

Cincom

SUPRA SERVER PDM

RDM Administration Guide
(VMS)

P25-8220-45



SUPRA[®] Server PDM RDM Administration Guide

Publication Number P25-8220-45

© 1990, 1995–2002 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This document contains unpublished, confidential, and proprietary information of Cincom.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 [®]	iD CinDoc [™]	MANTIS [®]
C+A-RE [™]	iD CinDoc Web [™]	Socrates [®]
CINCOM [®]	iD Consulting [™]	Socrates [®] XML
Cincom Encompass [®]	iD Correspondence [™]	SPECTRA [™]
Cincom Smalltalk [™]	iD Correspondence Express [™]	SUPRA [®]
Cincom SupportWeb [®]	iD Environment [™]	SUPRA [®] Server
CINCOM SYSTEMS [®]	iD Solutions [™]	Visual Smalltalk [®]
	intelligent Document Solutions [™]	VisualWorks [®]
gOOi [™]	Intermax [™]	

UniSQL[™] is a trademark of UniSQL, Inc.
ObjectStudio[®] 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 Administration Guide (VMS)*, P25-8220-45, is dated January 15, 2002. This document supports Release 2.4 of SUPRA Server.

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

Phone: 1-800-727-3525

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

Contents

About this book	ix
Using this document.....	ix
Document organization	x
Conventions	xi
SUPRA Server documentation series	xiii
Introduction to the Relational Data Manager	15
The role of the RDM in the SUPRA Server system.....	18
How RDM signs on to the database.....	19
How RDM handles view-open requests	21
SUPRA Server's three schema architecture	23
The internal schema: Physical Data Description.....	24
The conceptual schema: base views	25
The external schema: derived views.....	26
How the RDM fits into the three schema architecture.....	27
Views.....	28
Two types of views	29
How views are used	29
User views.....	30
RDM reports	31
RDM security.....	33
Example database	34
Parts of a view	35
Column definitions.....	36
Access definitions	50
View design considerations	73
How RDM constructs rows.....	74
Database penetration.....	75
Database sweep	77

Navigational constraints and boundary conditions	78
Processing derived views	80
Keyed access to data	83
Unique keys	86
Non-unique keys	88
Constant keys	90
Secondary access keys	92
Generic reads	93
Domains.....	96
Null values	98
Default values	101
Validation options	103
Join compatibility	105
Referential integrity with RDM	106
Integrity rules and checking	108
Foreign key value integrity	109
Insertion integrity.....	110
Update integrity.....	112
GET processing.....	115
Deletion integrity	116
Referential integrity examples	121
Shared column values	126
View-to-user relationships	128

Physical and logical database changes 131

Overview.....	131
Physical and logical database actions	131

Defining and testing views using DBAID 135

Invoking DBAID	136
Signing on to DBAID	138
Using DBAID commands.....	139
* command	146
= command.....	148
BIND command	149
BY-LEVEL command.....	150
BYE command.....	152
CAUTIOUS command	153
COLUMN-DEFN command	154
COLUMN-TEXT command.....	158
COMMIT command	159
COPY command.....	160
DEFINE command.....	161
DELETE command.....	162
DENY command.....	164

EDIT command	165
ERASE command	166
FIELD-DEFN command	167
FIELD-TEXT command.....	170
FORGET command	172
GET command.....	173
GO command.....	179
INSERT command	183
KEEP command.....	188
line-number command	189
LINESIZE command	190
LIST command.....	191
MARK command	193
MARKS command.....	194
OPEN command	195
PAGESIZE command	198
PERMIT command.....	199
PRINT-STATS command.....	200
RELEASE command.....	201
REMOVE command.....	202
RENUMBER command.....	203
RESET command	204
SAVE command.....	205
SHOW-NAVIGATION command	207
SIGN-OFF command	208
SIGN-ON command.....	209
STATS command.....	210
STATS-OFF command	211
STATS-ON command	212
SURE command	213
UNDEFINE command.....	214
UPDATE command.....	215
USER-LIST command	218
VIEW-DEFN command.....	219
VIEWS command.....	221
VIEWS-FOR-USER command	222

RDM status indicators 223

Function Status Indicators (FSIs).....	224
Column Attribute Status Indicators (ASIs).....	226
Validity Status Indicators (VSIs)	230

Optimizing view performance using bound and global views	231
Differences between bound and global views	231
Advantages of using global views.....	234
Changing view text: a note of caution.....	235
Bound views.....	236
Binding a view.....	236
Ensuring that you update a bound view.....	239
Global views.....	241
Creating a Global View file	243
Example Global View input files	257
Example Global View report file.....	258
Options for RDM access to the SUPRA Server directory.....	259
Running without the directory	259
Running with the directory and with Global Views.....	260
Running with the directory alone.....	260
Generating RDM reports	261
RDM reports	263
Stage one—specifying the reports to be produced.....	264
Stage two—generating the reports	268
DBAID quick reference	269
DBAID commands	269
Definitions	275
Status indicators	276
ASI values.....	276
FSI values.....	277
VSI values.....	278
Example RDM reports	279
DBA report format description	280
Domain usage report format description	285
Logical Data Item report format description.....	288
Physical Data Item report format description.....	291
Validation Table Usage report format description	294
Example user validation exits	295
Example database	305
Relations in the internal schema.....	305
Base views in the conceptual schema.....	310
Derived views in the external schema	314
Index	319

About this book

Using this document

To administer the RDM, this manual provides you with information necessary for:

- ◆ Understanding the overall purpose of the RDM and its role in the SUPRA Server system
- ◆ Understanding the parts of a view
- ◆ Creating views best suited to your needs by:
 - Understanding some important design considerations
 - Recognizing modifications you must make as a result of physical and logical database changes
 - Interpreting the status indicators RDM returns to show view processing results
- ◆ Designing and testing views
- ◆ Optimizing view performance
- ◆ Supplementing your understanding of the concepts explained in this manual by providing an example database called EXAMPL

Document organization

The information in this manual is organized as follows:

Chapter 1—Introduction to the Relational Data Manager

Introduces you to the Relational Data Manager. Shows you how the RDM fits in the SUPRA Server system, how RDM signs on to the SUPRA Server Directory, and describes SUPRA Server's three-schema architecture.

Chapter 2—Parts of a view

Describes the two parts of a view (column and access definitions), and provides the syntax for defining them.

Chapter 3—View design considerations

Provides information you should consider before defining your views.

Chapter 4—Physical and logical database changes

Shows you further actions you should take as a result of physical and logical changes to the database.

Chapter 5—Defining and testing views using DBAID

Shows you how to sign on to DBAID to test and define your views, and gives syntax for the DBAID commands.

Chapter 6—RDM status indicators

Explains each of the status indicators that RDM provides to show view processing results.

Chapter 7—Optimizing view performance using bound and global views

Explains how to use bound and global views to optimize view performance.

Chapter 8—Generating RDM reports

Shows you how to produce RDM reports. You must have SPECTRA to run the RDM reports. SPECTRA is not available in the OpenVMS AXP environment.

Appendix A—DBAID quick reference

Provides a quick reference for DBAID commands.

Appendix B—Example RDM reports

Provides example RDM reports, using the example database described in Appendix D.

Appendix C—Example user validation exits

Shows example user validation exits.

Appendix D—Example database

Describes the database used in examples throughout this manual.

Index

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 illustrates that four spaces appear between the keywords.	BEGN <i>b</i> <i>b</i> <i>b</i> <i>b</i> 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.	<u>(WAIT)</u> (NOWAIT)
	The example indicates that you can optionally enter either WAIT or NOWAIT. (WAIT is underlined to signify that it is the default.)	
Braces { }	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.	MONITOR {ON OFF}
	In the example you must enter ON or OFF when using the MONITOR statement.	

Convention	Description	Example
<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>	<p>(WAIT) (NOWAIT)</p> <hr/> <p><u>STATISTICS</u></p>
<p>Ellipsis points...</p>	<p>Indicate that the preceding item can be repeated.</p> <p>In the example you can enter multiple host variables and associated indicator variables.</p>	<p><i>INTO :host-variable [:ind-variable],...</i></p>
<p>UPPERCASE lowercase</p>	<p>In most operating environments, keywords are not case-sensitive, and they are represented in uppercase. You can enter them in either uppercase or lowercase.</p>	<p>COPY MY_DATA.SEQ HOLD_DATA.SEQ</p>
<p><i>Italics</i></p>	<p>Indicate variables you replace with a value, a column name, a file name, and so on.</p> <p>In the example you must substitute the name of a table.</p>	<p>FROM <i>table-name</i></p>
<p>Punctuation marks</p>	<p>Indicate required syntax that you must code exactly as presented.</p> <p>() parentheses . period , comma : colon ' ' single quotation marks</p>	<p><i>(user-id, password, db-name)</i> INFILE 'Cust.Memo' CONTROL LEN4</p>
<p>SMALL CAPS</p>	<p>Represent a required keystroke. Multiple keystrokes are hyphenated.</p>	<p>ALT-TAB</p>

SUPRA Server documentation series

SUPRA 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 for VMS Systems*, P25-9062.

Overview

- ◆ *SUPRA Server PDM Digest for VMS Systems*, P25-9062

Getting started

- ◆ *SUPRA Server PDM VMS Installation Guide*, P25-0147
- ◆ *SUPRA Server PDM VMS Tutorial*, T25-2263

General use

- ◆ *SUPRA Server PDM Glossary*, P26-0675
- ◆ *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022

Database administration tasks

- ◆ *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260
- ◆ *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130
- ◆ *SUPRA Server PDM Utilities Reference Manual (UNIX & VMS)*, P25-6220
- ◆ *SUPRA Server PDM Directory Views (VMS)*, P25-1120
- ◆ *SUPRA Server PDM Windows Client Support User's Guide*, P26-7500*
- ◆ *SPECTRA Administrator's Guide*, P26-9220**

Application programming tasks

- ◆ *SUPRA Server PDM Programming Guide (UNIX & VMS)*, P25-0240
- ◆ *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130
- ◆ *SUPRA Server PDM RDM Administration Guide (VMS)*, P25-8220
- ◆ *SUPRA Server PDM Windows Client Support User's Guide*, P26-7500*
- ◆ *MANTIS Planning Guide*, P25-1315**

Report tasks

- ◆ *SPECTRA User's Guide*, P26-9561**



Manuals marked with an asterisk (*) are listed twice because you use them for different tasks.



Educational material is available from your regional Cincom education department.

1

Introduction to the Relational Data Manager

The Relational Data Manager (RDM) processes applications' requests to access the physical data held in PDM and RMS data sets and presents it as though it were arranged in two-dimensional tables. These two-dimensional tables are referred to as logical views or simply, *views*. The Database Administrator (DBA) designs views from data stored on the SUPRA Server directory database SUPRAD. The DBA can create two kinds of views: base views or derived views. Base views access the physical data, derived views access only other views; not the physical data sets themselves.

One of the biggest advantages of the RDM is that it enables application programmers and end users to use these views to access the database without concern for the location of the data. RDM also provides:

- ◆ **Relational Data Manipulation Language (RDML).** Allows programmers to retrieve and modify database contents. Refer to the *SUPRA Server PDM Programming Guide (UNIX & VMS)*, P25-0240, for details on using RDML.
- ◆ **DBAID.** Allows DBAs to define and test views. DBAID also has commands that programmers can use to test whether a view meets application requirements. See “[Defining and testing views using DBAID](#)” on page 135 for information on using DBAID.
- ◆ **Binding views.** Allows the DBA to place pre-opened copies of logical views on the SUPRA Server Directory database SUPRAD. See “[Optimizing view performance using bound and global views](#)” on page 231 for more information on bound views.
- ◆ **Globalizing views.** Allows the DBA to place pre-opened copies of frequently used logical views in global memory. See “[Optimizing view performance using bound and global views](#)” on page 231 for more information on global views.



Both bound views and global views reduce processing overhead because the view-open is only performed once (at the initial RDM sign-on); subsequent RDM sign-ons simply map to the open view. Bound views are stored on the SUPRA Server Directory, global views are stored in global memory. See “[Optimizing view performance using bound and global views](#)” on page 231 for more information on how to use bound and global views and the differences between them.

File types supported by the RDM

In OpenVMS environments, the RDM can access both Physical Data Manager (PDM) data sets and RMS data sets. RDM accesses PDM data sets through the PDM; however, it accesses RMS data sets directly.

To use RMS data sets, define them to the Directory using the DBA utility (Create RMS Data Set function). Then define the views and physical access requirements. RDML programs written in COBOL, FORTRAN, BASIC, and MANTIS can then access data held on RMS data sets through RDM as they would data held on PDM data sets.

For information about using PDM data sets, refer to the *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130.



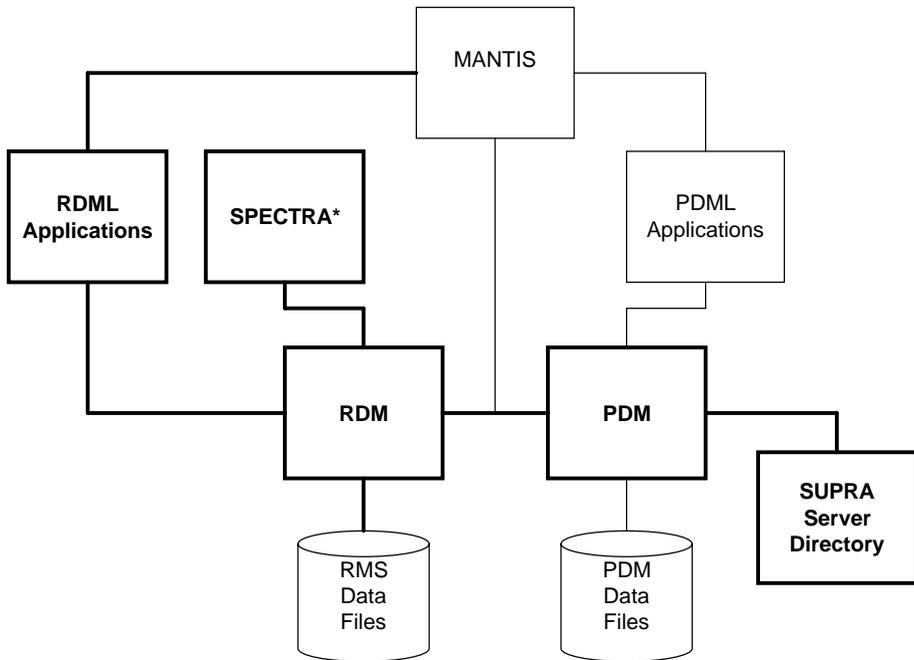
The SUPRA Server Physical Data Manager (PDM) does not access RMS data sets. Therefore, PDM recovery logging is not available for RMS data sets. However, VMS Recovery Unit Journaling is available as a product option.

Application programs that already access RMS data sets directly need not be changed. However, they are not insulated from change until they are rewritten using RDM for access.

All views must be stored on the SUPRA Server Directory database SUPRAD before application programs and users can access them. You can create views using either the DBA utility or the DBAID utility. The difference is that DBAID allows you to test your views before putting them into production; SUPRA DBA does not. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for more information on using DBA, and/or “Defining and testing views using DBAID” on page 135 for information on using DBAID.

The role of the RDM in the SUPRA Server system

The following figure shows the SUPRA Server system, including SUPRA Server components and other related products. As shown, the RDM receives application requests and accesses the physical data held on the Directory through the PDM. Note that the RDM accesses VMS RMS data sets directly (not through the PDM).

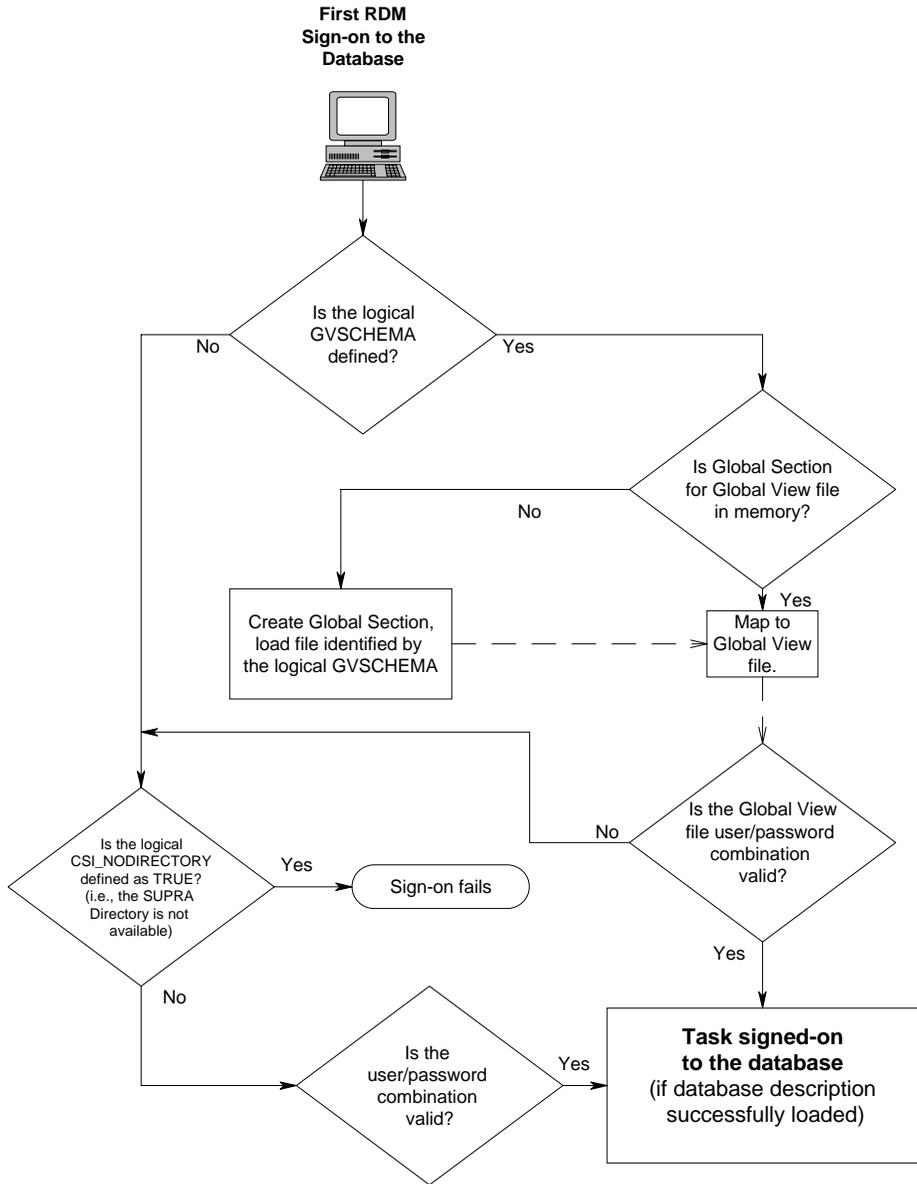


**SPECTRA is not available in OpenVMS AXP environments*

How RDM signs on to the database

The following figure shows what happens when RDM signs on to the database. If you are using global views, the RDM may or may not require access to the SUPRA Server Directory database.

Global views are opened and placed in global memory by the first RDM sign-on, remaining available for subsequent view requests. If you use global views, the RDM accesses the Global View file for run time information, not the SUPRA Server Directory. Therefore, you could place all your views in a Global View file and not require access to the SUPRA Server Directory at all. You also have the option of running your system using *both* a Global View file and the SUPRA Server Directory. For more information on using Global View files, see [“Optimizing view performance using bound and global views”](#) on page 231.

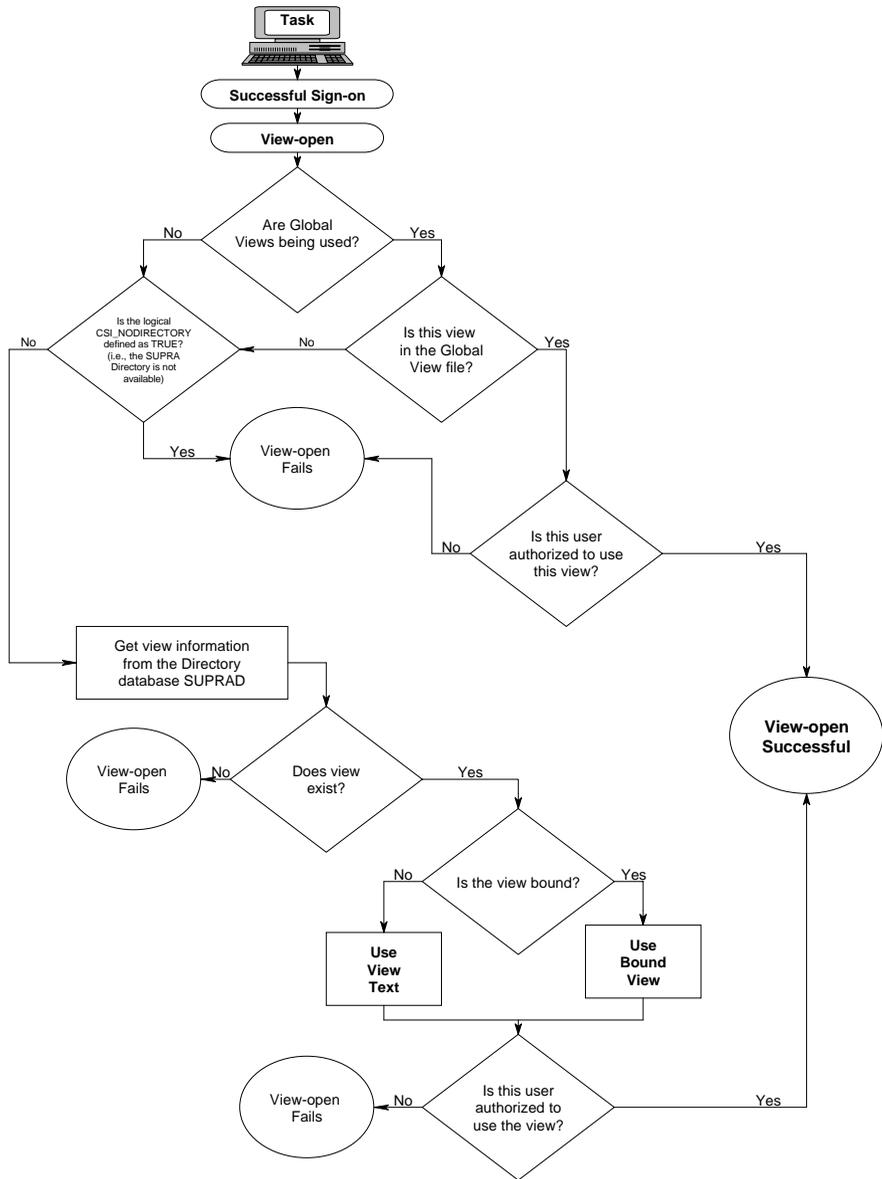


How RDM handles view-open requests

After successful sign-on to the database, RDM handles the view-open request. When RDM receives a view-open request, it does the following:

1. Checks if global views are being used. If so, RDM uses the global version of the view (stored in global memory at the first RDM sign-on).
2. If global views are not being used, RDM checks for a bound version of the view (stored on the SUPRA Server Directory database). If bound views are being used, RDM uses the bound version of the view.
3. If there is no global or bound version of the view, RDM uses the view definition text as stored on the SUPRA Server Directory database.

The following figure shows this process.



SUPRA Server's three schema architecture

SUPRA Server uses three schema architecture to provide a physical and logical implementation that is easy to maintain and that can be changed with minimum impact to applications. Three schema architecture consists of the internal, conceptual, and external schema.

The *internal* schema contains physical definitions of data as held on the SUPRA Server Directory database SUPRAD.

The *conceptual* schema contains a set of base views that map onto the logical data items in the internal schema. These base views define the integrity rules for the entire database.

The *external* schema is the view of data as accessed by application programs, MANTIS programs and SPECTRA* processes. It consists of a set of derived views that accesses the base views. Derived views never access the internal schema directly; however, they inherit the integrity constraints of the conceptual schema.

The RDM acts as insulation between application programs and the physical structure of data located in the internal schema.

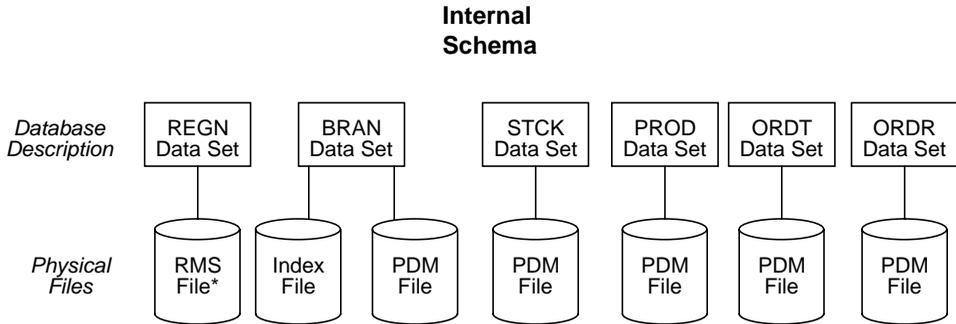


* SPECTRA is not available in OpenVMS AXP environments.

The internal schema: Physical Data Description

The internal schema is the lowest level of three schema architecture. It contains the physical description of data: record length, file layout, file type (PDM or RMS), location on the disk, recovery method, domain details, validation criteria, and so on. The Physical Data Manager (PDM) maintains the internal schema as database descriptions. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for information on defining a database description. You must create an internal schema before you can build a conceptual schema and an external schema.

An implementation consisting solely of an internal schema, where applications need to know the precise physical location and attributes of every piece of data, represents one schema architecture. The following figure illustrates this concept.



**RMS data sets are only accessed by the RDM, not the PDM*

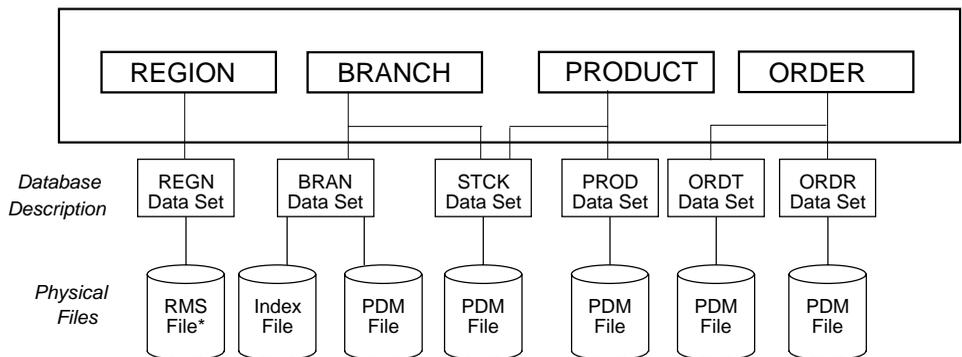
The conceptual schema: base views

The conceptual schema is the middle level of three schema architecture and contains the logical definition of the database. This logical definition consists of base views of data, which are relational views of data held in normalized tables.

The conceptual schema (containing normalized tables) is used to determine the physical implementation (internal schema) of your database. After you enter your physical database description, you enter the conceptual schema. To create your conceptual schema, you define base views using either DBA functions (refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260) or DBAID (see "Defining and testing views using DBAID" on page 135). The DBAID utility offers additional facilities to prototype and test views before relating them to users and putting them into production. Base views access the data sets directly and define referential integrity and data security. You must create a conceptual schema before you can build an external schema.

An implementation consisting of a database description (internal schema) and a set of base views (conceptual schema), with applications accessing data through the base views, represents two schema architecture. The following figure illustrates this concept.

Conceptual Schema



**RMS data sets are only accessed by the RDM, not the PDM.*

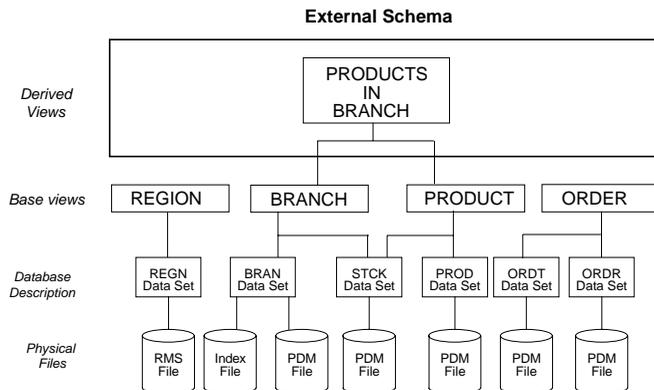
The external schema: derived views

The external schema is the top level of three schema architecture and consists of derived views, which access only other views. The views accessed by derived views can be either base views or other derived views, although we recommend that derived views access only base views to maintain optimum performance.

After defining base views, you create derived views using either DBA functions (refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260) or DBAID (see “Defining and testing views using DBAID” on page 135). Using DBAID, you can define your views, then test them *prior* to releasing them to users. When creating views using DBAID, you can define and open a base view and then define and open a derived view to access that base view without first saving the base view. Therefore, you can create both base and derived views in the DBAID test environment. Remember that you *cannot* save a derived view before you save the base view(s) it accesses.

Derived views inherit the security and integrity constraints associated with the base views that they access; however, you can also place more restrictive or higher levels of security on derived views. Application programmers, MANTIS programmers, and SPECTRA users access data through the derived views in the external schema. You must first define an internal schema and a conceptual schema before you can create an external schema. Both higher level schemas depend on the lower level schemas.

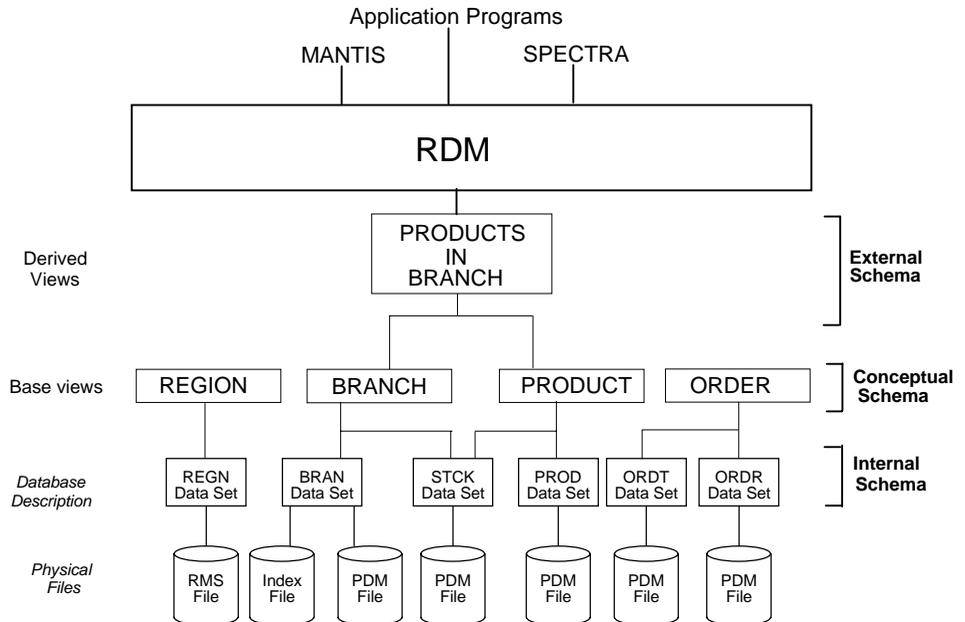
Therefore, three schema implementation consists of a database description (internal schema), a set of base views (conceptual schema), and a set of derived views (external schema). The following figure illustrates this concept.



How the RDM fits into the three schema architecture

The RDM uses the data views in the conceptual and external schemas to extract the physical data residing in the internal schema. RDM acts as insulation between the physical data and the application programmer; by presenting the data in views, the programmer need not be concerned for the physical structure of the data.

The following figure illustrates the Three Schema Architecture:



SPECTRA is not available in OpenVMS AXP environments.

Views

A view is a logical table of data consisting of logical data items drawn from one or more physical locations. RDM presents a view in a flat, two-dimensional, tabular format. Each two-dimensional table consists of rows and columns (see the following figure) that RDM maps to the physical database as you specify. Because RDM takes care of the physical navigation, users can manipulate database information without knowing its physical location or structure, or the integrity constraints placed upon it.

The diagram shows a table with three columns and three rows. The columns are labeled 'CUSTOMER NUMBER', 'CUSTOMER NAME', and 'CUSTOMER CLASS'. The rows contain data: (E40000, DOUG REED, Q1), (F80081, TOM LANGDON, B4), and (H22233, ATHENS INC, J1). A thick border highlights the middle row, and a thick border highlights the first column. A box labeled 'ROW' is positioned to the left of the middle row, and a box labeled 'COLUMN' is positioned below the first column.

	CUSTOMER NUMBER	CUSTOMER NAME	CUSTOMER CLASS
	E40000	DOUG REED	Q1
ROW	F80081	TOM LANGDON	B4
	H22233	ATHENS INC	J1

Two types of views

RDM uses two types of views: base and derived. Base views access the physical files described in the internal schema. Derived views can access only other views; base or derived. However, we recommend that derived views access only base views to maintain optimum performance.

Base views represent the conceptual schema, which is the logical description of the database. They insulate the derived views in the external schema from the physical data structures in the internal schema and provide the base level of security and referential integrity.

The main difference between base and derived views is that base views access only data sets and derived views access only other views. The DBA is the only one who ever needs to differentiate between the two types of views. The application programmer and SPECTRA user see no difference between a base view and a derived view. Both appear as tables of data and are known simply as views.

As the DBA, you can tailor views for specific uses or design them for multipurpose applications. You can allow users to perform powerful data maintenance actions (INSERT, UPDATE, and DELETE) with a view, or limit them to read-only access with no maintenance capabilities.

How views are used

Application programs use views to retrieve, insert, change or remove rows of data. By designing your business rules into your views, you have full control over how the RDML verbs GET, INSERT, UPDATE, and DELETE perform.

Application programmers use views by incorporating RDML statements into their programs. Refer to the *SUPRA Server PDM Programming Guide (UNIX & VMS)*, P25-0240, for details on how to use RDML statements in application programs.

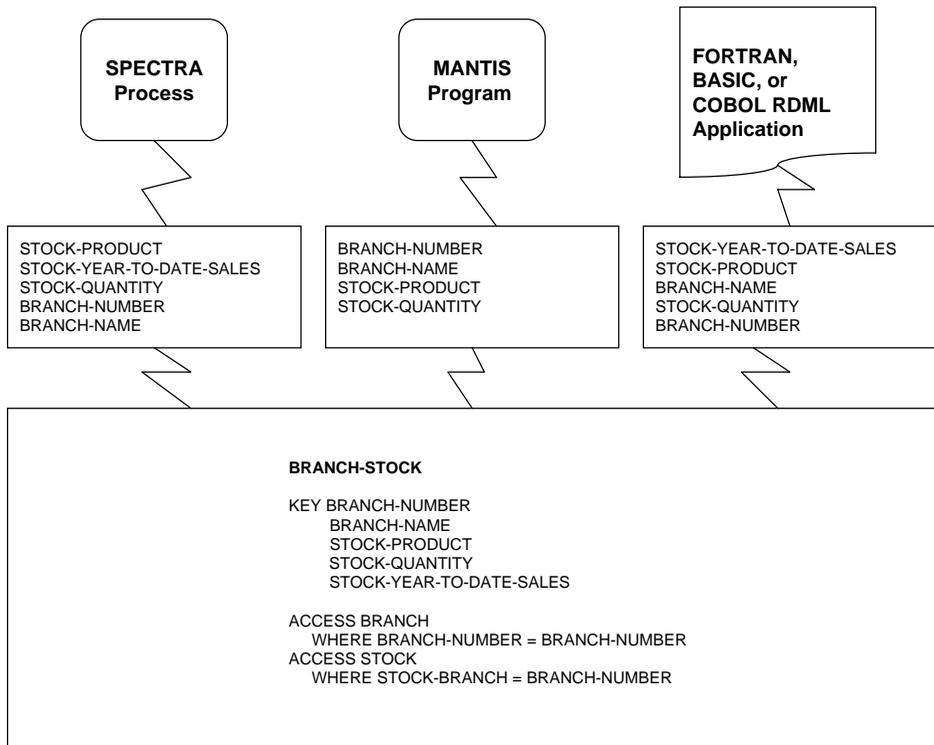
MANTIS programmers identify views with the VIEW statement. Then they use the GET, INSERT, DELETE, and UPDATE statements to access the data. Refer to your MANTIS documentation for additional details.

End users in OpenVMS VAX environments access views directly through SPECTRA, a relational query, update, and reporting tool.

User views

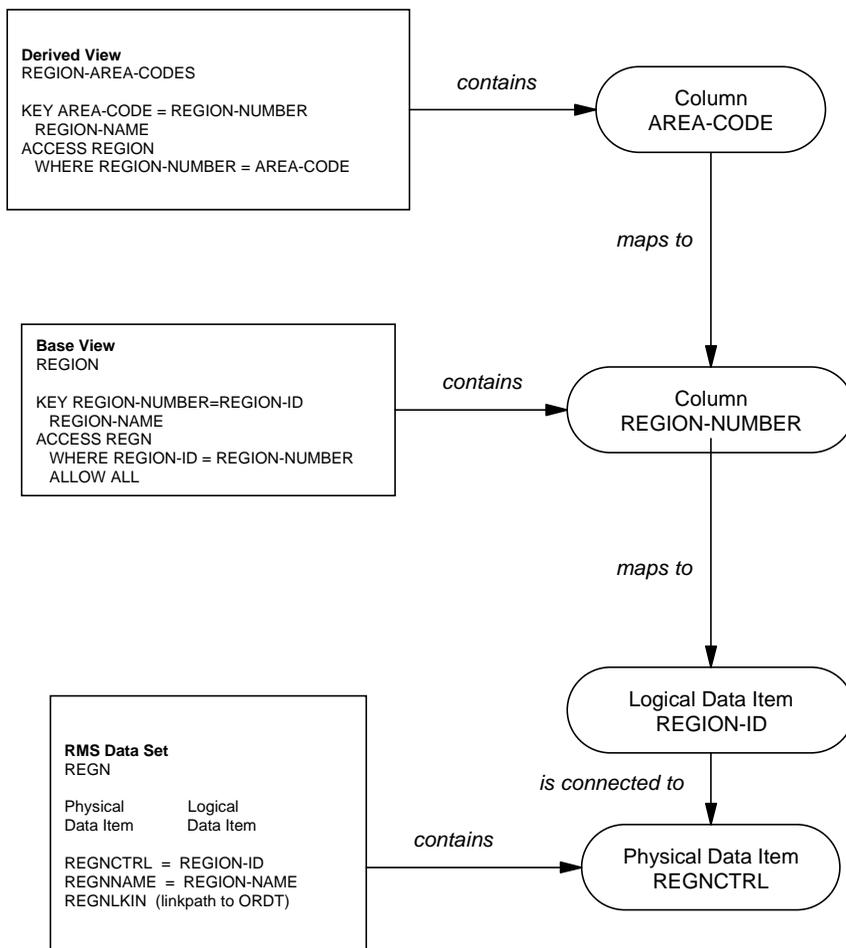
You do not need to define views to suit every possible need; programmers and SPECTRA users can subset views to suit their purposes. Such a subset is called a user view and can consist of columns from a single view or columns from multiple views. The user view can reorder the columns by specifying the column names in the order desired. By allowing end users and programmers to create their own user views, you can reduce the amount of administration and view design you need to do.

The following figure illustrates how different users could subset and reorder a view. Each user accesses the **BRANCH-STOCK** derived view that contains five columns. The SPECTRA user uses the entire view in the same sequence as specified in the View Definition. The MANTIS application uses only part of the available data, and also reorders the columns. The application program uses all the columns, but reorders them.



RDM reports

The implementation of three schema architecture can generate long connections between entities. The following figure illustrates such a connection. In this illustration, the AREA-CODE column in derived view REGION-AREA-CODES maps to the REGION-NUMBER column in base view REGION. The REGION-NUMBER column in turn maps to the REGION-NUMBER logical data item, which is connected to the physical data item REGNCTRL in the REGN RMS Data Set. The relationship is straightforward, but long.



RDM provides a comprehensive suite of reports that clarify these relationships and warn you of the lower or higher level impact of any proposed changes. These RDM reports follow the complex relationships between the physical data items, logical data items and columns, and the connections and interdependencies between base views and different levels of derived views. You can choose between the DBA and Application programmer reports, logical data item and physical data item cross reference reports, and validation table and domain usage reports. You can either select the scope of each report to encompass all databases, views, data sets, domains, and validation tables on the SUPRA Server Directory, or you can specify which entity or combination of entities you want to report on. See [“Generating RDM reports”](#) on page 261 and [“DBA report format description”](#) on page 280 for further information on the RDM reports.

RDM security

RDM controls security in two ways:

- ◆ View ACCESS Definitions
- ◆ User-to-view Relationships

The ACCESS definition of a view is described in “[Access definitions](#)” on page 50.

The user-to-view relationship defines which views a SUPRA Server user name is authorized to access (refer to the [SUPRA Server PDM Database Administration Guide \(UNIX & VMS\)](#), P25-2260, for details). You relate both base and derived views to users through DBA, DBAID or by using a Global View file.

Example database

Appendix C on page 305 provides an example database called EXAMPL, which includes descriptions of the relations in the internal schema, the base views in the external schema and the derived views in the conceptual schema.

All the examples in this manual are based on the EXAMPL database listed in “[Example database](#)” on page 305.

2

Parts of a view

Views provide great flexibility for accessing your data. When you create a view, you must specify:

- ◆ A column definition to identify the fields that are included in the view.
- ◆ An access definition to specify how RDM accesses the data.

You use either SUPRA DBA or the DBAID utility to define your views. The difference is that DBAID allows you to test your views before putting them into production; SUPRA DBA does not. However, note that views defined with DBAID that refer to physical data items in the column definition cannot be saved to the SUPRA Server Directory database SUPRAD.

Before actually defining your views, it is important to understand what makes up a view. This chapter describes the column and access definition requirements and considerations.

Column definitions

The column definition describes each logical data item in the view and its characteristics. The column definition must precede the access definition. You enter the name of each logical data item to be included in a view and an equivalent column name, if required. If you do not specify a column name, RDM uses the logical data item name as the column name.

Each column definition can be associated with one or more logical data items. Each logical data item has a one-to-one correspondence to a physical data item.

When you define a view using DBAID (see “[Defining and testing views using DBAID](#)” on page 135), specify the column and access definitions as lines of text, each preceded by a line number. When you define a view using DBA Functions (refer to the [SUPRA Server PDM Database Administration Guide \(UNIX & VMS\)](#), P25-2260, for details on logical data items), specify the column definition through the screen-based EDIT/EDT interface. When defining a view through DBA, you do not need to specify each line number as you do through DBAID. You can open and save views through DBA as you can through DBAID, however, you cannot test them through DBA.

Any view defined on the SUPRA Server Directory can be used by DBAID. However, views defined within DBAID that refer to physical data items in the column definition, rather than logical data items, cannot be saved on the SUPRA Server Directory. (DBAID allows definition and testing of views that refer to physical data items for compatibility with other platforms.)

Column definition requirements for *base views*:

KEY REQ FKEY NONUNIQUE KEY	[<i>column - name</i> =] <i>logical - data - item</i> [[=] <i>logical - data - item</i> ₂] ... [[=] <i>logical - data - item</i> _n]
---	---

[UNIQUE] CONST	[<i>column - name</i> =] <i>logical - data - item</i> [[=] <i>logical - data - item</i> ₂] ... [[=] <i>logical - data - item</i> _n] = constant
-----------------------	--

The column definition requirements for *derived views*:

KEY REQ NONUNIQUE KEY	[<i>column - name</i> =] <i>source - column - name</i> [[=] <i>source - column - name</i> ₂] ... [[=] <i>source - column - name</i> _n]
--------------------------------------	---

[UNIQUE] CONST	[<i>column - name</i> =] <i>source - column - name</i> [[=] <i>source - column - name</i> ₂] ... [[=] <i>source - column - name</i> _n] = constant
-----------------------	--

KEY

Description *Optional.* Indicates that the column is required in the view, is used as a logical key, and forms a unique key in combination with other unique keys.

Considerations

- ◆ KEY causes RDM to disregard rows with missing, null, or invalid occurrences of the KEY columns because a key column is required (see REQ description).
- ◆ You can specify a maximum of nine KEY and NON-UNIQUE KEY columns in a view.

Considerations for base views

- ◆ KEY causes direct reads to occur for a column when a value is given in GET, and the column maps to a physical key such as GET USING in DBAID.
- ◆ KEY causes a sequential search of data sets for a column when a value is given in GET, and the column is not a physical key such as GET USING.
- ◆ If the view is normalized, any one combination of unique key values will appear only once in a view.
- ◆ If the logical key is also a secondary key that allows duplicates, there is no point defining it as a unique key in the view.

REQ

Description *Optional.* Indicates the column is required when processing this view.

Considerations

- ◆ Required columns restrict the number of occurrences in the view. Valid rows in a base view must have an occurrence of the required column's physical record and, in the case of packed, numeric, or floating-point columns, must contain a valid non-null value for the required column.
- ◆ The REQ option affects the RDML commands in the following ways:
 - GET. All required (REQ) columns must be present, valid, and non-null, or RDM will return NOT FOUND on direct GETs. For a sequential (sweeping) GET, RDM skips the row.

When subsetting base views (making user views), if the programmer specifies a column list by using INCLUDE, one or more required columns can be omitted. However, the row is retrieved only if all required columns in that row are present.

For more information on the INCLUDE statement, refer to the [SUPRA Server PDM Programming Guide \(UNIX & VMS\)](#), P25-0240.

When using a user view based on a derived view, if a required column is not included in a user view, the required column must still be present, but is not returned to the program.

- INSERT or UPDATE. All required columns must be present, valid, and non-If a required column is missing from a user view of a base view, RDM returns an ASI of DATA. If a required column is not included in a user view of a derived view, RDM returns a status of FAIL.
- DELETE. No effect.

FKEY

Restriction This parameter is for base views only.

Description *Optional.* Indicates that this column may contain a null foreign key and is not required.

Consideration FKEY is used to enforce referential integrity in base views. It allows you to define a foreign key that may contain a null value. For the foreign key to support null values, use this syntax:

```
REQ column-name = foreign-key = primary-key
```

becomes

```
FKEY column-name = foreign-key = primary-key
```

if you want the foreign key to support null values.

NON-UNIQUE KEY

Description *Optional.* Indicates this column is required in the view and forms a non-unique logical key in combination with other unique or non-unique logical keys.

Considerations

- ◆ It causes direct reads for a column when a value is supplied on the GET, and the column is a physical key.
- ◆ It causes a sequential search (sweep) of files (in base views) or base views (in derived views) for a column when a value is supplied in the GET and the column is not a physical key.
- ◆ It causes the RDML processor to disregard rows with null or invalid occurrences of the KEY columns because a key column is required (see REQ description).
- ◆ It enables you to specify a value for the view logical key when selecting a row without requiring a unique occurrence of the key column.
- ◆ You can specify a maximum of nine KEY and NON-UNIQUE KEY columns.

[UNIQUE] CONST

Description *Optional.* Indicates that the column is required in the view, and the value of the column must be equal to the given constant for the row to qualify. The value of the constant is specified as part of the column description (=constant).

Default NON-UNIQUE unless you specify UNIQUE before CONST.

Considerations

- ◆ CONST must be specified if a constant is supplied.
- ◆ The value of the column must equal the constant for the row to qualify.
- ◆ All CONST columns are part of the logical key for the view.
- ◆ CONST columns are not returned in the view.
- ◆ Because CONST columns are not returned in the view, they cannot be used as a value when selecting a row (specifying values for a GET will map those values to KEY and NON-UNIQUE KEY columns, skipping any CONST columns. See “[Keyed access to data](#)” on page 83 for more information.

column-name=

- Restriction** This parameter is for base views only.
- Description** *Optional.* In base views, assigns a column name to the logical data item. The column name will then be used by derived views, application programs, and SPECTRA. In derived views, it assigns another name to the source column, to be later used by application programs and SPECTRA.
- Format** 1–26 alphanumeric characters and hyphens. The first character must be alphabetic. The last character cannot be a hyphen. Hyphens cannot be consecutive characters.

Considerations

- ◆ Use this option to assign a column name that will be more descriptive or meaningful when used in the application.
- ◆ Column names need only be unique within the view.
- ◆ If you do not specify a column name, RDM uses the logical data item name for base views, and the source-column-name in derived views.
- ◆ You must use a column name when the column name is equal to multiple logical data item names (base view) or multiple source column names (derived views). Otherwise, RDM will use the first name as the column name. For example:

```
PRODUCT-CODE = STOCK-PRODUCT = PRODUCT-ID
(column-name) (logical name) (logical name
                for STCKPROD)    for PRODCTRL)
```

logical-data-item

- Restriction** This parameter is for base views only.
- Description** *Required.* Identifies the logical data item that will be associated with the column being defined.
- Format** 1–26 alphanumeric characters and hyphens. The first character must be alphabetic. The last character cannot be a hyphen. Hyphens cannot be consecutive characters.

Considerations

- ◆ The logical data item definition must already exist on the SUPRA Server Directory database SUPRAD.
- ◆ You can use physical data item names instead of logical data item names when using DBAID to define a test view.
- ◆ If you use physical data items in the column definition in DBAID, you will be unable to save the view on the SUPRA Server Directory database SUPRAD.
- ◆ If you do not specify a column name, RDM uses the logical data item name for base views, and the source-column-name in derived views.

[[]=logical-data-item₂...]...[[]=logical-data-item_n]

- Restriction** This parameter is for base views only.
- Description** *Optional.* Specifies one or more logical data items that will map to a single column in the view (see Examples 3 and 4 at the end of this section).
- Format** 1–26 alphanumeric characters and hyphens. The first character must be alphabetic. The last character cannot be a hyphen. Hyphens cannot be consecutive characters.

Considerations

- ◆ This is a convenient method of mapping the same value to many logical data items in the physical database.
- ◆ If you specify multiple logical data items, you must specify a column-name or RDM uses the first logical data item name as the overall column name, which may result in unexpected behavior. See the description of column name.
- ◆ If the column is designated as a KEY, REQ, CONST, or NON-UNIQUE KEY, all logical data items specified will have the same constraint.
- ◆ RDM will access the logical data items using the order of the data sets in the ACCESS statements (see “[Access definitions](#)” on page 50), which does not have to be the same order that the logical data items are specified on this statement.
- ◆ When using GET to retrieve a row, the values of the columns in the view will be those of the last column accessed. The only exception is if the column is a KEY, CONST, or NON-UNIQUE KEY with the key value given. In this case, RDM compares each redundant column with the key value before returning the row.
- ◆ Logical data items must be from the same domain unless a domain override is specified. To override normal domain checking, include an additional equal sign as shown below:

```
column-name-a = = logical-name-a = logical-name-b
```

source-column-name

- Restriction** This parameter is for base views only.
- Description** *Required.* Indicates the name of the base view column being accessed.
- Format** 1–26 alphanumeric characters and hyphens. The first character must be alphabetic. The last character cannot be a hyphen. Hyphens cannot be consecutive characters.
- Consideration** The column must already exist in the view being accessed.

[[=]source-column-name₂]...[[=]source-column-name_n]

- Restriction** This parameter is for derived views only.
- Description** *Optional.* Specifies one or more base view columns that will map to a single column in the derived view.
- Format** 1–26 alphanumeric characters and hyphens. The first character must be alphabetic. The last character cannot be a hyphen. Hyphens cannot be consecutive characters.

Considerations

- ◆ This is a convenient method of mapping the same value to many columns.
- ◆ If you specify multiple source-column-names, you must specify a column name. See the description of column name.
- ◆ If the column is designated as a KEY, REQ, CONST, or NON-UNIQUE KEY, all column names specified will have the same constraint.
- ◆ The columns are accessed according to the order the base views are specified in the ACCESS statements (see “[Access definitions](#)” on page 50) which does not have to be the same order that the source column names are specified in this statement.
- ◆ When using GET to retrieve a row, the values of the columns in the view will be those of the last column accessed. The only exception is if the column is a KEY, CONST, or NON-UNIQUE KEY with the key value given. In this case, RDM compares each redundant column with the key value before returning the row.
- ◆ The columns must be from the same domain unless a domain override is specified. To override normal domain checking, include an additional equal sign as shown below:

```

REQ REGION-NUMBER = = BRANCH-REGION = REGION-NUMBER
(column name)      (source column   (source column
                    name from the    name from the
                    base view        base view
                    BRANCH)         REGION)
    
```

=constant

Description *Required* with CONST. Specifies the value to be assigned as a constant for this column.

Format You can specify the value as:

- ◆ X followed by hexadecimal digits enclosed in single quotes;
X'nnnnnn'
- ◆ Numeric characters (binary, packed, numeric, or floating point);
nnnnnnnnnn
- ◆ Alphabetic characters enclosed in single quotes; *'cccccc'*

Considerations

- ◆ The length of the value depends on the length of the column being defined.
- ◆ If the view is to be saved on the Directory, the length, including quotes, must not exceed 24 characters.

General considerations

- ◆ Any columns defined as KEY, REQ, CONST, or NON-UNIQUE KEY are required columns for the view. See the description of REQ on a column definition.
- ◆ More than one column in a row can be a logical key; however, the keys are treated as a compound key for that row.
- ◆ In a base view, if a related data set contains both a key column and a non-unique key column, the PDM processes the data set as if both were non-unique keys. A non-unique key makes the combination of keys non-unique.
- ◆ In a derived view, if a view contains two columns and one is declared a key and the other a non-unique key, the view will be processed as if both were non-unique keys. This is because a non-unique key makes the entire combination of keys non-unique.
- ◆ Column definition statements must precede the access definition statements.
- ◆ Column definition statements do not have to be in any particular order and need not correspond to the access path specified in the access definitions.
- ◆ A constant value must pass any validity checking required and cannot be the same as the null value. See “Null values” on page 98 for a description of null values.

Examples

- ◆ The column definition for this view indicates a stock-product row that may have multiple product values for each branch.

```
PRODUCTS-IN-BRANCH
```

```
0005 KEY BRANCH-NUMBER = CUSTOMER-BRANCH
0010 NON-UNIQUE KEY PRODUCT-CODE = STOCK-PRODUCT
0015 DESCRIPTION = PRODUCT-DESCRIPTION
```

- ◆ This view returns data about branches in Region Number 111:

```
REGION-111-INFO
0005 KEY BRANCH-NUMBER
0010 CONST REGION-NUMBER=BRANCH-REGION=111
0020 BRANCH-NAME
0025 BRNCH-CITY
0030 BRANCH-STATE
```

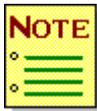
- ◆ The following example uses multiple fields (logical data items for base views; source column names for derived views) to define the column.

```
BRANCH-NUMBER = BRANCH-ID = STOCK-BRANCH = CUSTOMER-BRANCH
```

- With a GET, the value returned in BRANCH-NUMBER depends on which column (BRANCH-ID, STOCK-BRANCH, or CUSTOMER-BRANCH) is accessed last. RDM does not guarantee that these values are equal in this case.
- An INSERT of a value into BRANCH-NUMBER may insert the same value into BRANCH-ID in BRAN, STOCK-BRANCH in STOCK, and CUSTOMER-BRANCH in CUSTOMER on the physical database. See “[View design considerations](#)” on page 73.
- With an UPDATE, a change in BRANCH-ID will update BRANCH-NUMBER, STOCK-BRANCH, and CUSTOMER-BRANCH.

- ◆ The following example uses multiple fields (logical data items or source column names) to define the column as a logical key.

```
KEY BRANCH-NUMBER = BRANCH-ID = STOCK-BRANCH = CUSTOMER-BRANCH
```



This example is for explanation only. Because the fields are physical keys, an UPDATE actually would not be allowed.

All three columns will be treated as keys:

- With a GET, you will retrieve only those rows that have BRANCH-ID, STOCK-BRANCH, and CUSTOMER-BRANCH equal to the value given for BRANCH-NUMBER in the USING phrase. If no key value is supplied on the GET command, RDM does not guarantee that these values are equal as above.
- An INSERT of a value into BRANCH-NUMBER may insert the same value into all three columns (BRANCH-ID, STOCK-BRANCH, and CUSTOMER-BRANCH).

Access definitions

Following the column definitions in the view are the ACCESS statements. These statements describe the navigation RDM uses to find data in the PDM data sets or through the base views. The rest of this section describes the format of the ACCESS statements for both base and derived views.



For a discussion of important considerations in defining the ACCESS statements, see “[View design considerations](#)” on page 73.

In base views the access definition tells RDM how to navigate from one physical record to the next. Physical records can be in the same data set, or in a different data set.

In derived views the access definition describes how to navigate from view to view, how to access base and derived views, and the relationships between the views accessed. Different relationships between physical records determine how RDM navigates from one to the other; an inappropriate navigation technique degrades the overall efficiency of your applications. See “[Navigational constraints and boundary conditions](#)” on page 78 for an explanation of database navigation.

The access definition defines the physical navigation of the database and the maintenance operations allowed through this view. You may enter the optional clauses in any order. However, the order of the ACCESS statements as a whole is important because it controls the order in which the data sets are accessed.

You can use the ACCESS statement syntax to specify precisely which access method RDM should use. Alternatively, you can use a generalized form of the ACCESS statement, and RDM selects the optimum access strategy according to data set type. Both forms of the syntax are presented here:

- ◆ The first syntax format shows the generalized form of the syntax for base views that consists of an ACCESS statement and the WHERE clause.
- ◆ The second syntax format shows how the specific form of the ACCESS statement for base views varies depending on the data set type and access method desired.
- ◆ The third syntax format shows the generalized ACCESS syntax for derived views.
- ◆ The fourth syntax format shows RMS data set ACCESS syntax.

Once RDM opens a view and selects the access strategy, it uses that strategy until the view is released and re-opened. If the view is bound or global, RDM uses the selected access strategy until the view is rebound or reglobalized because RDM cannot switch access methods at run time.

For example, if RDM opens a view to access data using a secondary key and this key subsequently becomes unavailable because it has been de-populated, de-activated or corrupted, the view no longer works.

For views that are not bound or global, you can build logic in your program to release and re-open the view should the access method fail.

When you use the generalized form of the access method (see the first syntax format below), you need to be careful that RDM is not scanning the files when you do not want it to. RDM status indicators were designed to provide information about the operation of your views (see “[RDM status indicators](#)” on page 223). Also, the DBAID SHOW-NAVIGATION command allows you to check that RDM is using the correct access methods. See “[Defining and testing views using DBAID](#)” on page 135 for information about the DBAID commands STATS and SHOW-NAVIGATION.

The following shows the generalized ACCESS syntax for base views:

**ACCESS *data - set - name* [(*record - code* [;
: *record - code*]
:
+]]**

**[ONCE
SCAN]**

[REVERSE]

**[WHERE *column*₁ [=] *selection-criteria* [... AND *column*_{*n*}
 [=] *selection-criteria*]]**

[ALLOW [SHARED] [ALL] [INS] [DEL] [REP] [UPD]]

[GIVING *column*₁ [...*column*_{*n*}]]

The following shows the specific ACCESS syntax for base views:

```
ACCESS data - set - name [ ( record - code [ ;  
: record - code ]  
:  
+ ] ]
```

```
[ ONCE  
SCAN ]
```

```
[ REVERSE ]
```

```
VIA { linkpath  
      { secondary - key } }
```

```
[ USING item1, item2, ...itemn ]
```

```
[ WHERE column1 [=] selection - criteria [... AND columnn  
      [=] selection - criteria ] ]
```

```
[ ALLOW [ SHARED ] [ ALL ] [ INS ] [ DEL ] [ REP ] [ UPD ] ]
```

```
[ GIVING column1 [...columnn ] ]
```

```
[ ORDER [ column - name [ DESCENDING ] ] [ FIRST  
      NEXT  
      PRIOR  
      LAST ] ] ]
```

The following shows the generalized ACCESS syntax for derived views:

```
ACCESS view - name
```

```
[ USING ( item1, item2, ...itemn ) ]
```

```
[ WHERE column1 [=] selection - criteria [... AND columnn  
      [=] selection - criteria ] ]
```

[GIVING *column-1* [...*column-n*]]
 [ALLOW [ALL] [INS] [DEL] [REP] [UPD]]

The following shows RMS data set ACCESS syntax:

ACCESS *data-set-name*
 [VIA (*rms-key-name*)]
 [USING (*value*₁, *value*₂, ... *value*_n)]
 [ONCE]
 [WHERE *column*₁ [=] *selection-criteria* [... AND *column*_n
 [=] *selection-criteria*]]
 [ALLOW [SHARED] [ALL] [INS] [DEL] [REP] [UPD]]
 [GIVING *column*₁ [...*column*_n]]

The following are descriptions for the parameters used in the ACCESS statements.

ACCESS

Description *Required.* Identifies the statement as an access definition for the view.

data-set-name

Restriction This parameter is for base views and RMS data sets only.

Description *Required.* Identifies the data set to be accessed.

Format 4 alphanumeric characters. The first character must be alphabetic.

Consideration The data set description must already exist on the Directory.

view-name

- Restriction** This parameter is for derived views only.
- Description** *Required.* Identifies the base view to be accessed.
- Format** 1–26 alphanumeric characters and hyphens. The first character must be alphabetic. The last character cannot be a hyphen. Hyphens cannot be consecutive characters.
- Consideration** It must be a valid view defined on the Directory or a virtual view that has been opened using DBAID.
-

(record - code $\left[\begin{array}{l} ; \\ ; \\ ; \\ : \\ + \end{array} \right]$ **:record - code)**

- Restriction** This parameter is for base views only.
- Description** *Optional.* Specifies the record-code(s) to be retrieved from the coded record related data set specified in data-set-name.
- Format** Two-character record-code, as previously defined on the SUPRA Server Directory. The operator has a shorthand method available for specifying record-code processing. You can use this method either in place of the FROM clause or in conjunction with it.
- Options** The valid operators are:
- , Specifies an "or" condition. For example: (HD,CM) reads the data set until either an HD or a CM record is found.
 - ; Specifies embedded many-to-one relationships. For example (IT;HD) scans the related data set for a row with the HD record-code, starting from the row with the IT record code.
 - : Specifies embedded one-to-many relationships. For example (HD:IT) retrieves many IT coded records for each HD coded record retrieved.
 - + Specifies record-code groups and indicates a one-to-one relationship. For example: (HD+CM) retrieves only one CM coded record for each HD coded record retrieved. The specification also refuses to insert more than one CM coded record for each HD coded record.

Considerations

- ◆ You can specify combinations of record-codes and operators. For example, (HD+SH:IM) indicates that for every Header (HD) there is an associated Shipment (SH) and many Items (IM).
- ◆ The "or" operator is not recommended because users of the view need to know which physical data items are active under which record-codes. You should inform users if you rearrange the physical data items in the record-codes.
- ◆ Record-code specification depends on data set and record relationships. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for more information on coded records.
- ◆ The maximum number of record codes accessible from a single view is nine.

FROM *data-set* [(*record-code*)]

Restriction This parameter is for base views only.

Description *Optional.* Indicates that the named related data set (following ACCESS) is to be retrieved from a previously obtained occurrence of the data set specified in this FROM clause. This overrides the default navigation from primary to related data sets and allows related-to-related data set navigation.

Format

FROM	Specified as shown.
<i>data-set</i>	Data set name already existing on the Directory.
(<i>record-code</i>)	Two character record-code already on the Directory. The record-code must be enclosed in parentheses.

Consideration This statement is for related data sets only.

**[ONCE
SCAN]**

- Restriction** This parameter is for base views only.
- Description** *Optional.* Specifies how data is to be retrieved from the data set.
- Options** ONCE. Indicates that only the first related data set row will be retrieved from the data set. Establishes a one-to-one relationship between the previous data sets accessed and this one.
- SCAN. Indicates that the chain is to be scanned for the indicated data based on generalized selection logic and/or record code specifications, if present. Normally the scan indicates that a one-to-many relationship exists within the logical data.

[ONCE]

- Description** *Optional.* Indicates that only the first row will be retrieved.

[REVERSE]

- Restriction** This parameter is for base views only.
- Description** *Optional.* Indicates that the linkpath chain or secondary key is to be read in reverse order.
- Consideration** For secondary keys, this parameter is only available when REVERSE DIRECTIONAL search is enabled.

VIA { *linkpath*
secondary - key }

Restriction This parameter is for base views only.

Description *Optional.* Specifies how RDM should access the data set.

Considerations

- ◆ The VIA clause is allowed on PDM and RMS data sets.
- ◆ If you specify a secondary key name (dsetSKnn) on the VIA clause, you must specify the column that maps onto the physical data item connected to the secondary key on the USING clause so that RDM can perform a keyed read.
- ◆ When you're not using indexing (that is, secondary keys), the VIA clause applies to related data sets and cannot be used on the first ACCESS statement in a view.
- ◆ When you are not using indexing (secondary keys), the VIA clause applies to related data sets and defaults to a one-to-many relationship that meets all logical constraints (required columns).
- ◆ If you are using indexing (secondary keys) RDM will return an OPEN error if the index is not populated or activated.
- ◆ The name specified must be a valid linkpath or secondary key already defined on the Directory database SUPRAD.

VIA (*rms-key-name*)

Restriction	This parameter is for RMS data sets only.
Description	<i>Optional.</i> Specifies the index field used to access the RMS data set. Indexes can be either unique or non-unique. The default analysis of an alternate index will be to create a one-to-many relationship between the data set and its parent.
Format	The name must be a valid eight character name defined on the SUPRA Server Directory as an RMS key name.

Considerations

- ◆ You can use the VIA clause with the USING clause.
- ◆ You can use the VIA (*rms-key-name*) clause on the first ACCESS statement in a view.
- ◆ Excluding the VIA clause implies use of the first primary key defined by the KEY statement in the column definition.
- ◆ The RMS key named in this statement must be defined on the SUPRA Server Directory where it is related to a particular data item also defined on the SUPRA Server Directory. This relationship specifies the displacement within the record of the record key.
- ◆ The length of the RMS key column is defined separately from (and may or may not equal) its associated column. By specifying a length greater than that of the associated column, you can use a compound key (mapping to more than one contiguous column). Use generic keys by defining an RMS-key-name length less than that of the associated column length.
- ◆ RDM does not support non-contiguous key access to RMS data sets.

USING (*item*₁, *item*₂, ...*item*_n)

- Restriction** This parameter is for base views only.
- Description** *Optional.* Indicates that the primary data set is to be directly read using the specified key-column as the control-key.
- Format** The key-column may be a physical data item, logical data item, column, constant, or it may be constructed at run time from multiple columns and constants by using parentheses. For example:

```
ACCESS PRODUCT USING (COLUMN1, '252', COLUMN2)
```

Considerations

- ◆ While not a requirement, it is recommended that you also use these clauses on base view ACCESS statements.
- ◆ Compound physical keys are constructed from parts of the physical key as defined on the SUPRA Server Directory. Each part of the compound physical key must correspond to the sub-defined part of the key as specified on the Directory.
- ◆ If you specified a secondary key name on the VIA clause, the column you specify on the USING clause must be connected to that secondary key.

USING (*item*₁, *item*₂, ...*item*_n)

- Restriction** This parameter is for derived views only.
- Description** *Optional.* Indicates that a logical read using specified value(s) is to be done on the view, mapping values to logical keys.
- Format** Each value may be a column or constant.

Considerations

- ◆ You must use WHERE and/or USING clauses on derived view ACCESS statements. While not a requirement, it is recommended that you also use these clauses on base view ACCESS statements.
- ◆ Values specified must map to the logical keys of the accessed view. When specifying several values, you can omit values from the right-hand side of the group of values. Sub-definitions of logical keys are not allowed. For example, if you had the following base view, VIEW-A, defined:

```
VIEW-A

KEY COLUMN1
KEY COLUMN2
KEY COLUMN3
```

the following derived view, accessing VIEW-A, is valid:

```
VIEW-B

KEY COLUMN-A = COLUMN1
KEY COLUMN-B = COLUMN2
KEY COLUMN-C = COLUMN3
ACCESS VIEW-A
    USING (COLUMN-A, COLUMN-B)
```

- ◆ However, the following view, VIEW-C, is not valid because values were omitted, and they do not map to logical keys, from left to right:

```
VIEW-C

KEY COLUMN-A = COLUMN1
KEY COLUMN-B = COLUMN2
KEY COLUMN-C = COLUMN3
ACCESS VIEW-A
    USING (COLUMN-A, ,COLUMN-B)
```

- ◆ You could correct VIEW-C using either of the following:

```
WHERE (COLUMN1 = COLUMN-A) AND (COLUMN3 = COLUMN-B)
```

or

```
USING (COLUMN-A) WHERE (COLUMN3 = COLUMN-B)
```

- ◆ If the view that you are accessing has only one logical key, you can omit the parentheses.

USING (*item*₁, *item*₂, ...*item*_{*n*})

Restriction This parameter is for RMS data sets only.

Description *Required* (except on the first ACCESS statement). Directly reads the RMS indexed sequential data set using the specified RMS key as the control-key.

Format The key-column can be a logical data item, physical data item, column or constant, or it can be constructed at run time from multiple columns and constants by using parentheses. For example:

```
ACCESS SAMPLE USING (CUSTOMER-NUMBER, '252', ORDER-PRODUCT-CODE)
```

Considerations

- ◆ You can use the USING clause with the VIA clause for RMS data sets. In this case, the key field is the alternate key rather than the BASE key.
- ◆ RDM constructs compound physical keys from parts of the physical key. Each part of the compound physical key must map to the sub-defined part the key as specified on the Directory.
- ◆ If the Directory definition of the physical key field is

01	SAMPCTRL	15
02	SAMPCSTN	05
02	SAMPORIX	04
02	SAMPPRIX	06
03	SAMPPRD1	02
03	SAMPPRD2	04

you can specify the following compound key definitions:

```
USING ORDRCTRL
USING (SAMPCSTN, SAMPORIX, SAMPPRIX)
USING (SAMPCSTN, SAMPORIX, (SAMPPRD1, SAMPPRD2))
```

- ◆ You can force a generic read at RDM level by leaving columns out of the USING clause for a compound key. You can omit columns from the right only, but those column names you supply must be the same length as the columns in the sub-definition of the physical key. Using the previous example, you can specify a compound key as follows:

```
USING (SAMPCSTN, SAMPORIX)
```

- ◆ Only the first two parts of the compound key are supplied to force a generic read. You can use generic reads only from within RDM based on the logical keys and the USING clauses in the RDM ACCESS statements. Application programmers cannot specify the number of characters in any column used for a generic read. You must supply the partial key used for the generic read in parentheses, even if you use only one column.
- ◆ You can use the USING clause to establish a one-to-many relationship between multiple RMS data sets on RMS data sets and PDM data sets.

WHERE $column_1$ [=] selection criteria [... AND $column_n$ [=] selection-criteria]

Description *Optional.* The WHERE clause, used with the USING clause, provides additional selection criteria (specified access). If you use the WHERE clause without the USING clause, RDM selects the optimum access strategy (generalized access).

For base views, the optimum access strategy for PDM data sets is chosen in the following manner:

	Primary	Related	RMS
1st Choice	Control key	Linkpath	Control key
2nd Choice	Secondary key	Secondary key	Alternate key
3rd Choice	Sequential scan	Sequential scan	Sequential scan

For RMS data sets, when you use the WHERE clause without the USING clause, RDM selects the optimum access strategy as follows:

- ◆ Initially, RDM tries to map to the full control key value.
- ◆ If no match, RDM tries to map to an alternate key on the data set.
- ◆ If there is no match, RDM scans the data set.

Format

WHERE Specified as shown.

$column_{1..n}$ For base views and RMS data sets, $column_1$ must map to a data item in the data set.
For derived views it must be a column in the base view named in the ACCESS statement.

$[=]$ Specifies an equal comparison between the column and the value. You must use the double equals = (domain override) if both $column_1$ and value are connected to different domains. If $column_1$ and value are connected to the same domain, or if only one is connected to a domain, you need only enter the single equals.

selection criteria Specifies the value that $column_{1..n}$ must match. In a base view it may be a physical data item (only in a virtual DBAID testing view), logical data item, column, or constant. In a derived view it can be a column from a previously accessed view (not necessarily a column in this view) or a constant.

AND Optional. Allows you to specify additional qualifications for the ACCESS statement.

Considerations

- ◆ You must use the WHERE and/or the USING clause when defining the ACCESS statements. While not a requirement, it is recommended that you use these clauses on the first ACCESS statement in a view.
- ◆ Column₁ and selection criteria must be the same length.
- ◆ Use the VIA or USING clause to force alternate index or linkpath selection in base views.
- ◆ Use RDM statistics to measure performance of the derived view.
- ◆ Use of the WHERE clause without the USING clause is called generalized access. Generalized access allows RDM to select the most efficient method of retrieving data.
- ◆ You only specify the domain override (= =) once, between the first and second columns.
- ◆ Secondary keys must be listed in the view in the same order as they are found in the physical file.

ALLOW [SHARED] [ALL] [INS] [DEL] [REP] [UPD]

Description	<i>Optional.</i> Specifies which physical actions are allowed.
Format	Any combination of options is valid, for example: ALLOW INSERT DELETE Allows inserts and deletes but not updates. ALLOW UPDATE Allows updates but not inserts and deletes.
Options	SHARED Allow shared update on data items in a data set. Does not check "C" ASI status. Not for derived views. ALL Allow all forms of database modification. INS Allow row insertions on the database. DEL Allow row deletions from the database. REP Allow row replacements on the database. UPD Same as REP.

Considerations

- ◆ You can use the ALLOW parameter on as many ACCESS statements as required.
- ◆ If you omit this parameter, the data set is restricted to read-only processing.
- ◆ These options relate to physical I/O on the data set. They do not relate to the application program's Relational Data Manipulation Language (RDML).
- ◆ Using SHARED has no impact on the INSERT option. However, using SHARED causes RDM to skip the column comparison usually done by the automatic record holding facility before deletion and replacement of rows.
- ◆ For derived views the ALLOW clause on this ACCESS statement may not override any constraints imposed by the base or derived view that you are accessing, but it may further restrict the allowed actions.

GIVING *column*₁ [...*column*_n]

- Description** *Optional.* Overrides normal data movement. The data movement is physical data item to column in base views, and base column to derived column in derived views.
- Format** The keyword GIVING followed by zero or more column names as defined on the column definition.

Considerations

- ◆ All columns that can be filled by a particular ACCESS statement will be filled unless a GIVING clause restricts this process. Columns not filled can be filled by later ACCESS statements.

If you omit this clause in a derived view, all columns derived from columns in the accessed base view that have not been supplied by some previous ACCESS statement are filled with values using this ACCESS statement.

- ◆ If you omit column names on the GIVING parameter, no column values are obtained by accessing this data set or base view. The data set or base view will be used for navigation only.
- ◆ Using this clause lets you access a data set or base view more than once and retrieve only selected columns each time.

ORDER [*column - name* [DESCENDING]]

FIRST
NEXT
PRIOR
LAST

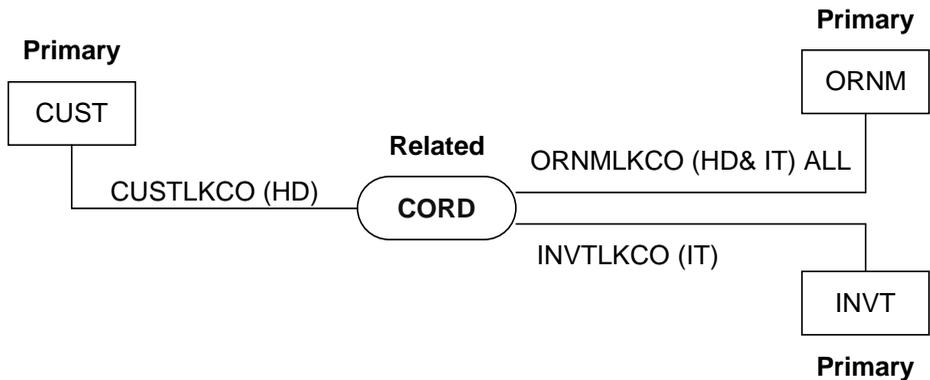
- Restriction** This parameter is for base views only.
- Description** *Optional.* Indicates that a predetermined ordering criterion applies to the related data set named in the ACCESS statement.
- Default** If you specify the ORDER clause, ordering defaults to ascending unless you specify descending. If you do not specify ordering in the ACCESS statement, the application program can use the FIRST/NEXT/PRIOR/LAST option on INSERT RDML to control ordering of the physical insertion of rows. However, this practice is *not* recommended.
- Format** The column name must be a logical data item defined in the view.

Considerations

- ◆ It is more efficient to use secondary keys to retrieve non-key columns in order than to use the ORDER clause.
- ◆ You cannot use the ORDER clause without the VIA clause.
- ◆ DBA unload/reload may be difficult or impossible for ordered data sets.
- ◆ Use the ORDER clause to retrieve rows; you must also use the ORDER clause to insert rows. If you do not insert rows in order, RDM will store them in logical sequence according to the INSERT option.
 - If the column type is a unique or non-unique key, and you use the GET USING command to retrieve a row, RDM will return the message OCCURRENCE NOT FOUND when it encounters the first row that is out of sequence. The row you are searching for may exist.
 - If the column is a unique or non-unique key, and you use the GET command without the USING clause to retrieve a row, RDM will ignore the ORDER clause and retrieve the rows in the order in which it inserted them.
 - If the column is not a key, RDM will ignore the ORDER clause and retrieve the rows in the order in which it inserted them.
- ◆ For record-codes, the linkpath and column used for ordering can be changed from ACCESS statement to ACCESS statement. Ordering information is used for the GET USING and INSERT functions.
- ◆ For INSERT, the specified ordering criterion is always used. The chain is entered on the basis of the ordering criterion, and the row is inserted at that position, regardless of any ordering keyword in the RDML statement. When inserting a row, the column in the ORDER clause is required (see definition of REQ in “[Column definitions](#)” on page 36).
- ◆ The ORDER clause makes two levels of positioning available. If a column is supplied in the ORDER clause, positioning of FIRST/NEXT/LAST/PRIOR applies only to multiple occurrences of the value for that column. If no column is supplied in the ORDER clause, positioning FIRST/NEXT/LAST/PRIOR applies to the entire chain for that data set. In both cases, the ORDER clause overrides FIRST/NEXT/LAST/PRIOR used with the RDML INSERT command.

- ◆ If the row is currently not positioned in the sequence of ordered columns, the NEXT function acts as LAST, and the PRIOR function acts as FIRST (placing it at the start of the sequence of ordered columns).
- ◆ The position on the linkpath is also affected by the REVERSE option because this accesses the linkpaths of related data sets in the reverse direction.

Examples The following examples give column and access definitions for the database shown in the following illustration:



- ◆ This view maintains and reports on basic customer information:

CUSTOMER VIEW

```

KEY CUSTOMER = CUSTOMER-NUMBER
NAME         = CUSTOMER-NAME
ADDRESS      = CUSTOMER-ADDRESS
CITY         = CUSTOMER-CITY
STATE        = CUSTOMER-STATE
ZIP          = CUSTOMER-ZIP
PHONE        = CUSTOMER-PHONE

ACCESS CUSTOMER
USING CUSTOMER
ALLOW ALL
  
```

- ◆ This view maintains and reports on basic ITEM information:

ITEMS VIEW

```
KEY ITEM = ITEM-CODE
ITEM-DESCRIPTION
ITEM-COST
ITEM-PRICE
ITEM-ON-HAND
ACCESS INVT
USING ITEM-CODE
ALLOW ALL
```

- ◆ This view allows the addition, modification and deletion of customer orders and the items included in the order:

ORDERS VIEW

```
KEY ORDER-NUMBER = ORNM-ORDER-NUMBER
CUSTOMER = CORD-CUSTOMER-NUMBER
AMOUNT = CORD-ORDER-TOTAL
ORDER-DATE = CORD-ORDER-DATE
SHIP-DATE = CORD-SHIP-DATE
KEY ITEM-NUMBER = CORD-ITEM-NUMBER
ITEM-QUANTITY = CORD-ITEM-QUANTITY
ITEM-PRICE = CORD-ITEM-PRICE
ACCESS ORNM
USING ORDER-NUMBER
ALLOW ALL
ACCESS CORD (HD)
ONCE VIA ORNMLKCO
ALLOW ALL
ACCESS CORD (IT)
FROM CORD (HD)
VIA ORNMLKCO
ALLOW ALL
ORDER ITEM-NUMBER
```

- ◆ This view provides access to orders on the basis of customers and would be used for reporting purposes only:

CUSTOMER-ORDERS VIEW

```
KEY CUSTOMER           = CUSTOMER-NUMBER
      NAME              = CUSTOMER-NAME
KEY ORDER-NUMBER      = CORD-ORDER-NUMBER
      AMOUNT           = CORD-ORDER-TOTAL
KEY ITEM-NUMBER       = CORD-ITEM-NO
      ITEM-QUANTITY    = CORD-ITEM-QTY
      ITEM-PRICE       = CORD-ITEM-PRICE
      ITEM-DESCRIPTION = INVT-ITEM-DESC
ACCESS CUSTOMER
      USING CUSTOMER
ACCESS CORD (HD)
      VIA CUSTLKCO
ACCESS CORD (IT)
      FROM CORD (HD)
      VIA ORNMLKCO
      ORDER ITEM-NUMBER
ACCESS INVT
      USING ITEM-NUMBER
```

- ◆ This view is a variation of Example 3. In this example, new CUSTOMER and INVT rows are automatically added by RDM if they are not already present when a new order is added:

ORDERS VIEW

```
KEY ORDER-NUMBER      =  ORNM-ORDER-NUMBER
CUSTOMER              =  CORD-CUSTOMER-NUMBER
AMOUNT                =  CORD-ORDER-TOTAL
ORDER-DATE            =  CORD-ORDER-DATE
SHIP-DATE             =  CORD-SHIP-DATE
KEY ITEM-NUMBER       =  CORD-ITEM-NUMBER
ITEM-QUANTITY         =  CORD-ITEM-QUANTITY
ITEM-PRICE            =  CORD-ITEM-PRICE
NAME                  =  CUSTOMER-NAME
ITEM-DESCRIPTION      =  INVT-DESCRIPTION

ACCESS ORNM
    USING ORDER-NUMBER
    ALLOW ALL

ACCESS CORD (HD)
    ONCE VIA ORNMLKCO
    ALLOW ALL

ACCESS CORD (IT)
    FROM CORD (HD)
    VIA ORNMLKCO
    ALLOW ALL
    ORDER ITEM-NUMBER

ACCESS CUSTOMER
    USING CUSTOMER
    ALLOW INSERT

ACCESS INVT
    USING ITEM-NUMBER
    ALLOW INSERT
```

3

View design considerations

When designing views, it is important that you understand the following:

- ◆ **Row construction** (see “[How RDM constructs rows](#)” on page 74)

Knowing how RDM constructs rows will help you define the access method best suited to your needs.

- ◆ **Keyed access** (see “[Keyed access to data](#)” on page 83)

Several types of keys are available for you to tell RDM how to access the data.

- ◆ **Integrity** (see “[Domains](#)” on page 96 and “[Referential integrity with RDM](#)” on page 106)

It is important to know how SUPRA Server handles domain attributes because RDM uses them to validate data and maintain data integrity (see “[Domains](#)” on page 96).

Another type of integrity is referential integrity, which ensures that two pieces of data representing the same fact do not become inconsistent (see “[Referential integrity with RDM](#)” on page 106).

- ◆ **Shared columns** (see “[Shared column values](#)” on page 126)

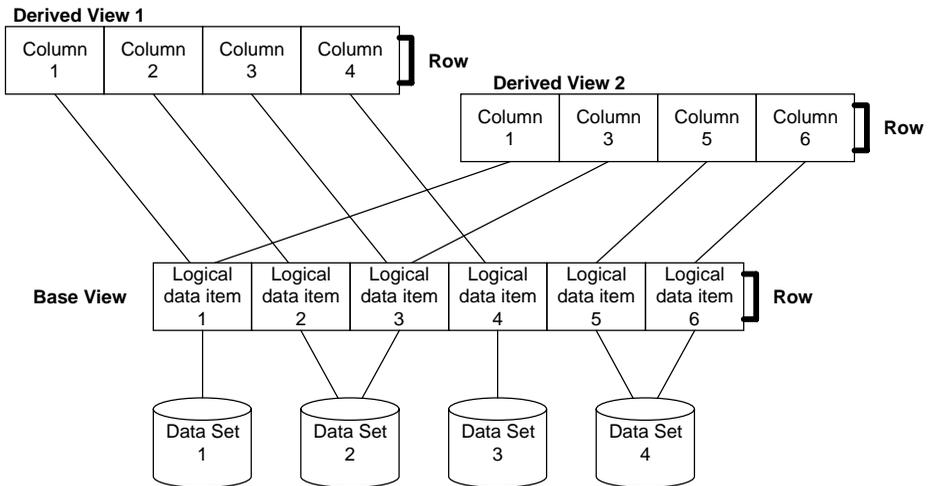
For base views, you can allow shared columns for efficient processing.

- ◆ **Security** (see “[View-to-user relationships](#)” on page 128)

You can control database security on a user-by-user basis by defining which users can use which views.

How RDM constructs rows

As discussed in “Views” on page 28, RDM presents data in a two-dimensional tabular format. RDM constructs the row based on the view definition by obtaining data from the data sets or base views named in the ACCESS statements. Rows created by RDM may be drawn from one or more data sets or base views and do not necessarily exist as a physical record anywhere on the database. The following illustration illustrates how RDM constructs rows:



When the RDM processes a GET, INSERT, UPDATE or DELETE command, it:

- ◆ Determines whether the view is a base or derived view.
- ◆ Opens it and all associated base views and data sets.
- ◆ Sets up the internal data structures for the application.

To the application there is no difference between a base and a derived view.

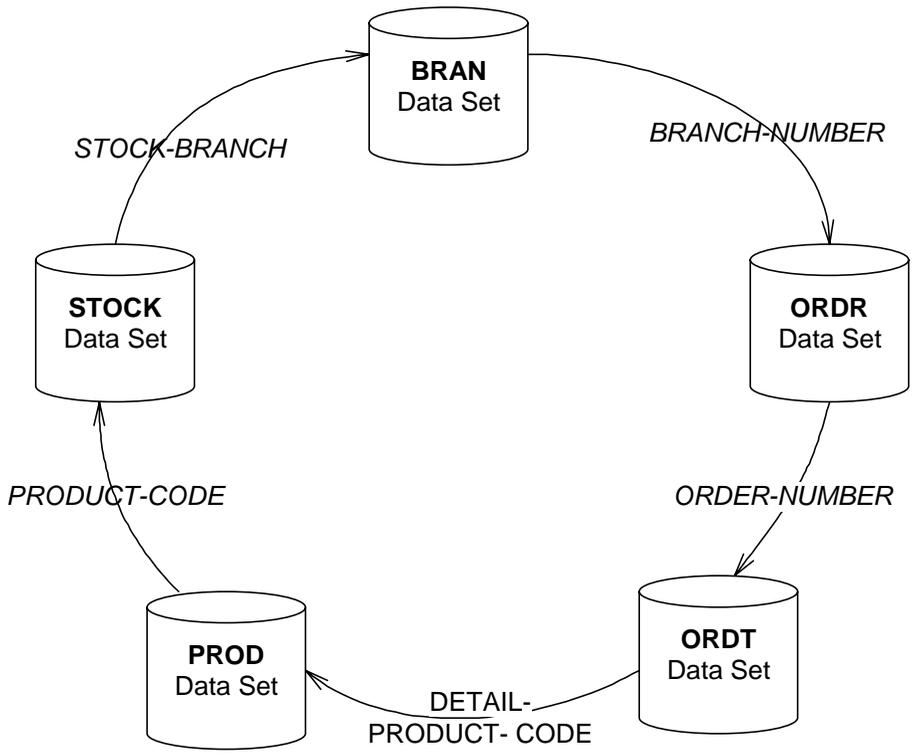
Database penetration

Database penetration is associated with the one-to-one keyed relationship. You penetrate (or access) the database by using a key value. At any time, you are pointing to one particular position in the database. Database penetration does not rely on anything you have previously done with the database. An example is retrieving a customer row using a particular customer number as the key.

You select database penetration by performing GET (with or without a key value), to establish your position within the database. Whether going from one or many data sets to another, using a key for retrieval implies a one-to-one keyed relationship from the source to the target.

Another kind of relationship is positional, implying location with relation to the position of other records instead of key value. For example, when retrieving records sequentially, getting the next physical record is a one-to-one positional relationship because the position of the first physical record finds the next. You do not use a key to get the next record. After the position is established based on the logical keys, you perform a positional GET without keys, or another "penetrating" GET with keys. A GET FIRST or GET LAST is guaranteed to access the database, while a GET NEXT or GET PRIOR is positional.

Penetration uses a base data set as the starting point, which simply retrieves the selected record in the data set specified by the first access definition in the view. From that record, you travel outward in one or more directions; each time you take a step, you can use that information to take additional steps. This would resemble a tree structure. However, navigation could also be circular. For example, you may want to search for all customers who have an account at a particular branch, regardless of any other branches they might patronize. The following illustration shows this circular navigation using some of the data sets described in the example database in [Appendix D](#) on page 305.



Database sweep

A database sweep involves taking a positional step either forward or backward. A database sweep occurs only on a one-to-many positional relationship. Examples of a database sweep are a primary-to-related data set relationship and a related-to-related data set relationship using record codes.

Sweeping occurs when you have already penetrated the database, that is, positioned yourself at a particular row or are accessing the database for the first time without using a key value (or index). You then move either forward or backward from your position; you can get either the next or the previous row. For example, in a logical view consisting of customer and orders, you first penetrate the database using the customer number. When you ask to read the first and subsequent orders, you rely on a database sweep based on positional relationships between the orders and the customer.



For performance reasons, we recommend that you use an index on a file instead of allowing database sweeping. For example, in the EXAMPL database listed in [Appendix D](#) on page 305, the order file contains an index on the customer number to minimize database sweeping.

When you sweep, there is an incremental movement, either forward or backward. You can sweep a data set without penetrating the database by starting at either the first or the last physical record (this is also called *scanning*).

Navigational constraints and boundary conditions

RDM enables you to identify points along the navigation path that must be reached for the navigation to be valid. Once those points are reached, you can also specify data items as required, causing navigation to be unsuccessful if the data item is not found.

Logical keys are always required. Therefore, when accessing a data set based on a logical key value that is not found, navigation stops. If, during database penetration, any required data items are not found, RDM returns a not-found status. However, if attempting a sweep through the database, RDM skips physical records that do not meet the constraints as if they do not exist. An example is the CUSTOMER-ORDER view that only returns rows for customers who have at least one order.

A boundary condition exists when you reach the end of a group of records. Scanning through a primary data set, the end of the data set is a boundary (see [Example 1](#)). If you are sweeping a related data set chain (based on a primary data set key), the end of the chain is a boundary (see [Example 2](#)). If, however, you scan the related data set without using a primary data set key, RDM gets the next primary data set record and navigates its associated related data set chain (see [Example 3](#)).

Examples two and three use these data sets:

BRAN		STCK			PROD	
3345	Cincinnati	3345	1122		1122	Wigits
1526	New England	1526			3500	
2221	Atlanta	3345			2500	
		2221				

Example 1: Boundary condition of end of primary data set. Using the CUSTOMER base view (see “[Example database](#)” on page 34 for view contents) your application just needs to scan the customers; none of the GET statements will include a key value for customer number. Instead, each GET will retrieve the next record in the file in physical order. In this case first customer number 111, then 222, then 333 and so on. The boundary condition is reached when the end of the file is reached. Note that you can use a secondary key on the customer data set if you want the information retrieved in a particular order, as has been done in the CUSTOMER view.

Example 2: Boundary condition for the end of related data set linkpath chain. Base View: BRANCH-PRODUCTS

```
KEY   BRANCH-NUMBER = BRANCH-ID
KEY   PRODUCT-CODE = STOCK-PRODUCT-ID = PRODUCT-ID
      PRODUCT-DESCRIPTION

ACCESS BRAN
      WHERE BRANCH-ID = BRANCH-NUMBER

ACCESS STCK
      WHERE STOCK-BRANCH-ID = BRANCH-ID

ACCESS PROD
      WHERE PRODUCT-ID = STOCK-PRODUCT-ID
```

This time you need to know all the products stocked in a particular branch. You do a database penetration by issuing a GET with the branch number as the key. Each subsequent GET will be a GET NEXT that will use the STCK related data set linkpath chain (BRANLKST) to get the order information for each order. The boundary condition is reached when there are no more products stocked by this branch (the end of the linkpath chain).

Example 3: Multiple boundary conditions. Using the BRANCH-PRODUCT view shown in example 2, you now need to get the order information for all customers. So, your application will scan the BRAN primary data set sequentially, sweeping the branch number linkpath in the STCK related data set. When the end of the related chain is reached for a branch number, the next branch record is read and all related records for that branch are found, and so on. You reach a boundary condition when you reach the last related record in the chain and when you reach the end of the BRAN data set.

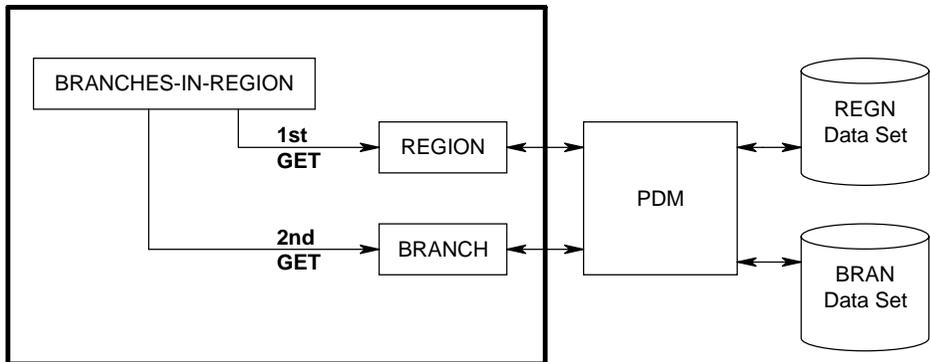
Processing derived views

Before you can use the RDML commands, GET, INSERT, UPDATE, and DELETE, the derived view must open the base view. Applications do not explicitly open a base view; it is opened on first use so that the view's internal data structure is available. Thus, opening a derived view results in opening one or more base views.

For example, when you open the PRODUCTS-IN-REGION derived view, the REGION, BRANCH, STOCK, and PRODUCT base views are also opened. In combination, these views can affect every data set in your physical database. After all views are opened, you can process the RDML commands.

Processing the GET command

When you issue a GET for the BRANCHES-IN-REGION base view, RDM issues a GET for the REGION base view, which causes a request on the REGN data set. If this operation returns data for the REGION base view, a GET is issued for the BRANCH base view. The GET results in a sweep of the BRAN data set searching for rows with the correct region-number. The following illustration shows the processing sequence.



Processing the INSERT command

This example uses the PRODUCTS-IN-REGION derived view to insert a new product into the stock of a branch in a given region.

Derived View: PRODUCTS-IN-REGION

View Text:

```

KEY REGION-NUMBER
    REGION-NAME
KEY BRANCH-NUMBER
    BRANCH-NAME
KEY STOCK-PRODUCT
    PRODUCT-DESC
ACCESS REGION
    WHERE REGION-NUMBER = REGION-NUMBER
    ALLOW UPDATE DELETE
ACCESS BRANCH
    WHERE BRANCH-REGION = REGION-NUMBER
    ALLOW ALL
ACCESS STOCK
    WHERE STOCK-BRANCH = BRANCH-NUMBER
        AND STOCK-PRODUCT = STOCK-PRODUCT
    ALLOW ALL
ACCESS PRODUCT
    WHERE PRODUCT-CODE = STOCK-PRODUCT

```

Because ACCESS statements in the REGION and PRODUCT base views do not allow INSERT, the REGION-NUMBER and STOCK-PRODUCT values must exist in the database before the INSERT can succeed. This derived view does not allow for insertion of new branches and stock into a branch without any restriction. The only reason to access the PRODUCT base view here is to provide the PRODUCT-DESC column. The integrity constraint between STOCK and PRODUCT (no STOCK-PRODUCT number is allowed that is not already in PRODUCT) is already defined in those base views.

Processing the UPDATE command

This view allows you to update existing branch names in a specified region, and to insert new branches. You cannot modify the region through this view.

Derived View: BRANCHES-IN-REGION

View Text:

```
KEY   REGION-NUMBER
      REGION-NAME
KEY   BRANCH-NUMBER
      BRANCH-NAME
ACCESS REGION
      ONCE
      USING REGION-NUMBER
ACCESS BRANCH
      WHERE BRANCH-REGION = REGION-NUMBER
      ALLOW INSERT UPDATE
```

Processing the DELETE command

This view keys into the REGN data set using the key REGION-NUMBER to delete a region. Any branches in the specified region are also deleted; the ACCESS statement to the BRAN data set has ALLOW DELETE. However, if any branch in the region contains customers, the entire delete operation is rejected. Neither the region nor any of its branches can be deleted if any one branch contains customers. To cascade delete the customers from the CUST data set, you would need to ALLOW DELETE on the ACCESS statement for the CUST data set as well.

Derived View: DELETE-REGION-WITH-NO-CUSTOMERS

View Text:

```
KEY   REGION-NUMBER = REGION-ID
REQ   BRANCH-NUMBER = BRANCH-ID
ACCESS REGN
      USING REGION-NUMBER
      ALLOW DELETE
ACCESS BRAN
      WHERE BRANCH-REGION-ID = REGION-ID
      ALLOW DELETE
ACCESS CUST
      WHERE CUSTOMER-BRANCH-ID = BRANCH-ID
```

Keyed access to data

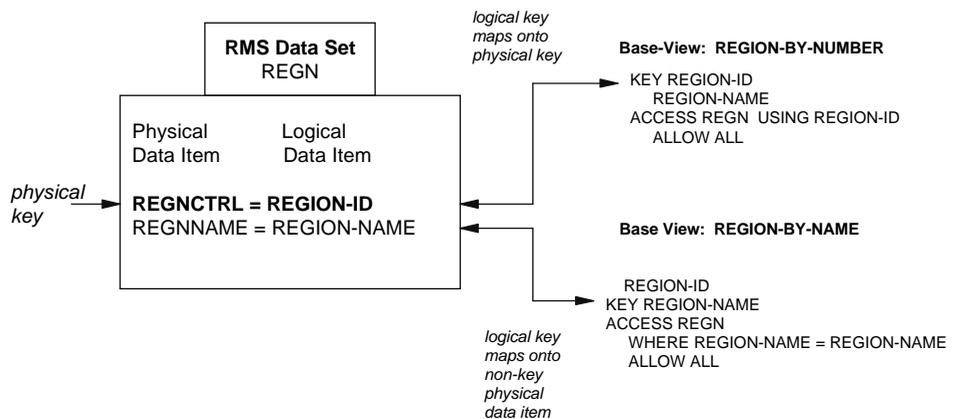
From the perspective of RDM, each apparently flat record described by a view has one or more logical keys. Each data set in the database has one or more physical keys. The logical key or combination of logical keys in your view may or may not map to a physical key.



Although it is possible to create a view that has no logical key, for performance reasons, it is *not* recommended.

You provide keyed access to data through the view definition. Each data set in the database can be accessed through a physical key, for example, customer number or part number. Provided you define a logical key that maps onto the physical key in the data set, RDM performs a physical keyed access. This is the most efficient way of accessing data.

If you do not provide keyed access (you either omit the key or define a logical key that does not map onto the physical key), a serial scan of the data set or view results. Even if a physical keyed read can be performed, you can still define a view that limits the data set to sequential access. The following illustration uses two base views to illustrate keyed and sequential access.



Base view REGION-BY-NUMBER has one logical key (KEY REGION-ID), which it uses as the logical key to the REGN data set (ACCESS REGN USING REGION-ID). Because this logical key maps onto the physical key REGNCTRL, RDM performs a keyed-read of the data set REGN. Physical keyed access provides good performance and is therefore useful for views that change the database (the RDML maintenance functions INSERT, UPDATE, and DELETE).

You can also access a data set through a secondary key. This is much more efficient than a serial scan, but not as efficient as using a physical control key. Secondary keys are returned in either ascending or descending order, or both, depending on the direction chosen during secondary key definition. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for details on how to define secondary keys.

Base view REGION-BY-NAME has one logical key (KEY REGION-NAME), which it uses as the logical key to the REGN data set (ACCESS REGN WHERE REGION-NAME = REGION-NAME). Because REGION-NAME does not map onto a physical key, RDM performs a sequential read of the data set REGN, scanning the physical file to locate the appropriate row. Scanning a data set is the least efficient method of access.

If, however, you define REGION-NAME as a secondary key in an index, the ACCESS statement ACCESS REGN WHERE REGION-NAME=REGION-NAME would cause RDM to use the index to obtain the data. This is much more efficient than a serial scan. You define indices and secondary keys during database definition. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260.

You can define a logical key two ways: (1) Using the keyword KEY (or NON-UNIQUE KEY) in the *column* portion of the view definition, or (2) Defining logical keys as part of the *access* portion of the view definition. The keys in the access portion of the view definition determine the access method to use.

In the following example (a derived view of the base view CUSTOMER), logical keys are placed on both CUSTOMER-NUMBER and CUSTOMER-NAME by defining the columns with the required KEY. CUSTOMER-NUMBER is defined as the logical key to the CUST data set on the ACCESS statement.

```
KEY CUSTOMER-NUMBER
KEY CUSTOMER-NAME
  CUSTOMER-ADDRESS
  CUSTOMER-STATE
ACCESS CUSTOMER
  WHERE CUSTOMER-NUMBER = CUSTOMER-NUMBER
ALLOW ALL
```

Each view can have zero to nine logical keys. Logical keys defined with the keyword KEY or NON-UNIQUE KEY can be used to supply any number of these key values for the view. A logical key in a view does not, by itself, cause RDM to perform a keyed access. A logical key enables the user to provide a value for the specified field if keyed access is desired. If you define a logical key that maps to a physical key on the database file (for example, a control key on a PDM primary data set), and the user program requests a read using that key, the access will work very quickly. RDM performs a keyed read of the data set and goes directly to the requested row. If, however, the user does not provide a value for that key, RDM performs a sequential read and treats the logical key as a required field.

Required columns are designated by the REQ option of the column definition and must be present for RDM to return a row. Imagine an example of customers and orders where each customer may have zero, one, or more than one order. If the order number is required, customer-order rows will be returned only if the customer has at least one order. If the customer number is required, all customer rows will be returned, regardless of whether the customer has any orders.

All keys, including logical keys, are always required columns. You can also assign fixed values or constants to impose constraints on the program and thus limit the application to retrieve or update selected rows.

You can define four different types of logical keys for a row: unique, non-unique, constant, and unique constant. You can define logical keys as unique or non-unique depending on your application requirements and record organization. However, if the logical key is also a secondary key that allows duplicates, there is no point defining it as a unique key in the view. The following sections give more details on these key types as well as information on designating columns as required or constant.

Unique keys

A unique key has one row for each key value. Remember that each row can ultimately map to one or more data sets. Therefore, using a unique key with unnormalized views may retrieve more than one row for each unique key.

When you define a simple or compound unique key (see “[Simple unique keys](#)” on page 86 and “[Compound unique keys](#)” on page 87), the program might not supply all the values. For example, if you define the customer number and order number as a compound unique key, the program can retrieve the row using zero, one, or two key values. In this way, the program can implement a generic read by specifying less than the total number of logical keys in the view. If the program specifies just customer number, RDM retrieves all orders for that customer.

If the logical key maps to the physical key of a data set that maintains uniqueness of the physical key, RDM will let the PDM maintain the uniqueness. If the column does not map to a unique key, RDM tries to keep the value unique by rejecting duplicate inserts.

Simple unique keys

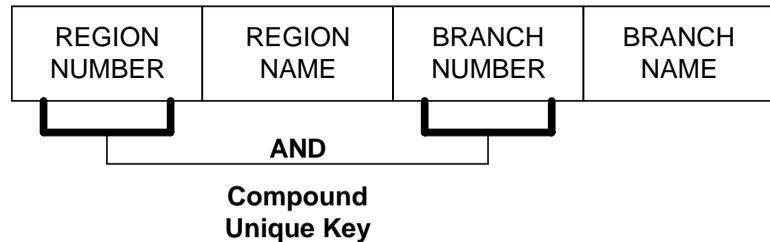
Think of a simple unique key as a selection criterion. It provides an equal comparison between a column and an application-specified value. An example of a simple unique key is the customer number, as shown in the following illustration. No two customers for a company should have the same customer number. Therefore, the key is unique.

CUSTOMER NUMBER	CUSTOMER NAME	CUSTOMER ADDRESS	CUSTOMER CITY	CUSTOMER STATE
--------------------	------------------	---------------------	------------------	-------------------


**Simple
Unique Key**

Compound unique keys

Another type of key is the compound unique key. In this case, more than one data item can be defined as a key with an "and" connection implied between the logical keys. An example of the compound unique key is illustrated in the following illustration.



Derived View: BRANCHES-IN-REGION

View Text:

```
KEY REGION-NUMBER
    REGION-NAME
KEY BRANCH-NUMBER
    BRANCH-NAME
ACCESS REGION
    ONCE
    USING REGION-NUMBER
ACCESS BRANCH
    WHERE BRANCH-REGION = REGION-NUMBER
ALLOW INSERT UPDATE
```

If your application includes both logical key values when issuing the GET, RDM will try to locate the row based on both values. RDM may do direct database penetration, scan or sweep navigation of the physical files, or some combination to find the specified row.

The key value of a compound unique key is the combination or concatenation of the logical key values. RDM keeps this combination unique. Using the BRANCHES-IN-REGION example, there may be several branches in a region, but there will be only a single row for the region number/branch number combination.

Non-unique keys

Non-unique keys differ from unique keys because RDM does not maintain the key value as unique. The column is still required, and the user can specify a value for the column to select rows. However, a single key value may return multiple rows. Do not confuse this with a generic search (see “**Generic reads**” on page 93), which may also return several rows for a given key value.

You can build views that have only unique keys, but still return several rows per unique key combination. This occurs when the unique logical keys do not uniquely specify a single row. Using the BRANCHES-IN-REGION view as an example, if you designate only region number as a unique logical key and the user specifies region number on a GET, multiple branch rows might be retrieved for each region number.



We recommend that you uniquely identify a single row whenever possible. It is required for INSERT and UPDATE operations.



With a non-unique key, the rows may be retrieved only sequentially and not based on a value, which could cause performance problems.

Simple non-unique keys

A simple non-unique key is a data item with the value in more than one row. If you can have more than one row with the same logical key, it is an unnormalized view and has non-unique keys. An example is a customer data set with comments about each customer. You neither date the comments nor supply another key, but you want to retrieve the comments for each customer. This is a non-unique, unnormalized view because multiple rows contain the same customer number. You define the customer number as a single key, and define access to the comment rows without specifying a key. When the program does its first GET using a customer number, it retrieves the first comment for that customer. The next GET retrieves the second comment, and so on. At the last comment for that customer, RDM finds a boundary condition (see [“Navigational constraints and boundary conditions”](#) on page 78) and returns a "not found" status.

Compound non-unique keys

A compound non-unique key is an extension of the simple non-unique key because you define more than one column as the logical key, and at least one of the logical keys is non-unique. All non-unique keys together still do not completely describe the row occurrence as unique. You can still retrieve more than one row with that same compound non-unique key.

You can define a non-unique key either by omitting a key that would uniquely define the view, or by explicitly defining a data item as a non-unique key. The difference is that the user can perform searches based on the value of a column explicitly defined as a non-unique key.

Constant keys

If you do not want the programmer to specify a value for the key in the application, supply a logical key with a fixed value. Do this either by entering the keyword `CONST` in the column definition and assigning it a literal value or by assigning the value to a logical data item name or column name in the `WHERE` clause of the access definition. RDM uses this value as though the program supplied the value as a key.

A `CONST` key can be unique or non-unique, depending on what you specify in your column definition. You can use a unique constant to prevent duplication of the constant value on insert. Constant values must pass data validity checking if specified, and cannot be null. A `CONST` is always a required column in the view.

You can use a `CONST` key for value-based security. For example, assume you want to define a view that retrieves only the customers from Texas. You could supply a constant of `TX` to the state field. Then the program can retrieve and update only Texas customers.

When you designate a column as a constant, RDM does not return the column value in the row. For example, the user of the view would never see the state value 'TX'.

Example Derived Views

View Text: CUSTOMERS-IN-TEXAS

```
KEY    CUSTOMER-NUMBER
        CUSTOMER-NAME
        CUSTOMER-ADDRESS
        CUSTOMER-CITY
CONST  CUSTOMER-STATE = 'TX'
        CUSTOMER-ZIP-CODE
        CUSTOMER-CLASS
        CUSTOMER-CREDIT-CODE
        CUSTOMER-CREDIT-LIMIT
        CUSTOMER-BRANCH
ACCESS CUSTOMER
        WHERE CUSTOMER-NUMBER = CUSTOMER-NUMBER
```

or

View Text: CUSTOMERS-IN-TEXAS-2

```
KEY    CUSTOMER-NUMBER
        CUSTOMER-NAME
        CUSTOMER-ADDRESS
        CUSTOMER-CITY
        CUSTOMER-ZIP-CODE
        CUSTOMER-CLASS
        CUSTOMER-CREDIT-CODE
        CUSTOMER-CREDIT-LIMIT
        CUSTOMER-BRANCH
ACCESS CUSTOMER
        WHERE CUSTOMER-NUMBER = CUSTOMER-NUMBER AND CUSTOMER-
STATE = 'TX'
```

Secondary access keys

As described in “[Unique keys](#)” on page 86, a unique key may map onto values found in more than one data set. In practice, this means that you can obtain a control key value from data-set-1 and use this value to access data-set-2. Thus, secondary access keys are copies of control key values found in other data sets. The data set containing the control key and the data set containing the secondary access key may be primary or related. It is the order in which you access the data sets that determines which data set contains the secondary access key.

You cannot modify secondary access keys because the physical key value for the secondary access key is held on the data set containing the control key. The PDM allows you to change this value only through the control key on the parent data set. If you attempt to open a view in which you allow updates on a data set accessed through a secondary key, RDM returns this message:

```
#nnnn DO NOT MODIFY SECONDARY ACCESS KEYS
```

where *nnnn* is the line in the view that generated the error.

In other words, if you specify the secondary access key in the column definition, you cannot insert or update rows on the data set accessed through that secondary key. To insert or update records on a data set accessed through a secondary key, omit the secondary key from the column definition. Alternatively, change the order of data set access.

Note that a secondary access key is different from a secondary key. You create secondary keys as part of the physical database during database definition to allow access to a data set via an alternate index.

Generic reads

Generic reads enable you to retrieve data using partial values as keys. You can omit characters from the right, substituting the wildcard character * (for equal or next match) or = (for equal only match). These characters are the default wildcard characters; however, you can specify your own wildcard characters by defining the logical names CSI_WILD_EN (for equal or next) and CSI_WILD_EQ (for equal only) as described in the general considerations at the end of this section. Refer to the *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130, for details on defining these logicals.

To illustrate a generic read, assume you are using a view, CUSTOMER-ORDER, containing the columns ORDER-CUSTOMER-NUMBER and ORDER-NUMBER where there is a secondary key associated with CUSTOMER-NUMBER:

Base View: CUSTOMER-ORDER

View Text:

```
NON-UNIQUE KEY ORDER-CUSTOMER-NUMBER = ORDER-CUST-ID
ORDER-NUMBER = ORDER-ID
ACCESS ORDR
WHERE ORDER-CUST-ID = ORDER-CUSTOMER-NUMBER
```

A series of GETs would return data in the following order:

NAME	CITY
-----	-----
ADAMS A	ABERDEEN
BROWN D	READING
CARSON C	ABERDEEN
COX D	READING
LOVE C	PORTSMOUTH
SMITH Fred	SLOUGH
SMITH P	LONDON
SMITH SM	MAIDENHEAD
SMYTH M	SWINDON

In this example, we are using the default wildcard characters. The wildcard (*) specifies an equal or greater than match instructing RDM to return a row where a key matches the partial key specified. If there is no key match, RDM will return the next row. RDM always returns a row for this option until it reaches the end of file boundary condition. Using the previous example, if you repeatedly issued the following statement:

```
GET CUSTOMER-ORDER USING C*
```

you would retrieve rows in the following order:

```
CARSON C                ABERDEEN
COX D                   READING
LOVE C                  PORTSMOUTH
SMITH Fred              SLOUGH
SMITH P                 LONDON
SMITH SM                MAIDENHEAD
SMYTH M                 SWINDON
OCCURRENCE NOT FOUND.
```

When RDM cannot find a row to match the partial key supplied, it returns rows that do not match until it reaches the end of file boundary condition.

General considerations for generic reads

- ◆ You must access the data set containing the partial key via a secondary key.
- ◆ The partial key specified must have a character data type. Generic read does not work on the other data types supported by RDM.
- ◆ You can specify your own wildcard characters by defining the following logical names in any logical name table available to your process:

```
$ DEFINE CSI_WILD_EN @
```

to use the @ character to specify an equal or next match

```
$ DEFINE CSI_WILD_EQ #
```

and to use the # character to specify an equal only match. You can substitute your own wildcard characters for @ and #.

- ◆ The wildcard character must be the rightmost character in the key. (You can omit parts of the key from the right only.) RDM ignores any values entered after the wildcard character. For example, the following two statements are both valid; however, RDM will process the second statement exactly like the first, ignoring the characters to the right of the wildcard.

```
GET view-name USING SM*
GET view-name USING SM*TH
```

Both statements retrieve the same rows.

- ◆ You can use compound generic keys; however, the wildcard character must still be the rightmost character in the string. For example, you could access a secondary key with two key parts as follows:

```
GET view-name USING 1234 TE=
```

However, the following statement is invalid because it includes characters to the right of the wildcard character.

```
GET view-name USING 12= TEMP
```

- ◆ You can specify values for logical key columns after the generic key, for example, for an index with one key part and a view with three logical key columns you could enter:

```
GET view-name USING SM* 1234 999
```

Domains

Every column in a view eventually maps to a physical data item defined on the SUPRA Server Directory database SUPRAD. When you define a data item on the SUPRA Server Directory, you also specify its physical characteristics such as length and format. You specify additional characteristics such as a validation option, default value, and null value when you define a domain and connect it to the data item. RDM checks the Global View file or the SUPRA Server Directory for these characteristics when processing RDML requests.

Domain attributes of null value, default value, retrieval validation, and validation type are especially important in processing views because RDM uses them to validate the data and maintain data integrity. In addition, domains ensure that relational joins are made only on columns of the same type. For instance, it makes no sense to create a join where

```
SALARY = RETAIL-PRICE
```

SALARY and RETAIL-PRICE may have the same format and length, so this join, although meaningless, would be permitted. By assigning each of these columns to a different domain, you avoid such oversights.

RDM performs some default validation such as checking that packed or numeric columns contain valid numbers. This check is made after the check for nulls but before any specified validation is done (see “[Null values](#)” on page 98, “[Default values](#)” on page 101, “[Validation options](#)” on page 103, and “[Join compatibility](#)” on page 105). RDM returns an ASI of “V” for columns that fail default validation checks. See “[RDM status indicators](#)” on page 223 for a description of ASIs.

You define domains and connect them to logical data items through DBA functions. The following screen illustration illustrates the contents of a sample domain as they appear during domain creation. You can see that null and default values, validation options, and validation exits are all specified through domains.

```

CINCOM SYSTEMS                                DOMAIN : REGION

 1 : DOMAIN-NAME                               :REGION
 2 : DOMAIN-FUNCTION                           :STRING
 3 : DOMAIN-UNIT                               :N/A
 4 : DOMAIN-FORMAT                             :CHARACTER
 5 : DOMAIN-LENGTH                             :2
 6 : DOMAIN-DECIMALS                           :0
 7 : DOMAIN-SIGNED                             :SIGNED
 8 : DOMAIN-NULLS-ALLOWED                     :NO
 9 : DOMAIN-NULL-VALUE                         :
10 : DOMAIN-DEFAULT-VALUE                     :
11 : DOMAIN-RETRIEVAL-VALIDATION              :NO
12 : DOMAIN-VALIDATION-TYPE                   :RANGE
13 : DOMAIN-MINIMUM-VALUE                     :01
14 : DOMAIN-MAXIMUM-VALUE                     :99
15 : DOMAIN-VALIDATION-EXIT-NAME              :
16 : DOMAIN-STATUS                             :O.K.

Enter field number (or <PF1> to exit) :

```

You can use the COLUMN-DEFN DBAID command to report on null, default, and validation information for each column in a view. In addition, it reports on the physical characteristics of the column, such as length and format. See [“Defining and testing views using DBAID”](#) on page 135 for more information on DBAID commands. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for more information on creating domains.

When DOMAIN RETRIEVAL validation is enabled, RDM validates the column for each retrieval (GET). Columns that fail the validation criteria are flagged with an invalid ASI. Required columns must be valid and non-null, or a row is not returned. However, data is returned for non-required columns with an invalid ASI.

The following sections discuss the components of null and domain support:

- ◆ [Null values](#)
- ◆ [Default validation](#)
- ◆ [Validation options](#)

Null values

The SUPRA Server Directory supports null values for all data types. You can specify in the domain whether a physical data item can be null and, if so, which value that data item should contain to represent a null. A null value represents missing or inapplicable information, and is independent of data type. Null values are distinct from blanks, zeros, the empty character string, or any other value, although you can use these values as the null string if you wish. To allow null values for a given logical data item, you connect the data item to the appropriate domain through DBA Functions (refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260).

For example, assume that you are organizing delivery routes. You specify the date, route, vehicle, time of departure, and driver. However, because you do not know which driver will be available for the dockside route and the city-south route, you specify the null value for both. The null value could be "Not yet allocated."

Inserting the null value for driver does not mean that nothing will be delivered on the dockside and city-south routes. Nor does it mean that the same driver will deliver on both routes. The null value "Not yet allocated" indicates that you do not know which driver will deliver. The necessary information is missing.

Note that the data type of the null value may differ from the data type of the column (shown in the DOMAIN-FORMAT field of the domain using the DBA utility). However the length of the null value is limited by the length of the field, regardless of data type.

Null values are optional; you define them in the domain details, which may then be connected to a data item through DBA. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for a description of how to specify a null value for a data item during domain definition.

For numeric or packed numeric data items without domains (for instance, those migrated from a previous release or newly created and not yet connected to a domain), RDM sets the NULL flag to "Y" and sets the null value to blanks. The NULL flag for data items of any other type (binary, character, or floating point) is set to "N."

GET Processing with null values

When RDM processes a GET request, each column that is equal to its null value is given an ASI of missing (-), and is set to zero for numeric type data or blanks for all other type data. Required columns must be non-null, as defined in the domain.

INSERT processing with null values

When RDM processes an INSERT command, all columns with a null ASI (an ASI of N) are set to their corresponding null value, which you specified in the domain details using the DBA utility.

The application program can insert a null value into a column by changing the ASI to "N" or by supplying the null value in the column. (See ["RDM status indicators"](#) on page 223 for a description of ASIs.) The DBAID user can insert a null value by inserting the keyword NULL into the column or by supplying the null value. Inserting the null value itself is NOT recommended because if the DBA changes the value of the null, you may have to change and recompile the application program that depends on it.

If a column in a view is required, the user cannot input null data. RDM rejects any attempt to insert a null value into a required column. Null values for foreign keys are allowed only if the foreign key is not required. (See ["Foreign key value integrity"](#) on page 109 for a description of null foreign keys.)

UPDATE processing with null values

When RDM processes an UPDATE command, all columns with a null ASI (an ASI of N) are set to their corresponding null value.

The application program can insert a null value into a column by changing the ASI to "N" or by supplying the null value in the column. The DBAID user can insert a null value by inserting the keyword NULL into the column or by supplying the null value. Inserting the null value itself is NOT recommended because if the DBA changes the value of the null, you may have to change and recompile the application program that depends on it.

If a column in a view is required, the user cannot input null data. RDM rejects any attempt to insert a null value into a required column. Null values for foreign keys are allowed only if the foreign key is not required. (See ["Foreign key value integrity"](#) on page 109 for a description of null foreign keys.)

DELETE processing with null values

When a view deletes a primary key, the Base View Definition can allow for foreign keys to be either cascade deleted or nullified, or it can restrict the delete (see “[Deletion integrity](#)” on page 116). To cascade delete, specify ALLOW DELETE on the data set that contains the foreign key so that foreign keys can be deleted. Alternatively, you can ensure that the foreign key restricts the delete by not adding an ALLOW on the relation with the foreign key (the target relation), forcing read-only access. This is useful, for example, if you want to delete a region but not all branches. With nullify delete, you can set each branch’s region number to a null value until the branch can be reassigned to a new region.

The following example is a base view that illustrates this.

Base View: DELETE-REGION-NULLIFY-BRANCH

View Text:

```
KEY   REGION-NUMBER = REGION-ID
      REGION-NAME
ACCESS REGN
      WHERE REGION-ID = REGION-NUMBER
      ALLOW DELETE
ACCESS BRAN
      WHERE BRANCH-REGION-ID == REGION-ID
      ALLOW UPDATE
```

See “[Foreign key value integrity](#)” on page 109 for details of how RDM can delete a primary key and nullify a foreign key.

MANTIS and SPECTRA support for nulls

Because MANTIS programs and SPECTRA processes cannot update ASI fields, to insert a null you must input the null value.

Default values

RDM uses the default value for a logical data item when no column in the user view maps to that logical data item, either because the user view subset does not include the mapping column or because the view does not contain the mapping column. You define default values in the domain details that can then be connected to a data item through DBA. The maximum length for a default value is 32 bytes. Any default that is less than 32 bytes is padded on the right with blanks.

Examples

- ◆ This example shows how RDM automatically uses the default value for a data item. The data set CUST contains the following physical and logical data items and default values, where appropriate:

Physical data item name	Column name/logical data item name	Default value
CUSTCTRL	CUSTOMER-NUMBER	
CUSTNAME	CUSTOMER-NAME	
CUSTCRCO	CUSTOMER-CREDIT-CODE	C3
CUSTCRLM	CUSTOMER-CREDIT-LIMIT	250

You could then construct the following derived view that would automatically use the default values for CUSTOMER-CREDIT-CODE and CUSTOMER-CREDIT-LIMIT:

Derived View: ADD-CUSTOMER-DEFAULT-VALUES

View Text:

```
KEY CUSTOMER-NUMBER
    CUSTOMER-NAME
    CUSTOMER-BRANCH
ACCESS CUSTOMER
    WHERE CUSTOMER-NUMBER = CUSTOMER-NUMBER
ALLOW ALL
```

An insert using the view with the values 1001 and McEwan Plastics would insert the following row into the database:

CUSTOMER-NUMBER	CUSTOMER-NAME	CUSTOMER-CREDIT-CODE	CUSTOMER-CREDIT-LIMIT
1001	McEwan Plastics	C3	250

- ◆ To insert a specified CUSTOMER-CREDIT-CODE or CUSTOMER-CREDIT-LIMIT column, you must include the column in the column definition of the view. For example, the following view would automatically use the default value for CUSTOMER-CREDIT-CODE, but would allow you to enter a value for CUSTOMER-CREDIT-LIMIT.

Derived View: ADD-CUSTOMER-DEFAULT-VALUES-2

View Text:

```
KEY CUSTOMER-NUMBER
   CUSTOMER-NAME
   CUSTOMER-BRANCH
   CUSTOMER-CREDIT-LIMIT
ACCESS CUSTOMER
   USING CUSTOMER-NUMBER
   ALLOW ALL
```

An insert using the view with the values 1002, Wick Potteries and 500 would insert the following row into the database:

CUSTOMER-NUMBER	CUSTOMER-NAME	CUSTOMER-CREDIT-CODE	CUSTOMER-CREDIT-LIMIT
1002	Wick Potteries	C3	500

Validation options

You specify validation options on the SUPRA Server Directory when defining domains. For details on how to define domains, refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260. The validation type tells RDM how to validate the contents of a column. The four validation options are:

- ◆ **Range checking.** Column values should be within a minimum and maximum range. A range value can be up to 32 bytes long.
- ◆ **Table checking.** Column values must match an entry in the associated validation table. A table value can be up to 72 bytes long.
- ◆ **Exit.** Uses a specified RDM user exit to verify the values of the column.
- ◆ **No validation.** Any validation must be done in your application programs.

You can specify only one validation option for each physical data item; all options are mutually exclusive.

RDM performs validation checking before each INSERT or UPDATE whenever a column in a view maps to a physical data item that uses one of the validation options. If the retrieval validation flag is set on retrieval, RDM validates the data immediately after the physical GET (at the base view level). On INSERT or UPDATE, RDM validates the data at the highest level (at derived view level if a derived view exists, otherwise at base view level).



If you use global views, validation information will be included in the Global View file. See [“Optimizing view performance using bound and global views”](#) on page 231 for more information on using global views.

Range checking

RDM verifies that a value in a column is within a specified range. You can specify the minimum value and the maximum value that RDM uses to validate. Range values are limited to 32 bytes in length, which is normally sufficient for data types other than character. For character columns that have lengths greater than 32 bytes, the range value is padded to the right with blanks before the comparison.

Example. Let's say our example company has seven credit codes: letters A–G. You can specify this range as the customer credit code domain CUSTOMER-CLASS. When processing an INSERT or UPDATE request, RDM ensures that a credit code of anything but A–G is rejected.

Table checking

RDM verifies that a value in a column is contained within a table of values stored on the SUPRA Server Directory database SUPRAD. You can build a table of values on the Directory and identify the name of the table for RDM to use. The DBA must create each validation table on the SUPRA Server Directory database SUPRAD. Each entry in the table can be a maximum of 72 bytes long.

Note that if a table contains many values, it is better to create another data set to store the values, and to use a foreign key to access them.

Example. There may be ten suppliers for a particular part. Whenever you place an order for that part, RDM verifies that the supplier you specify is one of the ten you are authorized to use. If the supplier is in the table, your order is processed.

Exits

You can write your own RDM user exits to perform any domain checking that the DBA has specified. RDM then calls the user exit when validating a value in a column. [Appendix C](#) on page 295 contains example user exits in C, FORTRAN, and PASCAL.

Note that you cannot use an RDM validation exit to translate a column value. (The validation exit must not change the value entered.)

Join compatibility

RDM ensures that any columns used in a join are from the same domain unless you explicitly use the domain override (=) in the access definition of the view. For example, you cannot join a column connected to a domain of numbers with a column connected to a domain of alphanumeric characters. The following ACCESS statement is incorrect because REGION-ID is from the REGION domain while BRANCH-REGION-ID is from the BRANCH-REGION domain.

```
ACCESS REGN
  WHERE REGION-ID = BRANCH-REGION-ID
```

However, this next example uses the extra equals sign to indicate that RDM should not perform normal domain checking. This ACCESS statement is allowed as long as REGION-NUMBER and BRANCH-NUMBER are the same length.

```
ACCESS REGN
  WHERE REGION-NUMBER == BRANCH-REGION-ID
ACCESS REGN
  WHERE REGION-NUMBER == BRANCH-REGION-ID
```

If one or both of the columns in a join do not have a domain, RDM only verifies that the length of both is the same.

Note that you only specify the domain override (=) once, between the first and second columns.

Referential integrity with RDM

Referential integrity ensures that two pieces of data representing the same fact do not become inconsistent. You set up your base views to maintain referential integrity. For the purpose of this discussion, we will use the following terminology:

- ◆ A *foreign key* is a data item in one data set that can only contain values found in the primary key of another data set.
- ◆ The *source relation* is the data set containing the foreign key as a data item; its rows depend on primary key values in another data set.
- ◆ The *target relation* is the data set containing the primary key values that match the foreign key values in the source relation; the rows in the source relation rely on this primary key.

The terminology is relative and expresses the relationship between only two data sets at a time: the target and the source. As shown in the following illustration, the values in foreign-key must first exist in primary-key-a.

Target relation:

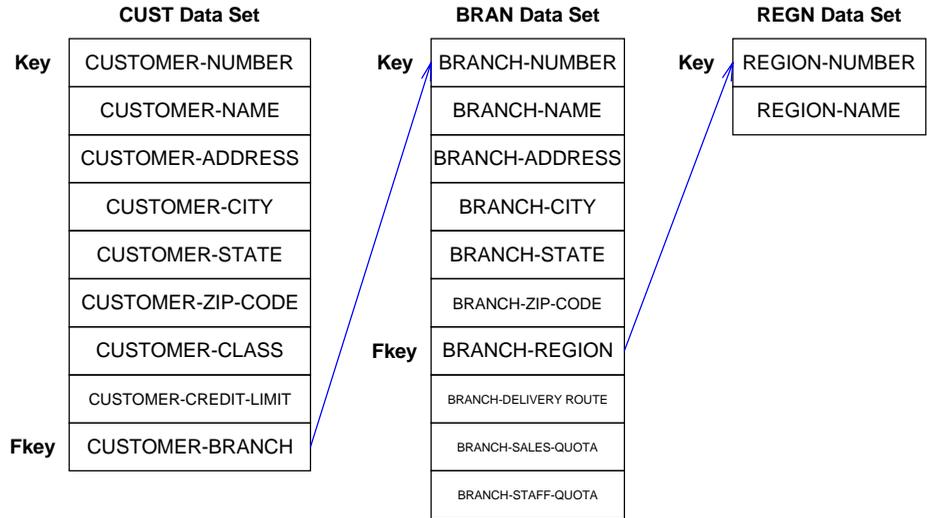
primary-key-a
---------------	-----	-----	-----

Source relation:

primary-key-b	...	foreign-key
---------------	-----	-------------

To ensure fast retrieval of the source relation, define a secondary key on the foreign key. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for a description of how to include data items in secondary keys.

The following illustration is an example of how RDM maintains referential integrity. (The data sets used here are based on the example database described in [Appendix D](#) on page 305.)



This illustration has two foreign keys:

- ◆ CUSTOMER-BRANCH is the foreign key from CUST to BRAN. CUST is the source relation; BRAN is the target relation.
- ◆ BRANCH-REGION is the foreign key from BRAN to REGN. BRAN is the source relation; REGN is the target relation.

Integrity rules and checking

RDM supports the following referential integrity rules:

- ◆ A foreign key value must exist in the target relation as a primary key. A primary key value must exist for each foreign key value in a source relation.
- ◆ Null values are allowed for a foreign key value.

RDM checks for referential integrity in two ways:

- ◆ **Foreign key value integrity.** When inserting or updating a row that contains a foreign key, the foreign key value must point to a valid primary key in the target relation or be null. This rule also applies if the foreign key consists of several key parts (subdefined fields). RDM performs INSERT or UPDATE integrity only if none of the key parts is null.
- ◆ **Deletion integrity.** RDM will not delete a row unless you first delete or nullify all foreign keys. This means that you cannot delete a primary key unless you also delete or nullify rows in a source relation that contain the key value in a foreign key.

You can implement referential integrity in the column definitions, ACCESS statements, or some combination.

Foreign key value integrity

To enforce foreign key value integrity, define the foreign key in the view. You can define a required foreign key or a foreign key that allows nulls. To define a foreign key, you must:

- ◆ Make the view column required and associated with both the foreign key in the source relation and the primary key in the target relation. For example:

```
REQ REGION-NUMBER == BRANCH-REGION-ID = REGION-ID
```

Or, identify the foreign key column with the keyword FKEY. For example:

```
FKEY REGION-NUMBER == BRANCH-REGION-ID = REGION-ID
```

- ◆ Access the target relation through its primary key by using the foreign key value from a source relation. For example:

```
ACCESS BRAN
  USING BRANCH-ID
  ALLOW INSERT UPDATE
ACCESS REGN
  WHERE REGION-ID == BRANCH-REGION-ID
  ALLOW INSERT UPDATE
```

If you use REQ, BRANCH-REGION (the foreign key) must be valid and non-null. If you use FKEY, BRANCH-REGION must be valid or null.

The rules for defining a foreign key are:

- ◆ The foreign key may consist of one or more columns. The parts of the foreign key do not have to be contiguous in the source relation; however, they must all come from the same physical data set.
- ◆ When the primary key in the target relation is subdefined and you are using the subdefined fields in your view, you must use all the parts of the foreign key to access the target relation through its primary key. The foreign key parts must provide the full primary key. In the view, this will cause a one-to-one access from the source relation to the target relation. You must not specify additional selection criteria (using the WHERE clause) on any data fields in the target relation.
- ◆ Each column that is part of the foreign key must be required and associated with equivalent parts of the foreign key from the source relation and the primary key from the target relation.
- ◆ Express all integrity constraints in base views. You must not use the FKEY option in derived views.

Insertion integrity

When you attempt an insert on a data set that contains a foreign key, RDM ensures that after the insert, the foreign key points to a valid primary key in the target relation or that the foreign key is null. A foreign key can be null only if you specify FKEY in the column definition. If you insert a non-null foreign key value and the primary key in the target relation does not exist, you can have RDM perform one of two actions:

- ◆ Reject the insert. You do this by not coding ALLOW INSERT or ALLOW ALL on the target relation. RDM marks the foreign key attribute with an ASI of "V" and sets the FSI to "D" or "X." (See "RDM status indicators" on page 223 for information on RDM status indicators.) For example:

Base View: CUSTOMER-INSERT-INTEGRITY

View Text:

```
KEY CUSTOMER-NUMBER = CUSTOMER-ID
REQ BRANCH-NUMBER = CUSTOMER-BRANCH-ID = BRANCH-ID
ACCESS CUST
    WHERE CUSTOMER-ID = CUSTOMER-NUMBER
    ALLOW INSERT
ACCESS BRAN
    WHERE BRANCH-ID = CUSTOMER-BRANCH-ID
```

- ◆ Automatically insert the primary key in the target relation. You do this by coding ALLOW INSERT or ALLOW ALL on the target relation. For example:

Base View: CUSTOMER-INSERT-INTEGRITY-2

View Text:

```
KEY CUSTOMER-NUMBER = CUSTOMER-ID
REQ BRANCH-NUMBER = CUSTOMER-BRANCH-ID = BRANCH-ID
ACCESS CUST
    WHERE CUSTOMER-ID = CUSTOMER-NUMBER
    ALLOW INSERT
ACCESS BRAN
    WHERE BRANCH-ID = CUSTOMER-BRANCH-ID
    ALLOW INSERT
```

If you have automatic insert of a new primary key, you may require validation of another foreign key in the automatically added row. In this case, you must also define the second foreign key. For example:

Base View: CUSTOMER-INSERT-INTEGRITY-3

View Text:

```
KEY CUSTOMER-NUMBER = CUSTOMER-ID
REQ BRANCH-NUMBER = CUSTOMER-BRANCH-ID = BRANCH-ID
REQ REGION-NUMBER == BRANCH-REGION-ID = REGION-ID
ACCESS CUST
  WHERE CUSTOMER-ID = CUSTOMER-NUMBER
  ALLOW INSERT
ACCESS BRAN
  WHERE BRANCH-ID = CUSTOMER-BRANCH-ID
  ALLOW INSERT
ACCESS REGN
  WHERE REGION-ID == BRANCH-REGION-ID
```



REGION-ID and BRANCH-REGION-ID are in different domains so that BRANCH-REGION-ID can be NULL. Therefore, domain override is required.

If a customer row is inserted with a BRANCH-NUMBER that does not exist, RDM inserts a branch row. However, before the branch is inserted, RDM checks that REGION-NUMBER points to an existing region row. If not, the insert fails. By placing ALLOW INSERT on the REGN data set, you could also make RDM perform automatic inserts on the REGN data set.

You can insert a null foreign key when the column is defined as FKEY (instead of REQ) either by placing an "N" into the ASI for the column or by supplying the actual null value; RDM does not perform INSERT referential integrity in this case because the primary keys cannot be null. Remember to use the FKEY syntax if the foreign key is likely to be null.

Note that inserting the actual null value is not recommended because the application is then dependent on that null value. Instead, let RDM insert the null value defined in the domain by setting the ASI for the column to "N." See [“Null values”](#) on page 98 for information on null values, and [“RDM status indicators”](#) on page 223 for a description of RDM Status Indicators.

Update integrity

When you update a foreign key, RDM ensures that, after the update, the foreign keys point to a valid primary key in the target relation or that the foreign key is null (provided the FKEY syntax was used). If you update the foreign key value and the primary key in the target relation does not exist, you can have RDM perform one of two actions:

- ◆ Reject the update. You do this by not coding ALLOW INSERT or ALLOW ALL on the target relation. RDM will set the foreign key ASI to "V" and the FSI to "D" or "X." For example:

Base View: CUSTOMER-UPDATE-INTEGRITY

View Text:

```
KEY   CUSTOMER-NUMBER = CUSTOMER-ID
REQ   CUSTOMER-BRANCH = CUSTOMER-BRANCH-ID = BRANCH-ID
ACCESS CUST
      WHERE CUSTOMER-ID = CUSTOMER-NUMBER
      ALLOW UPDATE
ACCESS BRAN
      ONCE
      WHERE BRANCH-ID = CUSTOMER-BRANCH-ID
```

- ◆ Automatically insert the primary key in the target relation. You do this by coding ALLOW INSERT or ALLOW ALL on the target relation. For example:

Base View: CUSTOMER-UPDATE-INTEGRITY-2

View Text:

```
KEY   CUSTOMER-NUMBER = CUSTOMER-ID
REQ   CUSTOMER-BRANCH = CUSTOMER-BRANCH-ID = BRANCH-ID
      BRANCH-NAME
ACCESS CUST
      WHERE CUSTOMER-ID = CUSTOMER-NUMBER
      ALLOW UPDATE
ACCESS BRAN
      ONCE
      WHERE BRANCH-ID = CUSTOMER-BRANCH-ID
      ALLOW INSERT
```

If the view defines other foreign keys in the automatically inserted target relation, insert integrity rules apply on the insertion. For example:

Base View: CUSTOMER-UPDATE-INTEGRITY-3

View Text:

```

KEY    CUSTOMER-NUMBER = CUSTOMER-ID
REQ    CUSTOMER-BRANCH = CUSTOMER-BRANCH-ID = BRANCH-ID
REQ    BRANCH-REGION == BRANCH-REGION-ID = REGION-ID
ACCESS CUST
      WHERE CUSTOMER-ID = CUSTOMER-NUMBER
      ALLOW UPDATE
ACCESS BRAN
      ONCE
      WHERE BRANCH-ID = CUSTOMER-BRANCH-ID
      ALLOW INSERT UPDATE
ACCESS REGN ONCE
      WHERE REGION-ID == BRANCH-REGION-ID

```



REGION-ID and BRANCH-REGION-ID are in different domains so that BRANCH-REGION-ID can be NULL. Therefore, domain override is required.

You can also specify updating on the target relation. For example, in the following view you could update both CUSTOMER-NAME and BRANCH-NAME.

Base View: CUSTOMER-UPDATE-INTEGRITY-4

View Text:

```

KEY    CUSTOMER-NUMBER = CUSTOMER-ID
      CUSTOMER-NAME
REQ    CUSTOMER-BRANCH = CUSTOMER-BRANCH-ID = BRANCH-ID
ACCESS CUST
      WHERE CUSTOMER-ID = CUSTOMER-NUMBER
      ALLOW UPDATE
ACCESS BRAN
      WHERE BRANCH-ID = CUSTOMER-BRANCH-ID
      ALLOW UPDATE

```

In the previous example, if you update the foreign key BRANCH-NUMBER, the update processing positions the BRAN data set on the row pointed to by the new foreign key value. This means that any update to BRANCH-NAME would apply to the branch row pointed to by the new foreign key value, not the branch row retrieved by the GET before the update. You must be very careful if you allow updating on both the source relation and the target relation.

You can allow both updates and inserts on the target relation. This means RDM can update the target relation if the primary key already exists, or insert the primary key if it does not exist.

If you define the column as FKEY instead of REQ, you can update a foreign key to null either by supplying the actual null value for the column or by placing an "N" into the ASI for the foreign key. RDM does not perform UPDATE referential integrity in this case because primary keys cannot be null. You can have a null foreign key only if you specify FKEY in the column definition portion of the view definition.



We do not recommend updating the foreign key to null by inserting the actual null value because the application is then dependent on that null value. Instead, let RDM insert the null value defined in the domain by setting the ASI for the column to "N." See ["Null values"](#) on page 98 for more information on null values, and ["RDM status indicators"](#) on page 223 for more information on RDM Status Indicators.

GET processing

Because a foreign key column is defined as required (REQ) and equivalent to the foreign key from the source relation and the primary key in the target relation, a GET RDML command must retrieve data from both the source relation and the target relation. This means that if an existing foreign key in the database is not valid, a view with the field defined as a foreign key is unable to retrieve the bad row. RDM will return an "occurrence not found" message because required data cannot be retrieved from the target relation—that is, the source foreign key and the target primary key must have the same value.

In the case of a null foreign key, RDM does not perform a GET on the target file because a null primary key is not allowed.

The following considerations and examples are for issuing GET statements. Each example uses DBAID syntax.

- ◆ When selecting with key values, always issue the first GET command as follows:

```
GET FIRST * USING value-1
```

- ◆ Issue any subsequent GETs with the same key value as follows:

```
GET NEXT * USING value-1
```

- ◆ Whenever the selection value changes, issue the GET command as follows:

```
GET FIRST * USING value-2
```

Deletion integrity

RDM will not allow a row to be deleted unless all foreign keys are first deleted or nullified. This means that you cannot delete a primary key if rows exist that contain foreign keys with the same value. To define referential integrity, you must access the source relation through its foreign key using the full primary key. For example:

```
ACCESS REGN
  WHERE REGION-ID = REGION-NUMBER
ALLOW DELETE
ACCESS BRAN
  WHERE BRANCH-ID = REGION-ID
```

You may define a secondary key on the foreign key to prevent sequential scans of the data set containing the foreign key. If the foreign key has multiple parts, include all the parts in the secondary key. For example:

```
ACCESS SAMP
  WHERE SAMPLE-NUMBER-SUB1 = SAMPLE-NUMBER-SUB1
    AND SAMPLE-NUMBER-SUB2 = SAMPLE-NUMBER-SUB2
    AND SAMPLE-NUMBER-SUB3 = SAMPLE-NUMBER-SUB3
ALLOW DELETE
ACCESS TEST
  WHERE BRANCH-TEST-SUB1 = REGION-SAMPLE-SUB1
    AND BRANCH-TEST-SUB2 = REGION-SAMPLE-SUB2
    AND BRANCH-TEST-SUB3 = REGION-SAMPLE-SUB3
```

You must not supply additional selection criteria on the WHERE clause for data fields in the source relation because RDM will use this additional criteria when checking the source relation.

If the source relation is an RMS data set, you should index the foreign key to improve performance. If the foreign key has multiple parts, include all parts in the alternate key. An alternate index is important because the source relation is not usually accessed through its primary key.

If you try to delete a primary key, and foreign keys with the same value still exist in the source relation, you can have RDM perform one of three actions:

- ◆ Reject the delete (restrict). You do this by coding `ALLOW DELETE` or `ALLOW ALL` on the target relation, but *not* on the source relation.
- ◆ Nullify the dependent foreign keys. You do this by specifying `ALLOW UPDATE` on the source relation and ensuring that no columns from the source relation are included in the column definition.
- ◆ Delete the dependent foreign keys (cascade). You do this by coding `ALLOW DELETE` or `ALLOW ALL` on the source relation as well as on the target relation. Provided the column definition does not contain any columns from the source relation, RDM deletes all occurrences of the foreign key in the source relation. If the source relation does provide columns, RDM deletes only one occurrence in the source relation. RDM deletes the primary key in the target relation when the last dependent foreign key is deleted.

When multiple relations depend on the source relation, RDM will "cascade delete" rows in all specified relations only if you specify `ALLOW DELETE` or `ALLOW ALL` on *each* relation.

To enforce referential integrity during a delete operation, use one of the following options:

- ◆ Restrict delete
- ◆ Nullify delete
- ◆ Cascade delete

Restrict delete

A delete operation will fail if any dependent rows (based on the foreign key) exist. Restrict Delete comes into play when there is a one-to-many relationship. One example of establishing a one-to-many relationship is accessing via a secondary key: RDM assumes a secondary key access is one-to-many. If you need a secondary key access to be one-to-one, then use the keyword ONCE with the ACCESS statement.

The following example shows a restrict delete:

Base View: DELETE-REGION-RESTRICT-BRANCH

View Text:

```
KEY REGION-NUMBER = REGION-ID
ACCESS REGN
WHERE REGION-ID = REGION-NUMBER
ALLOW DELETE
ACCESS BRAN
WHERE BRANCH-REGION-ID == REGION-ID
```



REGION-ID and BRANCH-REGION-ID are in different domains so that BRANCH-REGION-ID can be NULL. Therefore, domain override is required.

Nullify delete

When RDM performs a delete, it deletes the primary key but nullifies the foreign key. Follow these rules to nullify a foreign key:

- ◆ Allow UPDATE on the source relation; you must not ALLOW DELETE on the source relation.
- ◆ Access the source relation joining on the foreign key and the primary key from the target relation. To do this, specify that the foreign key is equal to the primary key in the WHERE clause of the ACCESS statement of the data set containing the foreign key. (See the second WHERE clause in the example below.)
- ◆ Ensure that the relation containing the foreign key data set supplies no columns.
- ◆ Allow DELETE on the target relation.
- ◆ Set the nulls allowed flag for the foreign key to Y. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for details of how to define null values for data items.

The following example shows a base view that can be used to delete the primary key and nullify the foreign key. In this example, the region is being deleted and any branches contained in that region will have a null value assigned to their BRANCH-REGION column. The ALLOW DELETE clause for the REGN primary data set designates that the region can be deleted. The ALLOW UPDATE clause on the BRAN data set designates that the BRANCH-REGION column can be nullified.

Base View: DELETE-REGION-NULLIFY-BRANCH

View Text:

```
KEY   REGION-NUMBER = REGION-ID
      REGION-NAME
ACCESS REGN
      WHERE REGION-ID = REGION-NUMBER
      ALLOW DELETE
ACCESS BRAN
      WHERE BRANCH-REGION-ID == REGION-ID
      ALLOW UPDATE
```

Cascade delete

When you perform a delete operation on a view, you must also delete all dependent rows (based on the foreign key). The following example shows a cascade delete:

Base View: DELETE-REGION-CASCADE-BRANCH

View Text:

```
KEY REGION-NUMBER = REGION-ID
    REGION-NAME
ACCESS REGN
    WHERE REGION-ID = REGION-NUMBER
    ALLOW DELETE
ACCESS BRAN
    WHERE BRANCH-REGION-ID == REGION-ID
    ALLOW DELETE
ACCESS CUST
    WHERE CUSTOMER-BRANCH-ID = BRANCH-ID
    ALLOW DELETE
```



REGION-ID and BRANCH-REGION-ID are in different domains so that BRANCH-REGION-ID can be NULL. Therefore, domain override is required.

DELETE is allowed for all three relations. If you omit the ALLOW DELETE (or ALLOW ALL) on any one relation, the delete will fail on *all* relations, as it becomes a restricted delete.

Referential integrity examples



In the following examples, REGION-ID and BRANCH-REGION-ID are in different domains so that BRANCH-REGION-ID can be NULL. Therefore, domain override is required.

- ◆ This view does not add a branch unless the region already exists. It does not allow REGION-NUMBER column to be updated unless the new value points to an existing region.

```
KEY   BRANCH-NUMBER = BRANCH-ID
      BRANCH-ADDRESS
      BRANCH-CITY
      BRANCH-STATE

REQ  REGION-NUMBER = BRANCH-REGION-ID = REGION-ID
ACCESS BRAN USING BRANCH-NUMBER
      ALLOW INSERT UPDATE
ACCESS REGN
      WHERE REGION-ID == BRANCH-REGION-ID
```

Notice that the foreign key in the BRAN data set is the same as the primary key in the REGN data set, and the REGN data set is accessed by its primary key with the foreign key value.

- ◆ This view checks to see if the region exists. If not, it adds a new region and then adds the branch. This difference between this view and the previous view is the ALLOW INSERT clause on the ACCESS statement for the data set REGN:

```
KEY BRANCH-NUMBER = BRANCH-ID
      BRANCH-ADDRESS
      BRANCH-CITY
      BRANCH-STATE

REQ  REGION-NUMBER = BRANCH-REGION-ID = REGION-ID
ACCESS BRAN USING BRANCH-NUMBER
      ALLOW INSERT UPDATE
ACCESS REGN
      WHERE REGION-ID == BRANCH-REGION-ID
      ALLOW INSERT
```

- ◆ This view accesses customer, then branch, then region. It allows updates and inserts into the CUST data set, and only updates to the BRAN data set. However, it allows neither updates nor inserts to the REGN data set, so the region must already exist.

```
KEY    CUSTOMER-NUMBER = CUSTOMER-ID
      CUSTOMER-NAME
REQ    BRANCH-NUMBER = CUSTOMER-BRANCH-ID = BRANCH-ID
      BRANCH-NAME
REQ    REGION-NUMBER == BRANCH-REGION-ID = REGION-ID
      REGION-NAME
ACCESS CUST
      ALLOW UPDATE INSERT
      WHERE CUSTOMER-ID = CUSTOMER NUMBER
ACCESS BRAN
      WHERE BRANCH-ID = CUSTOMER-BRANCH-ID
      ALLOW UPDATE
ACCESS REGN
      WHERE REGION-ID == BRANCH-REGION-ID
```

Thus, you can insert a new customer as long as the BRANCH-NUMBER already exists. You can then perform an update on the CUST and BRAN data set.

To update the column BRANCH-NUMBER, the new foreign key value must already exist in branch. Also, the update will reposition the BRAN data set to the new key value before updating BRANCH-NAME.

- ◆ This example shows how updating a foreign key can affect the positioning of the subsequent target relations.

```

KEY   CUSTOMER-NUMBER = CUSTOMER-ID
      CUSTOMER-NAME
REQ   CUSTOMER-BRANCH = CUSTOMER-BRANCH-ID = BRANCH-ID
      BRANCH-NAME
REQ   BRANCH-REGION == BRANCH-REGION-ID = REGION-ID
      REGION-NAME
ACCESS CUST
      WHERE CUSTOMER-ID = CUSTOMER-NUMBER
      ALLOW UPDATE
ACCESS BRAN
      WHERE BRANCH-ID = CUSTOMER-BRANCH-ID
ACCESS REGN
      WHERE REGION-ID == BRANCH-REGION-ID
      ALLOW UPDATE

```

Let's say a GET on this example returns a row with the following column values:

```

CUSTOMER-NUMBER = 15761
CUSTOMER-NAME   = CAROL JONES
CUSTOMER-BRANCH = 1000
BRANCH-NAME     = BRANCH 1000
BRANCH-REGION  = 100
REGION-NAME     = REGION 100

```

Now, let's say the columns are updated by the application as follows:

```

CUSTOMER-NUMBER = 15761
CUSTOMER-NAME   = CAROL LUCAS
CUSTOMER-BRANCH = 500
BRANCH-NAME     = BRANCH 1000
BRANCH-REGION  = 100
REGION-NAME     = WESTERN REGION

```

When the application issues an UPDATE telling RDM to change the data in the database, updates to CUSTOMER-NAME and CUSTOMER-BRANCH are applied as indicated. However, the change to CUSTOMER-BRANCH repositions the BRAN data set to the key value of 500. The ACCESS statement for the BRAN data set does not allow changes. So BRANCH-NAME for Branch 500 is not changed to 'BRANCH 1000.' Because REGN is accessed through a foreign key from BRAN, it is also repositioned (to Western Region, key value 500). Update processing to BRANCH-NAME and BRANCH-REGION is not performed due to the absence of ALLOW UPDATE on BRAN. Region remains positioned on key value 500. The update to REGION-NAME is now made to Region 500 (Western Region).

Even though foreign keys are defined as redundancies in the view, it is the ALLOW phrase on the source relation that controls whether you can update a foreign key. In the example, you cannot update BRANCH-REGION because there is no ALLOW UPDATE on the BRAN data set. Even though there is an ALLOW UPDATE on REGN and REGION-NUMBER is the same as in BRANCH-REGION, you cannot update BRANCH-REGION.

- ◆ This view allows for the region to be deleted if there are no dependent branches:

```
KEY   REGION-NUMBER - REGION-ID
      REGION-NAME
ACCESS REGN
      WHERE REGION-ID = BRANCH-REGION-ID
      ALLOW DELETE
ACCESS BRAN
      WHERE BRANCH-REGION-ID == REGION-ID
```

Notice that the source relation BRAN is accessed through its foreign key (BRANCH-REGION) using the primary key (REGION-ID). For optimal performance, consider defining a secondary key for BRANCH-REGION-ID.

- ◆ This view allows you to delete dependent branch rows, thereby allowing deletion of the region row. If there are no columns in the user view from the BRAN data set, deleting a region will also delete all branches dependent on it. If there are columns from BRAN in the user view, the program must delete each flat row by using a GET DELETE loop or by using DELETE ALL.

```
KEY REGION-NUMBER = REGION-ID
    REGION-NAME
ACCESS REGN
    WHERE REGION-IS = REGION-NUMBER
    ALLOW DELETE
ACCESS BRAN
    WHERE BRANCH-REGION-ID == REGION-ID
    ALLOW DELETE
```

Shared column values

You can share column values between views by specifying `ALLOW SHARED` in the `ACCESS` statement in the base view. You cannot use `ALLOW SHARED` in derived views. The advantages of allowing shared column values are:

- ◆ More efficient processing because automatic column value checking is bypassed when not needed.
- ◆ Modification of the same column in multiple views by the same task or other tasks. For example:

```
KEY BRANCH-NUMBER = BRANCH-ID
KEY BRANCH-REGION = BRANCH-REGION-ID
  REGION-NAME
ACCESS BRAN
  WHERE BRANCH-ID = BRANCH-NUMBER
ACCESS REGN
  WHERE REGION-ID == BRANCH-REGION-ID
ALLOW SHARED UPDATE
```



`REGION-ID` and `BRANCH-REGION-ID` are in different domains so that `BRANCH-REGION-ID` can be `NULL`. Therefore, domain override is required.

Using `SHARED` tells RDM that column values from a view can be shared between tasks and may change between a `GET` and a later `UPDATE` or `DELETE`. When the `SHARED` phrase is present, RDM does not check to see whether column values changed. If `SHARED` is not on the `ALLOW` phrase in an `ACCESS` statement, RDM performs a check on each column in the view, ensuring that column values have not changed. RDM does not check read-only columns that cannot be updated or deleted.

In the previous example, the only maintenance function that can be performed is UPDATE, and the only column that can be altered is REGION-NAME. Because SHARED is part of the ALLOW phrase, REGION-NAME is automatically altered and the automatic record hold and replace performed by RDM will not produce an error even if another view changes the value of the column. If you change the ACCESS statement to:

```
ACCESS REGN
WHERE REGION-ID = BRANCH-REGION-ID
```

and any other task changes the column, UPDATE or DELETE will fail. In the case of such a failure, RDM sets the FSI to D, the VSI and one or more ASIs to C and returns the following message:

```
COLUMN VALUE CHANGED BY ANOTHER VIEW
```

The changed columns are denoted by an ASI of "C." In this case, the "C" will be in the ASI column for REGION-NAME. The "C" VSI takes higher priority than other VSIs. (See ["RDM status indicators"](#) on page 223 for details about status indicators.)

View-to-user relationships

You control database security on a user-by-user basis by defining which users can use which views. This information is then stored on the SUPRA Server Directory database SUPRAD and optionally in the Global View file. Initially, the only user who can access a view is the user who creates it. Other users cannot access views that you have defined until you identify them as authorized users. You control user-to-view access through:

- ◆ Global views
- ◆ The DBAID commands PERMIT and DENY
- ◆ The DBA User authorization function from the Logical View Function menu

Globalizing views is a method of storing pre-opened views in a shared global section. You can control user-to-view access through the Global View file. Depending upon how you create your global views (interactively or in batch), you can relate users to a global version of a view even if those users have no access to the non-global version of the view. Furthermore, you can deny users access to the global version of a view even if those users are related to the non-global version of the view. See [“Optimizing view performance using bound and global views”](#) on page 231 for details of global view creation.

The DBAID PERMIT command relates a view to one or more users. The DBAID DENY command removes the relationship between a view and its users. With both commands, you can specify more than one user by separating consecutive user names with a space. See [“Defining and testing views using DBAID”](#) on page 135 for information on the syntax of both the PERMIT and DENY commands.

In the DBA utility, the User authorization option from the Logical View Function menu first prompts you for the name of the view to which you want to specify access. After you enter a valid view name, DBA displays a numbered list of existing authorized users. The last number in the list is blank to allow you to specify additional users. If you press RETURN without specifying a new user name, DBA prompts you to:

```
Specify function (Allow, Disallow), <PF4> to list or <PF1> to  
exit :
```

You can then remove user-to-view relationships (Disallow) or continue adding new user-to-view relationships (Allow). The DBA User authorization function is described in detail in the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260.

Because the user-to-view relationship is stored in the SUPRA Server Directory database, SUPRAD, any user-to-view relationship defined through the DBA User authorization function can be removed using the DBAID DENY command. Likewise, any user-to-view relationship defined through the DBAID PERMIT command can be removed through the DBA User authorization function.

You can relate both base and derived views to users. However, you should not relate users to derived views without authorizing them to use the base views accessed by the derived views. While the derived view accesses the base view, it can impose additional security on the user.

The following table shows BASE-VIEW-A and DERIVED-VIEW-B. BASE-VIEW-A allows all maintenance functions to its users, ALICE, MARY and JIM. User JIM is also related to DERIVED-VIEW-B, which accesses BASE-VIEW-A allowing read and update. Provided JIM accesses BASE-VIEW-A from DERIVED-VIEW-B, he is restricted to read and update access only. Thus, DERIVED-VIEW-B provides additional security by restricting its user (JIM) to read-only access to BASE-VIEW-A.

BASE-VIEW-A	DERIVED-VIEW-B
KEY CUSTOMER-NUMBER	KEY CUSTOMER = CUSTOMER-NUMBER
CUSTOMER-NAME	NAME = CUSTOMER-NAME
CUSTOMER-ADDRESS	STREET = CUSTOMER-ADDRESS
CUSTOMER-CITY	CITY = CUSTOMER-CITY
CUSTOMER-STATE	STATE = CUSTOMER-STATE
CUSTOMER-ZIP-CODE	ZIPCODE = CUSTOMER-ZIP-CODE
CUSTOMER-CLASS	
CUSTOMER-CREDIT-LIMIT	
CUSTOMER-BRANCH	
ACCESS CUST	ACCESS BASE-VIEW-A
USING CUSTOMER-NUMBER	USING CUSTOMER-NUMBER
ALLOW ALL	ALLOW UPDATE
<u>Authorized Users:</u>	<u>Authorized Users:</u>
ALICE	JIM
MARY	
JIM	

4

Physical and logical database changes

Overview

RDM insulates application programs from most changes to the physical database. However, certain changes require modifications, either to the application programs or to the views, to maintain the integrity of the database.

Physical and logical database actions

The following table lists physical and logical changes together with any necessary actions.

	Action				
	Change program	Recompile program	Modify view definition	Reglobalize / rebind view	Validate / compile DB
Data set changes					
Add a new data set			✓	✓	✓
RMS data set into PDM data set			✓	✓	✓
PDM data set into RMS data set			✓	✓	✓
Combine two data sets into one			✓	✓	✓
Delete a data set if it contains a column for a view	✓	✓	✓	✓	✓

	Action				
	Change program	Recompile program	Modify view definition	Reglobalize / rebind view	Validate / compile DB
Rename a data set			✓	✓	✓
Split one data set into several			✓	✓	✓
Change a PDM data set type			✓	✓	✓
Change a record length		✓		✓	✓
Change a linkpath location		✓		✓	✓
Change the physical key length	✓	✓	✓	✓	✓
Change the length of the base portion of a coded record				✓	✓
Change the position of the key in a primary record				✓	✓
Add or remove an index			✓	✓	✓
Physical changes					
Add new data items to a record					✓
Change data item length	✓*	✓	✓**	✓	✓
Change data item type	✓*	✓	✓**	✓	

* Does not apply to MANTIS programs

** Modify only if a constant column maps to the data item

	Action				
	Change program	Recompile program	Modify view definition	Reglobalize / rebind view	Validate / compile DB
No. of Decimal places	✓*	✓	✓**	✓	
Physical data item's location			✓	✓	✓
Delete data item from physical record if used by view and program	✓	✓	✓	✓	✓
Change null value, or nulls allowed				✓	
Change default value				✓	
Change validation type				✓	
Change validation data (range, table name, exit)				✓	
Logical changes					
Add columns to a view			✓	✓	
Change unique key to non-unique	✓	✓	✓	✓	
Change relationship and program depends upon relationship	✓	✓	✓	✓	

	Action				
	Change program	Recompile program	Modify view definition	Reglobalize / rebind view	Validate / compile DB
Define a new view	No change required				
Delete data item or column and program uses field or column	✓	✓	✓	✓	✓
Rename a column and program uses column on include	✓	✓	✓	✓	
Reorder columns			✓	✓	

5

Defining and testing views using DBAID

Through the DBAID Test Facility, you can define and test views before actually using them in production. You can also use the DBAID Test Facility to learn how the Relational Data Manager (RDM) works. A good way to implement new views is to check them out with DBAID prior to use.

Using DBAID, you can define a new view, open it, issue Relational Data Manipulation Language (RDML) commands, and examine the results. You can then change the view if necessary, reorder it for efficiency, or experiment with various navigation methods.

DBAID enables you to store the new view on the Directory. You can also load existing views from the Directory, change them to meet requirements, and then test them.

DBAID has commands that programmers can use to try out views defined for them. These commands are the programmers' subset of DBAID commands. With them, the programmer can learn the command functions. However, the programmer cannot update the Directory or define new views.

To edit a view, make sure you list the view text before you open the view. This makes the text of the view known to DBAID. Such a view is called a virtual view, and you can list any view in this way. However, if you open a view first, you must be authorized to use that view. See "[View-to-user relationships](#)" on page 128 for more information on user access authority.

When you define a base view using DBAID (see "[Defining and testing views using DBAID](#)" on page 135), specify the column definition as lines of text, each preceded by a line number. When you define a view using DBA Functions (refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260), specify the column definition through the screen-based EDT editor interface. When defining a view through DBA, you do not need to specify each line number as you do through DBAID. However, although you can open and save views through DBA as you can through DBAID, you cannot test them online.

DBAID can use any view defined on the Directory. However, views defined within DBAID that refer to physical data items in the column definition, rather than logical data items, cannot be saved on the Directory.

Invoking DBAID

If you use the procedure SUPRA_COMS:SUPRA_SYMBOL.COM, you will have a symbol DBAID that executes SUPRA_EXE:RUNDBAID.COM. This procedure checks for the existence of the logical CSI_SCHEMA. This logical is required to use RDM. If the logical is defined, it will be used. Otherwise, the procedure prompts you for your database name and uses it to define CSI_SCHEMA. Alternatively, you can select the DBAID Test Facility from the SUPRA Facilities screen, or set up your own symbol or command file. During any DBAID session, you can use views only if they are associated with one database description. This database description is defined using the logical name CSI_SCHEMA. For example, the following commands could be used to invoke DBAID:

```
$DEFINE CSI_SCHEMA TESTDB  
$RUN CSVDBAID
```

If you do not define CSI_SCHEMA, you receive the following message:

```
"ERROR NO LOGICAL NAME FOR USER DBMOD."
```

A command file, RUNDBAID, is supplied to invoke DBAID. For example, to invoke DBAID with database TESTDB, you could enter @RUNDBAID TESTDB or just @RUNDBAID with no parameters.

If the database contains RMS files, you can enable VMS RMS Recovery Unit Journaling. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for a description of how to enable Recovery Unit Journaling for physical files. Define the logical name CSI_RMS_RU_ON to be TRUE before invoking DBAID. This allows the transactions to the RMS files to be logged in a journal file that can be used to update or reset the RMS files if required.

You can define the logical CSI_RMS_RU_ON TRUE at the level of your database so it will be available to all users of that database. This will be in the group logical name table for a group-wide database, or in the system logical name table for a system-wide database. If you are using a multiple system-wide PDM, you can choose to define the logical in the CSI_PDM_ *pdmname* table. If you choose to define the logical in one of these shared logical name tables, you do not need to repeat the definition unless the machine on which you are working goes down. Alternatively, define CSI_RMS_RU_ON TRUE before invoking DBAID as follows:

```
$DEFINE CSI_RMS_RU_ON TRUE
$RUN CSVDBAID
```

This ensures that records in RMS data sets are rolled back to the last successful COMMIT point in the event of a system or application failure. This matches Task Level Recovery for PDM data sets.



RMS Recovery Unit Journaling will not work across a network. RMS files marked for Recovery Unit Journaling are inaccessible from a remote node running RDM applications.

Signing on to DBAID

You sign on to DBAID by responding to the prompt "PLEASE SIGN ON" with your 1–30 character user name. You must also supply a password if one was defined for you on the Directory. The password can be 1–8 alphanumeric or printable characters. You can enter the password on the same line as your user name, provided you precede the password with a space. If you enter your password on the same line as your user name, the password is displayed on the screen as you enter it. If you do not enter a password after your user name, DBAID displays a password prompt even if you do not have a password. Press RETURN in response to this prompt if you have no password; enter the password and press RETURN if you have a password. The password is not echoed when entered in response to the "Password:" prompt.

In the following example, INVENTORY-SYSTEM is the user name and SCALES is the password. When you successfully sign on, you receive the message "SUCCESSFUL COMPLETION - LEVEL 05." You can then begin entering commands.

```
SUPRA RELEASE 2.4
WELCOME TO DBAID - LEVEL -05

PLEASE SIGN ON.
>INVENTORY-SYSTEM SCALES
FSI: * VSI: = MSG: SUCCESSFUL COMPLETION - LEVEL 05
>
```

Using DBAID commands

Defining a view using DBAID involves the same syntax and considerations as when you define a view using the DBA functions explained in the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260. DBAID has four types of commands:

- ◆ Relational Data Manipulation Language (RDML) commands enable you to use test data with a defined view to make sure the view is properly defined.
- ◆ Editing commands enable you to change stored view definitions and to create new views to be tested before storing them on the Directory. After you have tested a new view with DBAID, you can store it on the Directory.
- ◆ System commands enable you to display information about the currently running DBAID Test Facility. Use these commands to display current DBAID users and active views.
- ◆ Built-in view commands enable you to inspect the view after it is opened.

The following table alphabetically lists all the DBAID commands within each category and provides a brief description and a section reference for detailed information. Commands available in the programmer's subset are indicated by a check mark (✓).

DBAID also provides a HELP facility. If you enter HELP in reply to a DBAID prompt, a list of topics available within the HELP facility displays. You can then select the topic by entering enough of the topic name for the selection to be unique. Some topics also have a list of subtopics.

Alternatively, you can directly access a topic by entering the topic name with the command HELP. For example:

```
>HELP DELETE
```

This entry gives information specific to DELETE. Note that to display information for the * command, you must use HELP ASTERISK. If you enter HELP *, the * acts as a "wildcard." Wildcard means that all topics are included in the help description.

RDML commands

Command	Programmer's subset	Description	Section reference
=	✓	Reissues the previous RDML command.	"= command" on page 148
BYE	✓	Exits DBAID.	"BYE command" on page 152
CAUTIOUS	✓	Prohibits an automatic COMMIT.	"CAUTIOUS command" on page 153
COMMIT	✓	Issues an RDM COMMIT command.	"COMMIT command" on page 159
DELETE	✓	Issues an RDM DELETE command.	"DELETE command" on page 162
ERASE	✓	Causes an RDM RESET to be issued when an "X" FSI is returned.	"ERASE command" on page 166
FORGET	✓	Removes the specific mark from the list of marks in use.	"FORGET command" on page 172
GET	✓	Issues an RDM GET command which retrieves and displays the requested row.	"GET command" on page 173
GO	✓	Issues multiple RDM GET commands and displays the rows in tabular format.	"GO command" on page 179
INSERT	✓	Issues an RDM INSERT command.	"INSERT command" on page 183
KEEP	✓	Prohibits an automatic RESET.	"KEEP command" on page 188

Command	Programmer's subset	Description	Section reference
MARK	✓	Issues an RDM MARK command. Marks the current position of the row established by the previous GET.	"MARK command" on page 193
OPEN	✓	Readies for use either a virtual or stored view.	"OPEN command" on page 195
RELEASE	✓	Issues an RDM RELEASE command. Closes all opened views and releases the occupied storage.	"RELEASE command" on page 201
RESET	✓	Issues an RDM RESET command.	"RESET command" on page 204
SIGN-OFF	✓	Signs off the user from DBAID.	"SIGN-OFF command" on page 208
SIGN-ON	✓	Identifies the user to DBAID.	"SIGN-ON command" on page 209
SURE	✓	Causes a COMMIT after each successful insert, update, or delete.	"SURE command" on page 213
UPDATE	✓	Issues an RDM UPDATE command.	"UPDATE command" on page 215

Editing commands

Command	Programmer's subset	Description	Section reference
DEFINE		Defines a name for a virtual view.	"DEFINE command" on page 161
EDIT		Readies a stored or virtual view for modification.	"EDIT command" on page 165
<i>line-number</i>		Deletes, adds, or replaces a line in the currently editable view.	"line-number command" on page 189
LIST		Lists a stored or virtual view and readies it for modification.	"LIST command" on page 191
RENUMBER		Renumbers a virtual view so that line numbering starts at 10 with each line incremented by 10.	"RENUMBER command" on page 203
UNDEFINE	✓	Removes a defined virtual view.	"UNDEFINE command" on page 214

System commands

Command	Programmer's subset	Description	Section reference
*	✓	Used with other commands to indicate the last view name used.	"* command" on page 146
BIND		Binds the view.	"BIND command" on page 149
COPY		Copies the definition of one view to another view.	"COPY command" on page 160
DENY		Removes the relationship between a user and a view on the Directory.	"DENY command" on page 164
LINESIZE	✓	Specifies the width of lines for DBAID output.	"LINESIZE command" on page 190
MARKS	✓	Lists all open MARKS and the views they are marking.	"MARKS command" on page 194
PAGESIZE	✓	Specifies the number of lines on the page/screen for DBAID output.	"PAGESIZE command" on page 198
PERMIT		Relates a view to a user on the Directory.	"PERMIT command" on page 199
REMOVE		Removes the view access definition, its binding, and the relationship between it and the database. This command is for use only by the DBA.	"REMOVE command" on page 202
SAVE		Saves a virtual view definition that has been opened with the OPEN command.	"SAVE command" on page 205
USER-LIST	✓	Displays the column definition for the view named.	"USER-LIST command" on page 218
VIEWS	✓	Displays all views active in DBAID.	"VIEWS command" on page 221

Built-in logical view commands

Command	Programmer's subset	Description	Section reference
BY-LEVEL	✓	Displays the column names in the view by level of occurrence.	"BY-LEVEL command" on page 150
COLUMN-DEFN	✓	Displays the full description of a column in a view.	"COLUMN-DEFN command" on page 154
COLUMN-TEXT	✓	Displays the short and long text for a column in a view.	"COLUMN-TEXT command" on page 158
FIELD- DEFN	✓	Displays the full description of a column in a view.	"FIELD-DEFN command" on page 167
FIELD- TEXT	✓	Displays the short and long text for a column in a view.	"FIELD-TEXT command" on page 170
SHOW-NAVIGATION	✓	Displays the access strategy used by the view.	"SHOW-NAVIGATION command" on page 207
VIEW- DEFN	✓	Displays a condensed description of the view.	"VIEW-DEFN command" on page 219
VIEWS-FOR-USER	✓	Lists the views related to the signed-on user, along with the short text for the view.	"VIEWS-FOR-USER command" on page 222

Statistics commands

Command	Programmer's subset	Description	Section reference
PRINT-STATS	✓	Displays current statistics for all views.	"PRINT-STATS command" on page 200
STATS	✓	Displays current statistics for all open views or a particular open view.	"STATS command" on page 210
STATS-OFF	✓	Displays the current statistics for all views and then disables the statistics gathering.	"STATS-OFF command" on page 211
STATS-ON	✓	Initializes statistics to zero then enables the gathering of statistics on user views on both the logical and physical level.	"STATS-ON command" on page 212

*** command**

You can use the asterisk (*) in DBAID for two functions: as a substitute for the last view name used or to denote a comment when editing a view.

*

General consideration

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

Examples

- ◆ Using * as a substitute for the last view name used:

```
OPEN VIEW
GET *      (Performs GET on VIEW)
OPEN VIEW2 = * column1,column5
GET *      (Performs GET on VIEW2)
```

◆ Using * to denote a comment:

```
* Comment line for column 1
COLUMN-1
* Comment line for column 2
COLUMN-2
* Comment line for ACCESS statement 1
ACCESS ....
* Comment line denoting end of access definition
```

Comment lines entered in the access definition of a view are saved as entered. However, comments entered before the first ACCESS statement in a view are saved at the top of the view, regardless of where you originally define them. For example, if you save the preceding view through DBA or DBAID, comment lines are reordered so that the next time you list the view, comment lines are reorganized as follows:

```
* Comment line for column 1
* Comment line for column 2
* Comment line for ACCESS statement 1
COLUMN-1
COLUMN-2
ACCESS ....
* Comment line denoting end of access definition
```

As you can see, comments before the first ACCESS statement in the view are all saved at the top of the view, whereas comments in the access definition remain as originally defined.

= command

The = command reissues the previous RDML command.

=

General consideration

The = command repeats the previous command exactly, even if the command was invalid.

Example

In this example, the = command reissues the GET NEXT command preceding it.

```
GET NEXT CUSTOMER-PRODUCT-VIEW
```

```
=
```

BIND command

The BIND command saves and binds the specified view.

BIND *view-name*

view-name

Description *Required.* Specifies the view to be saved and bound.

Format A valid view name.

Consideration You can enter * instead of a view name, causing DBAID to substitute the last view name used.

BY-LEVEL command

The BY-LEVEL command displays the column names in a view by level of occurrence, starting with level 0, followed by level 1, and so on. 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 are to be displayed.

Format A valid and opened view name.

Considerations

- ◆ If you omit the view name, RDM displays all column names for all your opened views, including columns from base views opened by derived views.
- ◆ You can enter * instead of a view name, causing DBAID to substitute the last view name used.

column-number

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

Format An integer value.

Considerations

- ◆ If you use this parameter, you must specify a view name.
- ◆ If you omit this parameter, all column names of the specified view display.

Example This example displays the column names in all opened views; in this case, the base view STOCK and the derived view PRODUCTS-IN-REGION. Columns are listed by level of occurrence.

```
>BY-LEVEL PRODUCTS-IN-REGION

! NUMBER !      VIEW NAME                !      FIELD NAME                LEVEL!
!   1    ! STOCK                        ! STOCK-NO                       0!
!   2    ! STOCK                        ! STOCK-PRODUCT                  0!
!   1    ! PRODUCTS-IN-REGION           ! REGION-NO                      0!
!   2    ! PRODUCTS-IN-REGION           ! REGION-NAME                    0!
!   3    ! PRODUCTS-IN-REGION           ! BRANCH-NO                      1!
!   4    ! PRODUCTS-IN-REGION           ! REGION-NAME                    1!
!   5    ! PRODUCTS-IN-REGION           ! STOCK-PRODUCT                  2!
!   6    ! PRODUCTS-IN-REGION           ! PRODUCT-DESC                   2!
!   1    ! *INT00003-PRODUCT            ! PRODUCT-CODE                   0!
!   2    ! *INT00003-PRODUCT            ! PRODUCT-DESC                   0!
!   1    ! *INT00002-STOCK              ! STOCK-BRANCH                   0!
!   2    ! *INT00002-STOCK              ! STOCK-PRODUCT                  0!
!   1    ! *INT00001-BRANCH             ! BRANCH-NO                      0!
!   2    ! *INT00001-BRANCH             ! BRANCH-NAME                    0!
!   3    ! *INT00001-BRANCH             ! BRANCH-REGION                 0!
!   1    ! *INT00000-REGION            ! REGION-NO                      0!
!   2    ! *INT00000-REGION            ! REGION-NAME                    0!
```

View names in the format **INTnnnn-viewname* are base views opened by the derived view immediately preceding them in the VIEW NAME column.

**INT* Indicates that this is a base view opened by a derived view.

nnnnn Indicates the order in which the base view was opened (00000 is the first view opened by the derived view, 00001 the second, 00002 the third, and so on).

-viewname Identifies the name of the base view.

It is important to identify how a base view was opened because each base view can be opened by more than one derived view, as well as independently. This is illustrated in the above example by the base view STOCK. STOCK is opened independently as the base view STOCK; at the same time, it is opened by the derived view PRODUCTS-IN-REGION. When STOCK is opened by PRODUCTS-IN-REGION, it is assigned the internal name **INT00002-STOCK* to distinguish it from the base view STOCK, opened independently, and to indicate that it is the third base view opened by PRODUCTS-IN-REGION.

BYE command

The BYE command exits the DBAID Test Facility.

BYE

General considerations

- ◆ The BYE command exits the CSVDBAID image.
- ◆ Any unsaved virtual views are erased.

CAUTIOUS command

The CAUTIOUS command disables the DBAID automatic COMMIT processing. This command is the opposite of the SURE command. When you use CAUTIOUS, DBAID does not automatically issue a COMMIT when an RDML INSERT, UPDATE, or DELETE command returns an "*" FSI. Instead, you must issue the COMMIT explicitly.

CAUTIOUS

General considerations

- ◆ DBAID normally issues a COMMIT after every successful RDML modification. The CAUTIOUS command is not required; however, it gives you more control over COMMIT commands when updating the database.
- ◆ CAUTIOUS only affects your database, not the SUPRA Server Directory (SUPRAD). An implicit COMMIT is issued to the SUPRA Server Directory by the DBAID system commands BIND, DENY, PERMIT, REMOVE, and SAVE. These COMMITs must be issued after the Directory is modified for them to take effect.

COLUMN-DEFN command

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

For compatibility purposes, you can use the **FIELD-DEFN** command in the same manner as the COLUMN-DEFN command.

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

view-name

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

Format Must be a valid and opened view.

Considerations

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

column-name

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

Considerations

- ◆ The column must already be part of the view.
- ◆ If you use this parameter, you must have specified a view name.
- ◆ If you omit this parameter, the COLUMN-DEFN command displays all column descriptors for each column of the specified view, one at a time.

Example

This example displays a description of one of the columns in the STOCK view. See the following table for an explanation of each column descriptor.

VIEW-NAME	(+) STOCK
COL-NAME	(+) STOCK-BRANCH
COL-POS	(+) 0
COL-LEN	(+) 4
COL-ASI-POS	(+) 33
COL-DEC	(+) 0
COL-OUTP-LEN	(+) 4
COL-MASK-LEN	(-) 0
COL-FORMAT	(+) C
COL-MASK	(-)
COL-HEADING	(-)
COL-DEL-OPT	(+) Y
COL-INS-OPT	(+) Y
COL-UPD-OPT	(+) N
COL-REDUND	(+) N
COL-CONSTANT	(+) N
COL-LEVEL	(+) 0
COL-KEY-NUM	(+) 1
COL-REQUIRED	(+) Y
COL-UNIQUE	(+) Y
COL-EDIT-TRANS	(+)
COL-ORDERING	(-)
COL-SIGNED	(+) Y
COL-NULLS-OK	(+) N
COL-NULL-LEN	(-) 0
COL-NULL-VAL	(-)
COL-DOMAIN	(-)
COL-VAL-TYP	(-)
COL-GET-VAL	(+) Y
COL-MIN-LEN	(-) 0
COL-MIN-VAL	(-)
COL-MAX-LEN	(-) 0
COL-MAX-VAL	(-)
COL-VAL-TABLE	(-)
COL-EXIT	(-)
COL-SRC-TYP	(+) F
COL-SRC-COL	(+) STOCK-BRANCH-ID
COL-SRC-REL	(+) STCKBRAN
COL-INT-REL	(+) STCK
COL-RC	(+)

Keys for the view

VIEW-NAME Name of the view

COL-NAME Name of the column

Data needed to read the column from the row

COL-POS Offset of column value from (0) start of row

COL-LEN Length of column value in bytes

COL-ASI-POS Distance ASI for column is offset from start of user buffer

Data needed to display the column

COL-DEC Number of decimal places

COL-OUTP-LEN Edited output length

COL-MASK-LEN Length of output mask

COL-FORMAT Column format

COL-MASK Column mask

COL-HEADING Column heading

Logical data about the column

COL-DEL-OPT Y = Column may be deleted

COL-INS-OPT Y = Column may be inserted

COL-UPD-OPT Y = Column may be updated

COL-REDUND Y = Column is redundant

COL-CONSTANT Y = Column is a constant

COL-LEVEL Level of occurrence

COL-KEY-NUM 0 - 9 = Column key number

COL-REQUIRED Y = Column is required

COL-UNIQUE Y = Column is unique

COL-EDIT-TRANS Reserved for future use

COL-ORDERING A = Ascending order, D = Descending order

COL-SIGNED Y = Column is signed

Data about null value for the column

COL-NULLS-OK	Y = Nulls are allowed
COL-NULL-LEN	Length of the null value
COL-NULL-VAL	Null value in external format

Validation criteria for the column

COL-DOMAIN	Domain name, if any
COL-VAL-TYP	Validation type, R = Range, T = Table, E = Exit
COL-GET-VAL	Y = Validation done after GET
COL-MIN-LEN	Length of minimum value
COL-MIN-VAL	Minimum value in external format
COL-MAX-LEN	Length of maximum value
COL-MAX-VAL	Maximum value in external format
COL-VAL-TABLE	Validation table name
COL-EXIT	Validation exit name

Source column data

COL-SRC-TYP	Source type for the column: F = Data set, V = View.
COL-SRC-COL	If COL-SRC-TYP is "F," this field contains the logical data item name; if COL-SRC-TYP is "V," this field contains the source column name.
COL-SRC-REL	If COL-SRC-TYP is "F," this field contains the physical data item name; if COL-SRC-TYP is "V," this field contains the source view name.
COL-INT-REL	If COL-SRC-TYP is "F," this field contains the data set name; if COL-SRC-TYP is "V," this field contains the user view name.
COL-RC	If COL-SRC-TYP is "F," this field contains the record code, if any.

COLUMN-TEXT command

The COLUMN-TEXT command displays the comments for a column in a view. For compatibility purposes, you can use the FIELD-TEXT command in the same manner as the COLUMN-TEXT command.

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

view-name

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

Format Must be a valid and opened view.

Considerations

- ◆ If you omit this parameter, the COLUMN-TEXT command displays the short and long text for all your opened views.
- ◆ You can enter * instead of a view name. This causes DBAID to substitute the last view name used.

column-name

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

Considerations

- ◆ The column must already be part of the view.
- ◆ If you use this parameter, you must specify a view name.
- ◆ If you omit this parameter, the COLUMN-TEXT command displays the comments for all columns.

COMMIT command

The COMMIT command issues an RDM COMMIT request. All updates since the last COMMIT are made permanent in the database.

COMMIT

General considerations

- ◆ DBAID automatically issues a COMMIT after every successful modification (INSERT, UPDATE, and DELETE). The COMMIT command is not required in DBAID.
- ◆ Automatic commit processing is turned off with the **CAUTIOUS** command and turned on with the **SURE** command.

COPY command

The COPY command copies the view definition of one view to another view.

COPY *view-name₁* *view-name₂*

view-name₁

- Description** *Required.* Identifies the name of the view to be copied.
- Format** Must be a valid view on the Directory for the database specified or a virtual view.

view-name₂

- Description** *Required.* Identifies the new name for the view being copied.
- Format** 1–30 alphanumeric characters and hyphens. The first character must be alphabetic.

Considerations

- ◆ After being copied, the new view is listed (see “[LIST command](#)” on page 191) and is available for editing.
- ◆ The copied view is virtual; it is not copied onto the Directory.

General consideration

DBAID first looks for a virtual view with the name *view-name₁*. If the view is not found, DBAID searches the Directory for the view. Once DBAID finds the view text on the Directory, it creates a virtual view *view-name₁*. DBAID then copies the contents of *view-name₁* to *view-name₂*, and lists *view-name₂*. In DBAID, both copies result as virtual views.

Example

This example copies CUSTOMER from the Directory and names it NEW-CUSTOMER. NEW-CUSTOMER is listed and is available for editing.

```
COPY CUSTOMER NEW-CUSTOMER
```

DEFINE command

The DEFINE command defines a new view to DBAID.

DEFINE *view-name*

view-name

Description *Required.* Specifies the name of a new view.

Format 1–30 alphanumeric characters and hyphens. The first character must be alphabetic.

General considerations

- ◆ The DEFINE command does not go to the Directory to retrieve a view. It creates a virtual view that exists only within the DBAID execution. You can eventually save this view on the Directory (see “[SAVE command](#)” on page 205).
- ◆ Once you issue the DEFINE command, you can use the line-number command (see “[line-number command](#)” on page 189) to define the columns in your view.

Example This example defines the view CUSTOMER to DBAID.

```
DEFINE CUSTOMER
```

DELETE command

The DELETE command issues an RDM DELETE request, which removes one or more row occurrences from the database.

DELETE [ALL] *view-name*

ALL

Description *Optional.* Deletes all rows that satisfy 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 that contains the row(s) to be deleted.

Format Must be a valid and opened view.

Considerations

- ◆ We recommend that before performing the DELETE, you perform a successful GET command that contains a FOR UPDATE clause, in case another task changes the row between the GET and the DELETE.
- ◆ You can enter * instead of a view name, causing DBAID to substitute the last view name used.

General considerations

- ◆ RDM deletes rows only if the ALLOW clause specifies DEL or ALL. See Example 4.
- ◆ RMS interfile integrity is maintained only for those RMS data sets within the view. RDM does not check indexes to other RMS data sets not included in the access definition.

Examples

- ◆ This example deletes one occurrence of SAMPLE-VIEW obtained by using the value in KEY1:

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

- ◆ This example deletes all occurrences of rows containing KEY1:

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

The above example works as if the following loop were performed:

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

- ◆ This example deletes all rows in SAMPLE-VIEW:

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

- ◆ This RMS example shows the statements used by the GET and DELETE commands to delete all rows from the CUST and ORDR data sets:

```
Name:    CUSTOMER-ORDER-VIEW
KEY CUSTOMER-NUMBER = CUSTOMER-ID = ORDER-CUST-ID
KEY ORDER-NUMBER = ORDER-ID
ACCESS CUST
  USING CUSTOMER-NUMBER
  ALLOW DELETE INSERT
ACCESS ORDR
  USING CUSTOMER-NUMBER
  ALLOW ALL
GET CUSTOMER-ORDER-VIEW
DELETE ALL CUSTOMER-ORDER-VIEW
```

DENY command

The DENY command revokes a user's privilege to use a view in the SUPRA Server Directory. The command removes the relationship between the user and the view entities on the Directory. This command can provide security because it allows the DBA to define in the Directory who can use a view. Note that use of global views can be used to override this security.

DENY *view-name user-name₁ [...user-name_n]*

view-name

Description *Required.* Specifies the name of the view to which the user is denied access.

Format Must be a valid view.

Consideration You can enter an * instead of a view name. DBAID then substitutes the last view name used.

user-name₁ [...user-name_n]

Description *Required.* Specifies the name of the user who is to be denied access to a view.

Format Must be a valid user ID, defined on the Directory.

Consideration You can specify more than one user in a single DENY command by separating each user name with a single space.

General considerations

- ◆ Use of global views may override the user-to-view relationship in the SUPRA Server Directory.
- ◆ You can use the DENY command to remove the relationship between a user and a view, regardless of whether the relationship was created with Directory maintenance or the PERMIT command.
- ◆ After successfully removing the relationship for each user, DBAID issues a **COMMIT** to the SUPRA Server Directory database SUPRAD.
- ◆ If an error occurs while removing the relationship, DBAID issues a **RESET** and terminates processing of the command.

EDIT command

The EDIT command prepares a view for modification.

EDIT *view-name*

view-name

Description *Required.* Identifies the name of the view to be edited.

Format Must be a valid view name.

Consideration You can enter * instead of a view name, causing DBAID to substitute the last view name used.

General considerations

- ◆ When you issue the EDIT command, the system first searches for a virtual view. If it is not found, the system then searches the Directory.
- ◆ Once you issue the EDIT command, you can use the line-number command (see “[line-number command](#)” on page 189) to modify your view.
- ◆ The EDIT mode is automatically entered after a [COPY](#), [DEFINE](#), or [LIST](#) command.

Example This example prepares the view CUSTOMER for modification.

```
EDIT CUSTOMER
```

ERASE command

The ERASE command causes DBAID to issue an automatic RDM **RESET** whenever an "X" FSI is returned from an RDML command. This command is the opposite of the **KEEP** command.

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 and opened view.

Considerations

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

column-name

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

Considerations

- ◆ The column must already be part of the view.
- ◆ If you use this parameter, you must have specified a view name.
- ◆ If you omit this parameter, the FIELD-DEFN command displays all column descriptors for each column of the specified view or all virtual views, one at a time.

Example

This example displays the full description of all columns in all open views. See the following table for an explanation of each column descriptor.

```
> FIELD-DEFN
VIEW-NAME                (+) CUSTOMER
FIELD-NAME                (+) CUSTOMER-NUMBER
FIELD-POS                 (+) 0
FIELD-LEN                 (+) 5
ASI-POS                   (+) 60
FIELD-DEC                 (+) 0
OUTPUT-LEN                (+) 5
MASK-LEN                  (+) 15
FORMAT                    (+) Z
EDIT-MASK                 (-)
HEADING                   (-)
DELETABLE                 (+) Y
INSERTABLE                (+) Y
REPLACEABLE               (+) N
FIELD-LVL                 (+) 0
KEY-NUMBER                (+) 1
REQUIRED                  (+) Y
UNIQUE                    (+) Y
EDIT-TRANS                (-)
ORDERING                  (-)
SIGNED                    (+) N
```

Column descriptor	Explanation
VIEW-NAME	The name of the view being described.
FIELD-NAME	The name of the column being described.
FIELD-POS	The position of the column in the user's buffer, starting at byte 0.
FIELD-LEN	The length of the column in bytes.
ASI-POS	The position of the ASI of this column in the user buffer.
FIELD-DEC	The number of decimal places in the column.
OUTPUT-LEN	The length of the output column.
MASK-LEN	The length of the edit mask.
FORMAT	The format of the column.
EDIT-MASK	Not implemented for this release.
HEADING	Not implemented for this release.
DELETABLE	Indicates whether the row may be deleted.
INSERTABLE	Indicates whether the row may be inserted.
REPLACEABLE	Indicates whether this column may be updated.
FIELD-LVL	Indicates the level of the row which contains this column.
KEY-NUMBER	Indicates which key column this is; 0 indicates the column is not a key, 1 is the first key, and so on, up to 9.
REQUIRED	Indicates the column must not be null when performing updates or inserts.
UNIQUE	Indicates the column is a unique key.
EDIT-TRANS	Not implemented for this release.
ORDERING	Indicates a linkpath is ordered using this column.
SIGNED	Whether or not the column is signed.

FIELD-TEXT command

The FIELD-TEXT command displays the comments for a column in a view.

FIELD-TEXT [view-name [column-name]]

view-name

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

Format Must be a valid and opened view.

Considerations

- ◆ If you omit this parameter, the FIELD-TEXT command displays the short and long text for all of your opened views.
- ◆ You can enter * instead of a view name, causing DBAID to substitute the last view name used.

column-name

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

Considerations

- ◆ The column must already be part of the view.
- ◆ If you use this parameter, you must specify a view name.
- ◆ If you omit this parameter, the FIELD-TEXT command displays the comments for all columns.

Example This example displays the comments for all columns in all open views:

```

>FIELD-TEXT
!!          VIEW NAME          !          FIELD NAME          !
!-----!-----!-----!
!STOCK          -!STOCK-QUANTITY          !
!-----!-----!-----!
!                                COMMENTS                                !
!-----!-----!-----!
!The total amount of inventory for the product at this branch location. !
.....
***MORE***
>

!          VIEW NAME          !          FIELD NAME          !
!-----!-----!-----!
!STOCK          !STOCK-BIN-LOCATION          !
!-----!-----!-----!
!                                COMMENTS                                !
!-----!-----!-----!
!The specific bin location in which the product can be found.          !
.....
***MORE***>

```

FORGET command

The FORGET command removes the specified mark and its resources from the list of marks in use. It also clears the name and resources allocated by a previous **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, the indicated mark is released and cannot be used 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 *literal*₁[*literal*₂...*literal*_{*n*}]]

NEXT
LAST
SAME
FIRST
PRIOR

Description *Optional.* Specifies the order of retrieval of rows.

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-value*" phrase for predictable results.
- ◆ If you are accessing the data set via a secondary key that supports REVERSE DIRECTIONAL search, you can use GET PRIOR.
- ◆ For a non-unique key:
 - GET NEXT. Retrieves the next occurrence of the row within the generic group immediately after the current row, or the first row if no current position exists.
 - 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. Performs a read reverse within the group of non-unique keyed rows.
- ◆ For RMS, GET LAST and GET PRIOR are not supported. To access base RMS data sets, supply the key.

view-name

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

Format Must be a valid and opened view.

Consideration You can enter * instead of a view name, causing 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 upon the current contents of the row.
 - ◆ If you do not need to be certain of the content of the row, use GET without the FOR UPDATE phrase. When the UPDATE or DELETE function is performed, automatic record holding performs the lock before modifying the row.
 - ◆ FOR UPDATE locks all physical resources until the row(s) are released by a COMMIT or RESET. Automatic COMMITs are issued on GET, INSERT, UPDATE, and DELETE, unless you have disabled it (see SURE) with a CAUTIOUS command. This practice can lead to system inefficiency and other tasks receiving a HELD status.
-

AT mark-name

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

Consideration If you use the AT phrase, you cannot also use the USING phrase.

USING *literal*₁[*literal*₂...*literal*_n]

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

Format The values must be part of a valid view. Use 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 definition. No more than nine keys are allowed in one view.
- ◆ Any omitted keys are treated as generic keys. Using generic keys is a convenient feature for allowing both direct access to a view and a sequential scan of many rows. All occurrences of a particular unspecified column are returned as long as the other keys are satisfied.

- ◆ You can force RDM to perform a generic read at the PDM level by omitting characters from the right of the key value, and replacing them with one of the wildcard characters:

* default character for equal or next match

= default character for equal or only match

Generic reads are more efficient at the PDM level than at the RDM level.

Note that you can specify your own wildcard characters by defining the logical name CSI_WILD_EQ to point to the equal/next wildcard character, and CSI_WILD_EN to point to the equal only wildcard character. Refer to the *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130.

- ◆ The order of specified keys in the USING phrase corresponds to the order of key declarations in your column definition. You can omit keys only to the right; you cannot skip a key, then give others after it.
- ◆ If you use the USING phrase, you cannot also use the AT phrase.

Examples

- ◆ This example returns the row whose key starts with the letters JO from the view LASTNAME. No position is specified, so NEXT is assumed. However, because this is the first GET, the first row with this key is returned.

```
GET LASTNAME USING 'JO*'
```

- ◆ This example returns the next row with the key JONES. When you have read all JONES's records, you get the message "OCCURRENCE NOT FOUND":

```
GET * USING 'JONES'
```

- ◆ This example returns the first JONES with the initials APQ:

```
GET * USING 'JONES' 'APQ'
```

- ◆ This example uses the default NEXT. The next row is returned—for example, the row following JONES APQ. You can use this command repeatedly to get all rows in a view, such as the rest of the Joneses, then all Robinsons, and so on:

```
GET *
```

- ◆ In this RMS example, GET retrieves data from the ORDR data set using the first physical key:

```
GET FIRST ORDER-VIEW USING 225600x100
```

- ◆ This RMS example uses the view REGION with the GET command to retrieve data from the REGN data set:

```
KEY      REGION-NUMBER = REGION-ID
        REGION-NAME
```

```
ACCESS REGN
        WHERE REGION-ID = REGION-NUMBER
        ALLOW ALL
```

* To restrict deletions of REGIONS that contain branches.

```
ACCESS BRAN
        WHERE BRANCH-REGION-ID == REGION-ID
```



REGION-ID and BRANCH-REGION-ID are in different domains so that BRANCH-REGION-ID can be NULL. Therefore, domain override is required.

- ◆ In this example, the physical key of the data set SAMP consists of three subparts. RDM performs a generic read when the application program supplies a value for either or for a combination of SAMPLE-KEY1 and SAMPLE-KEY2:

```
GET SAMPLE-VIEW USING CUST01
GET SAMPLE-VIEW USING CUST01 ORDR02
```

- ◆ All products for the requested order are returned when the values for SAMPLE-KEY1 and SAMPLE KEY2 are supplied. The GET command uses the full physical key if all three logical keys are supplied:

```
GET SAMPLE-VIEW USING CUST01
                        ORDR02
                        PART11
```

- ◆ This RMS example is similar to the preceding example. However, SAMPLE-KEY-2 is left out of the USING phrase to force a generic read of the SAMP data set:

```
GET SAMPLE-VIEW USING CUST02
```

- ◆ Fields can be left out of the USING clause only from the right. Generic reads can be performed from within RDM only if they are based on the logical key supplied within the USING clause in the ACCESS statements.

GO command

The GO command issues a **GET** request based on a single key, followed by a series of sweeping GET requests (the rows are displayed in a tabular format).

```
GO [ NEXT
    PRIOR ] view-name
```

```
[ START [ NEXT
          LAST
          SAME
          FIRST
          PRIOR
          AT mark - name ] ]
```

```
[ FOR number - of - rows ]
```

```
[ [ FROM
    USING ] literal1 [ literal2...literaln ] ]
```

[NEXT
[PRIOR]

Description *Optional.* Specifies the positional modifier to be used in subsequent retrievals after the initial access by the GET command.

Default NEXT

view-name

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

Format Must be a valid and opened view.

Consideration You can enter * instead of a view name, causing DBAID to substitute the last view name used.

START [NEXT
LAST
SAME
FIRST
PRIOR
AT mark - name]

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

Default FIRST if GO NEXT is specified
PRIOR if GO PRIOR is specified

FOR number-of-rows

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

Default 16,777,216

Format Integer value in the range 1–16,777,216.

Consideration GET NEXTs will be issued until the count is exhausted, or until the last row is retrieved, whichever occurs first.

[FROM
USING] literal₁ [literal₂... literal_n]

Description *Optional.* Identifies the values used 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 Key values are used only on the initial access; the scan is unqualified.
USING Key values are used for both the initial access 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 definition. You can omit keys only to the right; you cannot skip a key, then give others after it.
- ◆ Any omitted keys are treated as generic keys. Using generic keys allows for both direct access to a view and a sequential scan of many rows. All occurrences of a particular unspecified key are returned 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 definition.

General considerations

- ◆ The output is displayed in columns, where possible. If more data is to be displayed than fits on a screen, DBAID will determine a different format.
- ◆ After the GO command displays a page of rows (see “[PAGESIZE command](#)” on page 198), the prompt *****MORE***** is issued. You can continue the display on the next page after input of a blank line.
- ◆ At the end of the rows retrieved by GO, the prompt *****END***** is issued.
- ◆ “FOR *number-of-rows*” is not recommended for online use because it does not pause until the last screen.
- ◆ The GO command always looks ahead one 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 might appear to be skipped.

Examples

- ◆ The command "GO VIEW START AT VIEW-MARK1 USING (VIEW-KEY-VALUE)" issues the following sequence of RDM GET commands until a not-found FSI is returned:

```
GET VIEW AT VIEW-MARK1
GET NEXT VIEW USING (VIEW-KEY-VALUE)
GET NEXT VIEW USING (VIEW-KEY-VALUE)
.
.
.
```

- ◆ The command "GO PRIOR VIEW START LAST FROM (VIEW-KEY-VALUE)" issues the following sequence of RDM GET commands until a not-found FSI is returned:

```
GET LAST VIEW USING (VIEW-KEY-VALUE)
GET PRIOR VIEW
GET PRIOR VIEW
.
.
.
```

INSERT command

The INSERT command issues an RDM INSERT request. The INSERT places a row in the physical database based on the relative location specified.

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

NEXT
LAST
FIRST
PRIOR

Description	<i>Optional.</i> Specifies where the row will be inserted in relation to existing rows. The view definition may override this specification.
Default	NEXT If not positioned in the view, NEXT defaults to LAST, and PRIOR defaults to FIRST.
Options	For non-unique key values:
	NEXT Places a row after the current row. If no current position exists, the row is placed in the last position in the view.
	LAST Places a row in the last position of the view.
	FIRST Places a row in the first position in the view.
	PRIOR Places a row before the current row. If no current position exists, the row is placed in the first position in the view.

view-name

Description *Required.* Specifies the name of the view where you want the rows inserted.

Format Must be a valid and opened view.

Considerations

- ◆ You can enter * instead of a view name, causing DBAID to substitute the last view name used.
- ◆ After the column values are entered, the row is displayed. The message INSERT (Y/N) is displayed and a response is required. Y inserts the rows. N cancels the insert.

MASS

Description *Optional.* Inserts many rows.

Considerations

- ◆ The positioning parameter you specify is used by RDM on every insert command issued by mass insert.
- ◆ Rows are input immediately following this command after the prompts MASS INSERT PROCESSING INITIATED and ENTER "END." TO EXIT MASS INSERT.
- ◆ Rows are inserted as flat records. Separate the columns with commas. To insert rows that are longer than one line, terminate the list of values with a comma.
- ◆ If columns have no values, enter two consecutive commas to indicate their absence. This value is treated as a null value for packed or numeric columns, as a large number for binary columns, and as blanks for character columns.
- ◆ If columns contain single quotes (apostrophes), replace them with two single quotes (not double quotes) and enclose the entire string in single quotes. If columns contain spaces, enclose the entire string in single quotes.
- ◆ Specify "END." after you input all rows to be inserted into the view.
- ◆ To place multiple rows on a single line, leave a blank between rows. Do not specify the view name while doing a mass insert.
- ◆ Processing stops if ten errors are detected while using MASS insert; otherwise, enter "END." to terminate inserting.

General considerations

- ◆ If you use INSERT without MASS, DBAID prompts you for values even if the view does not allow inserts.
- ◆ Quotes can be used to include blanks in character strings.

Examples The following examples use INSERT in an online environment. The > indicates user input.

- ◆ This example inserts a row in the physical database:

```
>INSERT *
NUMBER
>9998
PRODUCT
>AAAA
INSTALLED
>100893

NUMBER                ( ) 9998
PRODUCT                ( ) AAAA
INSTALLED              ( ) 100893
INSERT (Y/N)?
>Y
FSI: * VSI: + MSG: SUCCESSFUL COMPLETION
```

- ◆ These examples use MASS to insert one or more rows without reference to column names. Use a blank to indicate the end of an inserted row. You can also enter one or more rows per line, using a comma to carry part of a row to the next line. Use END. to stop mass inserting:

```
>INSERT * MASS
MASS INSERT PROCESSING INITIATED.
ENTER "END TO EXIT MASS INSERT.
>9997,BBBB,100793
FSI: * VSI + MSG: SUCCESSFUL COMPLETION
```

```
>9996,CCCC,
>100683
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
>9995,DDDD,100593 9994,EEEE,100493 9993,FFFF,100393
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
>END.
MASS INSERT PROCESSING COMPLETED.
```

- ◆ In this example, the following view is used with the INSERT command. An insert is attempted on each data set. If at least one row is inserted, the operation is successful.

```
BRANCH-STOCK-PRODUCT
KEY BRANCH-NUMBER = BRANCH-ID
KEY PRODUCT-CODE = STOCK-PRODUCT-ID = PRODUCT-ID
PRODUCT-DESCRIPTION
ACCESS BRAN
WHERE BRANCH-ID = BRANCH-NUMBER
ALLOW ALL
ACCESS STCK
WHERE STOCK-BRANCH-ID = BRANCH-ID
ALLOW ALL
ACCESS PROD
WHERE PRODUCT-ID = STOCK-PRODUCT-ID
ALLOW ALL
```

- ◆ This example shows how to prohibit an insert on a particular view by not coding INS or ALL on the ALLOW clause. In this example, whether or not the PROD row exists, the ORDR row is not inserted. If the row exists, a message indicating that an invalid value is in a required column appears, and an ASI of V is returned on the key columns.

```
BRANCH-STOCK-PRODUCT
KEY BRANCH-NUMBER = BRANCH-ID
KEY PRODUCT-CODE = STOCK-PRODUCT-ID = PRODUCT-ID
    PRODUCT-DESCRIPTION
ACCESS BRAN
    WHERE BRANCH-ID = BRANCH-NUMBER
    ALLOW ALL
ACCESS STCK
    WHERE STOCK-BRANCH-ID = BRANCH-ID
    ALLOW ALL
ACCESS PROD
    WHERE PRODUCT-ID = STOCK-PRODUCT-ID
    ALLOW DEL
```

KEEP command

The KEEP command disables the DBAID automatic **RESET** feature specified with the **ERASE** command. This command is the opposite of the ERASE command. KEEP prohibits DBAID from issuing a RESET when it receives an FSI of "X" from the view. Instead, DBAID "keeps" the database as it is and allows the user to decide whether to RESET or not.

KEEP

General considerations

- ◆ KEEP is the default.
- ◆ KEEP does not affect the RESET that may be issued to the SUPRA Server Directory database, SUPRAD, by the DBAID systems commands (**REMOVE**, **SAVE**, **BIND**, **PERMIT**, and **DENY**) when an error occurs.

line-number command

The line-number command deletes, adds, or replaces a Data Definition Language (DDL) statement in the currently editable view.

line-number [ddl-statement]

line-number

Description *Required.* Indicates the number of the line to be deleted, added, or replaced.

Format 1–4 numeric characters.

Consideration If the line number is less than four digits, DBAID adds zeroes to the front of the number. For example, 10 becomes 0010. If the number is longer than four digits, it is shortened to the first four digits.

ddl-statement

Description *Optional.* Specifies the view definition statement to be added or replaced.

Format Must be a valid DDL statement.

Consideration If the line-number is used without a following ddl-statement line, the line is deleted from the view definition.

General considerations

- ◆ Before you can use this command, you must first issue a **COPY**, **DEFINE**, **EDIT**, or **LIST** command.
- ◆ You can enter a maximum of 200 lines in DBAID for a single view.

Example This example illustrates the use of various line-number commands:

```
>LIST VIEW1
  10 CUSTOMER-NUMBER
  20 CUSTOMER-ADDRESS
  30 CUSTOMER-PHONE-NUMBER
  40 ACCESS CUST
>10 KEY CUSTOMER-NUMBER    Replaces line 10
>15 CUSTOMER-NAME         Inserts line 15
>30                       Deletes line 30
>LIST VIEW1
  10 KEY CUSTOMER-NUMBER
  15 CUSTOMER-NAME
  20 CUSTOMER-ADDRESS
  40 ACCESS CUST
```

LINESIZE command

The LINESIZE command specifies the number of characters to be displayed on a line or displays the current line-size setting.

LINESIZE [*number-of-characters*]

number-of-characters

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

Default 77

Options 12-256

Consideration If you omit this parameter, the command displays the current LINESIZE setting.

LIST command

The LIST command displays a saved or virtual view and readies it for modification.

LIST *view-name*

view-name

Description *Required.* Identifies the view to be displayed.

Format Must be a valid view.

Considerations

- ◆ If the view is not a virtual view, DBAID searches the Directory for the view text, as long as the view is not opened.
- ◆ You can enter * instead of a view name, causing DBAID to substitute the last view name used.

General considerations

- ◆ The SUPRA Server Directory database, SUPRAD, is not available when the logical definition CSI_NODIRECTORY is defined as TRUE.
- ◆ Once you issue the LIST command, you can use the **line-number** command (see “**line-number command**” on page 189) to modify your view.
- ◆ If a LIST command returns the message "NO VIEW TEXT SINCE VIEW IS NOT VIRTUAL," the view was opened during the current session without first being listed using either the LIST command or the DEFINE command (see “**DEFINE command**” on page 161). To correct, do a RELEASE, then UNDEFINE, and then LIST. The view must be opened again to execute it.
- ◆ LIST automatically issues an EDIT (see “**EDIT command**” on page 165).
- ◆ Using LIST before OPEN reads the text for the view definition from the Directory without verifying that the DBAID user is related to the view.
- ◆ DBAID can create views when you enter text with COPY, DEFINE, EDIT, or LIST. If LIST is used on a view in the Directory, the text becomes a virtual view which DBAID can modify. Virtual views enable you to open a view without relating it to a user.
- ◆ This command can be used only by users with a Directory access authority of PRIVILEGED (PRIV) or DBA/UTILITIES (DA).

Example

The following lists the view BRANCHES-IN-REGION:

```
KEY   REGION-NUMBER
      REGION-NAME
KEY   BRANCH-NUMBER
      BRANCH-NAME
ACCESS REGION
      ONCE
      USING REGION-NUMBER
ACCESS BRANCH
      WHERE BRANCH-REGION = REGION-NUMBER
      ALLOW INSERT UPDATE
```

MARK command

The MARK command marks the current position of the 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 and opened view.
- Consideration** You can enter * instead of a view name, causing DBAID to substitute 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 row.

General considerations

- ◆ The AT clause in the GET command repositions the view at the position set by the MARK command.
- ◆ You can create any number of marks for a logical user view, but to conserve space, reuse marks when possible.

Example This example marks the current position of the row:

```
>MARK CUSTOMER AT REMEMBER-CUSTOMER
```

You can do other GETs on CUSTOMER and return to this mark immediately.

MARKS command

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

MARKS

Example This example lists all open marks and the views (CUSTOMER-PROD) they are marking:

```
>MARKS  
  
          MARK NAME                VIEW NAME  
MARK6                CUSTOMER-PROD  
MARK5                CUSTOMER-PROD  
MARK4                CUSTOMER-PROD  
MARK3                CUSTOMER-PROD
```

OPEN command

The OPEN command reads 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 to be used in DBAID.

Format 1–30 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ If you omit this parameter, the view name is used.
- ◆ Use this command with the column parameter to create many smaller user views from one common view.
- ◆ To open a view that is not listed or defined in the same session of DBAID, the user must be related to the view in the Directory or the Global View file, if any.

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 * instead of a view name, causing DBAID to substitute the last view name used.
- ◆ The LIST command makes view text available to DBAID. If you issue an OPEN command on a view instead of issuing a LIST command, RDM directly opens the view without making text available to DBAID. When RDM opens the view, it checks for a global version first, then checks for a bound version if no global version exists. If neither a global nor a bound version exists, RDM opens the copy of the view stored on the Directory. This will affect you if your view text differs from the global or bound version.



If the logical CSI_NODIRECTORY is defined as TRUE, you will not be able to access the SUPRA Server Directory database SUPRAD.

- ◆ If a virtual view was released, you must undefine and reopen the view with full specification.

column₁,...,column_n

Description *Optional.* Identifies the column(s) to be included in the user view.

Considerations

- ◆ The columns must already be part of the view being opened.
- ◆ You can continue the list of column names on successive lines by ending the current line with a comma. This will be necessary if the current line size is less than the space required to enter all columns in the row.
- ◆ The **USER-LIST** command displays the list of columns used to open the view.

General consideration

- ◆ OPEN returns the following message, containing information about the storage used:

nnnnn BYTES USED IN OPENING VIEW

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

Example

This example opens the user view CUSTOMER-BRANCH-ONLY. The view comprises a subset of all columns in the base view CUSTOMER.

```
>OPEN CUSTOMER-PRODUCT-ONLY = CUSTOMER CUSTOMER-NUMBER,CUSTOMER-
  BRANCH
```

Only CUSTOMER-NUMBER and CUSTOMER-BRANCH are returned by GET CUSTOMER-BRANCH-ONLY, even though CUSTOMER has 10 columns defined.

PAGESIZE command

The PAGESIZE command specifies the number of lines to be displayed on a screen/page or displays the current page-size setting.

PAGESIZE [*number-of-lines*]

number-of-lines

Restriction The value must be greater than 10.

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

Format 2 or more numeric characters.

Consideration If you omit this parameter, the command displays the current page-size setting.

General consideration

The initial page size is 24 lines.

PERMIT command

The PERMIT command relates a view to a user(s) on the Directory. This command provides security since it allows the DBA to define user-to-view authorization through DBAID.

PERMIT *view-name* *user-name*₁ [...*user-name*_{*n*}]

view-name

Description *Required.* The name of the view that you are relating to a user.

Format Must be a valid view.

Consideration You can enter * instead of a view name, causing DBAID to substitute the last view name used.

*user-name*₁ [...*user-name*_{*n*}]

Description *Required.* Specifies the name of the user you are relating to the view on the Directory.

Format Must be a valid user ID, defined on the Directory.

Consideration You can specify more than one user in a single PERMIT command by separating each user name with a single space.

General considerations

- ◆ The DBAID PERMIT command can be used instead of the Logical View User Authorization screen in the DBA utility.
- ◆ After successfully relating the view to each user, DBAID issues a COMMIT to the Directory database SUPRAD.
- ◆ If an error occurs while relating a user, DBAID issues a **RESET** to the Directory database SUPRAD and terminates processing of the command.

PRINT-STATS command

The PRINT-STATS command causes RDM to display the current statistics for all opened views. You can issue the command numerous times during a session after you have first issued a **STATS-ON** command.

PRINT-STATS

General considerations

- ◆ The STATS-ON command must precede the first PRINT-STATS command. If you do not first issue STATS-ON, PRINT-STATS has no effect.
- ◆ You can issue a **STATS-OFF** command to discontinue statistics gathering. The **BYE** and **SIGN-OFF** commands print statistics and then turn statistics gathering off.
- ◆ The PRINT-STATS command can be used to keep a statistical running total.

Example

In the following example, PRINT-STATS is used to print statistics after each RDML operation.

```
STATS-ON
GET NEXT BRANCH-LOCATION
.
.
PRINT-STATS
UPDATE BRANCH-LOCATION
.
.
PRINT-STATS
```

RELEASE command

The RELEASE command issues an RDM RELEASE, which closes a specific view or all views that are opened and releases the occupied storage.

RELEASE [*view-name*]

view-name

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

Format Must be a valid and opened view.

Considerations

- ◆ You can enter * instead of a view name, causing DBAID to substitute the last view name used.
- ◆ If you omit this parameter, all your opened views are released.

General considerations

- ◆ The definition of any view is retained, allowing subsequent retrieval and processing.
- ◆ This command does not affect virtual view text of the view(s).

REMOVE command

The REMOVE command removes the view and the relationship between it and the database from the Directory.

REMOVE *view-name*

view-name

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

Format Must be a listed (**LIST**) or edited (**EDIT**) view.

Considerations

- ◆ You can enter * instead of a view name, causing DBAID to substitute the last view name used.
- ◆ DBAID displays the following prompt asking for confirmation: "REMOVE EXISTING LOGICAL VIEW (Y/N?)."

General consideration

- ◆ You must list the view before removing it. This protects you from inadvertently removing views due to spelling errors.

Example This example removes PRODUCT-VIEW from the Directory. The view is still a virtual view in DBAID.

```
>REMOVE PRODUCT-VIEW
```

RENUMBER command

The RENUMBER command renumbers a virtual view so the line numbering starts at 10 with each line incremented by 10.

RENUMBER *view-name*

view-name

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

Format Must be a valid and opened view.

Consideration You can enter * instead of a view name, causing DBAID to substitute the last view name used.

RESET command

The RESET command issues an RDM RESET request. A RESET rolls back any database updates for the current user since the last COMMIT point.

RESET

General considerations

- ◆ Only use RESET after unsuccessful updates. DBAID issues a COMMIT after every successful update unless you have issued the CAUTIOUS command.
- ◆ DBAID does not automatically issue a RESET command when an "X" FSI is returned.

SAVE command

The SAVE command stores on the Directory a virtual view that was previously opened with an OPEN command.

SAVE *view-name* [BIND]

view-name

Description *Required.* Identifies the view to be stored on the Directory.

Format Must be a valid view-name.

Considerations

- ◆ The view must be created with COPY, DEFINE, EDIT, or LIST.
 - ◆ You can enter * instead of a view name, causing DBAID to substitute the last view name used.
 - ◆ The view must be a virtual view, and it must be opened.
-

BIND

Description *Optional.* Indicates you want to bind the view.

Consideration You can bind a view only if it is in the active database.

General considerations

- ◆ If the view being saved already exists, the system asks if you want to replace the existing view. If so, the new view replaces the old view on the Directory. This does not affect who can use the view.
- ◆ If the view did not exist previously, it must be related to users before it can be accessed by the application program.
- ◆ You cannot save a view that contains physical data items in the column definition. For example, the following view cannot be saved because CUSTCTRL is a physical data item.

```
KEY CUSTCTRL  
ACCESS CUST USING CUSTCTRL
```

- ◆ However, the following view can be saved:

```
KEY CUSTOMER-ID  
ACCESS CUST USING CUSTOMER-ID
```

- ◆ Physical data items can be used in the access definition.

Examples

- ◆ This example stores PRODUCT on the Directory:

```
>SAVE PRODUCT
```

- ◆ This example stores PRODUCT on the Directory and binds the view:

```
>SAVE PRODUCT BIND
```

SHOW-NAVIGATION command

The SHOW-NAVIGATION command allows you to verify the accuracy of the access paths used by RDM to access the underlying entities during a view open.

SHOW-NAVIGATION [*view-name*]

view-name

Description *Optional.* Specifies the view for which you wish to display details of access paths used.

Format Must be a valid and opened view.

Considerations

- ◆ You can enter * instead of a view name. DBAID returns information on the last view name used.
- ◆ If you omit this parameter, RDM returns information on all opened views in turn.
- ◆ Access methods include:
 - INDEXED - via RMS alternate key or PDM secondary key
 - KEYED - via control key
 - LINKPATH - via a linkpath
 - SCAN - via a sequential scan
 - RDML GET - via view-to-view access

Example The following example shows the access path used the view REGION-BY-NAME. REGION-BY-NAME is a base view because it accesses a data set, REGN, through the secondary index key REGNSKNM.

```
>SHOW-NAVIGATION REGION-BY-NAME
-----
VIEW NAME : REGION-BY-NAME                                     !
-----
LVL!   ACCESSED FILE/VIEW NAME   ! ACCESS METHOD ! ACCESS PATH NAME !
---!-----!-----!-----!
!0 ! REGN                          ! INDEXED      ! NAME              !
-----
```

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** 1–30 alphanumeric characters. Must be a valid user name already defined on the Directory.

password

- Description** *Optional.* Indicates the user's password.
- Format** 1–8 alphanumeric characters. Must be a valid password already defined on the Directory.
- Consideration** To SIGN-ON as another user during a DBAID session, you must first issue a **SIGN-OFF**.

General consideration

When you invoke DBAID, you effectively issue a SIGN-ON.

Example This example identifies Jane Doe to DBAID:

```
>SIGN-ON JDOE DBAPSWD
```

STATS command

The STATS command causes RDM to display the current statistics for all open views or for a view that you specify. You can issue the STATS command numerous times during a session after you issue a **STATS-ON** command.

STATS [*view-name*]

view-name

Description *Optional.* Specifies the view for which you wish to display statistics.

Format Must be a valid and opened view.

Consideration You can enter an * instead of a view name, causing DBAID to substitute the last view name used.

General considerations

- ◆ The **STATS-ON** command must precede the first STATS command. If you do not first issue STATS-ON, STATS has no effect.
- ◆ You can issue a **STATS-OFF** to discontinue statistics gathering.
- ◆ When you issue STATS, the statistics are displayed on your screen.
- ◆ You can issue STATS-OFF followed by STATS-ON, or just STATS-ON to reset the statistical information.
- ◆ The STATS command can be used to keep a statistical running total.

Example In the following example, STATS is used to display a running total after each RDML operation.

```
STATS-ON
GET NEXT STOCK
.
.
.
STATS
UPDATE STOCK
.
.
STATS
```

STATS-OFF command

The STATS-OFF command causes RDM to print the current statistics. After the statistics are printed, they are displayed.

STATS-OFF

General considerations

- ◆ The **STATS-ON** command must precede the STATS-OFF command.
- ◆ Issuing a STATS-OFF command without a preceding STATS-ON command has no effect.
- ◆ The **BYE** or **SIGN-OFF** commands also perform a STATS-OFF command.

STATS-ON command

The STATS-ON command causes RDM to initialize the statistics to zero and then begin gathering statistics. The DBA can use this command, in conjunction with the **STATS-OFF** or **PRINT-STATS** commands, to examine what user views do on both a logical and physical level.

STATS-ON

General considerations

- ◆ Statistics are gathered on a task basis, not on a system-wide basis.
- ◆ Use the STATS-OFF command to print statistics and then turn them off.
- ◆ Use the PRINT-STATS command to print statistics, but continue gathering a running total.
- ◆ You can use the **BYE** and **SIGN-OFF** commands to print statistics and then turn them off.

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; SURE causes RDM to automatically issue a COMMIT if an "*" FSI is returned by a RDML command that alters the database.

SURE

General consideration

This is the default setting.

UNDEFINE command

The UNDEFINE command removes the name and definition of a virtual view.

UNDEFINE { **ALL**
 { *view - name* } }

{ **ALL**
 { *view - name* } }

Description *Required.* Specifies which virtual views to remove.

Options **ALL** Removes all virtual views currently in use and issues an RDM RELEASE.

view-name Identifies the virtual view to be removed. This must be a valid view.

Consideration You can enter * instead of view name, causing DBAID to substitute the last view name used.

General considerations

- ◆ Storage being used by the view is relinquished, allowing it to be reclaimed for defining other views.
- ◆ This command does not remove a saved definition from the Directory.

Examples

- ◆ This example removes all views currently in use:

```
>UNDEFINE ALL
```

- ◆ This example removes the view CUSTOMER:

```
>UNDEFINE CUSTOMER
```

UPDATE command

The UPDATE command updates data values in the database. For RMS data sets, it also updates the relationships between files.

UPDATE *view-name* [*column*₁*:=literal*₁[,...,*column*_{*n*}*:=literal*_{*n*}]

view-name

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

Format Must be a valid and opened view.

Consideration You can enter * instead of a view name, causing DBAID to substitute the last view name used.

*column*₁*:=literal*₁[,...,*column*_{*n*}*:=literal*_{*n*}]

Description *Optional.* Identifies a column in the view which 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. A hexadecimal value is not allowed.

Considerations

- ◆ Each updateable column is displayed, and replacement values are accepted. Entering a null line does not change the column. Entering new data changes the column value in the row. After all updateable columns are processed, the prompt "UPDATE (Y/N)" displays and requires a response.
- ◆ You can use the "column:=literal" syntax when updating columns in the row. Only the columns you specify are updated; all others remain the same. To update a row, indicate the column you want to update, the :=, and the new value for the column.
- ◆ Single quotes are not required around character or numeric literals unless the literal contains spaces or commas.
- ◆ Single quotes are required for you to change the value of a column to blanks. A literal of spaces (keyed in) must be in single quotes. If you press RETURN, you do not affect the item's value.
- ◆ You cannot use the UPDATE function to modify columns designated as key values. Use the DELETE and INSERT commands to modify key items. See "[DELETE command](#)" on page 162 and "[INSERT command](#)" on page 183.
- ◆ In order to UPDATE a row, you must first retrieve the row using the [GET](#) command.
- ◆ UPDATE cannot change all the values in a defined column to a specific value. For example, you could not change all PROD-CODES to 'T100,' even if you wanted to.
- ◆ If the physical field being updated is an alternate key for RMS data sets, RDM maintains the secondary index in the same file as the primary index.

Example

This example updates the columns STOCK-QUANTITY and STOCK-BIN-LOCATION in the view STOCK:

```
>UPDATE STOCK STOCK-QUANTITY:=25, STOCK-BIN-LOCATION:=A3
```

If an alternate index is defined for the above view, RDM performs the update as follows:

- ◆ Before deleting the row from the DATE data set, RDM checks to see if any rows in the ORDR data set have the old value.
- ◆ If so, the delete is not performed on the DATE data set. RDM then attempts to insert the new value into the DATE data set.
- ◆ If a duplicate occurrence is found, the error is ignored.

USER-LIST command

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

USER-LIST *user-view-name*

user-view-name

Description *Required.* Identifies the user view or view to be displayed.

Format Must be a valid view.

Consideration You can enter * instead of a view name, causing DBAID to substitute the last view name used.

Example This example displays the list of columns for the B1 user view:

```
>OPEN B1=BRANCH BRANCH-NUMBER , BRANCH-ADDRESS , BRANCH-CITY , BRANCH-STATE
FSI: *   VSI: =   MSG:           4600 BYTES USED IN OPENING VIEW.
```

```
>USER-LIST B1
USER VIEW NAME :      B1
LOGICAL VIEW NAME : BRANCH
USER VIEW LIST :
BRANCH-NUMBER , BRANCH-ADDRESS , BRANCH-CITY , BRANCH-STATE , END.
```

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 is to be displayed.

Format Must be a valid and opened view.

Considerations

- ◆ You can enter * instead of a view name, causing DBAID to substitute the last view name used.
- ◆ If you omit this parameter, a condensed description of all your opened views displays.

Example This example displays a condensed description of the CUSTOMER view. The following table explains each displayed descriptor.

```

>VIEW-DEFN
VIEW-NAME                (+) CUSTOMER
INS-ORDER                (+) N
TOTAL-SIZE                (+) 63
TOTAL-FIELDS             (+) 3
TOTAL-LEVELS             (+) 1
TOTAL-DELETABLE          (+) 3
TOTAL-INSERTABLE         (+) 3
TOTAL-REPLACEABLE        (+) 3
TOTAL-REQUIRED           (+) 1
TOTAL-KEYS                (+) 1
TOTAL-NON-UNIQUE         (+) 0
    
```

View descriptor	Explanation
VIEW-NAME	The name of the view being described.
INS-ORDER	Indicates that inserts will be ordered, depending on the value of a column.
TOTAL-SIZE	The total number of bytes in the view, including ASIs.
TOTAL-FIELDS	The number of columns in the view.
TOTAL-LEVELS	The number of levels in the view.
TOTAL-DELETABLE	The number of deletable columns.
TOTAL-INSERTABLE	The number of insertable columns.
TOTAL-REPLACEABLE	The number of updateable columns.
TOTAL-REQUIRED	The number of required columns.
TOTAL-KEYS	The number of keys in the view.
TOTAL-NON-UNIQUE	The number of non-unique keys in the view.

VIEWS command

The VIEWS command displays all views currently active in DBAID.

VIEWS

General consideration

The information displayed with this command includes:

- ◆ User View - The name of the user view.
- ◆ Logical View - The name of the view this user view is part of.
- ◆ Status - Indicates whether the user view is opened or released.

Example

This example displays all views currently active in DBAID:

```
>VIEWS

          USER VIEW                LOGICAL 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 views related to the signed-on user together with the date and time of the most recent view-save.

VIEWS-FOR-USER

General consideration

The date and time are displayed in the format mm/dd/yy.

Example

This example displays the views related to the signed-on user:

```

VIEWS-FOR-USER
!          LOGICAL VIEW NAME          !   DATE   !   TIME   !
!-----!-----!-----!
! BASE-VIEW          ! 08/22/96 ! 13:17:24 !
! REGION            ! 08/22/96 ! 13:18:04 !
! BRANCH            ! 08/22/96 ! 13:27:32 !
! CUSTOMER          ! 08/22/96 ! 13:32:14 !
! PRODUCT           ! 01/16/96 ! 12:49:04 !
! BRANCH-SUBSET     ! 03/24/96 ! 15:12:24 !
! BRANCHES-IN-REGION ! 03/24/96 ! 15:17:58 !
! PRODUCTS-IN-REGION ! 03/24/96 ! 15:27:04 !
! REVIEW-DETAILS    ! 01/16/96 ! 12:51:13 !
! WRITING-DETAILS   ! 03/24/96 ! 15:33:18 !
! MANUALS           ! 01/16/96 ! 12:51:59 !
! AUTHOR            ! 01/16/96 ! 12:57:26 !
! PRODUCTION-DETAILS ! 01/16/96 ! 13:04:29 !
!-----!-----!-----!

```

6

RDM status indicators

RDM returns status indicators to the application program and to the DBAID user to indicate Relational Data Manipulation Language (RDML) processing results. The indicators are the same, regardless of whether the view is a base or derived view because base views pass the indicators to derived views.

The three types of status indicators that are returned after any RDML function call are as follows:

- ◆ **Function Status Indicators** indicate the success or failure of the function.
- ◆ **Column Attribute Status Indicators** indicate the status of each column in the row.
- ◆ **Validity Status Indicators** indicate the most severe column status within the row.

Function Status Indicators (FSIs)

Function Status Indicators (FSIs) reflect the success or failure of the RDML function. The FSI is returned to the application to let the program determine the next appropriate action. For MANTIS programs, refer to your MANTIS documentation for more information on status indicators. For FORTRAN, COBOL, or BASIC RDML applications, RDM returns the FSI to an application program in an area generated as part of the programmer-supplied ULT-CONTROL statement. The following two examples are of this generation: one COBOL and one FORTRAN.

Examples

- ◆ **COBOL.** Note that the asterisk indicates the statement that the programmer specifies in the source RDML program; the RDML preprocessor for COBOL generates all other statements.

```

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

```

- ◆ **FORTRAN.** Note that the C indicates the statement that the programmer specifies in the source RDML program; the RDML preprocessor for FORTRAN generates all other statements.

```

C      INCLUDE ULT-CONTROL
CHARACTER ULT_OBJECT_NAME*30,ULT_OPERATION*6,ULT_FSI*1,ULT_VSI*1,
+ULT_FILLER*2,ULT_MESSAGE*40,ULT_PASSWORD*8,ULT_OPTIONS*4,
+ULT_CONTEXT*4,ULT_LVCONTEXT*4
PARAMETER(ULT_CONTROL_LEN=100)
CHARACTER*(ULT_CONTROL_LEN) ULT_CONTROL
EQUIVALENCE (ULT_CONTROL(1:30),ULT_OBJECT_NAME(1:30))
+, (ULT_CONTROL(31:36),ULT_OPERATION(1:6)), (ULT_CONTROL(37:37),
+ULT_FSI(1:1)), (ULT_CONTROL(38:38),ULT_VSI(1:1))
+, (ULT_CONTROL(39:40),ULT_FILLER(1:2)), (ULT_CONTROL(41:80),
+ULT_MESSAGE(1:40)), (ULT_CONTROL(81:88),ULT_PASSWORD(1:8))
+, (ULT_CONTROL(89:92),ULT_OPTIONS(1:4)), (ULT_CONTROL(93:96),
+ULT_CONTEXT(1:4)), (ULT_CONTROL(97:100),ULT_LVCONTEXT(1:4))
CHARACTER*14 ULT_DATE_STAMP
DATA ULT_DATE_STAMP/' .19831114143849' /
C      ON ERROR
C          TYPE *, 'RDM control call failed', ULT_FSI
C          STOP
C      END ERROR-HANDLER
*

```

The FSIs have the following meanings:

FSI value	Meaning
*	SUCCESSFUL COMPLETION. The RDML function completed successfully.
D	DATA ERROR. The row contains invalid data. Check the ASIs to see which column contains invalid data.
F	FAILURE. The RDML function failed. Usually caused by a physical database problem returned to RDM.
N	NOT FOUND. The RDML processor cannot find an occurrence of the requested row.
R	DYNAMIC RESET. The PDM performed a dynamic reset on the database because of an earlier PDM failure. The PDM restarted automatically; however, you must reapply all modifications made since the last COMMIT or RESET.
S	SECURITY CHECK. The attempted RDML function violated a security constraint.
U	UNAVAILABLE RESOURCE. The resource required to complete this function was not available; for example, the data set was not open.
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.

A message associated with the FSI is accessible in the ULT-MESSAGE area for all returned indicators (see the preceding example).

Column Attribute Status Indicators (ASIs)

Column Attribute Status Indicators (ASIs) reflect the status of each column defined in a view. ASIs have a one-to-one mapping to each column. In FORTRAN, COBOL, or BASIC RDML programs, they are placed immediately after the last column in your view, for example:

COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	ASI ₁	ASI ₂	ASI ₃	ASI ₄
----------	----------	----------	----------	------------------	------------------	------------------	------------------

You can access the ASIs through names generated by the RDML preprocessor. During application program coding, the programmer specifies an INCLUDE statement for the view required. The RDML preprocessor generates definitions for all columns, followed by definitions for an ASI for each column. The name of the ASI for a column is the column name preceded by the four characters ASI-. The following two examples are of this generation; one for COBOL and one for **FORTRAN**.

Examples

- ◆ **COBOL**. Note that the asterisk indicates the statement that the programmer specifies in the source program; the RDML preprocessor for COBOL generates all other statements.

```
* 01  INCLUDE CUST-CONTACT.
01  LUV-CUST-CONTACT.
10  CUST-CONTACT.
20  CUST-NO  PIC S9(05).
20  CONTACT-NAME                PIC X(040).
20  CONTACT-TITLE                PIC X(040).
20  CONTACT-PHONE                PIC X(010).
10  ASI-CUST-CONTACT.
20  ASI-CUST-NO                  PIC X.
20  ASI-CONTACT-NAME            PIC X.
20  ASI-CONTACT-TITLE           PIC X.
20  ASI-CONTACT-PHONE           PIC X.
```

- ◆ **FORTRAN.** Note that the C indicates the statement that the programmer specifies in the source program; the RDML preprocessor for FORTRAN generates all other statements.

```

C      INCLUDE PART-COMP=V2 ( PART=PART-NAME , COMP=COMPONENT-NAME )
      CHARACTER*6 PART
      CHARACTER*6 COMP
      CHARACTER*1 ASI_PART,ASI_COMP
      EQUIVALENCE ( PART , PART_COMP ( 1:6 ) )
      EQUIVALENCE ( COMP , PART_COMP ( 7:12 ) )
      EQUIVALENCE ( ASI_PART , PART_COMP ( 13:13 ) )
      EQUIVALENCE ( ASI_COMP , PART_COMP ( 14:14 ) )
      PARAMETER ( PART_COMP_LEN=14 )
      CHARACTER* ( PART_COMP_LEN ) PART_COMP
      CHARACTER
      ULT$PART_COMP*30 , ULT$PART*16 , ULT$COMP*21 , ULT_END_VIEW
      +1*4
      DATA ULT$PART_COMP/ 'V2                                /ULT$PART
+ / '006C00PART-NAME , ' /ULT$COMP/ '006C00COMPONENT-NAME , /ULT_END_
VIEW
      +1/ 'END. ' /

```

The ASIs have the following meanings:

ASI value	Meaning
C	Returned when the column values are changed by another view. This check is made only when a GET statement (not GET FOR UPDATE) is followed by an UPDATE or DELETE statement. You can override this check by specifying SHARED on the ALLOW clause of an ACCESS statement.
V	Returned when the column is invalid (when a numeric column contains non-numeric data, when a column failed its validation checks [table, range, or user exit] or when a foreign key value is incorrect).
-	Returned when the column contains a null value, or when no physical record exists to supply the column value. The column in the row is set to blanks if it is a character, or zero if it is a binary, packed, numeric, or floating point data item; or, if the field contains the null value the column contains the null value. This ASI value only has meaning on GET RDML requests. This value only has meaning on GET RDML requests.
+	Returned if the column exists and was filled from a different accessed entity. (GET processing only.) An ASI of + is given to those columns generated by new physical records. GET FIRST returns ASIs of + to all columns, because it retrieves the first row in a view and must therefore access all associated physical records as new.
	The ASIs + and = indicate an occurrence of a new physical record when accessing a row. Whenever a new physical record is read, its physical data items generate ASIs of +, which are assigned to their corresponding columns in the row.
=	Returned if the column exists and its value was filled from the same accessed entity as the last access (those column values generated by unchanged physical records when a new row is read).
	The ASIs + and = indicate an occurrence of a new physical record when accessing a row. Whenever a new physical record is read, its physical data items generate ASIs of +, which are assigned to their corresponding columns in the row.
N	The application programmer can place an N in the ASI during UPDATES and INSERTS to set a column to its null value. This ASI value is never returned by RDM.

The three ways to use ASIs are:

- ◆ When you issue a GET command, certain columns returned may not have a value. Check this status (on columns that were not altered) with the ASI.
- ◆ If you receive an FSI indicating a data error, use the ASI to find which columns have illegal values.
- ◆ Programmers can use ASIs for validation of input. For example, when a logical view contains packed values, use the ASIs to avoid errors at run time that would be caused if you performed a calculation or move using an invalid packed decimal value. Do this by examining each ASI for such columns before performing the operation.

If the ASI for a column is V, the value is placed in the row even though it is not in a valid format. When a - ASI is returned, RDM placed a valid zero in numeric columns and spaces in character columns. For other ASI values, the column is valid.

Validity Status Indicators (VSIs)

Validity Status Indicators (VSIs) reflect the validity of the logical view after a RDML command causes a read of the physical database. The RDML processor returns the VSI to the program. In MANTIS, see the VSI function. For FORTRAN, COBOL, or BASIC RDML programs, the VSI is returned in an area generated as part of the programmer-supplied ULT-CONTROL statement (see the example in “[Function Status Indicators \(FSIs\)](#)” on page 224). You can use these indicators to determine the most significant ASI returned by RDM according to the following hierarchy:

VSI value	Meaning
C	The column value was changed by another logical 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.

The VSI enables the programmer to quickly determine if any additional processing of ASIs is needed to correct invalid data or to fill missing values.

7

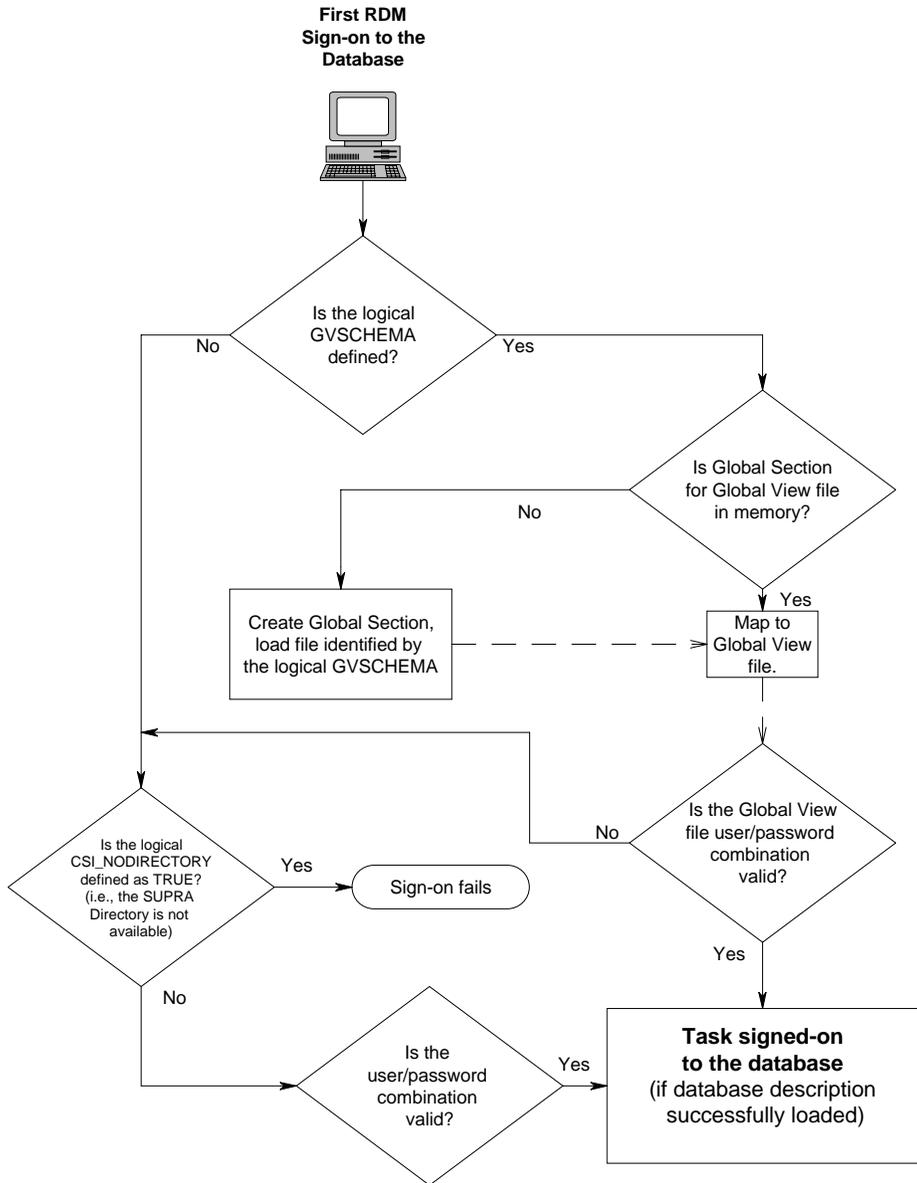
Optimizing view performance using bound and global views

Opening a view is resource intensive. To improve performance, SUPRA Server provides two methods to preopen views—bound and global—to reduce the time needed to open a view. These methods are discussed in the following sections.

Differences between bound and global views

A bound view is a preopened copy of a view that is stored in the SUPRA Server Directory. View binding improves performance on the initial access to a view, reducing the processing overhead of other requests to open the view.

A global view is placed in a Global View file, which is loaded into global memory during the first RDM sign-on (see the following figure).



Once loaded in memory, all other RDM tasks map to the global file to access the views. Performance improves because the first task to open the Global View file is the only one that requires initialization resources. All subsequent tasks simply map to the global section that contains the Global View file.



To reduce global memory usage, only globalize views that are used by multiple tasks concurrently.

Advantages of using global views



Both types of views can optimize system performance. However, Cincom recommends that you use global views because they are loaded in memory instead of requiring access to the Directory database SUPRAD for each view-open request. Because global views are stored in memory, they have the following advantages over bound views:

- ◆ Reduce the resources needed for system initialization.

The first user who signs on to RDM initializes the Global View file; all subsequent view requests map directly to the Global View file. Therefore, the processing overhead of opening a view occurs only once (see the figures under “[Differences between bound and global views](#)” on page 231 and “[Global views](#)” on page 241).

- ◆ Isolate production systems from development changes.

When using global views, tasks access the Global View file in memory; they don't access the SUPRA Server Directory database SUPRAD. This insulates your production database from changes that can occur as a result of defining and testing views.

- ◆ Eliminate the need for all sites to have access to the SUPRA Server Directory database SUPRAD.

Because tasks do not access the SUPRA Server Directory database SUPRAD, users do not require access to it. This allows you to run without a SUPRA Server Directory by setting the logical definition CSI_NODIRECTORY to TRUE (refer to the [SUPRA Server PDM System Administration Guide \(VMS\)](#), P25-0130).



If you choose to run without a Global View file or only some of your views globalized, a SUPRA Server Directory database (SUPRAD) *must be* available.



The figure under “[Differences between bound and global views](#)” on page 231 shows how the Global View file affects RDM sign-on processing.

Changing view text: a note of caution

Whether you use bound or global views, the view text is always stored separately from the bound or global copy. (The bound pre-opened copy is stored on the SUPRA Server Directory; the globalized pre-opened copy is stored in the Global View file, which may be in global memory.) If you change the view text, you *must* update your bound and global copies of the view to reflect the changes. If you fail to rebind the view or rebuild the Global View file with the most current copy of view, your bound and global views will become out of date. This could cause unpredictable results.

Changing the text of a view when using bound views only

When you save changes to a view using DBAID, DBAID automatically prompts you to rebind the view with the most current view text. You *must* answer YES to that prompt for DBAID to rebind the view with the most current view text. See “[Binding a view](#)” on page 236 for information on saving a view using DBAID.

Changing the text of a view when using global views

When a view is included in a Global View file and the view text changes, you must rebuild the Global View file so that it is updated. For information on how to rebuild the Global View file, see “[Creating a Global View file](#)” on page 243.

Changing the text of a view when bound views are included in a global view file

If a bound copy of a view exists and you are using a Global View file, the bound version is always included in the Global View file. This means that if the view text changes, you must rebuild *both* the bound version and the Global View file to ensure that all copies of the view are current.



Caution: Cincom recommends that you use global views only. If you use both bound views and a Global View file, you must make sure the view is current in three places (view text, bound version and global version), as opposed to just two (view text and global version) if you use only global views.

Bound views

A bound view is a pre-opened copy of a view that is stored in the SUPRA Server Directory database SUPRAD. View binding improves performance on the initial access to a view, reducing the processing time for application program requests that open the view.

Binding a view

You can bind a view using two different utilities: DBA and DBAID.

Using DBA to bind a view

To bind a view using DBA, select option 3, Logical Views, from the Function Selection for the DBA menu. The Logical View Function menu displays. Select option 2, Modify, and respond to the prompts.

```
CINCOM SYSTEMS      SUPRA DBA - LOGICAL VIEW FUNCTION

      Functions for logical views
1 : Examine
2 : Modify
3 : Create
4 : Delete
5 : Connect to database description
6 : Disconnect from database description
7 : List database descriptions using view
8 : List all logical views
9 : User authorization

      Enter choice no.: 2

Modify logical view name :
(<PF4> will select CUSTOMER) : CUSTOMER-REFERENCE

Database to which logical view refers : CUSTDB
```

To bind the view, press function key PF1 followed by B for bind. RDM attempts to open, save and bind the view you selected for modification. The view bind occurs only if the view-open and save to the SUPRA Server Directory were successful. If RDM successfully binds the view, it displays this message:

```
VIEW BINDING SUCCESSFUL
```

Binding a view using DBAID

To bind a view using DBAID, use either the DBAID BIND command or the DBAID SAVE command with the BIND qualifier. Note that a view must be open before it can be bound. The sample DBAID session in the following screen illustration shows how to bind a view using the BIND command. You can also use the SAVE command to bind a view (see the subsequent screen illustration).

```
>LIST CUSTOMER

                                CUSTOMER

0005 KEY CUSTOMER-NUMBER = CUSTOMER-ID
0010 REQ CUSTOMER-NAME
0015  CUSTOMER-ADDRESS
0020  CUSTOMER-CITY
0025  CUSTOMER-STATE
0030  CUSTOMER-ZIP-CODE
0035  CUSTOMER-PHONE-NUMBER
0040  CUSTOMER-FAX-NUMBER
0045  CUSTOMER-CLASS
0050  CUSTOMER-CREDIT-CODE
0055  CUSTOMER-CREDIT-LIMIT
0060 REQ CUSTOMER-BRANCH = CUSTOMER-BRANCH-ID = BRANCH-ID
0065 ACCESS CUST
0070   WHERE CUSTOMER-ID = CUSTOMER-NUMBER
0075   ALLOW ALL
0080 * To verify that CUSTOMER-BRANCH contains a valid branch on
0085 * INSERT and UPDATE.
0090 ACCESS BRAN
0095   WHERE BRANCH-ID = CUSTOMER-BRANCH-ID
>OPEN*
FSI: *   VSI: =   MSG:           4424 BYTES USED IN OPENING VIEW.
>BIND*
REPLACE EXISTING VIEW (Y/N)?
>Y
SAVED   19 LINES AS CUSTOMER                                12:40:42  04-JAN-1996
VIEW BINDING SUCCESSFUL.
>
```

Note that you do not need to list the view before you can open it and make the view text available for saving, if required; however, listing the text of a view allows you to check that you selected the correct view. The session in the following screen illustration shows how to bind a view using the SAVE command.

```
>LIST REGION

                                REGION
0005 KEY REGION-NUMBER = REGION-ID
0010 REGION-NAME
0015 ACCESS REGN
0020 WHERE REGION-ID = REGION-NUMBER
0025 ALLOW ALL
0030 * To restrict deletions of regions that contain branches.
0035 ACCESS BRAN
0040 WHERE BRANCH-REGION-ID == REGION-ID
>OPEN*
FSI: * VSI: = MSG:          1656 BYTES USED IN OPENING VIEW.
>SAVE* BIND
REPLACE EXISTING VIEW (Y/N)?
>Y
SAVED 8 LINES AS REGION                                12:20:51 04-JAN-1996
VIEW BINDING SUCCESSFUL.
>
```

Ensuring that you update a bound view

The Directory stores the bound version of a view *separately* from the text of a view. Therefore, if you change the text of the view and forget to update the bound version (rebind it), the bound version becoming out-of-date. An outdated bound view can produce unpredictable results, including failures such as access violations in application programs, MANTIS, or SPECTRA (SPECTRA is not available in OpenVMS AXP environments).

Deleting the bound view only

To delete the bound version of a view without deleting the unbound view text, first LIST the view in DBAID, then use the DBAID REMOVE command to remove both bound and unbound views. Then LIST the virtual view that remains and use the DBAID SAVE command to save the view definition on the Directory without binding it.

Deleting both the view definition and the bound view

Use the DBAID REMOVE command to delete both a view definition and the bound version of the view from the Directory. To guard against accidental deletions, you can delete only views that are virtual (views that you displayed using the DBAID LIST command). Even after you remove a view, the virtual text is still available until you leave DBAID. To remove the virtual view text, either enter the DBAID UNDEFINE command or enter BYE to exit DBAID.

In DBA, use the Logical View Functions menu and select the Delete view option to delete the view definition and the bound view.

Rebinding a view after making changes to view text

If you attempt to SAVE a view definition while a bound version of that view exists, DBAID asks if you want to rebind the view. If you reply "Y," DBAID replaces the current bound version with the new bound version. If you reply "N," DBAID saves the view on the Directory without re-binding it and the view text and bound copy become inconsistent.

Testing views: failing to rebind a view

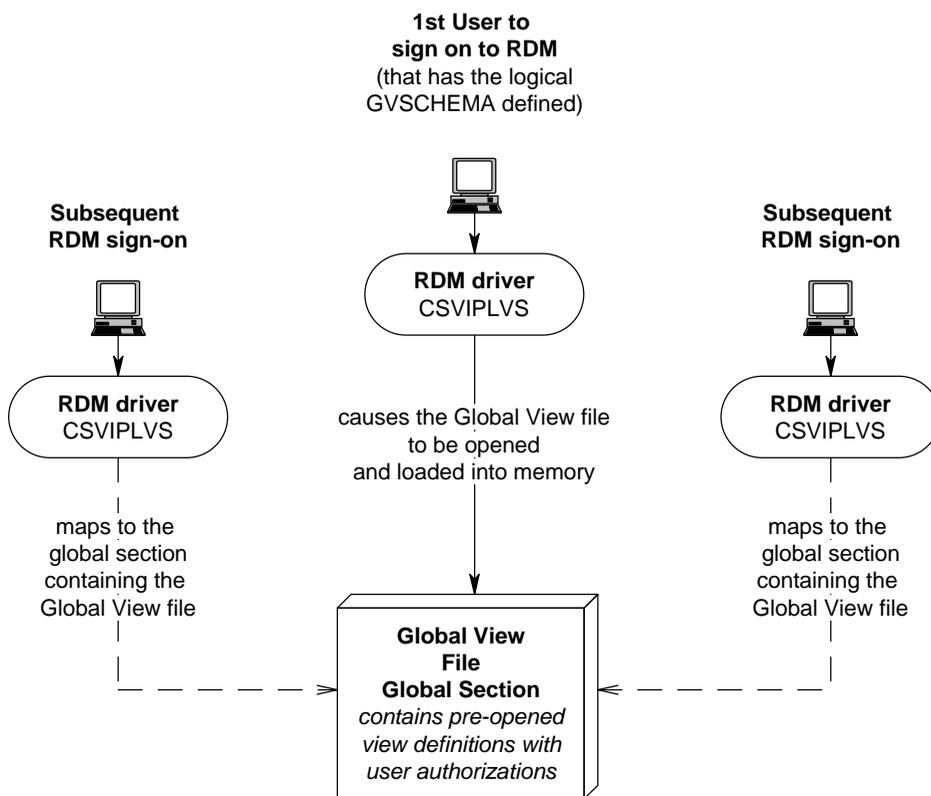
If you change the view definition and do not rebind the view, you will have a bound version that is different from the unbound version. This is useful if you are testing views before putting them into production, because applications will only be able to use the bound version (unless the view is globalized). Only after you rebind the view is it available to application programs.



If the bound view is in a Global View file, you must rebuild the Global View file before the new version of the view is available to application programs.

Global views

The Global View facility allows you to place commonly used views into a file that is loaded into global memory during the first RDM user initialization. This makes the views available to authorized users (as shown in the following figure). It also saves much of the processing overhead of opening views at the first application program access for each user task. In addition, view-open performance improves because separate view definitions need not be loaded for each user. Users share the copy of the view definition held in global memory.



To ensure that RDM applications use the global views, assign the logical name GVSCHEMA to the Global View file as shown:

```
$ DEFINE GVSCHEMA CSI_device:[directory] dbname.GBL
```

The database description associated with the Global View file identified by the logical GVSCHEMA must be identified by the logical CSI_SCHEMA. Refer to the *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130, for a complete description of these logicals.

You can load both base and derived views into global memory by including their view definitions in the Global View file. However, you should include a base view in the Global View file before you include any derived views that access it. Because the global view definitions are held in a file rather than on the Directory, they:

- ◆ Reduce the resources needed for system initialization
- ◆ Isolate production systems from development changes
- ◆ Eliminate the need for all sites to have access to the SUPRA Server Directory database SUPRAD

Available global views are used in preference to view definitions on the Directory. Therefore, opening a global view does not require SUPRA Server Directory access. This allows you to run without a SUPRA Server Directory by setting the logical definition CSI_NODIRECTORY to TRUE (refer to the *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130). If you choose not to use a Global View file or not all needed views are included in the Global View file, a SUPRA Server Directory must be available.

Once the global view is opened, it remains in the global memory until all users sign off from the database.

Creating a Global View file

The Global View file contains a list of global views and users authorized to access those views. Create a Global View file in one of two ways:

- ◆ Interactively, through the Global View file creation screens
- ◆ In batch, by creating an input text file containing details of the global views and authorized users, and making the following logical assignment:

```
$ DEFINE BATCH_GLOBAL_INPUT device:[directory] filename.ext
```

When you create a Global View file interactively, you can authorize only users who are already connected to the views on the Directory. Users who are not allowed to use a view on the Directory cannot be given access to the global version of that view.

However, when you use the batch Global View file creation facility, you can allow global view access to users who are not connected to a view on the Directory. In addition, batch Global View file creation allows all users connected to a view on the Directory to access the global version, unless you use the disallow clause to explicitly exclude them from the Global View file.

To initiate either method, select the Global View Creation option from the SUPRA Facilities menu or run the Global View Creation program CSVGLOBAL directly. CSVGLOBAL searches for a file identified by the logical name BATCH_GLOBAL_INPUT, which contains details of the global views and authorized users. See “[Batch Global View file creation](#)” on page 248 for more information.



CSVGLOBAL is a logical definition used to identify the Global View Creation utility. Refer to the *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130, for details of this logical definition.

If there is no logical translation for BATCH_GLOBAL_INPUT, the Global View Creation program CSVGLOBAL defaults to interactive mode and displays a series of screens. These screens prompt you to specify the following:

- ◆ Views to be used as global views
- ◆ Users who can access each global view

Reverse video indicates where you enter data. The arrow and tab keys position the cursor for entry. Press RETURN to transmit the data.

If you previously created a text file identified by the logical name BATCH_GLOBAL_INPUT, CSVGLOBAL automatically uses the details specified in that file. You do not need to enter any further information. CSVGLOBAL translates the logical name CSI_SCHEMA to identify the database that the specified global views should access.

Selecting views. You use the first Global View Creation menu (see the following screen illustration) to select the views you want to include in the Global View file. SUPRA Server translates the logical name CSI_SCHEMA to find the name of the database accessed by these views and displays that name at the top of the screen.

```
CINCOM SYSTEMS          GLOBAL VIEW CREATION          RELEASE 2.4
                          DATABASE NAME: QADBD1
                          Include   Restrict
                          View Y/N   User Y/N   View Name
                          Y
                          Y
                          Y      Y
                          TEST6D
                          TEST63
                          TEST6F
                          TEST6G
                          TEST7
                          TEST9
                          TEST10
                          TEST11
                          TEST12
                          TEST13
                          TEST14
                          TEST15
                          TEST16
                          TEST8
                          END SELECTION.
```

To select a view, enter Y in the “Include View” column. Any users who have access to the view on the Directory are allowed to access the global version of the view unless you enter Y in the “Restrict User” column. Press TAB to move the cursor between fields.

The last entry on the View Selection menu is END SELECTION. There may be another screen of views from which to choose. To display subsequent view selection screens, enter N in the “Include View” column next to END SELECTION. This displays the next screen (if there is one) or returns you to the top of the current screen if there is no next screen. When you are finished, press RETURN or enter Y next to END SELECTION, and press RETURN.

Restricting user access to global views. If you enter a Y in any Restrict User column, SUPRA Server displays the Restrict Users menu when you finish selecting views. The name of the view you are restricting displays at the top of the screen. The name of each user allowed to access the non-global version of the view is displayed in the Username column. To set the authority specifications, enter Y or N in the Authorize column, as shown in the following screen illustration. Press RETURN or ENTER to display the next restrict users screen.

```
CINCOM SYSTEMS          GLOBAL VIEW CREATION          RELEASE 2.4

view name:    TEST10

Authorize Y/N   Username
              Y      DATABASE-DESCRIPTIONS
              Y      SRV
              N      ALEC
```

Following the last view selected, the Global View facility then:

- ◆ Opens each view in turn, displaying the message OPENING VIEW: *view-name*. (The bound version is used, if one exists.)
- ◆ Creates the Global View file named *xxxxxx.GBL* (where *xxxxxx* is the 6-character database name).
- ◆ Displays a completion message in reverse video at the bottom of the screen until you press PF1 to exit.
- ◆ Creates a Global View report by view name in a file named *GVxxxxxx.LIS* (where *xxxxxx* is the database name). This report shows which views are included and which users have access to them.

Batch Global View file creation

Selecting Global View Creation from the SUPRA Facilities menu invokes the CSVGLOBAL program. Alternatively, you can run CSVGLOBAL directly from the command level by entering:

```
$RUN CSVGLOBAL
```

Refer to the *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130, for more information on the CSVGLOBAL logical definition.

The CSVGLOBAL program first searches for the logical name BATCH_GLOBAL_INPUT, pointing to a text file. If this logical name and corresponding file exist, CSVGLOBAL processes the file, using the details it contains about the global views and authorized users, to create a Global View file (batch Global View file creation). If the logical name does not exist, CSVGLOBAL defaults to interactive global view creation described in the preceding section. Use the DCL DEFINE command to make the logical assignment as follows:

```
$DEFINE BATCH_GLOBAL_INPUT device:[directory]filename.ext
```

where *device:[directory]filename.ext* is the full directory specification for the input text file. (The full directory specification is unnecessary if you run batch global view creation from the directory containing the input file.) Refer to the *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130, for complete details on this logical definition.

The input file can contain the following details:

- ◆ User name and password that you want to use to create global views
- ◆ Groups of users who should have similar rights to use views
- ◆ Global views and lists of users and/or groups allowed to access them
- ◆ The group ALL-VIEWS and lists of users and/or groups of users allowed to access them
- ◆ The group OTHER-VIEWS (those views not already specified) and lists of users and/or groups of users allowed to access them

Running the Global View Creation program in batch identifies the database to be accessed by the global views by translating the logical name CSI_SCHEMA. Therefore, before you run batch Global View file creation, make the following logical assignment:

```
$DEFINE CSI_SCHEMA database-name
```

where *database-name* is the 6-character name of your database. Refer to the *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130, for complete details on this logical definition.

Creating an input text file. You create a Global View input file using a standard text editor. Input files are divided into three parts:

- ◆ User definition
- ◆ Group definitions
- ◆ View definitions

The user definition is required and consists of the `USER` statement that specifies the user name that is creating the Global View file. You can include only one user definition.

Group definitions are optional and specify any number of users who are to have the same global view access rights. You can include as many group definitions as you want. You can also include one group *within* another group, provided you defined the included group earlier in the input file. Thus, group definitions provide greater flexibility in restricting user access to views.

At least one view definition is required. View definitions identify the views being made global and specify user access to those views by allowing and disallowing named users and groups of users. You can specify as many view definitions as you want; however, the view definition `ALL-VIEWS` overrides all previous view definitions. At least one user must be allowed access to at least one view.

An exclamation mark (!) anywhere on a line indicates that the text to the right of the exclamation mark is a comment.

User definition

USER *username* [,PASSWORD *password*].

Group definition

GROUP *group - name* = { *username* [, *username*] ... } ... }

View definition

VIEW[S] *view - name* [, *view - name*...]
ALL - VIEWS
OTHER - VIEWS

[:ALLOW { *username* [, *username*] ... } ...] . [] ...
[:DISALLOW { GROUP *group - name* [, GROUP *group - name*] ... } ...] . [] ...

User definition

USER *user-name*

Description *Required.* Identifies the user to access the Global View Creation program.

Format 1–30 alphanumeric characters and hyphens. The first character must be alphabetic.

Considerations

- ◆ The user name must exist on the SUPRA Server Directory (SUPRAD) and have a privilege of at least DATABASE ADMINISTRATOR (DA).
- ◆ The user definition is terminated with a period (.).

,PASSWORD *password*

Description *Optional.* Specifies the password associated with the user name.

Format 6 alphanumeric characters.

Considerations

- ◆ If the specified user name has a password, the password you enter must match.
- ◆ If the specified user name has a password, but no password is entered, the Global View Creation program prompts you to enter the password at the terminal. The password you type does not display on the screen.
- ◆ If running batch global view creation, you must include a password clause if a password is needed.
- ◆ If the user definition has a password, you must enter the period (.) that terminates the user definition after the password.

Group definition

GROUP *group-name* =

Description *Required* for each group definition. Specifies the name of a group of users, groups, or users *and* groups to have the same access to particular global views.

Format 1–30 alphanumeric characters.

Considerations

- ◆ Terminate the GROUP statement with a period (.).
- ◆ You can define each group name only once.
- ◆ If you define a group that contains one or more other groups, these groups *must* already be defined in a previous GROUP statement.
- ◆ You must specify a group name with a GROUP statement *before* it can be used in a VIEW[S], ALL-VIEWS, or OTHER-VIEWS statement.
- ◆ The group ALL-USERS is predefined. Do not attempt to specify it by using a GROUP statement.

user-name

Description *Optional*. Identifies the user(s) to be included in the group.

Format 1–30 alphanumeric characters and hyphens. The first character must be alphabetic.

Considerations

- ◆ All specified user names must exist on the SUPRA Server Directory database SUPRAD.
- ◆ You can specify combinations of user names and predefined group names in any GROUP statement. However, specify at least one user name *or* one group name.
- ◆ Separate each user name and GROUP group name specification with a comma (,).

GROUP *group-name*

Description *Optional.* Identifies a previously defined group or groups to be included in this group.

Format 1–30 alphanumeric characters.

Considerations

- ◆ All group names you include in one group must already be defined in a previous GROUP statement.
- ◆ You can specify combinations of predefined group names and user names in any GROUP statement. However, specify at least one group name *or* one user name.
- ◆ Separate each GROUP group name and user name specification with a comma (,).

View definition

VIEW[S] *view-name* [, *view-name...*]

Description *Optional.* Specifies a view or list of views as global views.

Format 1–30 alphanumeric characters and hyphens. Each specified view must already exist.

Considerations

- ◆ Terminate each view statement with a period (.).
- ◆ Include at least one of the following view statements in the view definition:
 - VIEW[S] *view-name* [, *view-name...*]
 - ALL-VIEWS
 - OTHER-VIEWS
- ◆ Separate each view name in the list by a comma (,).
- ◆ The VIEW[S] *view-name* [, *view-name...*] statements can be used with the OTHER-VIEWS statement. However, all VIEWS[S] statements must be defined *before* the OTHER-VIEWS statement in the input file.
- ◆ You can specify a maximum of 1000 global views.

ALL-VIEWS

Description *Optional.* Specifies all views on the Directory as global views.

Consideration The ALL-VIEWS statement overrides any previously specified views. A warning displays if any view definition precedes the ALL-VIEWS statement.

OTHER-VIEWS

Description *Optional.* Specifies all views other than those already specified in a VIEW[S] statement.

Consideration Include the OTHER-VIEWS statement *after* VIEW[S] statements.

; ALLOW

Description *Optional.* Allows specified users or groups of users access to named global views.

Considerations

- ◆ You can specify as many ALLOW and DISALLOW clauses as you want, in any order. However, if you ALLOW and DISALLOW a given user or group name several times, only the last ALLOW or DISALLOW applies.
- ◆ A user can use a global view only if one of the following conditions is met:
 - The user is connected to the view on the Directory and is not specified in any DISALLOW clause.
 - The user is in an ALLOW clause *and* is not in any subsequent DISALLOW clause.

; DISALLOW

Description *Optional.* Prevents specified users or groups of users from accessing named global views.

Considerations

- ◆ You can specify as many DISALLOW and ALLOW clauses as you want, in any order. However, if you DISALLOW and ALLOW a given user or group name several times, only the last DISALLOW or ALLOW applies.
- ◆ A user can use a global view only if one of the following conditions is met:
 - The user is connected to the view on the Directory and is not specified in any DISALLOW clause.
 - The user is in an ALLOW clause *and* is not in any subsequent DISALLOW clause.
- ◆ Include the statement DISALLOW GROUP ALL-USERS first, after the VIEW clause, to ensure that only those users explicitly specified in subsequent ALLOW clauses have access to the global views.

user-name

- Description** *Optional.* Identifies users who are allowed or disallowed access to the specified global view or views.
- Format** 1–30 alphanumeric characters and hyphens. The first character must be alphabetic.

Considerations

- ◆ All specified user names must exist on the SUPRA Server Directory (SUPRAD).
- ◆ User names connected to a view on the Directory automatically have access to the global version unless specifically excluded through a disallow statement.
- ◆ User names included in an ALLOW clause that are not connected to a view on the SUPRA Server Directory can access only the global version of the view.

GROUP *group-name*

- Description** *Optional.* Identifies a previously defined group allowed or disallowed access to the view or views specified.
- Format** 1–30 alphanumeric characters.
- Option** GROUP ALL-USERS

Considerations

- ◆ Specify only those group names that are already defined in a previous GROUP statement.
- ◆ The group ALL-USERS is predefined.

Example Global View input files

The following examples illustrate user, group, and view definitions as they appear in a global view input file.

- ◆ USER ALICE.


```

GROUP PRODUCTION = USER3, USER19, JIM .
! Group definition for group PRODUCTION .
ALL-VIEWS ; DISALLOW GROUP ALL-USERS ;
ALLOW GROUP PRODUCTION
; ALLOW ALICE.

! All views on the Directory are defined as global views.
! The DISALLOW GROUP ALL-USERS restricts access to all
! users, including those connected to the views on the
! Directory. Only the PRODUCTION group and user ALICE
! can access the global views.
```
- ◆ USER DATABASE-DESCRIPTIONS.


```

GROUP PRIV = ANN, DAVID, MARY !development personnel
GROUP ADMIN =JIM, SARAH . !administrative personnel
VIEWS PRODUCT, CUSTOMER;
DISALLOW GROUP ALL-USERS;
ALLOW GROUP ADMIN, GROUP PRIV, SAM.

! The views PRODUCT and CUSTOMER are defined as global
! views. View definition allows development personnel,
! administrative personnel, and user SAM access to the global
! views PRODUCT and CUSTOMER.
VIEW PRODUCT;
ALLOW GROUP ALL-USERS;
DISALLOW GROUP ADMIN ; ALLOW JIM.

! The view PRODUCT-STRUCTURE-VIEW is defined as a
! global view. View definition disallows the ADMIN group;
! however, user JIM who is also a member of the ADMIN group
! is given access through a subsequent ALLOW clause.
OTHER-VIEWS.

! All other views connected to this database are defined as
! global views. Because there is no ALLOW clause, these
! views are accessible to all users who are connected to
! them on the Directory.
```
- ◆ USER DATABASE-DESCRIPTIONS.


```

VIEWS ORDERS,PRODUCT;
DISALLOW GROUP ALL-USERS.
OTHER-VIEWS; ALLOW GROUP ALL-USERS.

! This input file includes all views in the global
! view file and allows all users to access all views
! except ORDERS and PRODUCT.
```

After Batch Global View Creation processes the input file, it:

- ◆ Opens each view in turn and displays the message:
`OPENING VIEW: view-name`
- ◆ Creates the Global View file named `xxxxxx.GBL` (where `xxxxxx` is the 6-character database name) and displays the message:
`FILE: data-base-name.GBL CREATED`
- ◆ Creates a Global View report by view name in a file named `GVxxxxxx.LIS` (where `xxxxxx` is the database name). This report shows which views are included and which users have access authority.

Example Global View report file

The following example shows the contents of a Global View report file for the database TESTDB. The filename is GVTESTDB.LIS.

CREATION OF GLOBAL VIEWS ON:	1-APR-1996 12:37:24
<hr/>	
VIEW NAME	USER NAME
<hr/>	
PRODUCT	JIM
CUSTOMER	SARAH
	ANN
	DAVID
	SAM
CUSTOMER-ADDRESS	JIM
PRODUCTS-IN-REGION	SARAH
	ANN
	HARRY
	JANE
	ALICE
RDM ERROR ON VIEW :PRODUCT	
#0003 KEY NOT SUPPLIED.	

The Global View report file also lists any errors encountered when RDM attempted to create the global views.

Options for RDM access to the SUPRA Server directory

When using RDM bound and global views, you have several options regarding access to the SUPRA Server Directory by applications. “Running without the directory” on page 259, “Running with the directory and with Global Views” on page 260 and “Running with the directory alone” on page 260 describe these options. The figure under “Differences between bound and global views” on page 231 shows you graphically what happens when you try to access the Directory using the different options.

Running without the directory

If you decide not to use the SUPRA Server Directory (the logical CSI_NODIRECTORY=TRUE), RDM gets run-time information only from the Global View file. The PDM gets run-time information from the database description file identified by the logical CSI_SCHEMA.

To run an application program without the SUPRA Server Directory, the following logicals must be defined:

```
$DEFINE CSI_NODIRECTORY TRUE
$DEFINE CSI_SCHEMA dbname
$DEFINE GVSHEMA dev:[dir]global-view-file.GBL
```

Note that you can map global views to a system-wide global section (instead of the usual group-wide global section) by defining:

```
$DEFINE GVSHEMA_SYS TRUE
```

Refer to the *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130, for complete descriptions of these logicals.

Running with the directory and with Global Views

If you have access to a SUPRA Server Directory and to a Global View file, the global view overrides the corresponding view on the SUPRA Server Directory. Likewise, a bound view takes preference over the unbound version. In this environment, when a program requests a specific view, SUPRA Server proceeds as follows:

- ◆ SUPRA Server uses that view if it is available in the Global View file.
- ◆ If no global view exists, SUPRA Server seeks a bound view on the SUPRA Server Directory.
- ◆ If neither a global view nor a bound view is available, SUPRA Server tries to use the unbound view text on the SUPRA Server Directory.

When using both global views and the SUPRA Server Directory, you must have the following logicals defined:

```
$ DEFINE CSI_SCHEMA database-name
$ DEFINE GVSCHEMA database-name
```

Running with the directory alone

If a SUPRA Server Directory is available and global views are not being used, define logicals as follows:

```
$ DEFINE CSI_SCHEMA database-name
```

Note that the logical CSI_NODIRECTORY must not be defined as TRUE (you can eliminate the definition) Also note that the logical GVSCHEMA should not exist. If it points to an invalid value (non-Global View file), a warning will be generated and no Global View file will be used.

8

Generating RDM reports

The RDM reports track the complex relationships between physical data items, logical data items, and columns, and between base views and different levels of derived views. You can generate reports for both the DBA and the programmer. In addition, you can list cross references of logical and physical data items and report on domain and validation table usage.



The RDM reports are now available in the OpenVMS Alpha environment beginning with SUPRA Server PDM release 2.4.

The RDM reports show all views defined on the SUPRA Server Directory, whether base or derived. Run the reports to find out which columns are available from your base views, what integrity and security constraints they impose, and what maintenance actions are allowed for each view. Until you understand your base views, you cannot efficiently design derived views that use them. When defining views through both DBAID and DBA (EDIT/EDT interface), you can list the text of your base views on the screen, and even use their view definitions as the basis for your derived views, thereby saving typing. See “[Defining and testing views using DBAID](#)” on page 135 for details on how to use DBAID to define and test views. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for details on how to define views through DBA.

Once you have either reported on all the base views or listed them on your screen, you can begin to define derived views to fulfill the needs of the users.

The scope of the five RDM reports varies according to the option you select from the report specification screen (illustrated in “[Stage one—specifying the reports to be produced](#)” on page 264). The reports are:

- ◆ DBA report
 - All views
 - Specified databases
 - Specified views
 - Online report for a specified view
- ◆ Domain usage report
 - All domains
 - Specified domains
- ◆ Logical data item cross reference report
 - All views
 - Specified databases
 - Specified views
- ◆ Physical data item cross reference report
 - All data sets
 - Specified databases
 - Specified data sets
 - Online report for indices on a specified database
- ◆ Validation table usage report
 - All validation tables
 - Specified validation tables

RDM reports

You run the RDM reports in two stages:

1. Execute the command file CSIREQ.COM to sign on to SPECTRA and specify the reports you want to generate.
2. Produce the reports you specified by executing the command file CSIREP.COM.

2. Sign on to SPECTRA with the user name REPORTS and no password to display the SUPRA Directory Reporting Request menu, shown below.

```

Central files not available, User identification changed
==>

                SUPRA DIRECTORY REPORTING REQUEST MENU
                =====

SELECT ONE OF THE TOPICS BELOW.  TYPE THE NUMBER AND PRESS ENTER.

1.  CANCEL all report specifications
2.  DBA reports for views
3.  LOGICAL DATA ITEM CROSS-REFERENCE reports for sets of views
4.  PHYSICAL DATA ITEM CROSS-REFERENCE reports for data sets
5.  DOMAIN USAGE reports
6.  VALIDATION TABLE USAGE reports

7.  EXIT the report request facility

1=TOP 2=HELP 3=END 4=EX 5=SPLT 6=INP 7=PRIOR 8=NXT 9=MRK 10=GET 11=MOVE 12=PUT
    
```

Select one of the RDM reports by typing its number at the command line and pressing ENTER.



You can cancel all preceding report selections by typing 1 at the command line and pressing ENTER.

After you select the report, the appropriate RDM report specification screen displays. The following screen illustration shows an example of the RDM Report Specification screen displayed by selecting option 2, DBA reports for views.

```
Ready
==>2/ENTER

                SUPRA DIRECTORY REPORTING REQUEST MENU
                =====
                DBA REPORTS
                -----

SELECT ONE OF THE TOPICS BELOW.  TYPE THE NUMBER AND PRESS ENTER.

 1  VIEWS for ALL databases defined on the SUPRA Directory
 2  VIEWS for a SPECIFIED database defined on the Directory
 3  VIEWS selected by NAME only
 4  ONLINE report for specified views
 5  RETURN to top menu
 6  EXIT the report request facility

1=TOP 2=HELP 3=END 4=EX 5=SPLT 6=INP 7=PRIOR 8=NXT 9=MRK 10=GET 11=MOVE 12=PUT
```

Select the scope of the report you want to produce by typing its number at the command line, and press ENTER. This displays the process name and a brief description of the report parameters you need to enter.

In this example, option 2, VIEWS for a SPECIFIED database defined on the Directory, the following screen displays, prompting you to type in the name of the database you want to report on.

```

Ready
==>2/ENTER
  JANUARY 4TH, 1996  14:03:00  PAGE 1

REP-DBA-SPEC-DB

This routing specifies the parameters for part 2 of the DBA report
for SPECIFIED databases on the Directory.  This report is view name
within database name sequence.

Enter the name of the database required or <CTRL Z> TO EXIT

1=TOP 2=HELP 3=END 4=EX 5=SPLT 6=INP 7=PRIOR 8=NXT 9=MRK 10=GET 11=MOVE 12=PUT

```

Enter the database name in *uppercase* and press RETURN. If you enter the database name in lowercase, even if the database exists, SPECTRA rejects the specification with the message:

```
META002 - DATABASE NOT DEFINED TO THE DIRECTORY
```

After you press RETURN, SPECTRA displays a message confirming your choice and prompts you to press CTRL-Z to return to the initial Directory Reporting Request menu displayed in the second illustration above.



You can exit without specifying a report parameter by pressing CTRL-Z.

Stage two—generating the reports

Execute the command file CSIREP.COM by entering:

```
@CSI_DBA:CSIREP
```

This command file executes the SPECTRA processes to generate the reports and output them to your default directory; that is, the VMS directory to which your process is set when you execute CSIREP.

The reports have these names:

- ◆ CSI_DBA_REP.DAT - DBA Report
- ◆ CSI_DOMN.DAT - Domain Usage Report
- ◆ CSI_XREF1.DAT - Logical Data Item Cross Reference Report
- ◆ CSI_XREF2.DAT - Physical Data Item Cross Reference Report
- ◆ CSI_VALT.DAT - Validation Table Usage Report

[Appendix B](#) on page 279 illustrates the format and layout of the reports.

A

DBAID quick reference

DBAID commands

*

An asterisk has two uses:

- ◆ As a substitute for the last view name used.
- ◆ To denote a comment line when entered in column one of a view.

=

Reissues the previous RDML command.

BIND *view-name*

Saves and binds the specified view.

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

Displays the column names in a view by level of occurrence starting with level 0, followed by level 1, etc.

BYE

Exits DBAID.

CAUTIOUS

Prohibits an automatic COMMIT.

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

Displays a full description of the columns in a view. Equivalent to the FIELD-DEFN command.

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

Displays the comments for the columns in a view. Equivalent to the FIELD-TEXT command.

COMMIT

Permanently applies all updates to the database made since the last COMMIT point.

COPY *view-name*₁ *view-name*₂

Copies the text of one view to another view.

DEFINE *view-name*

Defines a new view to DBAID.

DELETE [ALL] *view-name*

Deletes a row occurrence from the database.

DENY *view-name* *user-name*₁ [...*user-name*_n]

Revokes one or more users' privilege to use a view. Separate user names by one space.

EDIT *view-name*

Prepares a saved or virtual view for modification.

ERASE

Issues an RDM RESET if an "X" FSI is returned. This command is the opposite of KEEP.

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

Displays a full description of the columns in a view. Equivalent to the COLUMN-DEFN command.

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

Displays the comments for the columns in a view. Equivalent to the COLUMN-TEXT command.

FORGET *mark-name*

Removes the specified mark and frees the storage allocated by a previously issued MARK command.

GET

NEXT
LAST
SAME
FIRST
PRIOR

view-name

[FOR UPDATE]

[AT *mark-name*
USING *literal*₁[*literal*₂...*literal*_n]]

Retrieves and displays a row for the specified view.

GO

NEXT
PRIOR

view-name

[START

NEXT
LAST
SAME
FIRST
PRIOR
AT <i>mark-name</i>

]

[FOR *number-of-rows*]

[

FROM
USING

*literal*₁[*literal*₂...*literal*_n]]

Issues a GET request based on a single key, followed by a series of sweeping GETs. Displays the rows in tabular format.

HELP [*topic*]

Invokes the DBAID online Help facility.

INSERT

NEXT
LAST
FIRST
PRIOR

view-name [**MASS**]

Places a row in the physical database at the specified relative location.

KEEP

Prohibits an automatic RESET. The opposite of the ERASE command.

line-number [***ddl-statement***]

Deletes, adds, or replaces the ASD statement in the current virtual view.

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

Specifies the number of characters to be displayed on a line.

LIST *view-name*

Displays a saved or virtual view and readies it for modification.

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

Marks the current position of the row, as established by the previous GET command.

MARKS

Lists all open MARKs and the views they are marking.

OPEN [***user-view-name***] *view-name* [***column₁***[**...**,***column_n***]

Readies a stored or virtual view for use by DBAID.

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

Specifies the number of lines to be displayed on a screen/page.

PERMIT *view-name* *user-name₁* [**...***user-name_n*]

Relates a view to a user(s) on the Directory. Separate each user name with a single space.

PRINT-STATS

Causes RDM to display the current statistics without disabling them.

RELEASE [*view-name*]

Closes a specific view or all views that have been opened and releases the occupied storage.

REMOVE *view-name*

Removes the specified view from the Directory, together with any relationships between the view and the database and the view and any users.

RENUMBER *view-name*

Renumbers a virtual view so that the line numbering starts at 10, with each line numbered in increments of 10.

RESET

Forces a task level abend and rolls back any database updates since the most recent commit point.

SAVE *view-name* [BIND]

Stores on the Directory a view that was previously opened with an OPEN command.

SHOW-NAVIGATION [*view-name*]

Allows you to verify the accuracy of the access paths used by RDM to access the underlying entities during a view open.

SIGN-OFF

Signs off the user from DBAID.

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

Identifies the user to DBAID.

STATS [view-name]

Displays statistics for one or all open views, provided you previously entered the STATS-ON command.

STATS-OFF

Prints the current statistics; then terminates statistics gathering.

STATS-ON

Initializes statistics to zero and then begins gathering statistics.

SURE

Issues a COMMIT after each successful insert, update, or delete.

UNDEFINE { ALL view-name }

Removes the name and definition of a virtual view.

UPDATE VIEW-NAME [column₁=literal₁,...column_n=literal_n]

Updates data values in the database.

USER-LIST user-view-name

Displays the column definition for the specified user view.

VIEW-DEFN [view-name]

Displays a condensed description of a view.

VIEWS

Lists all the views currently active in DBAID.

VIEWS-FOR-USER

Lists the names and short text for the views related to the signed-on user.

Definitions

<i>column-name</i>	The name of a column (logical data item) in a view.
<i>column-number</i>	The number of the column whose name is to be displayed.
<i>ddl-statement</i>	A View Definition statement.
<i>line-number</i>	The number of a line in the view.
<i>literal</i>	A group of characters used to represent a data value.
<i>mark-name</i>	The name of the mark with which you are working.
<i>number-of-records</i>	The number of records processed or to be processed.
<i>topic</i>	Subject for which Help text is requested.
<i>user-name</i>	The name of a user as defined on the Directory.
<i>user-view-name</i>	The name of the user view with which you are working.
<i>view-name</i>	The name of the view with which you are working.
<i>password</i>	Password associated with the user-name on the Directory.

Status indicators

ASI values

Value	Meaning
C	Returned when the column values are changed by another view. This check is made only when a GET statement (not GET FOR UPDATE) is followed by an UPDATE or DELETE statement. You can override this check by specifying SHARED on the ALLOW clause of an ACCESS statement.
V	Returned when the column is invalid (when a numeric column contains non-numeric data, when a column failed its validation checks [table, range, or user exit] or when a foreign key value is incorrect).
-	Returned when the column contains a null value, or when no physical record exists to supply the column value. The column in the row is set to blanks if it is a character, or zero if it is a binary, packed, numeric, or floating point data item; or, if the field contains the null value, the column contains the null value. This ASI value only has meaning on GET RDML requests.
+	Returned if the column exists and was filled from a different accessed entity. (GET processing only.) An ASI of + is given to those columns generated by new physical records. GET FIRST returns ASIs of + to all columns, because it retrieves the first row in a view and must therefore access all associated physical records as new. The ASIs + and = indicate an occurrence of a new physical record when accessing a row. Whenever a new physical record is read, its physical data items generate ASIs of +, which are assigned to their corresponding columns in the row.
=	Returned if the column exists and its value was filled from the same accessed entity as the last access (those column values generated by unchanged physical records when a new row is read). The ASIs + and = indicate an occurrence of a new physical record when accessing a row. Whenever a new physical record is read, its physical data items generate ASIs of +, which are assigned to their corresponding columns in the row.
N	The application programmer can place an N in the ASI during UPDATES and INSERTS to set a column to its null value. This ASI value is never returned by RDM.

FSI values

Value	Meaning
*	Successful completion of the RDML function.
D	Data error: The row contains invalid or changed data (VSI=C). Check the ASI to find the column(s) containing the invalid value. Check the associated message in the ULT-MESSAGE area of ULT-CONTROL.
F	Failure: Indicates a major error. Something may be wrong with the database, or you may have tried to perform an illegal function on the user view. Check the associated message in the ULT-MESSAGE area of ULT-CONTROL.
N	Not found: Indicates a failure due to an occurrence problem that may be the result of a GET not found or an INSERT duplicate found. Check the associated message in the ULT-MESSAGE area of ULT-CONTROL.
S	Security check: Verify the RDML function and correct if necessary.
U	Unavailable resource: The resource required to complete this function was not available; retry later.
X	Reset recommended: While processing, RDML modifications were made to the database before the error condition was detected. Issue a RESET to restore the database. RESET overrides D, F, S, or U indicators.

VSI values

Value	Meaning
C	At least one column was 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 column in the row has changed.
=	No invalid, missing, or new physical occurrences were returned by this RDM function.

B

Example RDM reports

This appendix contains examples of the RDM reports and a description of their content. Each variable field in the report is given a key so that it can be identified in the format descriptions. These keys are included for the purpose of this description only and do not appear on the reports.

DBA report format description

You can generate three DBA reports. The format of all three is the same, as illustrated in the figure on the following page. The difference is in the scope of the report: the first report describes all views held on the Directory; the second describes views for a specified database and the third describes selected views.

In addition to the three DBA reports, a fourth option allows you to produce an online report for a specified view. This is the same as the third DBA report on selected views except that it produces the report file while you wait.

To produce the online report, select option 4 and enter the view name in response to the LV: prompt. After a pause when the RDM Reporting facility creates the report file CSI_DBA_QCK.DAT in your default directory, you are prompted to press RETURN for next page or press END to stop:

- ◆ Press RETURN to redisplay the LV: prompt.
- ◆ Type END and press RETURN to redisplay the initial Directory Report Request menu (see the screens illustrated under “[Stage one—specifying the reports to be produced](#)” on page 264).

The DBA report describes in detail each data item used in a view. Where the reported view is a base view, the report also describes the data provided for higher level “derived” views. In addition, the DBA report lists the authorized users, the comments, the access definition, and the databases to which the view is connected.

The following figure shows the DBA report. The table following the DBA report provides field descriptions.

DECEMBER 8TH, 1996 10:11:11 PAGE 1
 LOGICAL VIEW DBA REPORT PART 3 FOR : * SELECTED VIEWS *

LOGICAL VIEW NAME : BRANCH

PHYSICAL DATA ITEM	DOMAIN OVERRIDE	DATA TYPE	CONSTANT VALUE OR DOMAIN	LOGICAL DATA ITEM	COLUMN
BRANCTRL		Key		BRANCH-ID	BRANCH-NUMBER
BRANNAME		Data		BRANCH-NAME	BRANCH-NAME
BRANREGN	Y	Required	BRANCH-REGION	BRANCH-REGION-ID	BRANCH-REGION
REGNCTRL	Y	Required	REGION	REGION-ID	BRANCH-REGION
BRANADDR		Data		BRANCH-ADDRESS	BRANCH-ADDRESS
BRANCITY		Data		BRANCH-CITY	BRANCH-CITY
BRANSTAT		Data	STATES	BRANCH-STATE	BRANCH-STATE
BRANZIPC		Data	ZIP-CODES	BRANCH-ZIP-CODE	BRANCH-ZIP-CODE
BRANSLSQ		Data		BRANCH-SALES-QUOTA	BRANCH-SALES-QUOTA
BRANSTFQ		Data		BRANCH-STAFF-QUOTA	BRANCH-STAFF-QUOTA
BRANDELV		Data	DELIVERY-ROUTES	BRANCH-DELIVERY-ROUTE	BRANCH-DELIVERY-ROUTE

DATA PROVIDED FOR HIGHER LEVEL VIEWS :

FOR VIEW	COLUMN	SOURCE COLUMN	DATA TYPE	CONSTANT VALUE OR DOMAIN	D/OV
BRANCH-LOCATION	BRANCH-NUMBER	BRANCH-NUMBER	Key		
BRANCH-LOCATION	BRANCH-NAME	BRANCH-NAME	Data		
BRANCH-LOCATION	BRANCH-REGION	BRANCH-REGION	Required		
BRANCH-LOCATION	BRANCH-ADDRESS	BRANCH-ADDRESS	Data		
BRANCH-LOCATION	BRANCH-CITY	BRANCH-CITY	Data		
BRANCH-LOCATION	BRANCH-STATE	BRANCH-STATE	Data		
BRANCH-LOCATION	BRANCH-ZIP-CODE	BRANCH-ZIP-CODE	Data		
BRANCH-LOCATION	BRANCH-DELIVERY-ROUTE	BRANCH-DELIVERY-ROUTE	Data		
PRODUCTS-IN-REGION	BRANCH-NUMBER	BRANCH-NUMBER	Key		
PRODUCTS-IN-REGION	BRANCH-NAME	BRANCH-NAME	Data		
BRANCH-STOCK	BRANCH-NUMBER	BRANCH-NUMBER	Key		
BRANCH-STOCK	BRANCH-NAME	BRANCH-NAME	Data		
PRODUCTS-IN-BRANCH	BRANCH-NUMBER	BRANCH-NUMBER	Key		
REGION-111-INFO	BRANCH-NUMBER	BRANCH-NUMBER	Key		
REGION-111-INFO	REGION-NUMBER	BRANCH-REGION	Constant	111	
REGION-111-INFO	BRANCH-NAME	BRANCH-NAME	Data		
REGION-111-INFO	BRANCH-CITY	BRANCH-CITY	Data		
REGION-111-INFO	BRANCH-STATE	BRANCH-STATE	Data		
BRANCHES-IN-REGION	BRANCH-NUMBER	BRANCH-NUMBER	Key		
BRANCHES-IN-REGION	BRANCH-NAME	BRANCH-NAME	Data		

USERS :

DR

COMMENTS :

ACCESS DEFINITION :

ACCESS BRAN
WHERE BRANCH-ID = BRANCH-NUMBER
ALLOW ALL
* To verify that BRANCH-REGION contains a valid region
* on INSERT and UPDATE.
ACCESS REGN
ONCE
WHERE REGION-ID == BRANCH-REGION-ID
* To restrict deletions of branches containing customers.
ACCESS CUST
WHERE CUSTOMER-BRANCH-ID = BRANCH-ID
* To restrict deletions of branches that have stock.
ACCESS STCK
WHERE STOCK-BRANCH-ID = BRANCH-ID

USED BY DATABASES :

EXAMPL

The first half of the report in preceding figure details each item used in a view. The following information explains the fields in that half of the report:

Field	Explanation
PHYSICAL DATA ITEM	Each column in a view eventually maps to a physical data item through its logical data item equivalent, possibly via many levels of logical view. This field shows the physical data item name.
DOMAIN OVERRIDE	A "Y" in this field indicates that the "DOMAIN OVERRIDE" feature is enabled for this physical data item. Note that this field indicates only those domain overrides enabled through the column definition of the view. Domain overrides entered in the access definition are not listed here, they are given in the listing of the access definition (see field number 15).
DATA TYPE	This field shows the type of the physical data item: key, constant, data, and so on.
CONSTANT VALUE OR DOMAIN	This field contains the constant value associated with this data item in this view, if there is one. If there is no constant value, this field contains the domain to which the physical data item belongs, if there is one. If there is neither constant value nor domain associated with the data item, this field remains blank.
LOGICAL DATA ITEM	<p>This field contains the logical name associated with the physical data item on the Directory. This name must appear as a column in the base view and can be found by following the hierarchy of views to the lowest level.</p> <p>If the view being reported accesses a base view, the name of the base view is given underneath the logical data item name, preceded by the literal "FROM VIEW:".</p> <p>If the base view is replaced and the column in the derived view no longer maps onto a lower level view, the literal "..VIEW HIERARCHY INCONSISTENT" displays.</p> <p>If more than five derived views exist between the reported view and the base view, the literal "..MORE THAN 6 LEVELS ..." displays.</p>
COLUMN	The name in this view that refers to the logical data item. This column name is the same as the source column name, unless this view specifies an alternative name in its column definition.

Other views can use this reported view. The second half of the report shown in preceding figure details the fields directly used by other views. The following information explains those fields:

Field	Explanation
FOR VIEW	The view name for which data is provided by this view.
COLUMN	The name that the derived view (field 7) uses to refer to the source column (field 9). This column name is the same as the source column name, unless this view specifies an alternative name in its column definition.
SOURCE COLUMN	Matches the column in the base view, given in field 6.
DATA TYPE	The type of field in the higher level view (the view given in field 7).
CONSTANT VALUE / DOMAIN	Provides the constant value for the column when used in the view given in field 7.
D/OV	Indicates whether the domain override facility is used in the column definition of the derived view. Field 4 describes the constant value or domain if this field has no value.
USERS	List of the authorized users of the view.
COMMENTS	Any comments on the Directory for the view.
ACCESS DEFINITION	The access definition of the view.
USED BY DATABASES	The databases to which this view is connected.

Domain usage report format description

You can generate two Domain Usage reports. The format of both is the same, as illustrated in the following figure. The difference is in the scope of the report: the first report describes all domains held on the Directory; the second report describes selected domains. Note that you can select more than one domain for the report on selected domains.

The Domain Usage report lists the content of each domain and any comments. It then describes the physical and logical data items that use the domain, the base views that directly use the logical data item, and any alias or constant values associated with the logical data item.

The following figure shows the Domain Usage report. The table following the figure provides field descriptions.

```

                                DECEMBER 8TH, 1996 10:21:22 PAGE 1
                                DOMAIN USAGE REPORT PART 2 - * SELECTED DOMAINS *

                                DOMAIN NAME: STATES

----- DATA ITEMS -----
PHYSICAL | LOGICAL          USED DIRECTLY IN VIEWS  COLUMN          CONSTANT VALUE
-----|-----|-----|-----|-----
BRANSTAT          BRANCH-STATE          BRANCH
CADRBSTA          CADR-BILL-STATE
CADRSSTA          CADR-SHIP-STATE
CUSTSTAT          CUSTOMER-STATE        CUSTOMER
ORDRSTAT          ORDER-SHIP-STATE      ORDER

COMMENTS :
-----

VALIDATION TABLE: STATES
VALIDATION EXIT:
RETRIEVAL VALIDATION: NO
VALIDATION TYPE: TABLE
NULLS PERMITTED: YES
FUNCTION: STRING
UNITS: N/A
FORMAT: CHARACTER
SIGN: SIGNED
LENGTH: 2
DECIMAL PLACES: 0
MINIMUM VALUE:
MAXIMUM VALUE:
NULL VALUE:
DEFAULT VALUE:

```

DECEMBER 8TH, 1996 10:21:22 PAGE 2
 DOMAIN USAGE REPORT PART 2 - * SELECTED DOMAINS *

DOMAIN NAME: ZIP-CODES

----- DATA ITEMS -----				
PHYSICAL	LOGICAL	USED DIRECTLY	IN VIEWS	COLUMN
-----		-----		
				CONSTANT VALUE
BRANZIPC	BRANCH-ZIP-CODE			BRANCH
CADRBZIP	CADR-BILL-ZIP-CODE			
CADRSZIP	CADR-SHIP-ZIP-CODE			
CUSTZIPC	CUSTOMER-ZIP-CODE			CUSTOMER
ORDRZIPC	ORDER-SHIP-ZIP-CODE			ORDER

COMMENTS :

Valid range for all ZIP-CODES is between '00000' and '99999'.

VALIDATION TABLE:
 VALIDATION EXIT:
 RETRIEVAL VALIDATION: NO
 VALIDATION TYPE: RANGE
 NULLS PERMITTED: YES
 FUNCTION: STRING
 UNITS: N/A
 FORMAT: CHARACTER
 SIGN: SIGNED
 LENGTH: 5
 DECIMAL PLACES: 0
 MINIMUM VALUE: 00000
 MAXIMUM VALUE: 99999
 NULL VALUE:
 DEFAULT VALUE:

The table below explains the fields for the previous figures:

Field	Explanation
PHYSICAL	Physical data items to which the domain is connected.
LOGICAL	The logical name for each physical data item given in field 1. This report lists only the logical data item names used in base views.
USED DIRECTLY IN VIEWS	Names of the base views that use the data items.
COLUMN	Column name for the logical data item. The column name is the same as the logical data item name unless the views specify alternative names in their column definitions. The column name always takes precedence over the equivalent logical data item name when the two differ because the column is referred to by application programs, SPECTRA, and derived views.
CONSTANT VALUE	Constant value for the column, if any.
COMMENTS	Comments held on the Directory for the domain.

Logical Data Item report format description

You can generate three Logical Data Item reports. The format of all three is the same, as illustrated in the following figure. The difference is the scope of the report: the first report describes all the logical data items in all views; the second describes all views connected to specified databases (note that you can specify more than one database); and the third describes selected views.

The Logical Data Item Cross Reference report describes logical data items, the base and derived views in which they are used and any column names, along with any constant values or domains that are associated with them. The report is presented in logical data item order.

The following figure shows the Logical Data Item Cross Reference report.
 The table following the figure provides field descriptions.

DECEMBER 8TH, 1996 10:12:11 PAGE 1
 LOGICAL DATA ITEM CROSS-REFERENCE REPORT PART 2 FOR : EXAMPL

VIEW	FROM BASE VIEW	COLUMN	CONSTANT VALUE	DOMAIN OVERRIDE
LOGICAL DATA ITEM : BRANCH-ADDRESS		PHYSICAL DATA ITEM : BRANADDR DOMAIN :		
BRANCH			BRANCH-ADDRESS	
BRANCH-LOCATION	BRANCH		BRANCH-ADDRESS	
LOGICAL DATA ITEM : BRANCH-CITY		PHYSICAL DATA ITEM : BRANCITY DOMAIN :		
BRANCH			BRANCH-CITY	
BRANCH-LOCATION	BRANCH		BRANCH-CITY	
REGION-111-INFO	BRANCH		BRANCH-CITY	
LOGICAL DATA ITEM : BRANCH-DELIVERY-ROUTE		PHYSICAL DATA ITEM : BRANDELV DOMAIN : DELIVERY-ROUTES		
BRANCH			BRANCH-DELIVERY-ROUTE	
BRANCH-LOCATION	BRANCH		BRANCH-DELIVERY-ROUTE	
LOGICAL DATA ITEM : BRANCH-ID		PHYSICAL DATA ITEM : BRANCTRL DOMAIN :		
ADD-CUSTOMER-DEFAULT-VALUES	CUSTOMER		CUSTOMER-BRANCH	
BRANCH			BRANCH-NUMBER	
BRANCH-LOCATION	BRANCH		BRANCH-NUMBER	
BRANCH-PRODUCTS			BRANCH-NUMBER	
BRANCH-STOCK	BRANCH		BRANCH-NUMBER	
BRANCHES-IN-REGION	BRANCH		BRANCH-NUMBER	
CUSTOMER			CUSTOMER-BRANCH	
CUSTOMER-INSERT-INTEGRITY			BRANCH-NUMBER	
CUSTOMER-INSERT-INTEGRITY-2			CUSTOMER-BRANCH	
CUSTOMER-INSERT-INTEGRITY-3			CUSTOMER-BRANCH	
CUSTOMER-UPDATE-INTEGRITY			CUSTOMER-BRANCH	
CUSTOMER-UPDATE-INTEGRITY-2			CUSTOMER-BRANCH	
CUSTOMER-UPDATE-INTEGRITY-3			CUSTOMER-BRANCH	
CUSTOMER-UPDATE-INTEGRITY-4			CUSTOMER-BRANCH	
CUSTOMERS-IN-TEXAS	CUSTOMER		CUSTOMER-BRANCH	
CUSTOMERS-IN-TEXAS-2	CUSTOMER		CUSTOMER-BRANCH	
DELETE-REGION-WITH-NO-CUSTOMER			BRANCH-NUMBER	
ORDER			ORDER-BRANCH	
PRODUCTS-IN-BRANCH	BRANCH		BRANCH-NUMBER	
PRODUCTS-IN-REGION	BRANCH		BRANCH-NUMBER	
REGION-111-INFO	BRANCH		BRANCH-NUMBER	
STOCK			STOCK-BRANCH	

The table below explains the fields for the previous figure:

Field	Explanation
LOGICAL DATA ITEM:	The logical data item name.
PHYSICAL DATA ITEM:	The physical data item to which the logical data item maps.
DOMAIN:	The name of the domain used by the data item.
VIEW	The views that use the logical data item.
FROM BASE VIEW	If the views in field 4 use the logical data item via intermediate access to other views, this field contains the name of the view that uses the logical data item directly. If the view in field 4 uses the logical data item directly, this field remains empty.
COLUMN	The column name that refers to this logical data item. This column name is the same as the logical data item name, unless the views specify alternative column names in their column definitions.
CONSTANT VALUE	The constant value displays here if the logical data item has one.
DOMAIN OVERRIDE	This field contains a “Y” if the domain override feature is enabled.

In addition to the three DBA reports, a fourth option allows you to produce an online report for a specified view. This report is the same as the third DBA report on selected views except that it produces the report file while you wait.

To produce the online report, select option 4 and enter the view name in response to the LV: prompt. After a pause when the RDM Reporting facility creates the report file CSI_DBA_QCK.DAT in your default directory, you are prompted to press RETURN for next page or press END to stop:

- ◆ Press RETURN to redisplay the LV: prompt.
- ◆ Type END and press RETURN to redisplay the initial Directory Report Request menu (see the screens illustrated under “**Stage one—specifying the reports to be produced**” on page 264).

Physical Data Item report format description

You can generate three Physical Data Item reports. The format of all three is the same, as illustrated in the following figure. The difference between them lies in the scope of the report: the first report describes all data sets held on the Directory; the second describes data sets connected to a specified database and the third describes selected data sets. Note that you can select more than one database or data set for the second and third reports. All three reports first display details of any indices connected to each data set, followed by the physical to logical data item cross-reference.

The Physical Data Item Cross Reference report describes any connected indices, the physical data items, the logical data items to which they map, the columns used to access the logical data items, the base views they are used in, and any constant values. Physical data items are presented by data set in the order they are defined. Where a physical data item is subdefined, the parent data item is followed by its sub-data-items.

In addition to the three Physical Data Item reports, a fourth option allows you to produce an online report showing details of the indices connected to the specified database.

To produce the online report, select option 4, and enter the database name in response to the DBNAME: prompt. After a pause when the RDM Reporting facility creates the report file CSI_XREF2_QCK.DAT in your default directory, you are prompted to press RETURN for next page or press END to stop:

- ◆ Press RETURN to redisplay the DBNAME: prompt.
- ◆ Type END and press RETURN to redisplay the initial Directory Report Request menu (see the screens illustrated under “[Stage one—specifying the reports to be produced](#)” on page 264).

The index details in the report file CSI_XREF2_QCK.DAT are presented in the same format as the index details shown in the following figure. However, the physical to logical data item details are omitted.

The following figure shows the Physical Data Item Cross Reference report. The table following the figure provides field descriptions.

DECEMBER 8TH, 1996 10:12:11 PAGE 1
 PHYSICAL DATA ITEM CROSS REFERENCE REPORT - PART 2 FOR EXAMPL
 PRIMARY DATA SET : BRAN

RECORD PHYSICAL CODE DATA ITEM	LOGICAL DATA ITEM	COLUMN OR CONSTANT VALUE	USED DIRECTLY BY VIEWS
-----------------------------------	----------------------	--------------------------	------------------------

INDEX NAME:- RN FOR DATABASE:- EXAMPL AND FILE:- BRAN
 CORRUPT-ACTION: OPERATOR
 NULLS SORTED: HIGH
 READ VERIFY: YES
 FILE-SPECIFICATION: RDM_EXAMPLE:BRAN.IDX
 SHADOW-FILE-SPECIFICATION:

The secondary key is needed on the foreign key REGN.

KEY NAME:- BRANSKRE
 UNIQUE = NO
 DIRECTION = BOTH
 ORDERING = NO
 TYPE = DIRECT
 SORT = NO
 % DUPLICATES ALLOWED = 40

Allow key access to the Branch data set using the REGION-NUMBER

foreign key.

DATA ITEMS IN KEY:-
 BRANREGN

DATA ITEM CROSS-REFERENCE :-

BRANROOT			
BRANCTRL	BRANCH-ID	STOCK-BRANCH	STOCK
BRANCTRL	BRANCH-ID	ORDER-BRANCH	ORDER
BRANCTRL	BRANCH-ID	BRANCH-NUMBER	BRANCH-PRODUCTS
BRANCTRL	BRANCH-ID	BRANCH-NUMBER	DELETE-REGION-WITH-NO-CUSTOMER
BRANCTRL	BRANCH-ID	BRANCH-NUMBER	BRANCH
BRANCTRL	BRANCH-ID	CUSTOMER-BRANCH	CUSTOMER-INSERT-INTEGRITY
BRANCTRL	BRANCH-ID	CUSTOMER-BRANCH	CUSTOMER-INSERT-INTEGRITY-2
BRANCTRL	BRANCH-ID	CUSTOMER-BRANCH	CUSTOMER-INSERT-INTEGRITY-3
BRANCTRL	BRANCH-ID	CUSTOMER-BRANCH	CUSTOMER-UPDATE-INTEGRITY
BRANCTRL	BRANCH-ID	CUSTOMER-BRANCH	CUSTOMER-UPDATE-INTEGRITY-2
BRANCTRL	BRANCH-ID	CUSTOMER-BRANCH	CUSTOMER-UPDATE-INTEGRITY-3
BRANCTRL	BRANCH-ID	CUSTOMER-BRANCH	CUSTOMER-UPDATE-INTEGRITY-4

DECEMBER 8TH, 1996 10:12:11 PAGE 2
 PHYSICAL DATA ITEM CROSS REFERENCE REPORT - PART 2 FOR EXAMPL
 PRIMARY DATA SET : BRAN

RECORD CODE	PHYSICAL DATA ITEM	LOGICAL DATA ITEM	COLUMN OR CONSTANT VALUE	USED DIRECTLY BY VIEWS
	BRANREGN	BRANCH-REGION-ID	BRANCH-REGION	BRANCH
	BRANREGN	BRANCH-REGION-ID	REGION-NUMBER	CUSTOMER-INSERT-INTEGRITY-3
	BRANREGN	BRANCH-REGION-ID	BRANCH-REGION	CUSTOMER-UPDATE-INTEGRITY-3
	BRANDELV	BRANCH-DELIVERY-ROUTE	BRANCH-DELIVERY-ROUTE	BRANCH
	BRANSLSQ	BRANCH-SALES-QUOTA	BRANCH-SALES-QUOTA	BRANCH
	BRANSTFQ	BRANCH-STAFF-QUOTA	BRANCH-STAFF-QUOTA	BRANCH
	BRANLKST			

The table below explains the fields for the previous figure:

Field	Explanation
INDEX NAME: FOR DATABASE: AND FILE:	The index name, database name and data set name.
CORRUPT-ACTION:	<p>The action taken if the index is corrupt.</p> <ul style="list-style-type: none"> ◆ OPERATOR marks the index as unavailable and prompts the operator. ◆ CONTINUE marks the index as unavailable and continues processing without using the corrupt index file. ◆ POPULATE performs a dynamic populate on the corrupt index.
NULLS SORTED:	Where in the collating sequence nulls are sorted.
READ VERIFY:	Whether the PDM checks if the index is corrupt before reading it.
FILE SPEC: SHADOW FILE SPEC:	Main and shadow file specifications.
KEY NAME:	The name of the secondary key.
UNIQUE =	Whether or not the index supports duplicate secondary keys.
DIRECTION=	The direction in which the keys are sorted in the file (forward, reverse or both).
ORDERING=	Whether or not the index uses pointers to ensure that records with identical keys are retrieved in the order they occur in the data set file.
TYPE =	The pointer type stored with the secondary key in order to make sure that records with identical keys are retrieved in the order they occur in the file. Indirect using a control key or Direct using Relative Record Number (RRN).

Validation Table Usage report format description

You can generate two Validation Table Usage reports. The format of both is the same, as illustrated in the following figure. The difference between them lies in the scope of the report; the first report describes all validation tables on the Directory; the second report describes selected validation tables. Note that you can select more than one validation table.

The Validation Table report lists the values given in each validation table, any comments, and the domains that use the validation table.

The figure below shows the Validation Table report. The table following the figure provides field descriptions.

```
                                DECEMBER 12TH, 1996 09:35:33 PAGE 1
                                VALIDATION TABLE REPORT PART 2 - * SELECTED TABLES *

                                VALIDATION TABLE: CUSTOMER-CLASS

COMMENTS :
-----

PERMITTED DATA VALUES
-----

                                001
                                002
                                003

USED IN DOMAINS
-----

                                CUSTOMER-CLASS
```

The following table explains the fields in the above figure.

Field	Explanation
1	Comments held on the Directory for the validation table.
2	A list of valid values for a data item using a domain to which this validation table applies.
3	A list of domains that use this validation table.

C

Example user validation exits

This appendix contains sample validation exits in C, COBOL, and PASCAL. You can use validation exits to write more complex validation logic than is available using Range and Validation tables.

You define validation exits in the Directory using the DBA Domains functions: enter E in the Domain Validation Type field, enter the 8-character Validation Exit name in the Domain Validation Exit Name field and then connect the Domain to the Physical Field you want the Domain to apply. Then, using the following examples as guidelines, code, compile, and link your program.

At run time, when RDM is processing a field that has Validation Exit defined and you have a logical name called CSI_VAL_EXIT pointing to the validation image, RDM will dynamically load the image, find the exit program in the image and pass the parameters for your program to execute.

The parameter list consists of a list of 11 addresses.

Addr1 Address of the return code. The return code is a 4-byte binary integer. The values the validation exit should set are:

1 = Valid value or valid exit name

0 = Invalid value

Addr2 Address of the 8-character exit name.

Addr3 Address of the 30-character user name.

Addr4	Address of the 30-character view name.
Addr5	Address of the 30-character column name.
Addr6	Address of the value to be validated. The length and format may vary.
Addr7	Address of the 1-character field type: C Character P Packed Z Zoned B Binary F Floating Point
Addr8	Address of the length of the value. This is a 4-byte binary integer.
Addr9	Address of the 1-character sign flag for this field. Y The value is signed. N The value is not signed.
Addr10	Address of the number of decimals in the value. This is a 4-byte binary integer.
Addr11	Address of the 1-character operation type. This field indicates the type of RDML request that caused the call of the validation exit. G GET RDML I INSERT RDML U UPDATE RDML O Open of the view. Exit should return a GOOD status for this type.

Examples

- ◆ This example uses COBOL to illustrate how to define a user exit.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. ROUTEXIT.
*****
*
* This is an example of a SUPRA RDM validation exit.
*       * It checks that the data passed to it is numeric.
*
* If so, it returns a 'good' status (1), otherwise a 'bad'
* status (0).
*
*****
ENVIRONMENT DIVISION.
DATA DIVISION.
LINKAGE SECTION.
*****
*
* All parameters must be declared in the LINKAGE SECTION.
*
*****
01   RETURN-STATUS  PIC S9(8) COMP.
* Status code returned to RDM indicating success or failure.
*       Success = 1, Failure = 0.

01   EXIT-NAME      PIC X(8).
* Contains the name of the exit.

01   USER-NAME     PIC X(30).
* The RDM user name.

01   VIEW-NAME     PIC X(30).
* The name of the current RDM view.

```

```
01    COLUMN-NAME    PIC X(30).
* The RDM column name.

01    DATA-BUFFER    PIC X(2).
* The area containing the data to be validated. The PICTURE clause
* must reflect the length of the data area. It is only 2 bytes in
* length in this example. It may be longer than really necessary.
* Its true length is specified in the DATA-LENGTH parameter.

01    DATA-FORMAT    PIC X.
* Single character indicating the type of data in the DATA-BUFFER.

01    DATA-LENGTH    PIC S9(8) COMP.
* The actual length of the data in the DATA-BUFFER.

01    SIGNED          PIC X.
* Sign indicator. Y if signed, otherwise N.

01    DECIMALS        PIC S9(8) COMP.
* Number of decimal places allowed.

01    OPERATION       PIC X.
* The RDM operation being performed.
* G = Get, I = Insert, U = Update, D = Delete.
```

PROCEDURE DIVISION

 USING

 RETURN-STATUS

 EXIT-NAME

 USER-NAME

 VIEW-NAME

 COLUMN-NAME

 DATA-BUFFER

 DATA-FORMAT

 DATA-LENGTH

 SIGNED

 DECIMALS

 OPERATION.

PARAGRAPH-NAME .

 IF DATA-BUFFER IS NUMERIC

 MOVE 1 TO RETURN-STATUS

 ELSE

 MOVE 0 TO RETURN-STATUS .

 EXIT PROGRAM.

- ◆ This example uses PASCAL to illustrate how to define a user exit.

```
MODULE sample val exit;
(*****
(* This is a sample Pascal Validation Exit Module.          *)
*****)

TYPE

    char8  = Packed Array [1..8] Of Char;
    char30 = Packed Array [1..30] Of Char;
    char60 = Packed Array [1..60] Of Char;

[GLOBAL] PROCEDURE NOTBLANK(
    VAR status      : Integer;
        exit name  : char8;
        user name  : char30;
        view name  : char30;
        col name   : char30;
        value      : char60;
        format     : char;
        length     : Integer;
        signed     : Char;
        decimals   : Integer;
        operation  : Char);
```

```
(*****)  
(* Description - This Procedure checks if the data passed via *)  
(*           the variable VALUE begins with a Blank, if so, a *)  
(*           status of 0 is passed back to indicate an      *)  
(*           invalid value, otherwise, a 1 is returned to   *)  
(*           indicate a valid value.                        *)  
(* Input Parameters :                                     *)  
(*   exit name : 8 character validation exit name, i.e., the *)  
(*             name of this procedure.                    *)  
(*   user name : 30 character RDM user name.              *)  
(*   view name : 30 character View name.                 *)  
(*   col name  : 30 character Column name.               *)  
(*   value     : 60 character buffer containing the data to be *)  
(*             validated. As this buffer is 60 bytes long, *)  
(*             this procedure can be used by any column with *)  
(*             length of 60 bytes or less. NOTE that if you *)  
(*             ---- *)  
(*             require a longer buffer, you must change this *)  
(*             parameter ! The parameter LENGTH gives the  *)  
(*             actual length of the data.                  *)  
(*   format    : 1 character data type.                   *)  
(*   length    : 4 Byte integer containing the data length. *)  
(*   signed    : 1 character sign indicator, Y if signed,  *)  
(*             N otherwise.                                *)  
(*   decimals  : 4 Byte integer containing the number of decimal *)  
(*             places allowed.                             *)  
(*   operation: 1 Byte character RDM operation type.      *)  
(*             G for Get, I for Insert, U for Update, etc. *)  
(*             *)  
(* Output Parameters :                                   *)  
(*   status    : 4 Byte integer to contain result of validation, *)  
(*             1 if data is valid, 0 otherwise.           *)  
(*             *)  
(***)
```

```

BEGIN

    If (Operation <> '0') Then Begin
        If (value[1] = ' ') Then
            status := 0                Return Bad status
        Else
            status := 1;                Return Good status
        End
    Else
        status := 1;                    Always return good
    End
    status := 1;                        for Open
END;
END.

```

- ◆ This example shows how to compile the PASCAL exit module.

```

$PASCAL SAMPLE_VAL_EXIT
!$ This command will compile the module which resides in a file
!$ called SAMPLE_VAL_EXIT.PAS and puts the object module in a file
!$ called SAMPLE_VAL_EXIT.OBJ which will be used by the LINKER later.

```

- ◆ This example shows how to link the PASCAL exit module, after it is compiled.

```

! This is a sample option file for linking the sample validation
! exit module. It is used in the following form:
! LINK/SHARE=SAMPLE_VAL_EXIT.EXE SAMPLE_VAL_EXIT/OPT
!     where sample val exit.exe is the name of the Shareable image
! being created. It is this name that you assign the Logical
! name CSI_VAL_EXIT to, e.g.
! $Define CSI_VAL_EXIT dev :[directory]SAMPLE_VAL_EXIT.EXE
!
! Note all validation exit names must be made UNIVERSAL at link
! time.
SAMPLE_VAL_EXIT      ! Module containing validation exit routines.
UNIVERSAL=NOTBLANK  ! Must be UNIVERSAL in order for RDM to pick
[, EXIT2, EXIT3 ...] ! up at Run Time. THIS IS A REQUIREMENT.
GSMATCH=ALWAYS,0,0  ! To ensure upward capability.
! All exits can come from the same module or multiple modules
! containing one or more exit routines.

```

- ◆ This example uses C to illustrate how to define a user exit.

```

/*****
 * This is an example of an RDM validation exit.  It checks that
 * the data passed to it contains no digits and is not all blank.
 *****/

#include <ctype.h>
CITYEXIT(p_status, /* status to be returned      - integer  */
p_exit_name, /* name of validation exit      - 8 chars  */
p_user_name, /* name of RDM user             - 30 chars */
p_view_name, /* name of RDM view             - 30 chars */
p_column_name, /* name of column in view      - 30 chars */
p_data_buffer, /* data to be validated        - any length*/
p_data_format, /* format of data              - 1 char   */
p_data_length, /* length of data              - integer  */
p_signed, /* data signed or not          - 1 char   */
p_decimals, /* number of decimal places    - integer  */
p_operation) /* RDM operation               - 1 char   */
int *p_status;
char *p_exit_name;
char *p_user_name;
char *p_view_name;
char *p_column_name;
char *p_data_buffer;
char *p_data_format;
int *p_data_length;
char *p_signed;
int *p_decimals;
char *p_operation;

```

```
/*
*****
* ensure its all uppercase and does not contain digits      *
* and is not all spaces.                                     *
*****/
int i;                /* loop control variable             */
char spaces[ ]=" "; /* constant                           */
*p status = 1; /* indicate success until proven           */
                /* otherwise                               */
for (i = 0; i < *p data length; i++)
    if (isdigit(p data buffer[i]))
        *p status = 0; /* indicate failure               */
        break;        /* break loop                       */

if (strncmp(p data buffer,spaces,13) == 0)
    *p status = 0; /* indicate failure                   */
return;          /* return to caller                               */
*/
```

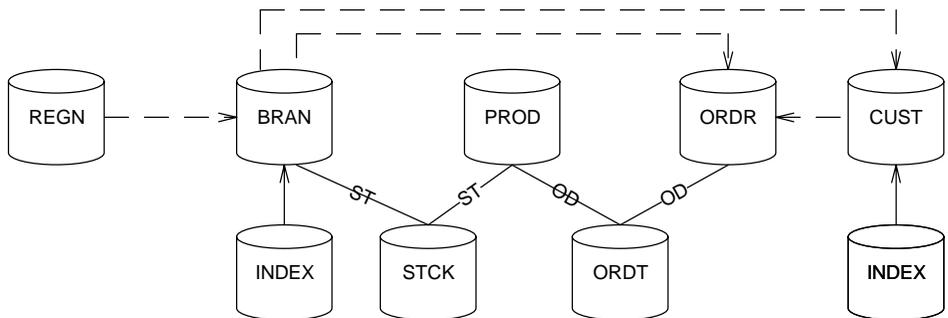
D

Example database

This appendix describes the EXAMPL database provided with SUPRA Server. The database described here is used throughout this manual for illustration. All three schemas, internal, conceptual, and external, are described in detail to illustrate the access paths and maintenance function constraints imposed at the different levels.

Relations in the internal schema

The internal schema consists of the physical description of the database and contains a mix of primary, related, and RMS data sets known as base relations. These data sets are shown in figure below. Note that the dashed lines represent foreign keys, and the solid lines represent linkpaths.



The data sets that comprise the internal schema are made up of physical data items and linkpaths. The following bulleted list shows each physical data item in a data set, its equivalent logical data item name (used in base views), any foreign key relationships, and connected domains or constant values.

◆ RMS data set REGN

File	Physical data item	Logical data item	Format	Domain or constant value
REGN	REGNCTRL=0002	REGION-ID	PIC X(2)	REGION
REGN	REGNAME=0030	REGION-NAME	PIC X(30)	

◆ PDM primary data set BRAN

File	Physical data item	Logical data item	Format	Domain or constant value
BRAN	BRANROOT=0008			
BRAN	BRANCTRL=0004	BRANCH-ID	PIC X(4)	
BRAN	BRANNAME=0030	BRANCH-NAME	PIC X(30)	
BRAN	BRANADDR=0040	BRANCH-ADDRESS	PIC X(40)	
BRAN	BRANCITY=0040	BRANCH-CITY	PIC X(40)	
BRAN	BRANSTAT=0002	BRANCH-STATE	PIC X(2)	STATES
BRAN	BRANZIPC=0005	BRANCH-ZIP-CODE	PIC X(5)	ZIP-CODES
BRAN	BRANREGN=0002	BRANCH-REGION-ID	PIC X(2)	BRANCH-REGION
BRAN	BRANDELV=0002	BRANCH-DELIVERY-ROUTE	PIC X(2)	DELIVERY-ROUTES
BRAN	BRANSLSQ=0009	BRANCH-SALES-QUOTA	PIC S9(7)V9(2) COMP	
BRAN	BRANSTFQ=0009	BRANCH-STAFF-QUOTA	PIC S9(9) COMP	
BRAN	BRANLKST=0008	Linkpath to STCK		

Notes

- BRANCH-NUMBER is a foreign key in the related data set STCK as STOCK-BRANCH and in the primary data set CUST as CUSTOMER-BRANCH.
- REGION-NUMBER is a foreign key in the primary data set BRAN as BRANCH-REGION.
- BRANCH-REGION has a secondary key (index) associated with it.
- Two domains are used for REGION and BRANCH-REGION because BRANCH-REGION may be set to the NULL value during a nullify delete.

◆ PDM related data set STCK

File	Physical data item	Logical data item	Format	Domain or constant value
STCK	STCKBRAN=0004	STOCK-BRANCH-ID	PIC X(4)	
STCK	BRANLKST=0008	Linkpath to BRAN		
STCK	STCKPROD=0008	STOCK-PRODUCT-ID	PIC X(8)	
STCK	PRODLKST=0008	Linkpath to PROD		
STCK	STCKQNTY=0009	STOCK-QUANTITY	PIC S9(9) COMP	
STCK	STCKBINL=0003	STOCK-BIN-LOCATION	PIC X(3)	
STCK	STCKYTDS=0009	STOCK-YEAR-TO-DATE- SALES	PIC S9(7)V9(2) COMP	

◆ PDM primary data set PROD

File	Physical data item	Logical data item	Format	Domain or constant value
PROD	PRODROOT=0008			
PROD	PRODCTRL=0008	PRODUCT-ID	PIC X(8)	
PROD	PRODDDESC=0040	PRODUCT-DESCRIPTION	PIC X(40)	

File	Physical data item	Logical data item	Format	Domain or constant value
PROD	PRODQUAN=0009	PRODUCT-QUANTITY	PIC X(9)	
PROD	PRODPRCE=0009	PRODUCT-PRICE	PIC S9(7)V9(2) COMP	
PROD	PRODGRUP=0003	PRODUCT-GROUP	PIC X(3)	
PROD	PRODLKST=0008	Linkpath to STCK		
PROD	PRODLKOD=0008	Linkpath to ORDT		

◆ PDM primary data set CUST

File	Physical data item	Logical data item	Format	Domain or constant value
CUST	CUSTROOT=0008			
CUST	CUSTCTRL=0008	CUSTOMER-ID	PIC X(8)	
CUST	CUSTNAME=0060	CUSTOMER-NAME	PIC X(60)	
CUST	CUSTADDR=0040	CUSTOMER-ADDRESS	PIC X(40)	
CUST	CUSTCITY=0030	CUSTOMER-CITY	PIC X(30)	
CUST	CUSTSTAT=0002	CUSTOMER-STATE	PIC X(2)	STATES
CUST	CUSTZIPC=0005	CUSTOMER-ZIP-CODE	PIC X(5)	ZIP-CODES
CUST	CUSTFONE=0013	CUSTOMER-PHONE-NUMBER	PIC X(13)	
CUST	CUSTFAXX=0013	CUSTOMER-FAX-NUMBER	PIC X(13)	
CUST	CUSTCLAS=0003	CUSTOMER-CLASS	PIC X(3)	CUSTOMER-CLASS
CUST	CUSTCRCO=0002	CUSTOMER-CREDIT-CODE	PIC X(2)	CREDIT-CODES
CUST	CUSTCRLM=0009	CUSTOMER-CREDIT-LIMIT	PIC S9(9) COMP	CREDIT-LIMIT
CUST	CUSTBRAN=0004	CUSTOMER-BRANCH-ID	PIC X(4)	

◆ PDM primary data set ORDR

File	Physical data item	Logical data item	Format	Domain or constant value
ORDR	ORDRROOT=0008			
ORDR	ORDRCTRL=0006	ORDER-ID	PIC X(6)	
ORDR	ORDRDATE=0008	ORDER-DATE	PIC X(8)	
ORDR	ORDRSHDT=0008	ORDER-SHIP-DATE	PIC X(8)	
ORDR	ORDRAMNT=0009	ORDER-AMOUNT	PIC S9(7)V9(2) COMP	
ORDR	ORDRBRAN=0004	ORDER-BRANCH-ID	PIC X(4)	
ORDR	ORDRLKOD=0008	Linkpath to ORDT		
ORDR	ORDRCUST=0008	ORDER-CUST-ID	PIC X(8)	
ORDR	ORDRSHIP=0090	ORDER-SHIP-TO		
ORDR	ORDRADDR=0040	ORDER-SHIP-ADDRESS	PIC X(40)	
ORDR	ORDRCITY=0030	ORDER-SHIP-CITY	PIC X(30)	
ORDR	ORDRSTAT=0020	ORDER-SHIP-STATE	PIC X(02)	STATES
ORDR	ORDRZIPC=0050	ORDER-SHIP-ZIP-CODE	PIC X(05)	ZIP-CODES
ORDR	ORDRFONE=0013	ORDER-SHIP-PHONE	PIC X(13)	

◆ PDM related data set ORDT

File	Physical data item	Logical data item	Format	Domain or constant value
ORDT	ORDTORDR=0006	DETAIL-ORDER-ID	PIC X(6)	
ORDT	ORDRLKOD=0008	Linkpath to ORDR		
ORDT	ORDTITEM=0002	DETAIL-ITEM-NUMBER	PIC X(2)	
ORDT	ORDTPROD=0008	DETAIL-PRODUCT-ID	PIC X(8)	
ORDT	PRODLKOD=0008	Linkpath to PROD		
ORDT	ORDTQNTY=0009	DETAIL-ORDER-QUANTITY	PIC S9(9) COMP	

Base views in the conceptual schema

The conceptual schema is described by base views that access the data sets directly by using the logical data item names. In some cases, the base view replaces the logical data item name with an alternative column name (the base view REGION uses the column name REGION-NUMBER instead of the logical data item name REGION-NO). In other cases, the base view uses the logical data item name as the column name.

Base View: REGION

View Text:

```
KEY REGION-NUMBER = REGION-ID
   REGION-NAME
ACCESS REGN
   WHERE REGION-ID = REGION-NUMBER
   ALLOW ALL
* To restrict deletions of REGIONs that contain branches.
ACCESS BRAN
   WHERE BRANCH-REGION-ID == REGION-ID
```

Base View: BRANCH

View Text:

```
KEY BRANCH-NUMBER = BRANCH-ID
   BRANCH-NAME
REQ BRANCH-REGION == BRANCH-REGION-ID = REGION-ID
   BRANCH-ADDRESS
   BRANCH-CITY
   BRANCH-STATE
   BRANCH-ZIP-CODE
   BRANCH-SALES-QUOTA
   BRANCH-STAFF-QUOTA
   BRANCH-DELIVERY-ROUTE
ACCESS BRAN
   WHERE BRANCH-ID = BRANCH-NUMBER
   ALLOW ALL
* To verify that BRANCH-REGION contains a valid region on
* INSERT and UPDATE.
ACCESS REGN
   ONCE
   WHERE REGION-ID == BRANCH-REGION-ID
* To restrict deletions of branches containing customers.
ACCESS CUST
   WHERE CUSTOMER-BRANCH-ID = BRANCH-ID
* To restrict deletions of branches that have stock.
ACCESS STCK
   WHERE STOCK-BRANCH-ID = BRANCH-ID
```

Base View: PRODUCT**View Text:**

```

KEY PRODUCT-CODE = PRODUCT-ID
   PRODUCT-DESCRIPTION
   PRODUCT-WAREHOUSE-QUANTITY
   PRODUCT-PRICE
   PRODUCT-GROUP
ACCESS PROD
   USING PRODUCT-CODE
   ALLOW ALL

* To restrict deletions of products that are in stock.
ACCESS STCK
   ONCE
   VIA PRODLKST

* To restrict deletions of products that are currently ordered.
ACCESS ORDT
   ONCE
   VIA PRODLKOD

```

Base View: STOCK**View Text:**

```

KEY STOCK-BRANCH = STOCK-BRANCH-ID = BRANCH-ID
KEY STOCK-PRODUCT = STOCK-PRODUCT-ID = PRODUCT-ID
   STOCK-QUANTITY
   STOCK-BIN-LOCATION
   STOCK-YEAR-TO-DATE-SALES
ACCESS STCK
   WHERE STOCK-BRANCH-ID = STOCK-BRANCH AND
   STOCK-PRODUCT-ID = STOCK-PRODUCT
   ALLOW ALL

* To verify that a valid branch exists on INSERT and UPDATE.
ACCESS BRAN
   ONCE
   WHERE BRANCH-ID = STOCK-BRANCH-ID

* To verify that STOCK-PRODUCT contains a valid product code on
* INSERT and UPDATE.
ACCESS PROD
   ONCE
   WHERE PRODUCT-ID = STOCK-PRODUCT-ID

```

Base View: ORDER

View Text:

```
KEY   ORDER-NUMBER = ORDER-ID
REQ   ORDER-CUST-NUMBER = ORDER-CUST-ID = CUSTOMER-ID
      ORDER-DATE
      ORDER-SHIP-DATE
      ORDER-SHIP-STREET = ORDER-SHIP-ADDRESS
      ORDER-SHIP-CITY
      ORDER-SHIP-STATE
      ORDER-SHIP-ZIP-CODE
      ORDER-SHIP-PHONE
      ORDER-AMOUNT
REQ   ORDER-BRANCH = ORDER-BRANCH-ID = BRANCH-ID
ACCESS ORDR
      WHERE ORDER-ID = ORDER-NUMBER
      ALLOW ALL
* To verify that the customer number is valid on INSERT and
  UPDATE.
ACCESS CUST
      ONCE
      WHERE CUSTOMER-ID = ORDER-CUST-ID
* To verify that ORDER-BRANCH is a valid branch number on
* INSERT and UPDATE.
ACCESS BRAN
      ONCE
      WHERE BRANCH-ID = ORDER-BRANCH-ID
```

Base View: ORDER-DETAIL

View Text:

```
KEY   DETAIL-ORDER-NUMBER = DETAIL-ORDER-ID = ORDER-ID
KEY   DETAIL-PRODUCT-NUMBER = DETAIL-PRODUCT-ID = PRODUCT-ID
      DETAIL-ITEM-NUMBER
      DETAIL-ORDER-QUANTITY
ACCESS ORDT
      WHERE DETAIL-ORDER-ID = DETAIL-ORDER-NUMBER
      ALLOW ALL
* To verify that a valid order exists on INSERT and UPDATE.
ACCESS ORDR
      WHERE ORDER-ID = DETAIL-ORDER-ID
* To verify that a valid product exists on INSERT and UPDATE.
ACCESS PROD
      WHERE PRODUCT-ID = DETAIL-PRODUCT-ID
```

Base View: CUSTOMER

View Text:

```
KEY CUSTOMER-NUMBER = CUSTOMER-ID
REQ CUSTOMER-NAME
    CUSTOMER-CLASS
    CUSTOMER-ADDRESS
    CUSTOMER-CITY
    CUSTOMER-STATE
    CUSTOMER-ZIPCODE
    CUSTOMER-CREDIT-CODE
    CUSTOMER-CREDIT-LIMIT
REQ CUSTOMER-BRANCH = CUSTOMER-BRANCH-ID = BRANCH-ID
ACCESS CUST
    WHERE CUSTOMER-ID = CUSTOMER-NUMBER
    ALLOW ALL
* To verify that CUSTOMER-BRANCH contains a valid branch on
  INSERT
  * and UPDATE.
ACCESS BRAN
    ONCE
    WHERE BRANCH-ID = CUSTOMER-BRANCH-ID
```

Derived views in the external schema

Once you define your base views of the database, you can derive other views from them. The derived views can contain columns from one base view, or columns from several base views. They can allow the same update options as the base views they access, or they can restrict the update options. The following three examples show derived views of increasing complexity:

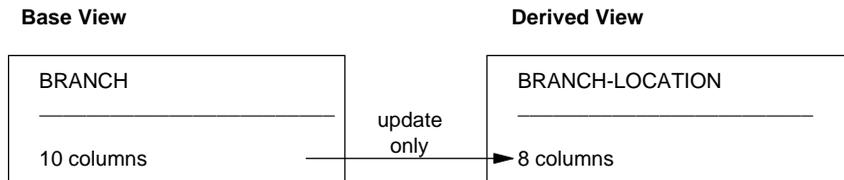
- ◆ This derived view is a subset of the BRANCH base view and excludes the BRANCH-SALES-QUOTA and BRANCH-STAFF-QUOTA columns.

Derived View: BRANCH-LOCATION

View Text:

```
KEY    BRANCH-NUMBER  
       BRANCH-NAME  
  
REQ    BRANCH-REGION  
       BRANCH-ADDRESS  
       BRANCH-CITY  
       BRANCH-STATE  
       BRANCH-ZIP-CODE  
       BRANCH-DELIVERY-ROUTE  
  
ACCESS BRANCH  
       USING BRANCH-NUMBER  
       ALLOW UPDATE
```

The figure below shows the base view and the number of columns it contains and the derived view and the number of columns it obtains from the base view. It also shows the update options specified for the derived view.



The derived view BRANCH-LOCATION could then be used by users who are not allowed to see the two quota columns. When defining this view, you do not have to enter all of the ACCESS statements that provide the integrity constraints, nor do you have to rewrite this view if the physical data set BRAN is broken apart or held in another physical file of a different name.

KEY identifies the logical key for the view. BRANCH-REGION does not have to be required in this view; however, if BRANCH-REGION is required, the base view will return and access only non-null, valid data for the column. Also, by specifying that it is required, you ensure that RDM can validate the required column in the derived view, thereby avoiding the need for RESETs to the database.

- ◆ You can design more complex derived views that access more than one base view. The derived view BRANCHES-IN-REGION accesses both the REGION and the BRANCH base views to produce a composite listing of branches within region.

Derived View: BRANCHES-IN-REGION

View Text:

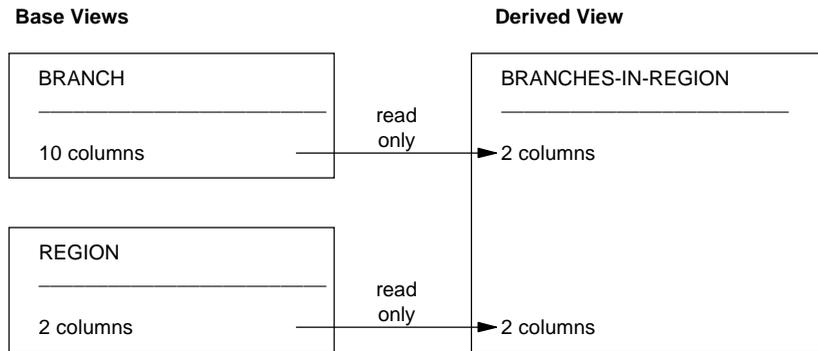
```

KEY    REGION-NUMBER
        REGION-NAME
KEY    BRANCH-NUMBER
        BRANCH-NAME
ACCESS REGION
        ONCE
        USING REGION-NUMBER
ACCESS BRANCH
        WHERE BRANCH-REGION = REGION-NUMBER
ALLOW INSERT UPDATE

```

To ensure data integrity, notice that you access BRANCH using a column, (BRANCH-REGION) that is not being used in BRANCHES-IN-REGION.

The figure below shows the names of the two base views accessed and the number of columns in each and the derived view and the number of columns it obtains from each base view. It also shows the update options specified for the derived view.



In addition to using two views to create a third derived view, the update options for the BRANCHES-IN-REGION derived views are changed. Even though REGION and BRANCH can be updated, the BRANCHES-IN-REGION view restricts access to read-only. When accessing a base view with a derived view, you can restrict view update capability but not extend it. For example, if the BRANCH base view does not have an ALLOW statement in its access definition (it was read-only), you would be unable to allow any update functions in any derived view that uses it.

- ◆ The PRODUCTS-IN-REGION derived view lists all products in stock in a specified region. The derived view accesses four base views for each row, and allows the user to perform different update functions on different base views. The following figure shows the base views and the number of columns in each and the number of columns that the derived view uses from each accessed base view. It also shows the update options specified for the derived view.

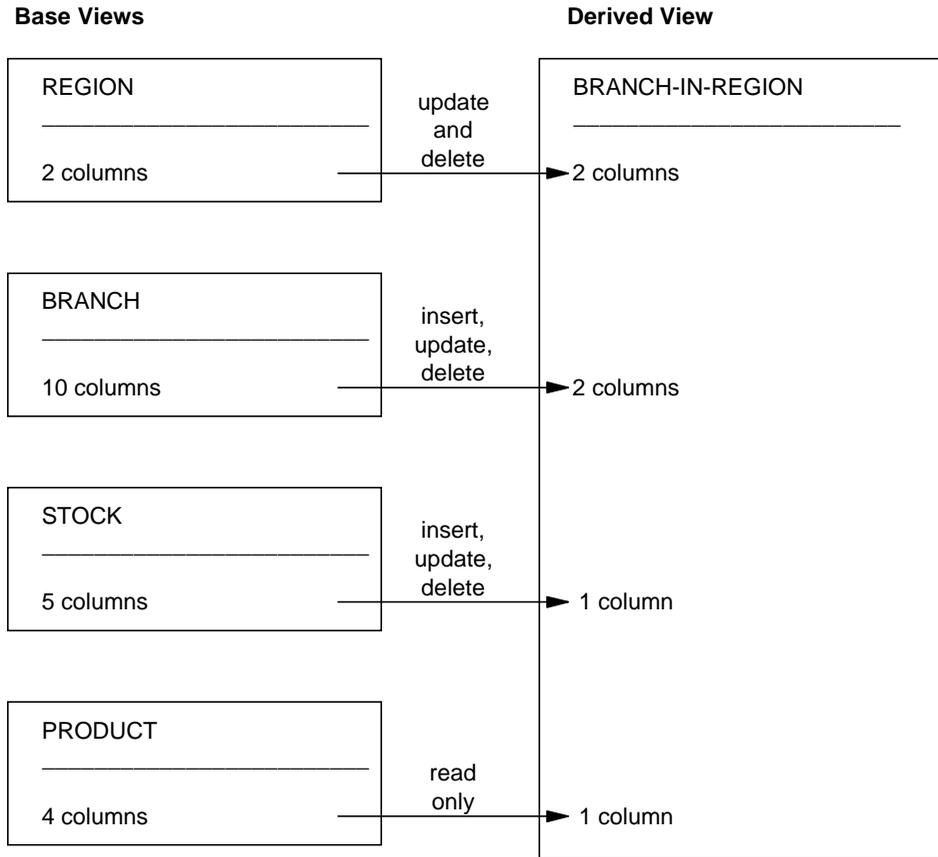
Derived View: PRODUCTS-IN-REGION

View Text:

```

KEY    REGION-NUMBER
        REGION-NAME
KEY    BRANCH-NUMBER
        BRANCH-NAME
KEY    PRODUCT-CODE = STOCK-PRODUCT = PRODUCT-CODE
        PRODUCT-DESCRIPTION
ACCESS REGION
        USING REGION-NUMBER
        ALLOW UPDATE DELETE
ACCESS BRANCH
        WHERE BRANCH-REGION = REGION-NUMBER
        ALLOW ALL
ACCESS STOCK
        WHERE STOCK-BRANCH = = BRANCH-NUMBER AND
        STOCK-PRODUCT = STOCK-PRODUCT
        ALLOW ALL
ACCESS PRODUCT
        WHERE PRODUCT-CODE = STOCK-PRODUCT

```



These views are used throughout this manual to illustrate RDM.

Index

*

* in binding a view 149

A

access

authority to use DBAID LIST
command 192

keyed and sequential 83

access definitions

base views 50

derived views 50

examples 69

location in view definition 50

order of access statements 50

parameters

ACCESS 54

ALLOW 66

data set name 54

FROM 56

GIVING 67

ONCE 57

ORDER 67

record-code 55

REVERSE 57

SCAN 57

USING 60

VIA 58

view name 55, 56, 57, 58, 59,
60, 62, 67

WHERE 64

syntax

generalized access for base
views 52

specific access for base
views 53

syntax for generalized access
for derived views 53

access methods

defining for views 51

run-time 51

access named global views 255
preventing 255

access paths used 207

Access portion of the view
definition 84

access statements
order 50

accessing the database See
Database penetration

ALL, using with DELETE 162

allow specified users access to
named global views 255

ALLOW, access definition
parameter

described 66

maximum 66

allowing data set updates 66

ASI 226

ASI. See Attribute Status
Indicators (ASI)

AT and USING 176

AT with MARK 193

Attribute Status Indicators (ASI)
226

authority for global views 247

authorize other users 128

automatic RDM RESET 166

B

base views

access definitions 50

defining direct reads 60

described 29

enforcing referential integrity 40

specifying logical data items 43
sweeping 40

batch global view creation 248

BATCH_GLOBAL_INPUT 248

bind a view

specified 149

using DBAID 237

blanks in character strings 185

bound views 231, 236

RDM access 51

rebinding 239

boundary condition 78

BYE command 152

- C**
- cascade delete 100, 120
 - CAUTIOUS command 153
 - opposite of SURE command 213
 - changes to database 131
 - changing view text 235
 - character and hexadecimal data
 - in USING clause of GET 176
 - character columns and range
 - checking 104
 - character data in GO command 180
 - circular navigation 75
 - clause
 - see parameter 58
 - close a specific view 201
 - Column Attribute Status Indicators (ASI) 226
 - column definitions
 - described 36
 - examples 69
 - for the user view named 218
 - order 48
 - parameters
 - column name* 42
 - CONST 41
 - constant value* 47
 - FKEY 40
 - KEY 38
 - logical data item name* 43
 - NONUNIQUE KEY 40
 - REQ 39
 - source column name* 45
 - position in view definition 48
 - syntax for 37
 - column descriptor
 - FIELD-DEFN 169
 - column name
 - COLUMN-TEXT 158
 - column names
 - defining for views 42
 - failing to specify 42
 - making meaningful 42
 - omitting on the GIVING parameter 67
 - column number
 - specifying in BY-LEVEL 150
 - COLUMN-DEFN command 154
 - COLUMN-DEFN DBAID 97
 - column-name
 - COLUMN-DEFN 154
 - columns
 - causing direct reads for physical keys 40
 - displaying description 167
 - maximum allowed as KEY 38
 - maximum allowed as NONUNIQUE KEY 38
 - modifying consideration 216
 - COLUMN-TEXT command 158
 - combining record codes 56
 - comments
 - displaying for a column 170
 - for column in view 158
 - in Access definitions 147
 - when editing a view (*) 146
 - COMMIT 159
 - after each successful insert, update or delete 213
 - and the SUPRA Server directory 153
 - disable auto COMMIT 153
 - compound keys
 - compound generic keys, wildcard position in 95
 - compound non-unique key 89
 - compound physical keys and logical keys 48
 - in the access definition, base views 60
 - in the access definition,RMS data sets 62
 - compound unique key 87
 - specifying in the access definition 59
 - conceptual schema
 - described 25
 - CONST parameter, column definition 41
 - constant keys 90
 - constant values
 - defining for a column 41
 - defining for views 47
 - null values 48
 - validity checking 48
 - COPY command 160
 - copy view definition to another view 160

creating a global view file
 described 243
 in batch 248
 interactively 245
 creating an input text file 249
 CSVGLOBAL 243, 248
 current line-size setting 190
 current page-size setting 198
 current statistics for open views
 210

D

data items not found 78
 data movement
 overriding 67
 data relationships
 many-to-one
 establishing using coded
 records 55
 one-to-many
 establishing using coded
 records 55
 establishing using SCAN 57
 one-to-one
 establishing using coded
 records 55
 establishing using ONCE 57
 data retrieval
 added selection criteria with
 WHERE 64
 defining in the access
 statement
 allowing physical actions 66
 defining a direct read 60
 ONCE 57
 reverse order 57
 SCAN 57
 efficient method 68
 many-to-one relationship 55
 one-to-many relationship 55
 one-to-one relationship 55
 ordering (ORDER) 67
 overriding normal data
 movement (GIVING) 67
 scanning 57
 using a linkpath 58
 using a secondary key 58

data sets 162
 defining for views 54
 navigation
 defining a data set for
 navigation only 67
 related to related 56
 data type
 of partial keys 95
 database changes, modifications
 required as a result of 131
 database name, specifying for
 batch global view execution
 248
 database penetration 75
 database sweep 77
 scanning 77
 database sweep 77
 database, how RDM signs on 19
 DBA report 262
 DBA to bind a view 236
 DBA unload-reload
 and ordered data sets 68
 DBA utility
 authorizing users 128
 DBAID
 commands 139
 creating the column definition
 36
 described 135
 help facility 139
 invoking 136
 maximum lines for a single view
 189
 password 138
 signing on 138
 sign-off 208
 sign-on 209
 test view
 considerations 43
 to bind a view 237
 treatment of line numbers 189

- DBAID commands 139
 - * 146
 - = 148
 - BIND 149, 269
 - BYE 152, 269
 - BY-LEVEL 150, 269
 - CAUTIOUS 153, 269
 - COLUMN TEXT 270
 - COLUMN-DEFN 154
 - COLUMN-TEXT 158
 - COLUMN DEFN 270
 - COMMIT 159, 270
 - COPY 160, 270
 - DEFINE 161, 270
 - DELETE 162
 - DELETE [ALL] 270
 - DENY 164, 270
 - EASE 270
 - EDIT 165, 270
 - ERASE 166
 - FIELD DEFN 270
 - FIELD TEXT 271
 - FIELD-DEFN 167
 - FIELD-TEXT 170
 - FORGET 172, 271
 - GET 173
 - GO 179, 271
 - HELP 271
 - INSERT 183, 272
 - KEEP 188, 272
 - line-number* 189
 - LINESIZE 190, 272
 - LIST 191, 272
 - MARK 193, 272
 - MARKS 194, 272
 - OPEN 195, 272
 - PAGESIZE 198, 272
 - PERMIT 199, 272
 - PRINT STATS 273
 - PRINT-STATS 200
 - programmer's subset 140
 - RELEASE 201, 273
 - REMOVE 202, 273
 - RENUMBER 203, 273
 - RESET 204, 273
 - SAVE 205, 273
 - SHOW NAVIGATION 273
 - SHOW-NAVIGATION 207
 - SIGN OFF 273
 - SIGN ON 273
 - SIGN-OFF 208
 - SIGN-ON 209
 - STATS 210, 274
 - STATS OFF 274
 - STATS ON 274
 - STATS-OFF 211
 - STATS-ON 212
 - substitute for last view name
 - used 269
 - SURE 213, 274
 - to denote a comment line 269
 - UNDEFINE 214, 274
 - UPDATE 215
 - UPDATE VIEW NAME 274
 - USER LIST 274
 - USER-LIST 218
 - VIEW DEFN 274
 - VIEW-DEFN 219
 - VIEWS 274
 - VIEWS FOR USER 274
- default validation 96
- default values 101
- default wildcard characters 93
- DEFINE command 161
- define new view to DBAID 161
- defining access definitions 50
 - parameters
 - ACCESS 54
 - ALLOW 66
 - data set name* 54
 - FROM 56
 - GIVING 67
 - ONCE 57
 - ORDER 67
 - record-code* 55
 - REVERSE 57
 - SCAN 57
 - USING 60
 - VIA 58
 - view name* 55, 56, 57, 58, 59, 60, 62, 67
 - WHERE 64
- syntax for
 - generalized access for base views 52
 - generalized access for derived views 53
 - specific access for base views 53

- defining column definitions 36
 - order of statements 48
 - parameters
 - column name* 42
 - CONST 41
 - constant value* 47
 - FKEY 40
 - KEY 38
 - logical data item name* 43
 - NONUNIQUE KEY 40
 - REQ 39
 - source column name* 45
 - position of in view definition 48
 - syntax for 37
- defining views
 - access definition 50
 - access definition parameters
 - ACCESS 54
 - ALLOW 66
 - data set name* 54
 - FROM 56
 - GIVING 67
 - ONCE 57
 - ORDER 67
 - record-code* 55
 - REVERSE 57
 - SCAN 57
 - USING 60
 - VIA 58
 - view name* 55, 56, 57, 58, 59, 60, 62, 67
 - WHERE 64
 - column definition
 - described 36
 - position 48
 - column definition parameters
 - column name* 42
 - CONST 41
 - constant value* 47
 - FKEY 40
 - KEY 38
 - logical data item name* 43
 - NONUNIQUE KEY 40
 - REQ 39
 - source column name* 45
 - examples 69
 - order of column definition
 - statements 48
 - syntax
 - access, generalized 52
 - access, generalized for
 - derived views 53
 - access, specific for base
 - views 53
 - syntax for
 - column definitions 37
 - tools for defining 35
- definitions 275
- delete
 - allowing 66
 - cascade 120
 - nullify 119
 - restrict 118
- DELETE 82
- delete bound version of a view 239
- DELETE command 162
- DELETE processing with null values 100
- deletion integrity 108, 116
- denote a comment when editing a view (*) 146
- DENY command 164
- derived views
 - and the ALLOW parameter 66
 - defining a logical read 60
 - described 29
 - processing 80
 - purpose of access definitions 50
 - specifying source column 46
- description of columns in view 154
- details of access paths used 207
- direct read
 - causing for physical key
 - columns 40
 - defining in the access
 - statement 60
- directory alone 260
- disable auto COMMIT 153
- disable automatic RESET 188

- display
 - column names in view by level of occurrence 150
 - comments for column in view 158
 - current statistics for open views 210
 - full description of columns in view 154
 - row for view 173
- display
 - condensed description of a view 219
- domain override 105
 - specifying 44, 65
- domain reports 262
- DOMAIN RETRIEVAL validation
 - enabled 97
- domain usage report 262

E

- EDIT command 165
- EDIT mode 165
- edit views 135
- efficient data retrieval 68
- enabling
 - DOMAIN RETRIEVAL 97
 - RU journaling 137
- END to stop mass inserting 186
- entering comments 147
- ERASE command 166
- errors during user authorization 199
- errors while denying user access to a view 164
- examples
 - global view report file 258
 - of view definition 69
- exit CSVDBAID image 152
- exit DBAID Test Facility 152
- exits 104
- external schema
 - described 26

F

- FIELD-DEFN command 167
- FIELD-DEFN output 169
- FIELD-TEXT command 170
- file access
 - to PDM files 17
 - to RMS files 17
- file types
 - supported by Relational Data Manager 17
- files
 - sweeping 40
- fixed value 90
- FKEY
 - column definition parameter 40
 - restriction 109
- forcing a generic read at PDM level 176
- foreign key 106
 - nullify 119
 - rules for defining 109
 - value integrity 108, 109
- FORGET command 172
- FROM
 - access definition parameter 56
 - in GO command 180
- FSI values 277
- FSI. See Function Status Indicators (FSI)
- function status indicator, values 225
- Function Status Indicators (FSI) 224

G

- generalized access syntax
 - base views 52
 - derived views 53
 - described 65
- generating reports 268
- generic keys
 - in GET 176
 - specifying in the access 59
- generic read 93
 - forcing 63
 - forcing at PDM level 176

GET 80
 considerations for using 115
 effect of required columns 39
 processing of with null values 99
 wildcard characters 95
 GET command 173
 GIVING, access definition
 parameter 67
 global view file 243, 245
 global view input file 249
 global views 232, 234
 and RDM access 51
 and the directory 260
 in preference to view definitions 242
 GO command 179
 group
 specifying in batch global input file 253

H

help facility
 DBAID 139

I

identify user to DBAID 209
 incremental movement 77
 index checking 162
 indexes
 specifying in the access definition 58
 initialize statistics to zero 212
 input text file 249
 INSERT 81
 effect of required columns 39
 INSERT command 183
 INSERT processing with null values 99
 insert records accessed through a secondary key 92
 insert rows
 longer than one line 184
 using the ORDER parameter 68
 insert, allowing 66
 insertion integrity 110
 INT prefix 151

integrity 108. *See also* referential integrity
 deletion 116
 insertion 110
 update 112
 internal schema
 described 24
 invalid values in KEY columns
 causing RDM to disregard 40
 invoking DBAID 136
 issue a GET request based on a single key 179
 issue RDM COMMIT request 159
 issue RDM DELETE request 162
 issuing a RESET 188

J

join compatibility 105

K

KEEP command 188
 KEY columns
 disregarding null occurrences in 40
 key value 87
 key values in UPDATE command 216
 KEY, column definition parameter 38
 keyed access 83
 keys 86
 partial values 93
 keys specified in GET 176
 keys. *See also* logical, Physical

L

length
 of constant values 47
 LEVEL 05 138
 level of occurrence 150
 lines to be displayed on a screen/page 198
 LINESIZE command 190
 linkpaths
 defining in the access statement 58
 list all open MARKs 194

- LIST command 191
 - with OPEN 196
- list views 135, 202
- lock out other users'
 - modifications 175
- logical data items
 - cross reference report 262
 - default for column definitions 43
 - defining for views 43
 - how RDM accesses multiple 44
 - mapping the same value to many 44
- logical keys 78
 - defining 84
 - number of 85
 - specifying more than one column as 48
- logical name GVSCHEMA 241
- logical views. *See* views
- logicals, for running
 - with directory alone 260
 - with no directory 259
 - with the directory and global views 260

M

- maintain uniqueness of physical keys 86
- MANTIS and SPECTRA support for nulls 100
- many-to-many relationship
 - establishing using coded-records 55
- mapping the same value to many logical data items 44
- mapping to a physical key 85
- MARK command 193
- mark the current position of a row 193
- MARKS command 194
- mass inserting 186
- MASS parameter in INSERT 184
- maximum
 - KEY columns in a view 38
 - NONUNIQUE KEY columns in a view 38
- maximum keys specified in GET 176
- maximum length for a default value 101
- maximum lines in DBAID for a single view 189
- meaningful names, assigning to columns 42
- modifying a view 165
- more than one user in a single PERMIT command 199
- multiple column names 42
- multiple logical data items
 - how RDM accesses 44
- multiple rows on a single line 184
- multiple users in a single PERMIT command 199

N

- navigation 75
 - defining a data set for navigation only 67
 - related data set to related data set 56
- network and RU journaling 137
- no validation 103
- NONUNIQUE KEY parameter,
 - column definition 40
- nonunique keys 88
 - combined with key columns 48
 - retrieval options 174
- NULL flag 98
- null values 98
 - and constant values 48
 - and DELETE processing 100
 - and GET processing 99
 - and INSERT processing 99
 - and required column 99
 - and UPDATE processing 99
 - defining for foreign keys 40
 - in KEY columns causing RDM to disregard 40
 - inserting 114
 - warning 114
- nullify delete 100, 119
- nullify foreign key 119
- number of characters to be displayed on a line 190
- number of lines to be displayed on a screen/page 198

O

- omit the key 83
- ONCE, access definition
 - parameter 57
- one-to-many relationship
 - establishing using coded-records 55
 - establishing using SCAN 57
- one-to-one keyed relationship 75
- one-to-one relationship
 - establishing 57
 - establishing using coded-records 55
- OPEN command 195
 - with LIST 196
- or operator
 - caution about using 56
 - described 55
- order
 - defining in the access statement 67
 - of access statements 50
 - of column definition statements 48
 - of keys in USING clause 176
 - of secondary keys 84
- overriding
 - domain checking 44
 - normal data movement 67
 - view access 164

P

- PAGESIZE command 198
- parameters
 - access definition
 - ACCESS 54
 - ALLOW 66
 - data retrieval 57
 - data set name* 54
 - FROM 56
 - GIVING 67
 - ONCE 57
 - ORDER 67
 - record-code* 55
 - related data sets 56
 - REVERSE 57
 - reverse order 57
 - SCAN 57

- USING 60
- VIA 58
- view name* 55, 56, 57, 58, 59, 60, 62, 67
- WHERE 64
- column definition
 - column name* 42
 - CONST 41
 - constant value* 47
 - FKEY 40
 - KEY 38
 - logical data item name* 43
 - NONUNIQUE KEY 40
 - REQ 39
 - source column name* 45
- partial key 95
- parts of a view 35
- password for DBAID 138
- PDM. See Physical Data Manager
- penetrating the database. See database penetration
- performance 84, 116
- PERMIT command 199
- physical actions, specifying for data sets 66
- physical changes to database 131
- physical data item cross reference report 262
- physical data items
 - using in the column definitions 43
- Physical Data Manager
 - data sets
 - how RDM accesses 17
 - physical key columns
 - causing direct reads for 40
 - physical keyed access 83, 84
 - position in the database 75
 - positional GET 75
 - positional modifier 179
 - positional relationship 75
 - prevent specified users from accessing named global views 255
 - print current statistics 211
 - PRINT-STATS command 200
 - programmer's subset
 - commands included in 140

- R**
- range checking 104
 - range values limit 104
 - RDM
 - access to the SUPRA Server directory 259
 - RDM access
 - strategy when specifying WHERE without USING 64
 - RDM COMMIT request 159
 - RDM DELETE request 162
 - RDM index checking 162
 - RDM reports 263
 - RDM RESET 166
 - RDM status indicators 230
 - RDM. See Relational Data Manager
 - RDML commands
 - effect of required columns 39
 - rebind a view 239, 240
 - record codes
 - and ordering 68
 - combining 56
 - defining in the access statement 55
 - recovery
 - for RMS data sets 17
 - recovery unit journaling, enabling 137
 - referential integrity 106
 - enforcing in base views 40
 - example 107
 - examples 121
 - RDM checking 108
 - rules 108
 - reissue previous RDML
 - command (=) 148
 - relate a view to user(s) on the Directory 199
 - related data set
 - defining in the access statement 56
 - navigation 56
 - relating views to users 128
 - Relational Data Manager
 - access methods and run-time 51
 - defining access method 51
 - described 15
 - file access to PDM and RDM data sets 17
 - file types supported 17
 - handling of view-open requests 21
 - reads of database 85
 - reports available 31
 - role in SUPRA Server system 18
 - role in three schema architecture 27
 - row construction 74
 - security 33
 - sign-on to the database 19
 - status indicators 226
 - relationships. See data relationships
 - RELEASE command 201
 - release occupied storage after closing a view 201
 - remove a view 202
 - REMOVE command 202
 - remove one or more row occurrences from the database 162
 - remove specified mark and resources from list of marks 172
 - remove the name and definition of a virtual view 214
 - RENUMBER command 203
 - replacements, allowing for rows 66
 - reports 97
 - description of 31
 - RDM 263
 - reposition a view 175
 - REQ parameter, column definition 39
 - required columns 85, 99
 - defining 39
 - defining a constant value 41
 - defining a non-unique key 40
 - defining unique logical key 38
 - parameters for specifying 48
 - RESET
 - automatic 166
 - disabling automatic 188
 - RESET command 204
 - reset statistical information 210

- restrict delete 118
 - restricting user access to global views 247
 - retrieve row for a view 173
 - retrieving data
 - in reverse order 57
 - reverse order
 - specifying 57
 - revoke user's privilege to use a view in SUPRA Server
 - Directory 164
 - RMS data sets 116, 162
 - access strategy when WHERE is not specified 64
 - defining RMS key for in access definitions 59
 - how RDM accesses 17
 - recovery 17
 - specifying direct reads 62
 - RMS files
 - enabling journaling 137
 - RMS keys
 - defining in the access statement 59
 - RMS, retrieving 174
 - roll back database updates 204
 - row occurrences in database, removing 162
 - RU journaling and network 137
 - run without a SUPRA Server
 - directory 259
 - run without a SUPRA Server
 - Directory 242
 - run-time
 - and RDM access methods 51
- S**
- SAVE command 205
 - save specified view 149
 - saved views in DBAID 191, 195
 - saving views 205
 - maximum length allowed 47
 - SCAN, access definition
 - parameter 57
 - scanning a data set 57
 - secondary keys 84
 - and partial keys 95
 - defined 92
 - defining in the access statement 58
 - reverse retrieval 57
 - USING parameter 58
 - security 33
 - overriding 164
 - sequential access 83
 - sequential search
 - defining for file or base view search 40
 - serial scan 83
 - shared columns 126
 - shared updates
 - allowing 66
 - consideration 66
 - SHOW-NAVIGATION command 207
 - sign user off from DBAID 208
 - signing on to DBAID 138
 - SIGN-OFF command 208
 - sign-on as another user during a DBAID session 209
 - SIGN-ON command 209
 - simple nonunique key 89
 - simple unique key 86
 - single quotes in UPDATE command 216
 - source columns
 - defining for views 45
 - source relation 106
 - specific access syntax 53
 - specify all views on the directory as global views 254
 - specify number of characters to be displayed on a line 190
 - specifying domain override 44
 - SPECTRA availability 239
 - START 180
 - starting DBAID 136
 - STATS command 210
 - STATS-OFF command 211
 - with PRINT-STATS 200
 - with STATS command 210
 - with STATS-ON command 212
 - STATS-ON command 212
 - with PRINT-STATS 200
 - with STATS command 210
 - with STATS-OFF command 211
 - status indicators 224
 - FSI values 277
 - VSI values 278
 - stop mass inserting 186

- storage
 - of global views 234
 - of views, releasing 201
- storage, freeing with the
 - UNDEFINE command 214
- SUCCESSFUL COMPLETION -
LEVEL 05 138
- SUPRA DBA
 - creating the column definition
with 36
 - difference from DBAID 35
- SUPRA Server directory
 - running without 259
 - view length allowed 47
- SUPRA Server Directory
and COMMIT 153
- SUPRA Server system
 - role of Relational Data Manager
18
 - three schema architecture 23
- SURE command 213
 - opposite of CAUTIOUS
command 213
- syntax
 - for access statements
 - generalized access for base
views 52
 - generalized access for
derived views 53
 - specific access for base
views 53
 - for column definitions 37

T

- table checking 104
- table entries
 - maximum 104
- table of values 104
- target relation 106
- test a view 240
- test view. *See* DBAID test view
- text file, input 249
- three schema architecture
 - conceptual schema 25
 - described 23
 - external schema 26
 - internal schema 24
 - role of RDM in 27
- tools for defining views 35

U

- UNDEFINE command 214
- unique constant 90
- unique key 86
 - retrieval options 174
- UPDATE 82
 - effect of required columns on
39
- UPDATE command 215
- update data values in the
database 215
- update integrity 112
- UPDATE processing with null
values 99
- update records accessed through
a secondary key 92
- user access
 - restricting 247
- user to view on Directory 199
- user to view relationship
removing 164
- user view column definition 218
- user views 30
- USER-LIST command 218
- users
 - relating views to 128
- USING with DBAID GET 176
- USING, access definition
parameter 58
- USING, in the GO command 180

V

- validation
 - checking 103
 - DOMAIN RETRIEVAL 97
 - exit 104
 - options 103
 - table 104
 - table usage report 262
- validity checking
 - and constant values 48
- Validity Status Indicators (VSI)
230
- VIA 58
- view definitions
 - examples 69
- view text 235
- view to user on Directory 199
- view, removing 202
- VIEW-DEFN command 219

VIEW-DEFN output 220

views

- access definitions 50
- access syntax
 - generalized syntax for base views 52
 - generalized syntax for derived views 53
 - specific syntax for base views 53
- column definitions 36
- creating with DBAID 36
- creating with SUPRA DBA 36
- defining
 - order of column definition statements 48
 - position of column definition 48
- defining access definition
 - parameters
 - ACCESS 54
 - ALLOW 66
 - data set name* 54
 - FROM 56
 - GIVING 67
 - ONCE 57
 - ORDER 67
 - record-code* 55
 - REVERSE 57
 - SCAN 57
 - USING 60
 - VIA 58
 - view name* 55, 56, 57, 58, 59, 60, 62, 67
 - WHERE 64
- defining column definition
 - parameters
 - column-name* 42
 - CONST 41
 - constant value* 47
 - FKEY 40
 - KEY 38
 - logical data item name* 43
 - NONUNIQUE KEY 40
 - REQ 39
 - source column name* 45

- defining column definition
 - syntax 37
- defining in the access
 - statement 55, 56, 57, 58, 59, 60, 62, 67
- described 28
- maximum number of lines in DBAID 189
- opening
 - how RDM handles 21
 - parts 35
 - relating users 128
 - repositioning 175
 - selecting for the global view file 246
 - sharing columns 126
 - types 29
 - user views 30
 - uses 29
- virtual views
 - displaying in DBAID 191
 - opening in DBAID 195
 - renumbering 203
 - text of 201
- VSI values 278
- VSI. See Validity Status Indicators (VSI)

W

- warning
 - null value 114
- WHERE, access definition
 - parameter
 - defining 64
 - specifying without USING in RMS data sets 64
- wildcard character
 - position of in key 95

