and the order number as keys) to find the number of the part ordered. Next, retrieve the CUSTOMER-VIEW (using the customer number as a key) to find the customer's name. Finally, using the part number as a key, retrieve the PRODUCT-VIEW to find the name of the part.

```

MOVE 12345 TO CUSTOMER NUMBER.
MOVE 67890 TO ORDER-NUMBER.
GET CUSTOMER-ORDER-VIEW USING CUSTOMER-NUMBER, ORDER-NUMBER.
GET CUSTOMER VIEW USING CUSTOMER-NUMBER.
GET PRODUCT-VIEW USING PART-NUMBER OF CUSTOMER-ORDER-VIEW.

```

## Using the MARK statement to save row position

The **MARK** statement tells RDM to mark the current position of the view established by the previous **GET**, **UPDATE**, or **INSERT**, for example:

```
MARK CUSTOMER-VIEW AT SAVE-LV.
```

The **AT** phrase specifies where the view **MARK** should be saved. You must define the field used, for example, **SAVE-LV**, in your program as a **PICTURE X(4)** field. You may use the **AT** phrase in the **GET** statement to reread the record at the position set by the **MARK** statement:

```
GET CUSTOMER-VIEW AT SAVE-LV.
```

## Using explicit and automatic record holding

With RDM, you can choose explicit or automatic record holding. This decision depends on program requirements and the process you use to modify a row.

### Explicit record holding

To specify explicit record holding, use the FOR UPDATE clause with the **GET** statement, GET FOR UPDATE. Explicit record holding invokes the record holding and enqueueing facilities of the underlying physical data manager and prevents other tasks from modifying the row.

Explicit record holding can cause a number of problems. A row is composed of selected physical fields from many physical files. Holding each of these physical fields can affect views in other tasks, even though the held view does not need the fields the other task wants to modify. Explicit record holding can also tie up resources for long periods and requires elaborate measures to release needed resources.

### Automatic record holding

Automatic record holding allows you to access a row using a **GET** statement, and to update or delete the row without an explicit request to hold the physical records. This allows more efficient processing because the required record is held immediately before the actual database modification takes place.

Modifications that affect your row can be made by other tasks between the time you access the row and the time you update or delete it. To prevent such modifications from being undetected, RDM checks each column value in the row. This ensures that the column value is the original value you retrieved. If any columns have been changed, a data error occurs, you receive a "D" FSI, RDM flags the changed columns with a "C" ASI, and produces a "C" VSI. The column values marked with "C" will not contain the new values but will contain the original values. This allows you to save the column values and retrieve the altered row to resolve the conflict.

## Handling error conditions

When anything other than an FSI value of "\*" is returned, RDM performs an automatic **RESET** and repositions you at the top of the view. (See "RDM status indicators" on page 80 for a list of possible FSI values and their meanings.) For example, if you perform a **GET** and then an **UPDATE** on a read-only view, the UPDATE will fail and RDM will reposition you at the top of the view. The next unqualified GET will return the first row in the view.

To avoid an automatic **RESET**, you need to code an error paragraph containing a NOT FOUND statement. The following example illustrates a sample error-handling paragraph:

```

PROD-TRAN.
  GET PROD FOR UPDATE USING TRAN-PROD
  NOT FOUND DO;
    PUT SKIP EDIT ('PROD-NOT-FOUND') (A);
    CALL ERROR-ON-PROD;
  END;
ERROR-ON-PROD: PROCEDURE;
  IF TIS-FSI='F' THEN
    DISPLAY 'DUE TO A MAJOR PROBLEM ENCOUNTERED
           WHILE ACCESSING THE LOGICAL USER
           VIEW PROD, THIS TASK IS NOW
           SIGNED OFF. ');
  SIGN-OFF;
  STOP;
END;
```

If you do not include an ERROR-ON-PROD paragraph in the program, the RDML compiler would have generated an automatic RESET as follows:

```

ERROR-ON-PROD: PROCEDURE;
  RESET;
END;
```

You can also add phrases (such as DUP KEY, ELSE, NOT FOUND, etc.) to your basic program statements to handle common exception conditions in your paragraph.

When coding an error handling paragraph, the paragraph name should have the format **ERROR-ON-viewname**. The view name must be the same as the name in the 01 level include statement, for example:

```
01 INCLUDE VIEW-NAME
```

If you start your view processing with a GET NEXT (default) followed by a USING phrase, GET NEXT USING KEY1, you have qualified the row, so GET NEXT USING returns a single row with the designated key.

If only one row has the specified logical key, a repeat of the same GET returns a NOT FOUND error. Because an error repositions you at the top of the view, another execution of the GET returns the correct row.

## Modifying rows

The DBA decides upon the modifications you can make to a row. There are three ways to modify a row:

- ◆ Update the data that already exists in the row (UPDATE statement)
- ◆ Delete the row (DELETE statement)
- ◆ Insert a new row (INSERT statement) (see “Using the INSERT statement” on page 94)

Issue a **COMMIT** command after each logical transaction (which may involve more than one change) to establish the modifications in the database.

## Updating rows

The **UPDATE** statement allows you to modify column contents. Before performing UPDATE, you must access the view by using a **GET** statement, for example:

```
GET ACCOUNT-DATA USING KEY1
UPDATE ACCOUNT DATA.
```

You cannot modify a view key using the UPDATE command. RDM does not permit replacing a view key because you need the view key to locate the view row to be replaced. To change a view key, first **DELETE** the old row, then **INSERT** a new one.

## Deleting rows

The **DELETE** statement removes a row from the system. Before performing DELETE, you must access the view by using a **GET** statement, for example:

```
GET SAMPLE-VIEW USING KEY1
DELETE SAMPLE-VIEW.
```

This example deletes the one occurrence of SAMPLE-VIEW obtained, based on the value of KEY1.

The phrase DELETE ALL deletes all rows that would have been retrieved by a GET FIRST followed by GET NEXTs using the parameters of the GET statement just prior to the **DELETE**. In other words, the DELETE ALL will delete all rows that depend on the key value specified on the latest **GET**:

```
GET SAMPLE-VIEW USING KEY1
DELETE ALL SAMPLE-VIEW.
```

This deletes all rows with the key value specified.

## Using the INSERT statement

The **INSERT** statement adds a new user row to the database.

```
INSERT ACCOUNT-DATA.
```

If you are inserting a user row in nonuniquely keyed rows, you can control the placement of the new row within the set of rows with the same key value. You cannot determine the location if the DBA has already defined an order for the view. The phrases NEXT, FIRST, LAST, or PRIOR may be added to the INSERT command. For example, INSERT NEXT ACCOUNT-DATA instructs the system to insert the new row after the current row (the last row accessed) in the view.

If the view is uniquely keyed, order is already determined. If the value of the keys to be inserted already exists, the DUP KEY condition results and RDM performs the action specified on the DUP KEY phrase in the **INSERT** statement.

```
INSERT ACCOUNT-DATA  
DUP KEY GO TO ALREADY-THERE.
```

Some constraints apply when inserting information. You can always add a new entity (a customer), assuming you have space on the database. Typically, you cannot add a new relationship until all the entities being related exist. You cannot add a relationship between an employee and a department until you have added the department and employment entities.

However, you can add an entity and a relationship in one operation. For example, you can add a new employee and his first department assignment in a single **INSERT** request, provided the DBA has allowed this operation.

## Using the COMMIT/RESET statements

The **COMMIT** statement makes the changes to the database (**INSERT**, **DELETE**, **UPDATE**) permanent. The **RESET** statement instructs the system to perform the standard error recovery procedure for dealing with the previous RDML request, (to undo all database changes made by this task since the last COMMIT).

When Task Level Recovery (TLR) is active, the COMMIT statement sends all pending updates to disk. A RESET backs out any database updates since the last COMMIT and continues processing from the RESET.

If TLR is not active, a RESET statement prints an error message in the task. The task abend is intentional, and the system prints messages on the job log indicating the last function statement issued prior to the RESET. Normally, standard database recovery procedures are performed, depending on the physical data manager being used.

In the CICS environment, RDM COMMIT/RESET logic works according to Dynamic Transaction Backout (DTB) processing. A COMMIT makes all updates permanent to the database and takes a CICS syncpoint. A RESET backs out any database updates since the last COMMIT and continues processing from the RESET.

Under CICS DTB, a rollback is performed. If you encounter an error condition in an online environment, you can back out of the modification by using the RESET function. This erases all modifications issued since the last COMMIT command.

For more information about DTB processing, refer to the *SUPRA Server PDM CICS Connector Systems Programmer's Guide (OS/390 & VSE)*, P26-7452.

## Handling errors requiring a recompile

RDM has several checks to ensure that the program you are running is current and that the user view it uses is the same as other applications in the system. When an RDML command is issued in an application program, RDM checks to see if the columns in the view, as defined in the Directory, are the same as when the program was last compiled. If not, an FSI status code is returned and the program must be recompiled and relinked.

Changes requiring a recompile are:

- ◆ Data type change (packed to zoned decimal, etc.).
- ◆ Deleted column (if the column is not part of your user view, you need not recompile).
- ◆ Column length change.
- ◆ A change in the number of decimal places.

Application systems are often composed of several separately compiled programs that depend on common definitions of data items. These programs call each other to perform special tasks. RDM checks on each RDML call to make sure that the definition of the user view is the same for each program. If you compile a program or subroutine with the same user view name as another program or subroutine and the user view definition does not match, RDM generates an error message. The data used to perform this error checking is contained in the field list generated at compile time by the RDML compiler.

For information on executing the RDML compiler, see [“Compiling and linking an RDM COBOL application program”](#) on page 123.

# 4

## RDM COBOL application program statements

This chapter contains format descriptions and usage considerations for the two groups of RDM COBOL program statements:

- ◆ DATA DIVISION statements
- ◆ PROCEDURE DIVISION statements

Some RDM COBOL program statements have specific underlying file system restrictions. For more information on the restrictions for PDM file systems, refer to the *SUPRA Server PDM RDM PDM Support Supplement (OS/390 & VSE)*, P26-8221. For information on VSAM restrictions, refer to the *SUPRA Server PDM RDM VSAM Support Supplement (OS/390 & VSE)*, P26-8222.

## DATA DIVISION statements

The DATA DIVISION of a COBOL program describes the format and characteristics of data in an application program. You can code the following statements in either the WORKING-STORAGE SECTION or the LINKAGE SECTION of your program if using OS/VS Batch COBOL or CICS Command Level COBOL.

### INCLUDE view-data

The INCLUDE statement indicates the views needed for your program and where (in the DATA DIVISION) to place them.

---

***level-number***[***user-view-name***]

**INCLUDE *view-name* [(*user-column-list*)].**

---

---

#### ***level-number***

**Description**     *Required*

**Options**         01–29

---

#### ***user-view-name***

**Description**     *Optional.* Assigns a name to the user view.

**Format**            Must follow COBOL data-item naming standards.

---

#### ***view-name***

**Description**     *Required.* Indicates the view you want to use.

**Format**            Must be a valid view name.

---

**(user-column-list)**

**Description**     *Optional.* Indicates the columns from the particular view you want to use.

**Format**             A list of columns in the specified view separated by commas and all enclosed by parentheses. The list may be spread over several lines, if necessary.

**Considerations**

- ◆ If you do not include required columns in the view, you cannot perform **INSERT**s and some **UPDATE**s on the view.
- ◆ Modifying key order in your user view could adversely affect performance. The DBA has defined the key order on the Directory to maximize performance.

**General considerations**

- ◆ Code an **INCLUDE** statement in the **DATA DIVISION** of your **COBOL** program to identify the view you want to use.
- ◆ Code an **INCLUDE** statement in the **LINKAGE SECTION** of your **COBOL** subroutines when user rows are being passed as parameters.
- ◆ Using column lists can enhance the performance of your application because **RDM** optimizes the physical accesses required based on the list of columns in the **INCLUDE** statement.
- ◆ Each user view acts independently of all other user views you include, even if the user views come from the same view.

**Examples**

The following examples will generate data definitions that are placed in the WORKING-STORAGE SECTION:

```

*
*01 CUSTOMER INCLUDE CUST (CUST-NO,NAME,CITY).
01 RDM-CUSTOMER.
10 CUSTOMER.
20 CUST-NO PIC S9(05).
20 NAME PIC X(040).
20 CITY PIC X(020).
10 ASI-CUSTOMER.
20 ASI-CUST-NO PIC X.
20 ASI-NAME PIC X.
20 ASI-CITY PIC X.
*
*01 INCLUDE CUS-PRD (CUST-NO,PROD-NO).
01 RDM-CUS-PRD.
10 CUS-PRD.
20 CUST-NO PIC S9(05).
20 PROD-NO PIC X(004).
10 ASI-CUS-PRD.
20 ASI-CUST-NO PIC X.
20 ASI-PROD-NO PIC X.
*
*01 CONTACT INCLUDE PROD.
01 RDM-CONTACT.
10 CONTACT.
20 PROD-NO PIC X(004).
20 PROD-DESC PIC X(040).
20 PROD-RENT PIC S9(07)V9(02) USAGE COMP-3.
20 PROD-MAINT PIC S9(07)V9(02) USAGE
COMP-3.
20 PROD-PURCH PIC S9(07)V9(02) USAGE COMP-3.
10 ASI-CONTACT.
20 ASI-PROD-NO PIC X.
20 ASI-PROD-DESC PIC X.
20 ASI-PROD-RENT PIC X.
20 ASI-PROD-MAINT PIC X.
20 ASI-PROD-PURCH PIC X.

```

## INCLUDE TIS-CONTROL

Use the INCLUDE TIS-CONTROL statement to include the special view, TIS-CONTROL, in a program.

---

***level-number* INCLUDE TIS-CONTROL.**

---

---

### ***level-number***

**Description**     *Required.*

**Options**         01–29

### **General considerations**

- ◆ You must include the TIS-CONTROL view in each program issuing an RDML request. Subroutines which are passed views, but which perform no access themselves, do not need this special view.
- ◆ Code an INCLUDE TIS-CONTROL statement in the LINKAGE SECTION of your application program if it is being passed from a calling module.
- ◆ The level-number must be between 01 and 29.
- ◆ Use the TIS-OPTIONS field to specify DEBUG and TRACE; code as follows:

```
DEBUG - DEBUG is on
NBUG  - DEBUG is off
TRAC  - TRACE is on
```

See “[Specifying TIS-CONTROL](#)” on page 79 for instructions on coding DEBUG and TRACE in your application program.

**Example** To add the special view TIS-CONTROL to your program, code the following statement:

```
01 INCLUDE TIS-CONTROL.
```

**Example output**

```
*
*01 INCLUDE TIS-CONTROL.
01 TIS-CONTROL.
    10 TIS-OBJECT-NAME          PIC X(30).
    10 TIS-OPERATION.
        20 TIS-ID                PIC X(2).
        20 TIS-OPCODE            PIC X.
        20 TIS-POSITION          PIC X.
        20 TIS-MODE              PIC X.
        20 TIS-KEYS              PIC X.
    10 TIS-FSI                  PIC X.
    10 TIS-VSI                  PIC X.
    10 FILLER                    PIC X(2).
    10 TIS-MESSAGE              PIC X(40).
    10 TIS-PASSWORD              PIC X(8).
    10 TIS-OPTIONS               PIC X(4).
    10 TIS-CONTEXT               PIC X(4).
    10 TIS-LVCONTEXT             PIC X(4).
```

---

## PROCEDURE DIVISION statements

This section presents alphabetically the RDML PROCEDURE DIVISION statements. Examples accompany each statement. Some examples also show the expanded code generated by the RDML Compiler.

When coding batch COBOL applications, it may be advisable to code an RDM command within an IF statement or a nested IF statement. Follow these rules for coding RDM commands within an IF statement:

- ◆ The expansion of an RDM command generates an imbedded IF statement to determine if an error has occurred on this RDM request. The generated IF statement ends with an explicit period ('.').
- ◆ Due to the above rule, if you include the RDM command in an IF statement, it must be the last statement coded in the IF statement.
- ◆ To issue multiple RDM commands within the scope of one IF statement, you can do either:
  - Place each RDM command in its own paragraph or section and perform each command.
  - Place each group of RDM commands in its own section or paragraph and perform the entire section or paragraph.

## COMMIT

The COMMIT statement issues a COMMIT to the underlying physical data manager in those environments where TLR or DTB is supported.

---

### COMMIT.

---

#### General considerations

- ◆ In those environments where TLR is supported, the COMMIT statement either returns a successful or restart status. In other environments, the COMMIT statement always returns a successful status.
- ◆ In a CICS DTB environment, a CICS syncpoint function is performed.
- ◆ To maintain view context in the CICS pseudoconversational mode, issue a COMMIT instead of a **SIGN-OFF** before task termination. The next program executed from the same terminal can continue to use RDM as if the task had not terminated.

#### Example

The COMMIT statement identifies the recovery point of the task which precedes it.

```
COMMIT .  
.  
.  
.  
.
```

## DELETE

The DELETE statement removes a row from the database.

---

**DELETE [ALL] *view-name*.**

---

---

### ALL

**Description** *Optional.* Deletes all view rows that depend on the logical keys specified by the previous **GET** for this view.

**Consideration** This statement uses the parameters of the GET statement issued just prior to the DELETE.

---

### *view-name*

**Description** *Required.* Specifies the view you want to use in your deletion.

**Format** Must be a valid, open view.

## General considerations

- ◆ The DELETE statement removes an entire row.
- ◆ DELETES will not be performed if data integrity will be compromised. In other words, a customer record cannot be deleted until all outstanding orders for that customer have been deleted.
- ◆ An "X" failure status from a DELETE request must be followed by a RESET to ensure database integrity. If you provide an error-handling paragraph which does not RESET following an "X" status on DELETE, it is possible that part of the modification will be done and part not done.
- ◆ The DELETE ALL command deletes all rows in a view if the program specifies a GET without a qualifying USING phrase.
- ◆ The RDML Precompiler program recognizes DELETE as a view request on two occasions: 1) when DELETE is followed by a valid view name, and 2) when DELETE is followed by ALL.
- ◆ When DELETE is followed by ALL, the specified view is either valid or invalid. If valid, the preprocessor generates code to handle the request. If invalid, the preprocessor generates an error message in the first listing, and the DELETE statement is commented out to the COBOL compiler.
- ◆ If you use any other DELETE statement, the preprocessor issues a warning message that the statement was skipped and was assumed to be a COBOL statement. The DELETE statement is given to the COBOL compiler as is, and no code is generated to handle the statement because it was not considered a legal view request.
- ◆ The DBA may disallow DELETES.

## Examples

- ◆ The following example deletes the one occurrence of SAMPLE-VIEW based on the value of KEY1:

```
GET SAMPLE-VIEW USING KEY1.  
DELETE SAMPLE-VIEW.
```

- ◆ This example deletes all user view rows that depend on the value in KEY1:

```
GET SAMPLE-VIEW USING KEY1.  
DELETE ALL SAMPLE-VIEW.
```

- ◆ This has the same effect as the following set of statements:

```
GET FIRST SAMPLE-VIEW USING KEY1.  
MORE.  
DELETE SAMPLE-VIEW.  
GET NEXT SAMPLE-VIEW USING KEY1.  
NOT FOUND GOTO DONE.  
GOTO MORE.  
DONE.
```

## FORGET

The FORGET statement frees the storage allocated by a previously issued **MARK** statement.

---

**FORGET** *data-item* [**NOT FOUND** *cobol-imperative-statement*]  
**[ELSE** *cobol-imperative-statement*].

---

### *data-item*

**Description** *Required.* Specifies what **MARK** information should be forgotten.

**Format** Must follow COBOL data-item naming standards.

### Considerations

- ◆ The data-item field must be a PIC X(4) data-item.
- ◆ You must define the data-item field in the DATA DIVISION of the program, and the field must contain information passed back by a previously issued **MARK** statement.

---

### **NOT FOUND** *cobol-imperative-statement*

**Description** *Optional.* Indicates what should be done if the mark information cannot be released.

**Considerations** RDM may not find a mark value if one of the following conditions is true:

- ◆ The mark has previously been forgotten by another FORGET statement or by a **RELEASE** statement.
- ◆ The data-item was never marked by a **MARK** statement.
- ◆ The marked data-item was somehow changed or moved.

---

### **ELSE** *cobol-imperative-statement*

**Description** *Optional.* Indicates what to do if the mark information release is done.

**Consideration** The program falls through to the next statement if you do not specify an ELSE clause.

### General considerations

- ◆ Once you have issued a FORGET statement, the indicated mark is released and cannot be regained without issuing a new **MARK** statement.
- ◆ After a successful FORGET, set the data-item field to spaces.

## GET

The GET statement identifies the row to be retrieved from the view indicated.

---

```

GET 
  NEXT
  LAST
  SAME
  FIRST
  PRIOR
 view-name [FOR UPDATE USING] 
  data-item1[...data-item9]
  AT data - item

[NOT FOUND cobol - imperative - statement] [ELSE cobol - imperative - statement]

```

---

NEXT
LAST
SAME
FIRST
PRIOR

<b>Description</b>	<i>Optional.</i> Indicates the order of row retrieval.
<b>Default</b>	NEXT
<b>Options</b>	<p>GET NEXT retrieves the next row with the specified keys. If no keys are supplied, the next sequential row is returned. If no current row exists, GET NEXT operates as GET FIRST.</p> <p>GET LAST retrieves the last row in the view with the specified keys. If no keys are given, RDM returns the last row.</p> <p>GET SAME retrieves the row just accessed, if a current row exists. If no current row exists, a NOT FOUND condition is signaled.</p> <p>GET FIRST retrieves the first row in the view with the specified keys. If no keys are given, RDM returns the first row.</p> <p>GET PRIOR retrieves the previous row with the specified keys. If no current row exists, GET PRIOR operates as GET LAST.</p>

### Considerations

- ◆ If the underlying file system cannot perform the GET PRIOR or GET LAST functions, an error results.
- ◆ A series of GET NEXTs loops back to the first row and continues if the statement has no NOT FOUND.
- ◆ A GET PRIOR view without a USING phrase returns a row if there is a currently established position for a given key in a row. However, after processing all prior rows for the key, RDM returns the message: "PDM DOES NOT SUPPORT THIS OPERATION."

---

### *view-name*

**Description**     *Required.* Specifies the name of the view you want to use.

**Format**             Must be a valid *view-name*.

---

### FOR UPDATE

**Description**     *Optional.* Allows you to lock out other users' modifications to the rows you are retrieving.

### Considerations

- ◆ The FOR UPDATE phrase allows you to perform modifications that depend on the current contents of the row.
- ◆ If you do not need to be certain of the content of the row, you can use a GET without the FOR UPDATE phrase. When RDM performs the UPDATE or DELETE function, the "automatic hold" facility performs the lock before modifying the row.
- ◆ Using FOR UPDATE may decrease overall system performance. If any column values have changed before performing a DELETE or UPDATE, the function produces a data error ("D" FSI) and flags the changed columns with a "C" ASI.

---

**USING *data-item1*[...*data-item9*]**

**Description**     *Optional.* Specifies the key values to use in accessing the view.

**Format**             The data items must be part of a valid view

**Considerations**

- ◆ The number of keys specified in the GET statement must be less than or equal to the number of keys in your specified logical view.
- ◆ Any omitted keys are treated as generic keys. The use of generic keys is a convenient feature for allowing both direct access to a row and a sequential scan of many rows. RDM returns all occurrences of a particular unspecified column if the other keys are satisfied.
- ◆ The order of specified keys in the USING phrase must correspond to the order of key declarations (left to right) in your Programmer's Report or your user view (see the INCLUDE statement, "**INCLUDE view-data**" on page 98). You cannot omit a key that occurs between two keys you want to specify. (For example, you cannot include KEY1 and KEY3 without including KEY2.)
- ◆ The USING phrase cannot be used with a GET SAME statement or with an AT phrase.
- ◆ If there is only one row for a given key and you try to use the same key with a GET USING statement to access the row a second time, you receive an "OCCURRENCE NOT FOUND" message. This message indicates there are no more occurrences with this particular logical key specification. In order to access this same row most efficiently, use a GET SAME statement instead.
- ◆ The logical key can use up to nine data items.

### **AT data-item**

**Description** *Optional.* Repositions a view based on the mark obtained by a previous MARK statement.

**Format** Must be part of a character field of length 4 (PIC X(4)).

#### **Considerations**

- ◆ The data-item is a storage location that contains information generated by a previous MARK statement.
- ◆ The USING and AT phrases cannot be used in the same GET statement.
- ◆ The AT phrase cannot be specified in a statement using the FIRST, NEXT, PRIOR, LAST, or SAME positional qualifiers.

---

### **NOT FOUND cobol-imperative-statement**

**Description** *Optional.* Indicates what RDM is to do if it finds no data.

**Considerations** Data may not be found due to one or more of the following reasons:

- ◆ No data is available for a keyed GET.
- ◆ All the existing data is exhausted for a generic GET.
- ◆ All the data available to the user view is exhausted for a nonkeyed GET.
- ◆ A series of GET NEXTs loops back to the first row and continues if the program does not check for a NOT FOUND.

---

### **ELSE cobol-imperative-statement**

**Description** *Optional.* Indicates what RDM is to do if good data is found.

**Consideration** The program falls through to the next statement if you do not specify an ELSE clause.

## Examples

- ◆ The following statement retrieves the first row in the view PROD that matches the supplied key value. The column PROD-TRAN contains the key value used for retrieving the row.

```
GET PROD USING PROD-TRAN.
```

- ◆ This statement retrieves the view using the KEY PROD-TRAN. The USING phrase indicates the key to use for retrieving the view.

```
GET PROD FOR UPDATE USING PROD-TRAN.
```

- ◆ This statement retrieves a view that you marked and saved for later access.

```
GET PROD AT PROD-MARK.
```

- ◆ Repeatedly issuing this request retrieves all PROD rows in the view.

```
GET PROD.
.
.
.
```

- ◆ This request retrieves the row for update.

```
GET PROD FOR UPDATE.
```

- ◆ The following statements retrieve rows in the specified order, i.e., NEXT, LAST, SAME, FIRST, PRIOR:

```
GET NEXT PROD.
.
.
.
GET LAST PROD.
.
.
.
GET SAME PROD.
.
.
.
GET FIRST PROD.
.
.
.
GET PRIOR PROD.
.
.
.
```

## INSERT

The INSERT statement inserts a new row into the view.

---

```

INSERT [
  NEXT
  LAST
  FIRST
  PRIOR
] view-name [DUP KEY cobol-imperative-statement]
    
```

---

**NEXT**  
**LAST**  
**FIRST**  
**PRIOR**

**Description** *Optional.* Specifies where to insert the row relative to its current position.

**Default** NEXT

**Options**

INSERT NEXT places the row after the current row, provided the keys are the same. If the keys are different or if no current row exists, INSERT NEXT operates as INSERT LAST.

INSERT LAST places the row into the view so that a subsequent GET LAST command using the same key values retrieves it.

INSERT FIRST places the row in the view so that subsequent GET FIRST commands using the same key values retrieve it.

INSERT PRIOR places the row in the view before the current row, provided the keys are the same. If the key values are different or if there is no current row, INSERT PRIOR operates as INSERT FIRST.

### Considerations

- ◆ If the DBA specified ordering in the view definition, or if the physical data manager does not allow program control of ordering, the specification on the INSERT statement is ignored.
- ◆ To ensure data integrity, an "X" failure status from an INSERT request must be followed by a **RESET**. If you provide an error-handling paragraph which does not RESET following an "X" status on INSERT, it is possible that part of the modification will be done and part not done.

---

**view-name**

<b>Description</b>	<i>Required.</i> Specifies the name of the view into which you want the rows inserted.
<b>Format</b>	Must be a valid, open view.

---

**DUP KEY *cobol-imperative-statement***

<b>Description</b>	<i>Optional.</i> Indicates what RDM should do if the row to be inserted is uniquely keyed, and if the value of the keys to be inserted already exists in the database.
--------------------	--

**General considerations**

- ◆ The DBA and/or the physical data manager being used may disallow ordering.
- ◆ For the INSERT to be successful, you must supply all keys and required columns, and they must be valid and non-null.
- ◆ Your application program can update a column with a null value by changing the ASI to "N" or by supplying the null value in the column.

**Examples** The following examples show various ordering possibilities available for use with the INSERT statement:

```

INSERT NEXT PROD.
      .
      .
      .
INSERT LAST PROD.
      .
      .
      .
INSERT FIRST PROD.
      .
      .
      .
INSERT PRIOR PROD.
      .
      .
      .

```

## MARK

The MARK statement records the current position of the view established by the last **GET**, **UPDATE**, or **INSERT** statement.

---

### **MARK** *view-name* **AT** *data-item*

---

---

#### *view-name*

**Description** *Required.* Indicates the view you want to mark.

**Format** Must be a valid, open view.

---

#### **AT** *data-item*

**Description** *Required.* Specifies where to save the MARK information.

**Format** Must be a PIC X(4) data-item

**Consideration** Cincom recommends that you define this column in the DATA DIVISION of the program and initialize it to the value spaces prior to use.

#### **General considerations**

- ◆ The AT phrase in the **GET** statement (see “**GET**” on page 109) is used to reposition the view at the position set by the MARK statement.
- ◆ You can create any number of MARKs for a view, but to conserve internal memory space, it is best to reuse MARKs or to **FORGET** them whenever possible. Each MARK requires its own data item in the DATA DIVISION.
- ◆ The number of MARKs that a program can have outstanding at any time is limited by the size of the available storage. When the program no longer requires a particular MARK, issue a FORGET command for the data-item.

**Example**

In this example the current position of the user view PROD is marked and saved at PROD-MARK:

```
WORKING-STORAGE SECTION.  
.  
.  
.  
01 PROD-MARK PIC X(4).  
.  
.  
.  
PROCEDURE DIVISION.  
.  
.  
.  
MARK PROD AT PROD-MARK.  
.  
.  
.  
GET PROD AT PROD-MARK.  
.  
.  
.
```

## RELEASE

The RELEASE statement closes a specific view or all open views, and frees internal storage space allocated to the RDML processor.

---

**RELEASE** [*view-name*].

---

---

### *view-name*

**Description** *Optional.* Specifies the view to be released.

**Format** Must be a valid, open view.

**Consideration** If you omit this parameter, the RELEASE statement releases all of your open views.

### General considerations

- ◆ The RELEASE statement is helpful when you are accessing multiple views. However, if you issue it without a *view-name*, RELEASE removes all MARKs (see “MARK” on page 116) and loses the current position in all views being used.
- ◆ The RDML Precompiler program only recognizes a RELEASE request as valid for a view if a period follows the statement. If the COBOL preprocessor finds anything other than a period, it issues a warning message saying that it skipped the statement and assumed it to be a COBOL statement.
- ◆ If you issue RELEASE without a *view-name*, reset the MARK fields in the application to spaces.

### Example

```
RELEASE .  
.  
.  
.
```

## RESET

The RESET statement attempts to undo any update, delete, or insert requests issued since the last COMMIT.

---

### RESET.

---

#### General considerations

- ◆ If you supply no error handling paragraph, the COBOL preprocessor generates an error routine that issues a RESET request to RDM.
- ◆ In a non-TLR batch program, this operation prints an error message and the task abends.
- ◆ The RDML Precompiler program recognizes a RESET request as valid only if it is followed by a period. If it finds anything other than a period, the preprocessor returns a warning message that the statement was skipped and was assumed to be a COBOL statement.
- ◆ In the CICS DTB environment, a CICS rollback function is performed.
- ◆ The RESET command restores your database to the last COMMIT point and you lose position on all views. Therefore, the GET SAME, DELETE, or UPDATE commands are not valid after a RESET. A GET NEXT command positions you on the first row while a GET PRIOR command positions you on the last row after a RESET.

#### Example

In this example, you indicate a reset:

```
RESET.
```

```
.  
. .  
. . .
```

## SIGN-OFF

The SIGN-OFF statement informs RDM that access to the system is no longer desired.

---

### SIGN-OFF.

---

#### General considerations

- ◆ The SIGN-OFF statement also releases all storage areas that were acquired to service RDML requests.
- ◆ Issue a SIGN-OFF at the end of every application program unless it is a CICS pseudoconversational application program. A CICS pseudoconversational application program transfers its RDM context to the next program run from the same terminal. Use a COMMIT, instead of a SIGN-OFF, for pseudoconversational operation.
- ◆ SIGN-OFF also causes a COMMIT.

#### Example

In this example, USER1 signs off the system:

```
.  
. .  
. .  
SIGN-OFF.
```

## SIGN-ON

The SIGN-ON statement identifies the user to RDM.

---

**SIGN-ON** *user-name* [*password*]

---



---

### *user-name*

**Description** *Required.* Indicates user's name.

**Format** Must be assigned in the Directory.

**Consideration** The user-name must be a COBOL data item name and not a literal.

---

### *password*

**Description** *Optional.* Indicates the user's password. The password is required if the user has an assigned password in the Directory.

**Format** Must be assigned in the Directory.

**Consideration** If a password is specified, it must be a COBOL data item name and not a literal.

### General consideration

A SIGN-ON request implicitly issues a **RELEASE** request and frees any previously allocated storage space.

**Example** In this example, SST signs on to the system:

```
DATA DIVISION
  01  USER-NAME  PIC X(3) VALUE 'SST'
      .
      .
      .
PROCEDURE DIVISION
  SIGN-ON USER-NAME
```

## UPDATE

The UPDATE statement updates column values in the database.

---

**UPDATE *view-name*.**

---

---

### *view-name*

**Description**     *Required.* Indicates view name you want to update.

**Format**            Must be a valid, open view

### General considerations

- ◆ Before performing an UPDATE, you must access the view using a GET statement.
- ◆ You can use the GET FOR UPDATE phrase before you use the UPDATE function when computing a new value for a row (incrementing a counter, etc.). If you are using the UPDATE function to place a value in a row, you need not issue a GET FOR UPDATE statement which is not dependent on the values already present.
- ◆ You cannot update a view key. By altering the view key, you are requesting a repositioning of the view, not a modification of the current row. To "update" a view key, you must first delete the old row, and then insert a new one.
- ◆ To ensure database integrity, a **RESET** must follow an "X" failure status from an UPDATE request. If you provide an error-handling paragraph that does not RESET following an "X" status on **DELETE**, only part of the modification may be done.
- ◆ The DBA may disallow updates.
- ◆ Your application program can update a column with a null value by changing the ASI to "N" or by supplying the null value in the column.

### Example

The statement UPDATE PROD indicates that you want to update the view PRODUCT:

```
GET PROD USING ---  
MOVE NEW-DATA TO PRODUCT-FIELD  
UPDATE PROD.  
.  
.  
.
```

# 5

## Compiling and linking an RDM COBOL application program

This chapter presents information on the RDML Precompiler, including instructions for executing the precompiler and linking considerations for each operating system.

The RDML Precompiler converts RDML statements into standard COBOL source code by using information on the Directory to generate and use data definitions. The standard COBOL compiler then converts the COBOL source code into object code. When the program executes, it issues RDML commands. RDM processes these commands by issuing commands to the underlying physical data manager (e.g., the SUPRA PDM). RDM then presents the results of these commands to the program in the form of rows of views.

See “[OS/390 and VSE samples and procedures](#)” on page 125 for the samples and procedures for executing the RDML Precompiler in your operating environment.

---

## Executing the RDML precompiler

The RDML Precompiler generates COBOL source statements accepted by the standard IBM COBOL compilers. You must use the APOST compile option (strings delimited by apostrophes, not quotes). If this is not the default at your installation, specify the options as follows:

- ◆ **OS/390**—Specify DELIM=APOST in the JCL PARM options for the COBOL compile step.
- ◆ **VSE**—Specify CBL APOST as the first source statement in the preprocess input.



---

Cincom also recommends you stipulate the NOSEQ option to avoid meaningless warning messages from the COBOL compiler.

---

The parameters for precompiling an RDM COBOL application program are as follows:

**OS/390**

---

```
omit PARM altogether for batch, COBOL
PARM=',COBOL-II' for batch, COBOL2
PARM='DFH' for CICS, COBOL
PARM='DFH,COBOL-II' for CICS, COBOL2
```

---

**VSE**

---

```
PARM='FCOBOL' for batch, COBOL
PARM='FCOBOL,COBOL-II' for batch, COBOL2
PARM='DFH' for CICS, COBOL
PARM='DFH,COBOL-II' for CICS, COBOL2
```

---

---

## Linking a compiled program

The following operating system dependent considerations apply when you are linking a compiled application program:

**OS/390**

---

In batch OS/390, COBOL applications are linked with CSVILUV.

---

In OS/390 CICS, COBOL applications are linked with CSVNICIC.

---

**VSE**

---

In batch VSE, COBOL applications are linked with CSVIOSVS.

---

In VSE CICS, COBOL applications are linked with CSVNICIC.

---

# A

## OS/390 and VSE samples and procedures

This appendix presents the samples and procedures for running the following tasks in OS/390 or VSE environments:

- ◆ RDML Precompiler
- ◆ Runtime support
- ◆ Batch DBAID
- ◆ Batch reports

### OS/390 samples and procedures

Description	Single-task		Central	
	Sample	Procedure	Sample	Procedure
Batch COBOL RDML Precompiler and COBOL compile		TISCOBBL		TISCOBCL
CICS COBOL RDML Precompiler and COBOL compile		TISCCBBL		TISCCBCL
Batch RDM COBOL Runtime Support		TISCGOBL		TISCGOCL
Batch DBAID	TXJBDAID	TISAIDBL	TXJCDAID	TISAIDCL
Batch RDM Impact of Change Report	TXJICRPT	TISICRBL		TISICRCL
Batch RDM Reports	TXJREPRT	TISRPTBL		TISRPTCL

## VSE samples

Description	Single-task	Central
RDML COBOL Precompiler	TXJCOBPP	
COBOL compile and link of precompiled RDML batch COBOL applications	TXJCOBCL	
CICS precompile COBOL compile and link of precompiled RDML CICS COBOL applications	TXJCOBCI	
Execute batch RDM COBOL applications	TXJCOBGO	
Batch DBAID	TXJBDAID	TXJCDAID
Batch RDM Impact of Change Report	TXJICRPT	
Batch RDM Reports	TXJREPRT	

# B

## Sample RDM COBOL application program

```
*****
* EXECUTES THE COMPLETE SET OF RDML STATEMENTS ACCESSING THROUGH
* THE PROD, PROD WITH A FIELD LIST, INVC AND BRANCH-CITY-STATE
* LOGICAL VIEWS
*****
*
IDENTIFICATION DIVISION.                                LVC00020
PROGRAM-ID. COBL0019.                                  LVC00030
ENVIRONMENT DIVISION.                                  LVC00060
INPUT-OUTPUT SECTION.                                  LVC00070
FILE-CONTROL.                                          LVC00080
    SELECT TRAN-IN ASSIGN TO UT-S-CRDFILE.             LVC00090
    SELECT PRINT-FILE ASSIGN TO UT-S-PRINTOUT.         LVC00100
                                                        LVC00110
DATA DIVISION.                                         LVC00120
FILE SECTION.                                          LVC00130
FD  TRAN-IN                                            LVC00140
    LABEL RECORDS ARE OMITTED                          LVC00150
    RECORDING MODE IS F                                LVC00160
    DATA RECORD IS TRAN-RECORD.                        LVC00170
                                                        LVC00180
```

01	TRAN-RECORD.	LVC00190
05	TRAN-ALL PIC X(80).	LVC00200
05	TRAN-SUB REDEFINES TRAN-ALL.	LVC00210
*	Q - QUIT, D - DEBUG, T - TRACE, N - NDBG	LVC00220
10	OPTIONS PIC X.	LVC00230
*	G- GET, U- UPDATE, I- INSERT, D- DELETE, M- MARK	LVC00240
*	C- COMMIT, A- RESET, R- RELEASE, S- SIGN-ON, OR F- SIGN-OFF	LVC00250
10	OPCODE PIC X.	LVC00260
*	F- FIRST, N- NEXT, P- PRIOR, L- LAST, A- AT, S- SAME	LVC00270
10	POSIT PIC X.	LVC00280
*	U- UPDATE, R- RELEASE	LVC00290
10	HOLD PIC X.	LVC00300
*	0 THROUGH 9 OR MARK NUMBER IF 'GA' OR 'M' COMMANDS USED	LVC00310
10	KEYS PIC X.	LVC00320
*	EITHER A VIEW NAME OR A MARK NAME.	LVC00330
10	OBJECT-NAME PIC X(30).	LVC00340
		LVC00350
05	TIS-SIGN-ON REDEFINES TRAN-ALL.	LVC00360
10	TIS-USER-ID PIC X(30).	LVC00370
10	TIS-PASS PIC X(8).	LVC00380
		LVC00390
05	PROD-DATA REDEFINES TRAN-ALL.	LVC00400
10	INP-PROD-NO PIC X(4).	LVC00410
10	INP-PROD-DESC PIC X(40).	LVC00420
10	INP-PROD-RENT PIC 9999999V99.	LVC00430
10	INP-PROD-MAINT PIC 9999999V99.	LVC00440
10	INP-PROD-PURCH PIC 9999999V99.	LVC00450
		LVC00460
FD	PRINT-FILE	LVC00860
	RECORDING MODE IS F	LVC00870
	LABEL RECORDS ARE OMITTED	LVC00880
	DATA RECORD IS PRINT-REC.	LVC00890
01	PRINT-REC PIC X(133).	LVC00900
		LVC00910

```

WORKING-STORAGE SECTION.                                LVC00920
                                                         LVC00930
***** LVC00940
*           LOGICAL VIEW DEFINITIONS                    * LVC00950
***** LVC00960
01  INCLUDE TIS-CONTROL.                                LVC00970
01  INCLUDE BRANCH-CITY-STATE.                          LVC01010
01  INCLUDE INVC.                                       LVC01010
01  INCLUDE PROD.                                       LVC01010
***** LVC01100
*           MARK VALUE DEFINITIONS                      * LVC01110
***** LVC01120
01  MARKS.                                              LVC01130
    05  MARK0           PIC X(4) VALUE SPACES.          LVC01140
                                                         LVC01240
01  TOP-OF-FORM      PIC X VALUE '1'.                    LVC01250
01  CNT-PAGE         PIC 9(2) COMP-3 VALUE 0.            LVC01260
01  HEAD-LINE.                                           LVC01270
    02  FILLER        PIC X(30) VALUE SPACE.            LVC01280
    02  HEAD-TEXT     PIC X(23) VALUE                    LVC01300
    02  FILLER        PIC X(28) VALUE SPACE.            LVC01310
    02  HEAD-PG-NAM   PIC X(4) VALUE 'PAGE'.            LVC01320
    02  HEAD-PAGE     PIC Z(3).                          LVC01330
                                                         LVC01340
01  ABNORMAL-LINE.                                       LVC01350
    02  FILLER        PIC X(1) VALUE SPACE.              LVC01360
    02  ERR-MSG       PIC X(41)                          LVC01370
           VALUE '**** UNEXPECTED END OF FILE ON INPUT ****'. LVC01380
    02  FILLER        PIC X(91) VALUE SPACES.            LVC01390
                                                         LVC01400
01  BAD-CODE-LINE.                                       LVC01410
    02  FILLER        PIC X(1) VALUE SPACE.              LVC01420
    02  ERR-MSG       PIC X(41)                          LVC01430
           VALUE '**** INVALID COMMAND DETECTED ****'.   LVC01440
    02  FILLER        PIC X(91) VALUE SPACES.            LVC01450
                                                         LVC01460

```

01	MESSAGE-LINE.		LVC01470
02	MESSAGE-CC	PIC X VALUE SPACES.	LVC01480
02	FILLER	PIC X(5) VALUE 'FSI: '.	LVC01490
02	OUT-FSI	PIC X.	LVC01500
02	FILLER	PIC X(6) VALUE ' VSI: '.	LVC01510
02	OUT-VSI	PIC X.	LVC01520
02	FILLER	PIC X(10) VALUE ' MESSAGE: '.	LVC01530
02	OUT-MESSAGE	PIC X(40).	LVC01540
			LVC01550
01	REDISPLAY-LINE.		LVC01560
02	REDISPLAY-CC	PIC X VALUE SPACES.	LVC01570
02	REDISPLAY-PROMPT	PIC X(2) VALUE '> '.	LVC01580
02	OUT-OPTIONS	PIC X.	LVC01590
02	OUT-OPCODE	PIC X.	LVC01600
02	OUT-POSIT	PIC X.	LVC01610
02	OUT-MOD	PIC X.	LVC01620
02	OUT-KEYS	PIC X.	LVC01630
02	OUT-OBJECT-NAME	PIC X(30).	LVC01640
02	FILLER	PIC X(91) VALUE SPACES.	LVC01650
			LVC01660
01	PRINT-OUT-RECORD.		LVC01670
05	PRINT-OUT-CC	PIC X VALUE SPACES.	LVC01680
05	OUT-REC	PIC X(131).	LVC01690
05	OUT-SIGN-ON REDEFINES OUT-REC.		LVC01700
10	OUT-USER-ID	PIC X(30).	LVC01710
10	OUT-PASS	PIC X(8).	LVC01720
05	OUT-PROD REDEFINES OUT-REC.		LVC01730
10	OUT-PROD-NO	PIC X(4).	LVC01740
10	OUT-PROD-DESC	PIC X(40).	LVC01750
10	OUT-PROD-RENT	PIC 9999999V99.	LVC01760
10	OUT-PROD-MAINT	PIC 9999999V99.	LVC01770
10	OUT-PROD-PURCH	PIC 9999999V99.	LVC01780

```

05 OASI-PROD REDEFINES OUT-REC.                                LVC01790
    10 OASI-PROD-NO      PIC X.                                LVC01800
    10 FILLER            PIC X(3).                            LVC01810
    10 OASI-PROD-DESC   PIC X.                                LVC01820
    10 FILLER            PIC X(39).                          LVC01830
    10 OASI-PROD-RENT   PIC X.                                LVC01840
    10 FILLER            PIC X(9).                            LVC01850
    10 OASI-PROD-MAINT  PIC X.                                LVC01860
    10 FILLER            PIC X(9).                            LVC01870
    10 OASI-PROD-PURCH  PIC X.                                LVC01880
*   05 OUT-CUST REDEFINES OUT-REC.                            LVC01890
*   05 OUT-CUS-PRD REDEFINES OUT-REC.                        LVC01900
*   05 OUT-CONTACT REDEFINES OUT-REC.                       LVC01910
01 EOF                PIC XXX VALUE 'NO '.

PROCEDURE DIVISION.

    PERFORM INITIALIZATION.
    PERFORM READ-TRAN-FILE UNTIL EOF = 'YES'.
    PERFORM TERMINATION.
    STOP RUN.

INITIALIZATION.
    SIGN-ON CINCOM CINCOM.
    RESET.
    OPEN INPUT TRAN-IN.
    READ TRAN-RECORD AT END MOVE 'YES' TO EOF.
    GET INVC FOR UPDATE.
    GET FIRST BRANCH-CITY-STATE
        NOT FOUND PERFORM PRINT-MESSAGE.
    GET LAST BRANCH-CITY-STATE USING BRANCH-CITY,
        BRANCH-STATE.
    GET NEXT PROD USING PRODUCT-CODE.
    GET PRIOR PROD.
    GET PRIOR USER-NAME.

```

```
READ-TRAN-FILE.  
    MOVE INP-PROD-DESC TO PRODUCT-DESC.  
    MOVE INP-PROD-NO TO PRODUCT-CODE.  
    INSERT NEXT PROD.  
    INSERT NEXT USER-NAME.  
    INSERT INVC DUP KEY PERFORM DUP-KEY.  
    INSERT FIRST PROD.  
    INSERT LAST BRANCH-CITY-STATE.  
    INSERT PRIOR PROD.  
    MARK PROD AT PRODUCT-CODE.  
PRINT-MESSAGE.  
    WRITE PRINT-REC FROM ABNORMAL-LINE.  
    DELETE ALL BRANCH-CITY-STATE.  
    FORGET PRODUCT-CODE NOT FOUND  
    PERFORM ERROR-FORGET.  
    UPDATE BRANCH-CITY-STATE.  
    UPDATE USER-NAME.  
ERROR-FORGET.  
    MOVE 'BAD RESET' TO OUT-MESSAGE.  
    WRITE PRINT-REC FROM MESSAGE-LINE.  
TERMINATION.  
    DELETE PROD.  
    DELETE USER-NAME.  
    RELEASE PROD.  
    COMMIT.  
    SIGN-OFF.
```

# Index

\*

\* in DBAID 31

=

= command  
defined 29  
example 32  
syntax 32

## A

ALL clause  
in DELETE command 40  
in DELETE statement 105  
application programming 18  
ASI. See Attribute (Column)  
Status Indicator (ASI)  
asterisk (\*), in DBAID 31  
AT phrase  
in GET command 48  
in GET statement 112  
in MARK command 58  
in MARK statement 116  
Attribute (Column) Status  
Indicator (ASI) 80, 91  
and packed values 83  
and zoned values 83  
defined 82

## B

batch DBAID, using 31  
batch environment, and data  
base recovery 95  
built-in logical view commands.  
See DBAID built-in view  
commands  
BYE command  
defined 29  
syntax 33

BY-LEVEL command  
defined 29  
example 35  
syntax 34

## C

catalogued procedures 125  
CAUTIOUS command  
defined 29  
syntax 36  
characters per line, specifying 57  
CICS environment, and data  
base recovery 95  
column list, displaying 70  
COBOL compiler 124  
COBOL program statements 97  
COBOL Programmer's Report 21  
defined 76  
COBOL programs, and RDM 17,  
21  
COBOL view program, writing 77  
column  
as a key 23  
defined 20  
modifying contents of 93  
column in view  
displaying description of 37  
displaying text for 37  
column names in view, displaying  
34  
COLUMN-TEXT command  
defined 29  
example 38  
syntax 37  
COMMIT  
automatic 67  
prohibiting automatic 36  
COMMIT command  
defined 29  
syntax 39  
COMMIT statement  
defined 25  
example 104  
syntax 104  
using 95  
compilers 18, 124  
current program, checking for 96

- D**
- DATA DIVISION statements 98
    - INCLUDE TIS-CONTROL 101
    - INCLUDE view-data 98
  - DATA DIVISION, coding 77
  - data integrity and DELETE statement 106
  - data manipulation, statements for 25
  - data model overview 20
  - DBAID
    - batch environment 31
    - formatting guidelines 31
    - signing-off 65
    - signing-on 66
  - DBAID built-in view commands 29
    - BY-LEVEL 34
    - COLUMN-TEXT 37
    - FIELD-DEFN 43
    - VIEW-DEFN 72
    - VIEWS-FOR-USER 74
  - DBAID commands 28
    - built-in view, defined 27
    - categories 27
    - RDML, defined 27
    - system, defined 27
  - DBAID RDML commands 29
    - = 32
    - BYE 33
    - CAUTIOUS 36
    - COMMIT 39
    - DELETE 40
    - ERASE 42
    - FORGET 45
    - GET 46
    - GO 50
    - INSERT 53
    - KEEP 56
    - MARK 58
    - OPEN 60
    - RELEASE 63
    - SIGN-OFF 65
    - SIGN-ON 66
    - SURE 67
    - UPDATE 68
  - DBAID system commands 28
    - LINESIZE 57
    - MARKS 59
    - PAGESIZE 62
    - USER-LIST 70
    - USERS 71
    - VIEWS 73
  - DBAID utility 26, 27
  - DEBUG 79, 101
  - DELETE command
    - defined 29
    - examples 41
    - syntax 40
  - DELETE statement
    - defined 25
    - examples 106
    - syntax 105
  - DTB. *See* Dynamic Transaction Backout (DTB)
  - DUP KEY phrase, in INSERT statement 115
  - duration of last request, displaying 71
  - Dynamic Transaction Backout (DTB)
    - and CICS environment 95
    - and COMMIT statement 104
    - and RESET statement 119
- E**
- ELSE clause
    - in FORGET statement 107
    - in GET statement 112
  - ERASE command
    - defined 29
    - syntax 42
  - error handling 92, 96
  - exit, from DBAID utility. *See* BYE command
- F**
- FIELD-DEFN command
    - defined 29
    - example 44
    - syntax 43
  - FOR phrase, in GO command 51

FOR UPDATE phrase  
 in GET command 48  
 in GET statement 110  
 FORGET command  
 defined 29  
 syntax 45  
 FORGET statement  
 defined 25  
 syntax 107  
 formatting guidelines, DBAID 31  
 FROM phrase, in GO command  
 51  
 FSI. See Function Status  
 Indicator (FSI)  
 Function Status Indicator (FSI)  
 80, 91, 92  
 defined 81

## G

GET command  
 defined 29  
 syntax 46  
 GET statement  
 defined 25  
 examples 113  
 syntax 109  
 using 86  
 GO command  
 defined 29  
 syntax 50

## I

INCLUDE statement  
 example 100  
 syntax 98  
 INCLUDE TIS-CONTROL  
 statement  
 examples 102  
 syntax 101  
 INSERT command  
 defined 29  
 examples 55  
 syntax 53  
 INSERT statement  
 defined 25  
 examples 115  
 syntax 114  
 using 94

## K

KEEP command  
 defined 30  
 syntax 56  
 key  
 compound nonunique 23  
 compound unique 23  
 defined 20  
 defining to directory 24  
 in a logical view 23  
 nonunique 23  
 unique 23

## L

lines, specifying number to  
 display 62  
 LINESIZE command  
 defined 28  
 syntax 57  
 linking a compiled program 124

## M

MARK command  
 defined 30  
 syntax 58  
 MARK statement  
 example 117  
 syntax 116  
 MARKS command  
 defined 28  
 example 59  
 syntax 59  
 MASS phrase, with INSERT  
 command 54

## N

NBUG 101  
 non-null value 23  
 NOT FOUND clause  
 in FORGET statement 107  
 in GET statement 112  
 null values 21  
 with INSERT statement 83, 115  
 with UPDATE statement 122

- O**
- OPEN command
    - defined 30
    - example 61
    - syntax 60
  - OS/390, catalogued procedures 125
- P**
- packed fields, and null value 83
  - packed values, and ASI 83
  - PAGESIZE command
    - defined 28
    - syntax 62
  - precompiler
    - execution of 124
  - PROCEDURE DIVISION
    - statements
      - COMMIT 104
      - DELETE 105
      - FORGET 107
      - GET 109
      - INSERT 114
      - MARK 116
      - RELEASE 118
      - RESET 119
      - SIGN-OFF 120
      - SIGN-ON 121
      - UPDATE 122
  - PROCEDURE DIVISION, coding 103
  - processing time used, displaying 71
  - programming with RDM,
    - overview 17
  - pseudoconversational
    - applications, signing off/on 85
  - pseudoconversational program
    - and COMMIT statement 104, 120
    - and SIGN-OFF statement 120
- R**
- RDM. See Relational Data Manager (RDM)
  - RDML. See Relational Data Manipulation Language (RDML)
  - recompile, when required 96
  - record
    - adding 94
    - inserting 94
  - record holding
    - automatic 91
    - explicit 91
  - recovery
    - in batch environment 95
    - in CICS environment 95
    - of data 24
    - of database 95
  - Relational Data Manager (RDM)
    - administration of 20
    - benefits of 17
    - overview 18
    - status indicators 80
    - unsuccessful function 92
  - Relational Data Manipulation Language (RDML)
    - compiler 18, 123
    - parameters 124
    - reissue 32
  - RELEASE command
    - defined 30
    - syntax 63
  - RELEASE statement
    - defined 25
    - example 118
    - syntax 118
  - Report, COBOL Programmer's 21, 76
  - request count, displaying 71
  - RESET
    - automatic 92
    - prohibiting automatic 56
    - using 95
    - with ERASE command 42
  - RESET command
    - defined 30
    - syntax 64
  - RESET statement
    - defined 25
    - example 119
    - syntax 119
  - rollback, CICS 119

**row**

- adding with INSERT 114
- defined 20
- deleting with DELETE
  - statement 105
- example 21
- modifying 93
- passing to external modules 79
- removing 93
- removing from database, with
  - DELETE command 40
- retrieving, discussion of 86
- save position of 90
- using DELETE 93
- using INSERT 94
- using UPDATE 93
- with GET command 46
- with GET FIRST command 47
- with GET FIRST statement 109
- with GET LAST command 47
- with GET LAST statement 109
- with GET NEXT command 47
- with GET NEXT statement 109
- with GET PRIOR command 47
- with GET PRIOR statement 109
- with GET SAME command 47
- with GET SAME statement 109
- with GET statement 109
- with nonunique key 88
- without a key 88

**S**

- sample RDM COBOL application
  - program 127
- save position, with MARK
  - statement 90
- security 17
- signing off 25
- signing off/on, and PROCEDURE
  - DIVISION 85
- signing on 25
- SIGN-OFF command
  - defined 30
  - syntax 65
- SIGN-OFF statement
  - defined 25
  - example 120
  - syntax 120

**SIGN-ON command**

- defined 30
- example 66
- syntax 66

**SIGN-ON statement**

- defined 25
- example 121
- syntax 121

- sign-on time, displaying 71

- START clause, with GO
  - command 51

- station number, displaying 71

**storage**

- for FORGET statement 107
- freeing 25, 45
- maintaining 86

- RELEASE command 63

- sign-off statement 120

- sign-on statement 121

- with FORGET command 45

- with RELEASE statement 118

**SURE command**

- defined 30
- syntax 67

- system commands *See* DBAID
  - system commands

- system performance, effect of
  - FOR UPDATE clause on 109

**T**

- tabular format display 50

- Task Level Recovery (TLR)

- and COMMIT statement 104

- task, restarting with RESET 119

- time of sign-on, displaying 71

- TIS-CONTROL 79

- TLR. *See* Task Level Recovery (TLR)

- TRACE 79, 101

**U**

- unsuccessful function (RDM) 92

- UPDATE command

- defined 30

- example 69

- syntax 68

UPDATE statement  
  defined 25  
  example 122  
  syntax 122  
user name, displaying 71  
user view  
  changing 22  
  creating 22  
  creating from logical view 78  
  defined 20, 21  
  example 21  
USER-LIST command  
  defined 28  
  syntax 70  
USERS command  
  defined 28  
  example 71  
  syntax 71  
users, displaying information  
  about 71  
USING phrase  
  in GET command 49  
  in GET statement 111  
  in GO command 51  
utility, DBAID 26, 27

## V

validation of data 80  
Validity Status Indicators (VSI)  
  80, 91  
  defined 83  
value, defined 20  
view  
  changing 22  
  closing 63  
  creating 22  
  defined 20, 21  
  displaying description of 72  
  example 21  
  preparing for use by DBAID 60  
  to use in program 77  
  with MARK statement 116  
  with RELEASE statement 118  
  with UPDATE statement 122

view program, writing in COBOL  
  76  
view record, inserting in physical  
  database 54  
VIEW-DEFN command  
  defined 29  
  example 72  
  syntax 72  
views  
  active, displaying 73  
  listing by signed-on user 74  
  multiple, accessing 89  
VIEWS command  
  defined 28  
  example 73  
  syntax 73  
VIEWS-FOR-USER command  
  defined 29  
  example 74  
  syntax 74  
VSE samples 126  
VSI. See Validity Status Indicator  
  (VSI)

## Z

zoned fields, and null value 83  
zoned value, and ASI 83