

Cincom

SUPRA SERVER PDM

RDM VSAM Support Supplement
(OS/390 & VSE)

P26-8222-62



SUPRA[®] Server PDM RDM VSAM Support Supplement (OS/390 & VSE)

Publication Number P26-8222-62

© 1987, 1990, 1993, 1998, 2000, 2002 Cincom Systems, Inc.
All rights reserved

This document contains unpublished, confidential, and proprietary information of Cincom.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

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

AD/Advantage [®]	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 [®]
 gOOj [™]	intelligent Document Solutions [™]	VisualWorks [®]
	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 VSAM Support Supplement (OS/390 & VSE)*, P26-8222-62, is dated January 15, 2002. This document supports Release 2.7 of SUPRA Server PDM in IBM mainframe environments.

We welcome your comments

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

Cincom Technical Support for SUPRA Server PDM

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

E-mail: helpna@cincom.com

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	vii
Using this document.....	vii
Document organization	vii
Revisions to this manual	vii
Conventions	viii
SUPRA Server documentation series	x
Accessing VSAM files with RDM	13
Prerequisites for KSDS access	13
Accessing VSAM files	14
Using VSAM with RDM	15
Describing VSAM files on the Directory	15
Adding file entities for VSAM files	16
Adding file entities for KSDS alternate indices	18
Adding physical fields for base files	19
Adding physical fields for alternate indices	20
Defining base views	21
Defining columns in base views	22
Defining ACCESS in base views	31
Format 1—Generalized ACCESS syntax	32
Format 2—Specific ACCESS syntax	32
Using the Relational Data Manipulation Language	39
Using the GET command.....	39
Using the DELETE command	40
Using the INSERT command.....	41
Using the UPDATE command	42
Using alternate indices	43
Alternate index considerations	43
Using generic key access	44

Environment-dependent considerations	45
Batch considerations	46
CICS considerations	47
User exits	49
Function parameter	51
Status parameter	52
Sample KSDS database and base views	53
Sample database	53
CUST file	54
PROD file	54
ORDR file	55
Example view definitions	56
Example view 1	56
Example view 2	56
Example view 3	57
Example view 4	57
Example view 5	58
Example view 6	58
Example view 7	59
Example view 8	59
Example view 9	60
Example view 10	60
Example view 11	61
Example view 12	61
Example view 13	61
Example view 14	62
Example view 15	62
Example view 16	63
Example view 17	63
Example view 18	64
Example view 19	65
Example view 20	66
Index	67

About this book

Using this document

The *SUPRA Server PDM RDM VSAM Support Supplement (OS/390 & VSE)*, P26-8222, provides the database administrator (DBA) with reference information on how to access VSAM key-sequenced data sets (KSDS) with the SUPRA Relational Data Manager (RDM). This manual also provides information on how to define a base view. In order to use this manual effectively, you should have knowledge of KSDS file structure and a working knowledge of RDM.

Document organization

The information in this manual is organized as follows:

Chapter 1—Introduction to accessing VSAM files with RDM

Provides an overview on accessing KSDS files with RDM.

Chapter 2—Using VSAM with RDM

Explains how to describe VSAM files on the Directory, how to define views for these files, and how to use different techniques to access these files.

Chapter 3—Environment-dependent considerations

Discusses batch and CICS considerations for RDM VSAM support.

Chapter 4—User exits

Describes the parameters passed to the user exits before and after accessing a VSAM file.

Appendix—Sample KSDS database and base views

Describes a sample KSDS database and shows some views which access the database.

Index

Revisions to this manual

There are no revisions to the content of this manual for this release.

Conventions

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

Convention	Description	Example
Constant width type	Represents screen images and segments of code.	<pre>PUT 'customer.dat' GET 'miller\customer.dat' PUT '\DEV\RMT0'</pre>
Slashed b (<i>b</i>)	Indicates a space (blank). The example indicates that four spaces appear between the keywords.	<pre>BEGNbbbSERIAL</pre>
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. The example indicates that you can optionally enter a WHERE clause.	<pre>[WHERE <i>search-condition</i>]</pre>
	Stacked items enclosed by brackets represent optional alternatives, one of which can be selected. The example indicates that you can optionally enter either WAIT or NOWAIT. (WAIT is underlined to signify that it is the default.)	<pre><u>(WAIT)</u> (NOWAIT)</pre>
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. The example indicates that you must enter ON or OFF when using the MONITOR statement.	<pre>MONITOR {ON OFF}</pre>

Convention	Description	Example
Underlining (In syntax)	<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>	<pre>[<u>WAIT</u>] [<u>NOWAIT</u>]</pre>
	<p>Underlining also indicates an allowable abbreviation or the shortest truncation allowed.</p> <p>The example indicates that you can enter either STAT or STATISTICS.</p>	<pre><u>STATISTICS</u></pre>
Ellipsis points...	<p>Indicate that the preceding item can be repeated.</p> <p>The example indicates that you can enter multiple host variables and associated indicator variables.</p>	<pre>INTO :host-variable [:ind- variable],...</pre>
UPPERCASE lowercase	<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>	<pre>COPY MY_DATA.SEQ HOLD_DATA.SEQ</pre>
<i>Italics</i>	<p>Indicate variables you replace with a value, a column name, a file name, and so on.</p> <p>The example indicates that you must substitute the name of a table.</p>	<pre>FROM <i>table-name</i></pre>
Punctuation marks	<p>Indicate required syntax that you must code exactly as presented.</p> <p>() parentheses . period , comma : colon ' ' single quotation marks</p>	<pre>(<i>user-id</i>, <i>password</i>, <i>db-name</i>) INFILE 'Cust.Memo' CONTROL LEN4</pre>
SMALL CAPS	<p>Represent a keystroke. Multiple keystrokes are hyphenated.</p>	<pre>ALT-TAB</pre>

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 (OS/390 & VSE)*, P26-9062.

Overview

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

Getting started

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

General use

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

Database administration tasks

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

Application programming tasks

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

Report tasks

- ◆ *SPECTRA User's Guide*, P26-9561



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



Educational material is available from your regional Cincom education department.

1

Accessing VSAM files with RDM

The Relational Data Manager (RDM) supports VSAM access to only key-sequenced data sets (KSDS). There are other types of VSAM files besides key-sequenced data sets; but in this manual a VSAM file is always a key-sequenced data set. You can access a VSAM key-sequenced data set with RDM using either the data set's prime key or any of its alternate keys. RDM does not require that a KSDS alternate key be unique. You need not change your existing VSAM data sets to access them through RDM. You must describe your VSAM data sets on the Directory to use RDM KSDS access.



There are two types of VSAM key-sequenced data sets that RDM can access: "native" VSAM key-sequenced data sets, and VSAM key-sequenced data sets that are SUPRA PDM (Physical Data Manager) files. This manual discusses only native VSAM key-sequenced data sets. For more information on RDM access to PDM files, regardless of whether they are VSAM, refer to the *SUPRA Server PDM RDM PDM Support Supplement (OS/390 & VSE)*, P26-8221.

RDM can access any combination of native VSAM key-sequenced data sets, and SUPRA Physical Data Manager (PDM) files.

Prerequisites for KSDS access

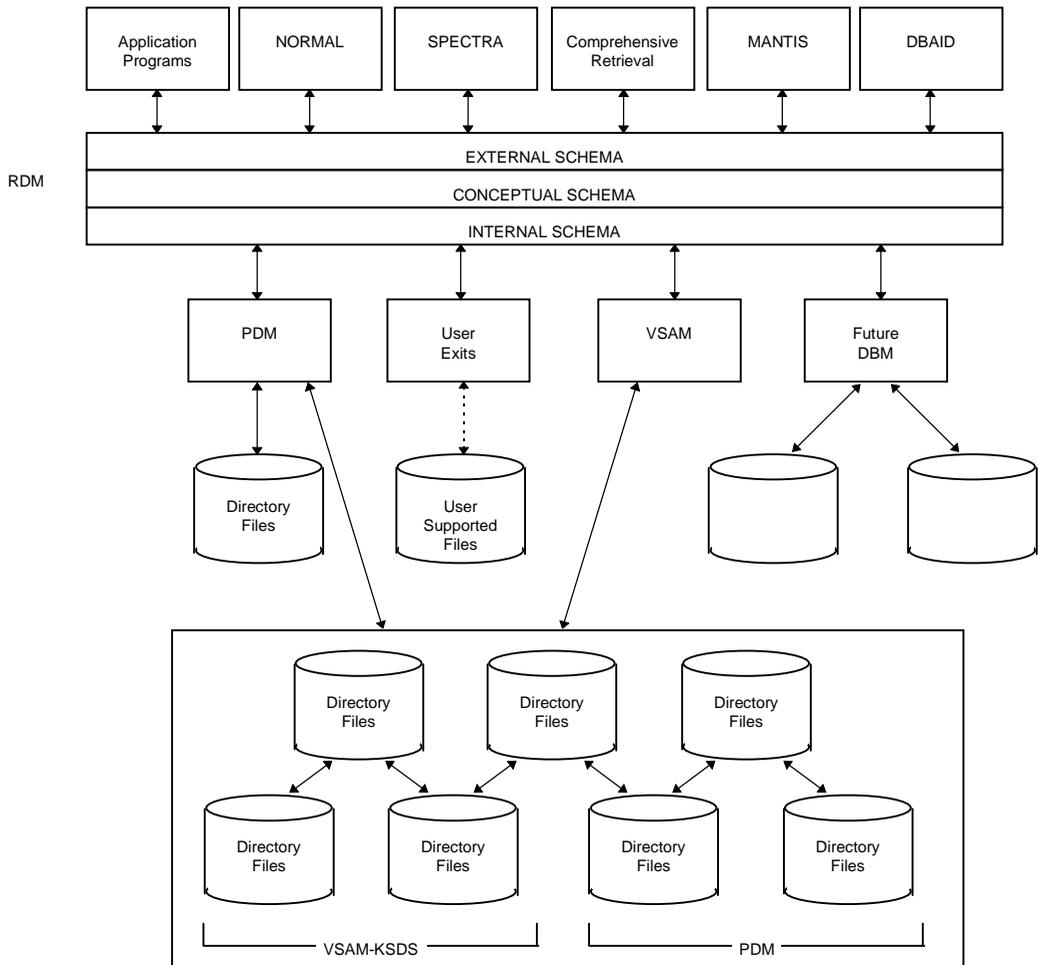
In order to use RDM KSDS access, your IBM operating environment must be OS/390 or VSE. You must have defined and created the VSAM data sets, including any alternate indices desired, using IBM's Access Method Services utility IDCAMS. For more information on Access Method Services, refer to the IBM *OS/390 DFSMS Access Method Services for Catalogs* or *VSE/VSAM Commands and Macros* manual. You must describe any VSAM base files and alternate index files you want RDM to use on the Directory. You must describe the layouts of the base files' data records on the Directory. "Using VSAM with RDM" on page 15 gives detailed instructions for adding these descriptions to the Directory. For more information on the Directory, refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250.

RDM VSAM support includes support for variable length records.

Accessing VSAM files

RDM applications can access native VSAM files in much the same way they access other files. Copy book libraries are unnecessary because the record formats of the VSAM files are stored on the Directory. With RDM, application programs and end users can manipulate and access data without any information about the data's physical location or structure.

The following figure shows how RDM and the conceptual schema insulate the external schema (which applications use) from the internal schema (which consists of the physical structure).



2

Using VSAM with RDM

This chapter explains how to describe VSAM files on the Directory, how to define views for these files, and how to use different techniques to access these files. This chapter also points out the performance considerations for various access techniques. These considerations reflect the physical structure of VSAM files, and you should take them into account in order to design efficient applications.

Describing VSAM files on the Directory

Before describing your VSAM files on the Directory, you must define and create your VSAM files, including alternate indices, using IBM's Access Method Services utility IDCAMS. For more information on IDCAMS, refer to the appropriate IBM VSAM manual.

You must describe your VSAM files on the Directory using either batch or online Directory maintenance:

- ◆ You must add one File entity for each file
- ◆ You must add physical fields for each file
- ◆ You may add external fields for a file
- ◆ You need not add internal record entities, because RDM uses only the "BASE." internal record (which you identify as your record code when creating Physical Fields in Directory Maintenance. Coded records are not supported.

This section gives examples of online Directory maintenance screens for adding File entities and physical field entities for VSAM files. For more information on using Directory maintenance, refer to the *SUPRA Server PDM Directory Online User's Guide (OS/390 & VSE)*, P26-1260, or the *SUPRA Server PDM Directory Batch User's Guide (OS/390 & VSE)*, P26-1261.

Adding file entities for VSAM files

To add a File entity using online Directory maintenance, specify the File category on the category menu screen. The next screen is the File command menu screen, where you specify the Add command. Also on the File command menu screen, specify the schema name and file name:

```
ENTER NAMING DATA:  
SCHEMA:                CUSTOMER  
FILE:                  ORDR
```

This file name is the logical file name you code in an ACCESS statement in a logical view that access this file.

The next screen is the first File Add screen, shown below:

```
FILE: ADD                TIS/XA DIRECTORY MAINTENANCE                1 OF 3  
SCHEMA: CUSTOMER                FILE: ORDR  
  
LAST UPDATE 12.00.00 08/01/2000 v: 0000 USER: CSI-DBA  
  
DDNAME;                ORDR  
DATA SET NAME:  
  
FILE TYPE:                OTHER  
FILE ACCESS METHOD:        KSDS  
FILE DEVICE TYPE:        3390  
FILE DEVICE ASSIGNMENT:  
LOGICAL RECORD LENGTH:    41  
TOTAL TRACKS:            0  
RECORDS PER BLOCK/CI:    0  
BLOCKS PER TRACK:        0  
VSAM CONTROL INTERVAL:    4096  
TOTAL VSAM CONTROL INTVL: 0  
  
COMMAND: F
```

You must specify the following values:

- ◆ DDNAME must be the name which identifies the file to the operating system or TP monitor. It is convenient to make this name the same as the logical file name, if possible; Directory maintenance assumes, by default, that they are the same. This name must be one of the following:
 - The file's actual DD name or DLBL name as specified in the JCL for executing batch applications (refer to the JCL reference manual for your operating system).
 - The file name as specified in the CICS File Control Table (refer to IBM's *CICS Resource Definition (Macro)*) or the File Resource definition in the CICS System Definition (CSD) file. (Refer to IBM's *CICS Resource Definition (Online)*).
- ◆ FILE TYPE must be OTHER.
- ◆ FILE ACCESS METHOD must be KSDS.
- ◆ FILE DEVICE TYPE must be a valid VSAM disk device, such as 3390. (This data is not used by RDM, but Directory maintenance requires it.)
- ◆ FILE DEVICE ASSIGNMENT is used in the VSE environment and must be a valid VSE device SYS number.
- ◆ LOGICAL RECORD LENGTH must agree with the record length defined in the VSAM catalog.
- ◆ VSAM CONTROL INTERVAL must be a number that is a valid VSAM control interval size and is compatible with the logical record length. (This number is not used by RDM, but Directory maintenance requires it.) Values you can use here include:
 - The file's actual control interval size.
 - A multiple of 2048 that exceeds the logical record length by at least seven.

Press ENTER and complete the screens that follow according to standard Directory Maintenance requirements.

Adding file entities for KSDS alternate indices

An alternate index file does not contain user data records; it exists only to provide an alternate set of key values for accessing a VSAM base file. This enables applications, including RDM, to access the base file as if it were a file with a different key but the same set of data records.

To use an alternate index, you must define a File entity on the Directory for the path connecting the alternate index and its base cluster. Everything in [“Adding file entities for VSAM files”](#) on page 16 about adding a File entity applies equally to base File entities and path file entities. Note, however, that you use the base logical file name in ACCESS statements, but never the name of the path file entity.

You must use a special naming convention on the Directory to indicate to RDM the precise relationship between a base file and its alternate indices: the alternate key physical field name on the base file is the concatenation of the base logical file name and the path logical file name. The following example illustrates the convention:

Example

RDM reads the Directory and finds that it includes two File entities whose access method is KSDS, whose file type is OTHER, and whose logical file names are ORDR and CUIX. RDM also finds that the file ORDR has a physical field ORDRCUIX. This indicates to RDM that:

- ◆ ORDR is a base VSAM file
- ◆ ORDCUIX is an alternate key to the file ORDR
- ◆ CUIX is the path for the file ORDR that corresponds to the alternate key ORDCUIX

Note that this naming convention depends on the logical file name, which is specified on the File command menu (as shown in [“Adding file entities for VSAM files”](#) on page 16), and which is not necessarily the same as the file’s DD or DLBL name.

Nothing in the File entities or the field entities on the Directory indicates to RDM whether an alternate key is unique or nonunique.

[“Using alternate indices”](#) on page 43 explains the use of alternate indices. [“Sample KSDS database and base views”](#) on page 53 describes in detail a KSDS data base including alternate indices, and shows views for accessing the data base.

Adding physical fields for base files

You must add physical fields to the Directory for the data you require. You must specify that the physical fields belong to the "BASE." internal record. You must add the physical field for the prime key using a special naming convention: The first four characters of the physical field name must be the logical file name, and the last four characters must be CTRL. The physical field name for the prime key for the file ORDR would be ORDRCTRL.

You must add a physical field for each alternate key, using the naming convention described in ["Adding file entities for KSDS alternate indices"](#) on page 18. The position and length of your base and alternate key fields on the Directory must agree exactly with the key fields as you defined them to VSAM using IDCAMS. You can define other fields according to any application conventions you define because the VSAM catalog contains no information about these fields.

The cumulative length of the Physical Field entities must match the record length for the file defined in the VSAM catalog. For variable-length records, this number must match the length of the longest logical record that occurs in the file.

The following figure shows the first of two physical field add screens. The second screen (not shown) contains validation options.

After you add the physical fields, you can relate the physical field names to external field names for ease of use. See “[Sample KSDS database and base views](#)” on page 53 for a sample data base with physical fields and external fields listed.

```
PHYFLD: ADD                                TIS/XA DIRECTORY MAINTENANCE      1 OF 2
SCHEMA: CUSTOMER                          FILE: ORDR
INTREC: BASE.                             PHYFLD: ORDRCTRL
LAST UPDATE 12.00.00 08/01/2000 v: 0000  USER: CSI-DBA

PARENT:                                     TOP.
POSITION:                                  END.
FUNCTION:                                  STRING
UNIT:
DATA FORMAT:                              C
PHYSICAL FIELD LENGTH:                    9
NUMBER DECIMAL PLACES:                    0
SIGNED OPTION:                            N
KEY REFERBACK:
LINKPATH TYPE:                            P
SEQUENCE FIELD:
SEQUENCE FIELD TYPE:
NULLS ALLOWED OPTION:                      N
NULL VALUE:

COMMAND: F
```

Adding physical fields for alternate indices

The Physical Field entity defines the physical and logical characteristics of a data base field for base and alternate index VSAM File entities.

Directory maintenance requires each File entity to have a logical record length of one or more. Therefore, each path File entity must have at least one physical field. However, because RDM does not use it, you may define it without any characteristics at all.

Defining base views

The Relational Data Manager (RDM) treats data as if it were arranged in tables (or “relations”). Each time an RDML command is issued, RDM uses a view to access a row in the table defined by that view.

You define a view (and therefore a table) with the view definition statements in DBAID; DBAID then stores the view definition on the Directory. (You can also define a view on to the Directory using Directory Maintenance; see the Directory Maintenance manuals for details.) A logical view is stored on the Directory in the LV category. The entity name of such an LV entity is the logical view name you use in the RDML commands you code in your applications and during DBAID sessions.

The examples in this manual include only views and RDML commands as you can use them in DBAID. For information on issuing RDML commands from application programs, refer to the *SUPRA Server PDM RDM COBOL Programming Guide (OS/390 & VSE)*, P26-8330, or the *SUPRA Server PDM RDM PL/1 Programming Guide (OS/390 & VSE)*, P26-8331.

A base view definition consists of two types of statements:

- ◆ **Column Definition.** Define the columns making up a row of the base view.
- ◆ **Access Definition.** ACCESS statements which define how the columns for the view are physically accessed. You enter ACCESS statements after the Column Definition statements

Defining columns in base views

The column definition, entered as part of the base view definition, defines each column to include in the view and each column's characteristics. You must define a column for each field included in a particular view.

KEY NONUNIQUEKEY REQ FKEY [[UNIQUE]CONST]	[<i>column - name</i> = [=]] <i>field - name</i>
[[=] = <i>field - name1</i> [... = <i>field - namen</i>]] [= <i>constant</i>]	

KEY

Description *Optional.* Indicates that this column is required in the view, is used as a logical key, and forms a unique key either by itself or when combined with other KEY fields.

Considerations

- ◆ This column is required. The General Considerations explain the effect of required columns on RDM's view processing.
- ◆ The maximum number of KEY and NONUNIQUE KEY columns which may be defined in a view is nine.
- ◆ If the column is a physical key, and you issue a GET USING command with a value for the column, RDM does a direct (or indexed) read.
- ◆ If the column is not a physical key, and you issue a GET USING command with a value for the column, RDM does a sequential search of the file for an equal condition on the column.

NONUNIQUE KEY

Description *Optional.* Indicates that this column is required in the view, is used as a logical key, and forms a nonunique key either by itself or when combined with other KEY and NONUNIQUE KEY fields.

Considerations

- ◆ If specified, this column is required. The General Considerations explain the effect of required columns on RDM's view processing.
- ◆ The maximum number of KEY and NONUNIQUE KEY columns which may be defined in a view is nine.
- ◆ If the column is a physical key, and you issue a GET USING command with a value for the column, RDM does a direct (or indexed) read.
- ◆ If the column is not a physical key, and you issue a GET USING command with a value for the column, RDM does a sequential search of files for an equal condition on the column.
- ◆ This option allows you to specify a value for the view key in the GET USING command without requiring a unique occurrence of the key column.

REQ

Description *Optional.* Indicates that this is a required field for this view.

Consideration If specified, this column is required. The General Considerations explain the effect of required columns on RDM's view processing.

FKEY

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

Consideration A foreign key column must be identified in the view either as a required field or as an FKEY field in order to enforce referential integrity. Refer to the *SUPRA Server PDM RDM Administration Guide (OS/390 & VSE)*, P26-8220, for more information about referential integrity.

[UNIQUE] CONST

Description *Optional.* Indicates that this column is required in the view, and that the value of the column must be equal to the given constant for RDM to GET the row. The default is nonunique unless you specify UNIQUE before CONST.

Considerations

- ◆ If specified, this column is required. The General Considerations explain the effect of required columns on RDM's view processing.
- ◆ The value of the constant is specified as part of the column data:
`CONST field-name = constant`
- ◆ All CONST columns are part of the logical key, but are not returned in the row.
- ◆ You must not use a CONST field after a USING clause in the RDML.
- ◆ Constants must be valid and non-null.

column-name = [=]

Description *Optional.* Assigns a name to the column which an application program can use.

Format 1–30 alphanumeric characters and the special characters #, -, _, and \$. The first character must be alphabetic or a special character. If the first character is a # or \$, the second character must be alphabetic.

Considerations

- ◆ This option allows you to assign a name that might be more descriptive and meaningful to the application.
- ◆ Column names need to be unique only within the view.
- ◆ If you do not specify a column name, applications use the field name (see below) to identify the field.
- ◆ If you specify multiple field names for a column, you must specify a column name. If you do not, RDM considers the first field name to be the column name.
- ◆ The field(s) must be from the same domain as the column unless an override is specified. To override the normal domain checking, include the optional extra equal sign as shown below:

```
REQ REGION = = BRANCH-REGION
```

field-name

Description *Required.* Indicates the name of the external field or physical field whose data is returned in the column.

Format Must conform to the naming conventions for external field and physical field names, as imposed by the Directory.

Considerations

- ◆ The external field or physical field definition must already exist on the Directory.
- ◆ If you code a physical field name, RDM assumes the first four letters of that name are the name of the file containing the physical field. If that assumption is false, then you must not code the physical field name. Instead, you must define an external field name for that field and use it for the field name.
- ◆ If the corresponding Physical Field entity on the Directory has a validation option of E, the corresponding validation exit entry point must exist.

****[=] = field-name1[...= field-namen]**

Description *Optional.* Specifies one or more external or physical fields, each of which corresponds to the same column in the view. These fields are called redundant fields.

Format Field names must be those of actual fields defined on the Directory.

Considerations

- ◆ If you specify multiple field names, you must specify a column name. RDM interprets the first identifier in a column definition for redundant fields as the column name.
- ◆ If you code a physical name, RDM assumes the first four letters of that name are the name of the file containing the physical field. If that assumption is false, then you must not code the physical field name. Instead, you must define an external field name for that field and use it as the field name.
- ◆ If the column has any of the attributes KEY, REQ, CONST, or NONUNIQUE KEY, then all of the fields named are required fields. The General Considerations explain the effect of required fields on RDM's view processing.
- ◆ If the column has any of the attributes KEY, CONST, or NONUNIQUE KEY, then when processing a GET, RDM ensures that all of the fields named have the same given value when a key is supplied. If the column has none of these attributes, the fields may have different values during GET processing, and the value returned to the user is that of the field most recently accessed.
- ◆ RDM supplies data from fields to columns in the order specified in the ACCESS statements, which is not necessarily the order on this statement.
- ◆ The fields must be from the same domain unless an override is specified. To override the normal domain checking, include the optional extra equal sign as shown below:

REQ REGION = = BRANCH-REGION = REGION-NO

= constant

Description *Optional.* Assigns a constant value to this column.

Format You may specify the value as either:

X'nnnnnn' Hexadecimal

nnnnnnnn Numeric (Binary, Packed, or Zoned)

'ccc' Character

Considerations

- ◆ The length of the value depends on the length of the field being defined.
- ◆ You must assign a constant value to a CONST column.

General considerations

- ◆ Any columns defined as REQ, KEY, CONST, or NONUNIQUE KEY are required columns for the view.
- ◆ Required columns restrict the number of occurrences in the view. A valid row must have an occurrence of the required column's physical record and a valid non-null value in the field supplying data to the column.
- ◆ Required columns affect the operation of RDM, depending on the command, in the following ways:
 - On a direct GET, RDM takes the NOT FOUND option unless all fields supplying data to required columns are present, valid and non-null.
 - On a sweeping GET, RDM skips the physical record unless all required fields are present, valid and non-null.
 - On any GET, a required column in the base view need not be defined in a derived view accessing that base view.
 - On an INSERT or an UPDATE, RDM returns an error unless all required columns are present, valid and non-null.
 - On a DELETE, required columns have no effect. Every DELETE must be preceded by a GET, where required columns have the effects described above.
- ◆ If a view contains several key columns and at least one NONUNIQUE KEY column, RDM processes the view with a nonunique logical key.
- ◆ At the physical level, more than one data element in a record can be a view key; however, the keys are treated as a compound key for that physical record.
- ◆ All column definition statements must precede the ACCESS statement(s).
- ◆ The order of the column definition statements does not affect the order of physical record processing.
- ◆ The constant in a CONST column definition cannot be a null pattern.
- ◆ The constant must pass the validity checking if the column has validation criteria associated with it.

Examples

- ◆ The Column Definition for this view indicates a customer-product row which may have multiple product values for each customer.

```

100 KEY                CUSTOMER-NUMBER = CUST-NO
200 NONUNIQUE KEY     PRODUCT-NUMBER = ORDR-PROD-NO
300                   DESCRIPTION = PROD-DESCRIPTION
400                   DATE-OF-SALE = ORDR-SALE-DATE
500                   SALE-AMOUNT = ORDR-SALE-AMOUNT
600                   FULL-PRICE = PROD-PRICE

```

- ◆ This view returns data about employees who have a DURATION-IN-DEPARTMENT of 0. This indicates that they are currently in the department.

```

100 KEY                EMPLOYEE-NUMBER
200 CONST              DURATION-IN-DEPARTMENT=0
300                   NAME=EMPLOYEE-NAME
400                   SKILL-CODE=SKILL-CODE-FOR-EMPLOYEE
500                   DESCRIPTION=SKILL-DESCRIPTION

```

- ◆ This example shows the usage of multiple field names.

```
100 CUSTOMER-NUMBER = CUST-NO = AUDIT-CUST-NO
```

- With a GET, the value returned in CUSTOMER-NUMBER depends on which field (CUST-NO or AUDIT-CUST-NO) is accessed last. RDM does not guarantee that these two values are equal in this case.
- An INSERT of a value into CUSTOMER-NUMBER results in the same value being inserted into CUST-NO and AUDIT-CUST-NO on the physical data base.
- With an UPDATE, a change in CUSTOMER-NUMBER updates both CUST-NO and AUDIT-CUST-NO.

- ◆ This example shows multiple column names as a key.

```
100 KEY CUSTOMER-NUMBER = CUST-NO = AUDIT-CUST-NO = SALES-CUST-NO
```

RDM treats all three fields as keys:

- With a GET, you retrieve only those records that have CUST-NO, AUDIT-CUST-NO, and SALES-CUST-NO equal to the value given for CUSTOMER-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 in the preceding example.
- An INSERT of a value into CUSTOMER-NUMBER causes the same value to be inserted into all three fields (CUST-NO, AUDIT-CUST-NO, and SALES-CUST-NO).
- With an UPDATE, a change in CUSTOMER-NUMBER updates CUST-NO, AUDIT-CUST-NO, and SALES-CUST-NO.
- CUSTOMER-NUMBER is not treated as a field name. If you intended to define four redundant fields in this column, this statement would be an error that RDM cannot detect. However, you can recognize and correct such an error, for example, by coding a column definition as follows:

```
100 KEY      CUSTOMER-NUMBER-COLUMN = CUSTOMER-NUMBER
                                     = CUST-NO
                                     = AUDIT-CUST-NO
                                     = SALES-CUST-NO
```

- ◆ In this example, both fields (CUST-NO and AUDIT-CUST-NO) are set to five on all functions:

```
100 CONST CUSTOMER-NUMBER = CUST-NO = AUDIT-CUST-NO = 5
```

Defining ACCESS in base views

You can use two types of ACCESS statements to retrieve data from KSDSs:

- ◆ **Generalized Access Syntax (Format 1).** Uses the WHERE clause and RDM determines the access strategy.
- ◆ **Specific Access Syntax (Format 2).** Uses the USING and VIA clauses to specify which key to use when accessing the file.

If you use the Generalized Access Syntax, RDM selects an access strategy when the view is opened. RDM determines whether the data should be accessed by using the file's prime key, by using an alternate key, or by scanning the file.

When you use the Generalized Access Syntax, you need to be aware of the possibility that RDM is scanning the entire file, even though this might not be what you intend or expect. RDM statistics were designed to provide you with information about the operation of your views; they can help you detect this sort of situation. The DBA user command SHOW-NAVIGATION, issued after the view OPEN, displays the ACCESS strategy selected for the view.

Format 1—Generalized ACCESS syntax

ACCESS *filename*

WHERE *field - 1* [=] *value* [... **AND** *field - n* [=] = *value*]

[**ONCE**]

[**ALLOW** [**SHARED**][**ALL**][**INSERT**][**DELETE**][**UPDATE**]
[**REP**]]

[**GIVING** *column1* [... *columnn*]]

Format 2—Specific ACCESS syntax

ACCESS *filename*

WHERE *field - 1* [=] *value* [... **AND** *field - n* [=] = *value*]

[**ONCE**]

[**VIA** { *external - field - name*
physical - field - name }]

[**USING** *key - field*]

[**ALLOW** [**SHARED**][**ALL**][**INSERT**][**DELETE**][**UPDATE**]
[**REP**]]

[**GIVING** *column1* [... *columnn*]]

Descriptions for each element of these formats follow. The elements are described in the order in which they appear in Format 2 above.

ACCESS

Description *Required.* Identifies the statement as a navigation definition for the base view.

filename

Description *Required.* Identifies the file to be accessed.

Format 4 alphanumeric characters and/or the special characters @, #, \$, or -. The first character must be alphabetic, #, \$, or @.

Consideration The file definition must already exist on the Directory. Code the same name as the relevant FI entity on the Directory.

WHERE *field-1* = [=] *value* [... AND *field-n* [=] = *value*]

Description *Optional.* The WHERE clause, used in conjunction with the USING and/or VIA clauses, provides additional selection criteria. If you use the WHERE clause without the USING clause, RDM selects the optimum access strategy in the following manner. If the fields include the prime key of the file to be accessed, or the leftmost portion of the prime key (a generic key), RDM uses the prime key. If the fields include an alternate key for the file, RDM uses that alternate key. If RDM can not use either the prime key or an alternate key, RDM scans the file.

Format	WHERE	Specified as shown.
	<i>field-1</i>	Specifies the target of data for this search criterion. Field-1 must be a field in the file. It may be an external field or a physical field.
	=[=]	Separates field-1 from value. If both fields have domains, the domains must be the same unless you override this restriction by specifying two equal signs. For example: ACCESS E\$CU WHERE CUST-NAME = = CUST-NO
	<i>value</i>	Specifies the source of the data for this search criterion. It can be a physical field or external field from a preceding ACCESS statement in this view, or a column name from this view or a constant. Constant values must be valid and non-null.
	AND	<i>Optional.</i> Allows specification of additional qualifications for the ACCESS.

Considerations

- ◆ Except for base files (the first file accessed in the view), you must use either a USING or WHERE clause.
- ◆ If you wish to control which alternate index is selected, use the VIA clause.
- ◆ Use RDM statistics to measure the performance of the view when using the WHERE clause without the USING clause.
- ◆ Column values used in a join must be valid and non-null. That is, to be considered equal they must be equal, valid, and non-null.
- ◆ If *field-1* (or-*n*) names a physical key, performance improves.

ONCE

Description *Optional.* Establishes a one-to-one relationship between the previous record accessed and the currently accessed record.

Consideration You may use the ONCE clause with the WHERE, USING, and/or VIA clauses.

VIA $\left[\begin{array}{l} \textit{external - field - name} \\ \textit{physical - field - name} \end{array} \right]$

Description *Optional.* Specifies the alternate key field used to access the KSDS file specified in the file name field.

Format The name specified must be an external field or physical field for an alternate key described on the Directory.

Considerations

- ◆ You may use the VIA clause with the USING clause. The VIA clause must immediately precede the USING clause.
- ◆ You may use the VIA clause on the first ACCESS statement in a view.
- ◆ RDM does not perform domain checking when you use the VIA clause.
- ◆ The field in the VIA clause must be defined as an alternate key in the VSAM catalog and on the Directory.

USING key-field

Description *Optional.* Indicates that a keyed read is to be done on the KSDS file using the specified key field as the physical key.

Format The key field may be a physical field, external field, column, constant, or it may be constructed at run time from multiple fields and constants by use of parentheses. For example:

```
ACCESS ORDR USING (CUST-NO,'X100')
```

Considerations

- ◆ You can use the USING clause with the VIA clause for KSDS files. In that case, the key field is the alternate key rather than the prime key. The VIA clause must immediately precede the USING clause.
- ◆ If you use the USING clause without the VIA clause, the field in the USING clause is the prime key.
- ◆ You can establish one-to-many relationships within KSDS files through RDM. In the following example, a single key value from the CUST file returns several records from the ORDR file because the full prime key of ORDR is the concatenation of CUST-NO and ORDR-PROD-NUM.

```
KEY CUST-NO
    CUST-NAME
    ORDR-PROD-NUM
    ORDR-CUST-PO-NO
ACCESS CUST USING CUST-NO
ACCESS ORDR USING CUST-NO
```

For any given CUST-NO value, several records may be returned from the ORDR file, while only one is returned from the CUST file.

- ◆ You can construct compound physical keys from parts of the physical key as defined in the Directory. Each part of the compound physical key must map to the subdefined part of the key as specified in the Directory. You must use the parenthesis in the USING clause to indicate the use of the subdefinition of the control key field.

If the Directory definition of the physical key field was

```
01  ORDRCTRL  9
02  ORDRCUIX  05
02  ORDRPRIX  04
03  ORDRPRD1  01
03  ORDRPRD2  03
```

you could specify the following compound key definitions:

```
USING  ORDRCTRL
USING  (ORDRCUIX, ORDRPRIX)
USING  (ORDRCUIX (ORDRPRD1, ORDRPRD2))
```

- ◆ You may force generic key access by leaving fields out of the USING clause for a compound key. The partial key (generic key) that remains must be contiguous and must include the leftmost character of the full key. The field names you supply must be the same length as the fields in the sub-definition of the physical key. Using the previous example, a compound key could be specified as follows:

```
USING  (ORDRCUIX)
```

- ◆ You can use generic key access from within RDM based on the generic keys supplied in the USING clauses or WHERE clauses in ACCESS statements; see [“Using generic key access”](#) on page 44 and [“Sample KSDS database and base views”](#) on page 53 for examples. Application programmers cannot specify the number of characters in any field used for a generic read. You must enclose the generic key in parentheses in the USING clause.
- ◆ RDM does not perform domain compatibility checking when you specify USING. You must specify the WHERE clause for domain checking to be performed.
- ◆ Constant values must be valid and non-null.

ALLOW [SHARED][ALL][INSERT][DELETE][

UPDATE
REP

]

Description	<i>Optional.</i> Specifies what physical actions are to be allowed for the specified file.
Format	Any combination is valid, for example: ALLOW INS DEL Allows inserts and deletes but not updates. ALLOW UPD Allows updates but neither inserts nor deletes.
Options	<p>SHARED Allows columns to be shared between views.</p> <p>ALL Allows all three forms of data base modifications.</p> <p>INSERT Allows inserts on the data base.</p> <p>DELETE Allows deletes from the data base.</p> <p>UPDATE / REP Allows updates or record replacements on the database.</p>

Considerations

- ◆ You may place the ALLOW phrase on as many ACCESS statements as are required to properly maintain the view.
- ◆ If you omit this clause, the file is accessed for read-only processing.
- ◆ These options relate to physical I/O on the file, not to the application program's RDML.
- ◆ Use of SHARED causes RDM to skip the field comparison processing normally performed by the automatic record holding facility before you delete or replace records. This field comparison detects and reports changed field values. Use of SHARED has no impact on INSERT.

GIVING column1...[columnn]

Description *Optional.* Overrides the normal physical field to column data movement.

Format The keyword GIVING followed by zero or more column names as defined on the column definition.

Considerations

- ◆ The GIVING clause allows you to access a file more than once and retrieve selected physical fields for each access. If several ACCESS statements use the same file, you can specify which columns to use for each access of the file.
- ◆ If you omit column names on the GIVING clause, RDM uses the file for navigation only.
- ◆ If you omit this clause, all columns derived from physical fields in the file and not supplied by some previous ACCESS statement are filled with values using this ACCESS statement. Therefore, if all columns supplied by fields of a file are filled from preceding ACCESS statements, RDM can optimize its use of a view by removing the entire ACCESS statement from the view.

Using the Relational Data Manipulation Language

RDM VSAM supports the GET, DELETE, UPDATE, and INSERT Relational Data Manipulation Language (RDML) commands. The following sections discuss specific considerations as they apply to RDM VSAM support. For additional information on RDML, refer to the appropriate language-oriented programming guide (see Associated Documentation in the front of this manual).

Using the GET command

The GET command retrieves data from the data base. RDM VSAM does not support GET LAST and GET PRIOR. The following GET command retrieves data from the ORDR file using the full physical key:

```
GET FIRST ORDER-VIEW USING 00225X100
```

The following views retrieve data from the ORDR file. The only difference between the two views is that the first view uses the WHERE clause, and the second view uses the USING clause on the ACCESS statement. The two views are equivalent.

```
KEY  ORDR-CTRL
      ORDR-INSTALL-DATE
      ORDR-MAINT-DATE
ACCESS ORDR WHERE ORDR-CTRL = ORDR-CTRL
```

```
KEY  ORDR-CTRL
      ORDR-INSTALL-DATE
      ORDR-MAINT-DATE
ACCESS ORDR USING ORDR-CTRL
```

Using the DELETE command

RDM does not delete records that depend upon each other. This feature ensures the integrity of your data. If you want to delete records from your KSDS data base, use the GET command to retrieve the record you want deleted. Then issue a DELETE command on the record as follows:

```
DELETE CUST-ORDER-VIEW
```

Use the following view with the DELETE command to delete records from the CUST and ORDR files. When a DELETE command is issued, RDM deletes the record in the ORDR file. A check is then made for an ORDR record which has the same CUST-NO value as the record in the CUST file. If no other records exist, then the CUST record is also deleted. The deletion of records in the CUST file is also prohibited if the ALLOW clause does not specify DELETE or ALL. Assume the key field CUST-NO is only part of the key to ORDR, making it a generic or partial key to ORDR, which makes ORDR a one-to-many child of CUST.

```
NAME: CUST-ORDER-VIEW
```

```
KEY CUST-NO
```

```
ACCESS CUST USING CUST-NO ALLOW DELETE INSERT
```

```
ACCESS ORDR CUST-NO ALLOW ALL
```

This same view coded with the WHERE clause on the ACCESS statement instead of the USING clause looks like this:

```
KEY CUST-NO
```

```
ACCESS CUST WHERE CUST-NO = CUST-NO
```

```
ALLOW DELETE INSERT
```

```
ACCESS ORDR WHERE ORDR-CUST-NUM = CUST-NO
```

```
ALLOW ALL
```

Using the INSERT command

The INSERT command inserts a record into your KSDS data base. For example:

```
INSERT CUST-ORDER-PROD-VIEW
```

Use the following view with the INSERT command to insert a record into the data base. An insert is attempted on each file. If a record is inserted into at least one file, the operation was successful.

```
NAME: CUST-ORDER-PROD-VIEW
```

```
KEY CUST-NO
```

```
    CUST-NAME
```

```
KEY ORDR-PROD-NUM
```

```
    PROD-DESC
```

```
ACCESS CUST USING CUST-NO ALLOW ALL
```

```
ACCESS ORDR USING (CUST-NO, ORDR-PROD-NUM) ALLOW ALL
```

```
ACCESS PROD USING ORDR-PROD-NUM ALLOW ALL
```

If you want to prohibit an insert on a particular view, do not code INSERT or ALL on the ALLOW clause. For example, suppose you do not want a product inserted when processing an order on the ORDR file if that product does not already exist in the PROD file. By not coding INSERT or ALL on the PROD file ALLOW clause, RDM first checks the PROD file before inserting a record in the ORDR file. If the PROD record does not exist, the ORDR record is not inserted.

```
NAME: CUST-ORDER-VIEW
```

```
KEY CUST-NO
```

```
    CUST-NAME
```

```
KEY PRODUCT-NUMBER=ORDR-PROD-NUM=PROD-NUMB
```

```
ACCESS CUST USING CUST-NO ALLOW ALL
```

```
ACCESS ORDR USING (CUST-NO, ORDR-PROD-NUM) ALLOW ALL
```

```
ACCESS PROD USING ORDR-PROD-NUM ALLOW DEL
```

The following view produces the same results as the above view except that it uses the WHERE clause to allow RDM to determine the access strategy instead of the USING clause.

```

KEY CUST-NO
  CUST-NAME
KEY PRODUCT-NUMBER=ORDR-PROD-NUM=PROD-NUMB
ACCESS CUST WHERE CUST-NO = CUST-NO ALLOW ALL
ACCESS ORDR WHERE ORDR-CUST-NUM = CUST-NO
  AND ORDR-PROD-NUM = ORDR-PROD-NUM
  ALLOW ALL
ACCESS PROD WHERE PROD-NUMB = ORDR-PROD-NUM
  ALLOW DEL

```

Using the UPDATE command

RDM updates column information and relationships between files within your data base. If you want to update records within your KSDS data base, use the GET command to retrieve the record you want to update, then issue an UPDATE command on the record as follows:

```
UPDATE DATE-OF-SALE-VIEW
```

You can use the following view with the UPDATE command to enter customer, product, and subsidiary customer information within your KSDS data base.

```

NAME:  DATE-OF-SALE-VIEW
KEY ORDR-CUST-NUM
KEY ORDR-PROD-NUM
ORDR-SUB-CUST-NO
ACCESS ORDR USING (ORDR-CUST-NUM, ORDR-PROD-NUM) ALLOW ALL
ACCESS CUST USING ORDR-SUB-CUST-NO ALLOW ALL

```

If you want to use the generalized form of the ACCESS statement and have RDM determine the access strategy, code the following two ACCESS statements instead of the previous two:

```

ACCESS ORDR WHERE ORDR-CUST-NUM = ORDR-CUST-NUM
  AND ORDR-PROD-NUM = ORDR-PROD-NUM
  ALLOW ALL
ACCESS CUST WHERE CUST-NO = ORDR-SUB-CUST-NO
  ALLOW ALL

```

When updating a physical field that is a foreign key, RDM automatically maintains the secondary KSDS file. For example, if you change the value for ORDR-SUB-CUST-NO in the above view, RDM attempts to delete the old value and then insert a new value.

Using alternate indices

You can use an alternate key to retrieve, update, and maintain records in VSAM base files.

See “[Adding file entities for KSDS alternate indices](#)” on page 18 for information about describing your alternate indices on the Directory.

Alternate index considerations

- ◆ If you are using an alternate index for updating, we recommend you use SHAREOPTIONS 3 or 4 for that KSDS file. For more information on SHAREOPTIONS, see the *IBM OS/VS2 Access Method Services* or *VSE/VSAM Commands and Macros* manual.
- ◆ RDM supports nonunique indices under batch, but not under CICS.
- ◆ You should not have two views using the same alternate index for the same task at the same time or the results are unpredictable.
- ◆ We recommend using RDM's referential integrity features to maintain your data's integrity. Refer to the *SUPRA Server PDM RDM Administration Guide (OS/390 & VSE)*, P26-8220, for more information about referential integrity.

Using generic key access

To use generic key access, supply only part of the physical key in the view or in the application. The part you supply (the generic key) must be contiguous and must include the left-most character of the physical key. For example, in the next view, the physical key of the ORDR file consists of two sub-parts. RDM performs a generic key access when the application program supplies a value for only CUST-NUM on the GET. RDM returns all products for the requested customer when the value for CUST-NUM is supplied.

```
NAME: ORDER-VIEW

KEY CUST-NUM = ORDR-CUST-NUM
KEY ORDR-PROD-NUM
   ORDR-CONTACT-NO
ACCESS ORDR USING (CUST-NUM, ORDR-PROD-NUM)
```

A generic retrieval from this view would look like this:

```
GET ORDER-VIEW USING CUST-NUM
```

The following view is the same as the above view except it uses the WHERE clause instead of the USING clause to specify the two sub-parts:

```
KEY CUST-NUM = ORDR-CUST-NUM
KEY ORDR-PROD-NUM
   ORDR-CONTACT-NO
ACCESS ORDR WHERE ORDR-CUST-NUM = CUST-NUM
   AND ORDR-PROD-NUM = ORDR-PROD-NUM
```

The next view provides the same results as the previous views, except ORDR-PROD-NUM is left out of the USING clause to force a generic read on the ORDR file. You can do generic key access from within RDM based on the generic key supplied in the ACCESS statement's USING clause.

```
KEY CUST-NUM = ORDR-CUST-NUM
KEY ORDR-PROD-NUM
   ORDR-CONTACT-NO
ACCESS ORDR USING (CUST-NUM)
```

You can also perform a generic key access using the WHERE clause on the ACCESS statement:

```
KEY CUST-NUM = ORDR-CUST-NUM
KEY ORDR-PROD-NUM
   ORDR-CONTACT-NO
ACCESS ORDR WHERE ORDR-CUST-NUM = CUST-NUM
```

3

Environment-dependent considerations

RDM VSAM support depends on the support available for a particular operating environment. Batch RDM VSAM support depends directly on VSAM and CICS RDM VSAM support depends on CICS File Control.

RDM VSAM support takes full advantage of any record holding, logging, and recovery facilities available for each environment. Refer to the IBM *OS/VS VSAM Programmer's Guide* or IBM *VSE/VSAM: Using Commands and Macros* for batch processing or IBM's *CICS Recovery and Restart Guide* for CICS processing.

Batch considerations

In a batch environment, you must do the following:

- ◆ You must include in your JCL a DD or DLBL statement for each base VSAM file (see “[Describing VSAM files on the Directory](#)” on page 15). You must also include a DD or DLBL statement for each alternate index. The JCL statement for an alternate index refers to neither the index file containing the alternate index nor to the alternate index field in the base cluster, but rather to the PATH connecting the two. The DSN or file name must match the value of the NAME parameter on the IDCAMS DEFINE PATH control statement.
- ◆ You can create the load module CSVIBVRP to define the VSAM resource pool for RDM. RDM VSAM fully supports local resource sharing but does not support global resource sharing. To create CSVIBVRP, use the list form of the IBM supplied macro, BLDVRP, as follows:

```
CSVIBVRP CSECT
          BLDVRP MF=L,TYPE=LSR[ ,STRNO=number][ ,KEYLEN=length] ,
          BUFFERS=(size(number),size(number),...)
          END
```

Refer to the IBM *VSE/VSAM: Using Commands and Macros* manual for information on the BLDVRP macro.



Because local shared resources (LSR) are the only type available under VSE, you must omit the TYPE=LSR parameter.

- ◆ You must make the module CSVIBVRP available to RDM VSAM. You can place it in a library from which RDM can dynamically load it, or you can link it with the RDM VSAM batch support module or phase. The RDM VSAM batch support module for OS/390 is CSVIVSAM, and the RDM VSAM batch support phase for VSE is CSVJVSAM.
- ◆ If you do not create your own CSVIBVRP, RDM operates under the default resource pools installed in VSAM by your systems programmers.

CICS considerations

In a CICS environment, you must define any VSAM files you wish to access through RDM. If you define them in the CICS File Control Table (FCT), refer to IBM's *CICS Resource Definition (Online)* for information on creating File Resources.

The CICS File Resource name is the DD name or DLBL name you entered in the Directory File Description. The File definition must specify the following parameters:

```
ADD(YES) BROWSE(YES) DELETE(YES) UPDATE(YES)
```



Under CICS, RDM does not support access to base clusters via paths from alternate indices.

You must define a Program Resource for the CICS VSAM support module. The module is named CSVCVSAM for OS/390 and CSVDVSAM for VSE.

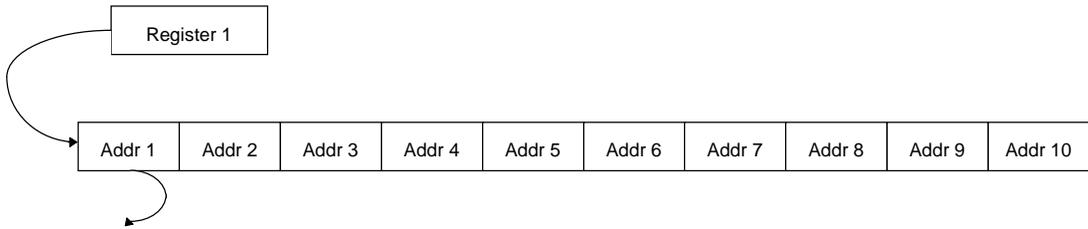
4

User exits

The function exit CSVXFUNC and status exit CSVXSTAT allow you to insert processing routines before and after database operations to satisfy any special requirements for your installation. This chapter describes the parameters passed to the user exits before and after accessing a VSAM file. Refer to the [SUPRA Server PDM RDM Administration Guide \(OS/390 & VSE\)](#), P26-8220, for general information about all RDM user exits.

You can use the CSVXFUNC exit to bypass or change a database operation and the CSVXSTAT exit to analyze the results of a database operation. The list of parameters that are passed to your routine is the same for the CSVXFUNC and CSVXSTAT exits except for the SKIP/LOOP parameter.

These parameters are:



Addr1	-FUNC	5-byte function
Addr2	-STATUS	4-byte status code
Addr3	-FILE	4-byte file name
Addr4	-KEYLENGTH	4-byte key length
Addr5	-DD NAME	8-byte DD name
Addr6	-KEY	Key value
Addr7	-END	END.
Addr8	-DATA	Variable length data area
Addr9	-MODE	4-byte mode parameter (RLSE, RSEQ, ESEQ, or END.)
Addr10	-SKIP/LOOP	1-byte return code 1 - Skip physical data manager call (CSVXFUNC) 1 - Loop physical data manager call (CSVXSTAT) All other values - do not skip/repeat physical data manager call

Function parameter

The function parameter value can be one of the following:

- ◆ **KSDGF.** Get first
- ◆ **KSDGN.** Get next
- ◆ **KSDUP.** Update
- ◆ **KSDIN.** Insert
- ◆ **KSDDL.** Delete

If the function is a GET FIRST (KSDGF) or a GET NEXT (KSDGN), the following table shows what the RDM action is, depending on the mode.

Mode parameter	Function	
	KSDGF (Get first)	KSDGN (Get next)
END.	Direct read and hold record.	Retrieve next record and hold record.
RLSE	Direct read.	Retrieve next record.
ESEQ (batch only)	Alternate index is in sequential or browse mode. Return to direct access.	Not possible.
RSEQ (batch only)	Switch alternate index to sequential or browse mode, and read record with key value requested.	Alternate index is currently in browse mode. Read next sequential record.

Status parameter

The following table lists all the valid status codes for the Status Parameter:

Code	Description
****	Successful completion
BCTL	Key out of range
DUPM	Duplicate key
END.	End of file or record not found
FAIL	VSAM macro or CICS DFHFC macro failed
FNAV	File could not be opened
FULL	File is full; record could not be added
HELD	Record held by another task
IOER	I/O error occurred
IVWV	An update request attempted to change the key of reference
MRNF	Record not found
NHLD	Record not held
NOCO	Insufficient virtual storage
NOVS	Environment dependent interface module for RDM VSAM not found

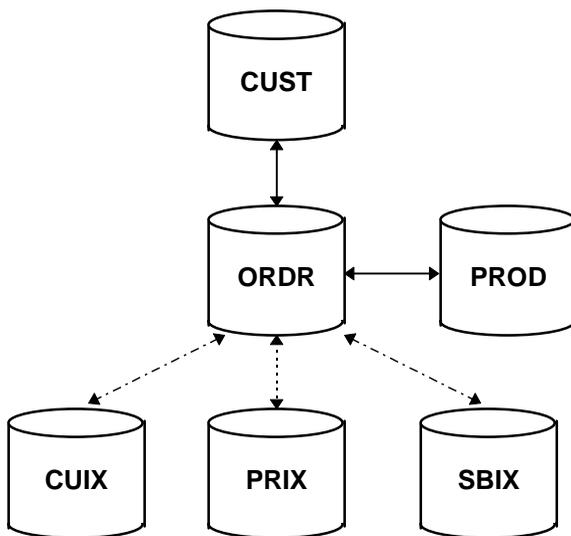
A

Sample KSDS database and base views

This appendix describes a sample KSDS database and shows some views which access the database.

Sample database

The following figure shows a diagram of the sample database. Following the diagram is a description of the files.



←.....→ Logical links maintained by RDM

←.....→ Alternate index maintained by VSAM

The preceding figure illustrates the database consisting of the files on the following pages.

CUST file

A native VSAM key-sequenced data set (KSDS). The prime key for this file is the customer number field. There are no alternate indices. The file contains the following customer information fields:

Physical field	External field	Column name
CUSTCTRL	CUST-NO	CUST-NO
CUSTNAME	CUST-NAME	NAME
CUSTADDR	CUST-ADDR	ADDRESS
CUSTCITY	CUST-CITY	CITY
CUSTSTAT	CUST-STATE	STATE
CUSTZIPC	CUST-ZIP	ZIP
CUSTPHNE	CUST-PHONE	PHONE

PROD file

A native VSAM key-sequenced data set (KSDS). The prime key for this file is the customer number field. There are no alternate indices. The file contains the following customer information fields:

Physical field	External field	Column name
PRODCTRL	PROD-NUMB	PROD-NO
PRODDISC	PROD-DESC	PROD-DESC
PRODLCHG	PROD-LEASE-CHRG	PROD-RENT
PRODMCHG	PROD-MAINT-CHRG	PROD-MAINT
PRODPRCE	PROD-PURCH-PRICE	PROD-PURCH

ORDR file

A native VSAM key-sequenced data set (KSDS). The prime key for this file is a concatenation of the customer number and product number fields. The file contains the following order information fields:

Physical field	External field	Column name
ORDRCTRL	ORDR-CTRL	ORDER-NO
ORDRCUIX	ORDR-CUST-NUM	CUST-NUM
ORDRPRIX	ORDR-PROD-NUM	PROD-NUM
ORDRIDTE	ORDR-INSTALL-DATE	INSTALL-DATE
ORDRBCHG	ORDR-REG-CHG-DATE	CHARGE-DATE
ORDRMDTE	ORDR-MAINT-DATE	ORDER-MAINT
ORDRSBIX	ORDR-SUB-CUST-NO	SUB-CUST-NO
ORDRCONN	ORDR-CONTRACT-NO	CONTRACT-NO
ORDRPONO	ORDR-CUST-PO-NO	PO-NO

Three alternate indices are defined for the ORDR file:

- ◆ **CUIX.** Nonunique index on the customer number. The alternate key field name is ORDR-CUST-NUM.
- ◆ **PRIX.** Nonunique index on the product number. The alternate key field name is ORDR-PROD-NUM.
- ◆ **SBIX.** Nonunique index on the subsidiary office number. This number maintains satellite and remote offices. The alternate key field name is ORDR-SUB-CUST-NO.

Example view definitions

The following examples describe views which access the files shown in “[Sample database](#)” on page 53.

Example view 1

This view accesses only one file, CUST, using the file’s prime key, CUST-NO. The ALLOW clause on the ACCESS statement allows the replacement, retrieval, insertion, and deletion of customers.

```
KEY CUST-NO
    ADDRESS = CUST-ADDR
    NAME = CUST-NAME
    STATE = CUST-STATE
ACCESS CUST WHERE CUST-NO = CUST-NO
    ALLOW UPDATE DELETE INSERT
```

Example view 2

This view is the same as the above view, except that it uses a USING clause instead of a WHERE clause to specify a direct keyed read using the customer number as the key.

```
KEY CUST-NO
    ADDRESS = CUST-ADDR
    NAME = CUST-NAME
    STATE = CUST-STATE
ACCESS CUST USING CUST-NO
    ALLOW UPDATE DELETE INSERT
```

Example view 3

The following view can be used to ensure integrity when you are performing deletes: this view deletes a customer's orders when deleting a customer.

```

KEY CUST-NO
   CUST-ADDR
   CUST-NAME
   CUST-CITY
ACCESS CUST WHERE CUST-NO = CUST-NO
   ALLOW ALL
ACCESS ORDR WHERE ORDR-CUST-NUM = CUST-NO
   ALLOW ALL

```

Example view 4

This view shows the use of the nonunique alternate key, ORDR-PROD-NUM. It also shows the use of column names. The order file is used in this view for delete maintenance only; a delete of a product will cause a delete of all orders for that product.

```

KEY PROD-NO = PROD-NUMB
   PROD-DESC
   PROD-RENT = PROD-LEASE-CHRG
   PROD-MAINT = PROD-MAINT-CHRG
   PROD-PURCH = PROD-PURCH-PRICE
ACCESS PROD USING PROD-NO
   ALLOW ALL
ACCESS ORDR VIA ORDR-PROD-NUM USING PROD-NUMB
   ALLOW DELETE

```

Example view 5

This view is the same as the preceding view, except that it uses the WHERE clause instead of the VIA and USING clauses. RDM cannot access the ORDR file using the full prime key because you are specifying only part of the prime key (ORDR-PROD-NUM). RDM cannot do a generic access using the partial prime key because ORDR-PROD-NUM is not the leftmost part of the prime key. Instead, RDM looks for an alternate index. In this case, RDM chooses to access the ORDR file via the PRIX index.

```
KEY PROD-NO = PROD-NUMB
  PROD-DESC
  PROD-RENT = PROD-LEASE-CHRG
  PROD-MAINT = PROD-MAINT-CHRG
  PROD-PURCH = PROD-PURCH-PRICE
ACCESS PROD WHERE PROD-NUMB = PROD-NO
  ALLOW ALL
ACCESS ORDR WHERE ORDR-PROD-NUM = PROD-NUMB
  ALLOW DELETE
```

Example view 6

This view shows the use of two keys. It uses the customer number to index through the order list to get all orders associated with that customer. In this view, an insert can succeed only if the customer exists.

```
KEY CUST-NO
  CUST-NAME
KEY PROD-NUM = ORDR-PROD-NUM
  INSTALL-DATE = ORDR-INSTALL-DATE
  SUB-CUST-NO = ORDR-SUB-CUST-NO
ACCESS CUST USING CUST-NO
ACCESS ORDR VIA ORDR-CUST-NUM USING CUST-NO
  ALLOW DELETE
```

Example view 7

This view allows the insertion of items into a purchase order. Inserts will only succeed in this view only if both the customer and product exist. Notice that there are two columns specified on the ACCESS ORDR statement. These columns make up the full control key.

```

KEY CUST-NO
      PO-NO = ORDR-CUST-PO-NO
REQ  PROD-NUM = ORDR-PROD-NUM = PROD-NUMB
      NAME = CUST-NAME
      PROD-DESC = PROD-DESC
ACCESS CUST USING CUST-NO
ACCESS ORDR USING (CUST-NO,PROD-NUM)
      ALLOW ALL
ACCESS PROD USING PROD-NUM

```

Example view 8

This view is the same as example 7, except that it uses the WHERE clause on the ACCESS statements. Notice that there are two columns specified on the ACCESS ORDR statement. These columns make up the full control key.

```

KEY CUST-NO
      PO-NO = ORDR-CUST-PO-NO
REQ  PROD-NUM = ORDR-PROD-NUM = PROD-NUMB
      NAME = CUST-NAME
      PROD-DESC = PROD-DESC
ACCESS CUST WHERE CUST-NO = CUST-NO
ACCESS ORDR WHERE ORDR-CUST-NUM = CUST-NO
      AND ORDR-PROD-NUM=PROD-NUM
      ALLOW ALL
ACCESS PROD WHERE PROD-NUMB = PROD-NUM

```

Example view 9

This view uses a nonunique alternate key, ORDR-CUST-NUM, to access the order file to get the customer's associated orders and then to access the product file to get a description of each product ordered.

```
KEY CUST-NO
    CUST-NAME
REQ ORDR-PROD-NUM
    PROD-DESC
ACCESS CUST USING CUST-NO
ACCESS ORDR VIA ORDR-CUST-NUM USING CUST-NO
    ALLOW ALL
ACCESS PROD USING ORDR-PROD-NUM
    ALLOW ALL
```

Example view 10

This view retrieves a customer's orders and then prints the description of the product ordered. Notice this view shows an example of using the GIVING clause with the WHERE clause.

RDM uses the partial prime key to access the ORDR file even though there is an alternate index on the ORDR-CUST-NUM (CUIX) field. This is because the partial prime key, ORDR-CUST-NUM, is a valid generic key for generic key access. RDM first tries to use a prime key or a generic partial prime key before choosing an alternate index.

```
KEY CUST-NO
    CUST-NAME
KEY PROD-NUM = ORDR-PROD-NUM
    PROD-DESC
ACCESS CUST WHERE CUST-NO = CUST-NO
ACCESS ORDR WHERE ORDR-CUST-NUM = CUST-NO GIVING PROD-NUM
ACCESS PROD WHERE PROD-NUMB = ORDR-PROD-NUM
```

Example view 11

This view first accesses the product file and then goes to the order and customer files to determine which customers ordered a certain product. If the application programmer leaves CUST-NUM off of the GET command, a scan of the ORDR file occurs and results are meaningless. The programmer should supply both parts of the logical key.

```
KEY PROD-NUMB
   PROD-DESC
KEY CUST-NUM = ORDR-CUST-NUM
   CUST-NAME
ACCESS PROD USING PROD-NUMB
ACCESS ORDR USING (CUST-NUM, PROD-NUMB)
ACCESS CUST USING ORDR-CUST-NUM
```

Example view 12

This view is the same as example 11, except that it uses the WHERE clause on the ACCESS statements.

```
KEY PROD-NUMB
   PROD-DESC
KEY CUST-NUM = ORDR-CUST-NUM
   CUST-NAME
ACCESS PROD WHERE PROD-NUMB = PROD-NUMB
ACCESS ORDR WHERE ORDR-CUST-NUM = CUST-NUM
                   AND ORDR-PROD-NUM = PROD-NUMB
ACCESS CUST WHERE CUST-NO = ORDR-CUST-NUM
```

Example view 13

This view is the same as examples 11 and 12, except that it uses a nonunique alternate key, ORDR-PROD-NUM. This returns meaningful results even if the programmer omits the customer number (the second part of the logical key) from the GET.

```
KEY PROD-NUMB
   PROD-DESC
KEY CUST-NUM = ORDR-CUST-NUM
   CUST-NAME
ACCESS PROD USING PROD-NUMB
ACCESS ORDR VIA ORDR-PROD-NUM USING PROD-NUMB
ACCESS CUST USING ORDR-CUST-NUM
```

Example view 14

This view is similar to examples 11, 12, and 13, except it uses the VIA clause to specify an alternate key and the WHERE clause to add a selection criterion. This view is more efficient than the one in Example view 13 if the programmer supplies the customer number logical key on the GET. In Example view 13, RDM reads an order record and uses the customer number in it as the key to retrieve customer information. Only after the retrieval from the CUST file does RDM compare the data read against the supplied logical key value for the customer number. In this example, RDM uses the customer number to qualify the ORDR record and skips the retrieval from CUST if the customer number doesn't match/ If there is no customer number on the GET, the two views are about equally efficient.

```
KEY PROD-NUMB
   PROD-DESC
KEY CUST-NUM = ORDR-CUST-NUM
   CUST-NAME
ACCESS PROD USING PROD-NUMB
ACCESS ORDR VIA ORDR-PROD-NUM USING PROD-NUMB
      WHERE ORDR-CUST-NUM = CUST-NUM
ACCESS CUST WHERE CUST-NO = CUST-NUM
```

Example view 15

This view shows the use of a nonunique alternate key, ORDR-CUST-NUM. The view looks for each customer number and associated product number in the order file and then accesses the customer and product files. The customer name, product number, and description are then retrieved. An insert to this view works only if the customer already exists.

```
KEY CUST-NUM = ORDR-CUST-NUM = CUST-NO
   CUST-NAME
REQ ORDR-PROD-NUM
   PROD-DESC
ACCESS ORDR VIA ORDR-CUST-NUM USING CUST-NUM
      ALLOW ALL
ACCESS CUST USING ORDR-CUST-NUM
ACCESS PROD USING ORDR-PROD-NUM
      ALLOW ALL
```

Example view 16

This view is the same as example 15, except that it uses the WHERE clause. RDM uses the prime key to access the ORDR file because it finds a partial mapping on the left part of the control key.

```

KEY CUST-NUM = ORDR-CUST-NUM=CUST-NO
  CUST-NAME
REQ ORDR-PROD-NUM
  PROD-DESC
ACCESS ORDR WHERE ORDR-CUST-NUM = CUST-NUM
  ALLOW ALL
ACCESS CUST WHERE CUST-NO = ORDR-CUST-NUM
ACCESS PROD WHERE PROD-NUMB = ORDR-PROD-NUM
  ALLOW ALL

```

Example view 17

This view is essentially the same as view 16, but it uses a compound key (CUST-NUM, ORDR-PROD-NUM) rather than a nonunique alternate key.

```

NONUNIQUE KEY CUST-NUM = ORDR-CUST-NUM = CUST-NO
  CUST-NAME
REQ ORDR-PROD-NUM
  PROD-DESC
ACCESS ORDR USING (CUST-NUM, ORDR,PROD-NUM)
  ALLOW ALL
ACCESS CUST USING ORDR-CUST-NUM
ACCESS PROD USING ORDR-PROD-NUM
  ALLOW INSERT

```

Example view 18

This view accesses the CUST file two times using different keys. Notice that the first ACCESS to the customer file does not contain an ALLOW clause, although the second ACCESS to the customer file does. This is done to ensure that a customer must exist to insert orders. However, a new subsidiary may be added to the file. The ACCESS statement on the PROD file also does not contain an ALLOW clause; this ensures that no orders can be added for a product that does not exist.

```
NONUNIQUE KEY CUST-NUM = ORDR-CUST-NUM = CUST-NO
      NAME = CUST-NAME
REQ PROD-NUM = ORDR-PROD-NUM = PROD-NUMB
      PROD-DESC
      SUB-CUST-NO = ORDR-SUB-CUST-NO
      SUB-NAME = CUST-NAME
ACCESS ORDR VIA ORDR-CUST-NUM USING CUST-NUM
      ALLOW ALL
ACCESS CUST USING ORDR-CUST-NUM GIVING NAME
ACCESS PROD USING ORDR-PROD-NUM
ACCESS CUST USING SUB-CUST-NO GIVING SUB-NAME
      ALLOW ALL
```

Example view 19

This view is similar to view 18, except that it shows the use of the WHERE clause. RDM accesses the ORDR file using the partial prime key instead of the alternate key; this is because the partial prime key is a valid generic key for generic key access.

```
NONUNIQUE KEY CUST-NUM = ORDR-CUST-NUM = CUST-NO
      NAME = CUST-NAME
REQ PROD-NUM = ORDR-PROD-NUM = PROD-NUMB
      PROD-DESC
      SUB-CUST-NO = ORDR-SUB-CUST-NO
REQ SUB-NAME = CUST-NAME
ACCESS ORDR WHERE ORDR-CUST-NUM = CUST-NUM
      ALLOW ALL
ACCESS CUST WHERE CUST-NO = ORDR-CUST-NUM GIVING NAME
ACCESS PROD WHERE PROD-NUMB = ORDR-PROD-NUM
ACCESS CUST WHERE CUST-NO = SUB-CUST-NO
      GIVING SUB-NAME
      ALLOW ALL
```

Example view 20

This is a fairly complex view. It access the ORDR file via an alternate index, and it also specifies a compound WHERE clause on that file for additional selection criteria. Notice that the view accesses the CUST file twice, once to obtain the customer name and the other to obtain the subsidiary customer name.

```
KEY PROD-NUMB
    PROD-DESC
    ORDR-INSTALL-DATE
    ORDR-MAINT-DATE
KEY CUST-NUM = ORDR-CUST-NUM
    NAME=CUST-NAME
KEY SUB-CUST-NO = ORDR-SUB-CUST-NO
    SUB-CUST-NAME = CUST-NAME
ACCESS PROD USING PROD-NUMB
ACCESS ORDR VIA ORDR-PROD-NUM USING PROD-NUMB
    WHERE ORDR-CUST-NUM = CUST-NUM
        AND ORDR-SUB-CUST-NO = SUB-CUST-NO
ACCESS CUST WHERE CUST-NO = CUST-NUM GIVING NAME
ACCESS CUST WHERE CUST-NO = SUB-CUST-NO GIVING SUB-CUST-NAME
```

Index

A

- access
 - generic key 44
 - syntax 31
- access definition statements 21
- ACCESS parameter 33
- ALLOW phrase 37
- alternate indices
 - adding file entities 18
 - overview 43
 - physical fields 20
- alternate key field 34

B

- base files, adding physical fields
 - 19
- base views
 - column definition 22
 - defining 21
 - defining ACCESS 31
- batch considerations 46

C

- CICS considerations 47
- column definition statements 21
- columns
 - constant value 27
 - defining 22
 - naming 25
- conceptual schema 14
- constant values 27
- copy book libraries 14
- CSVIBVRP load module 46
- CSVXFUNC exit 49

D

- database example 53
- ddname, defining 17
- DELETE command 40
- Directory
 - adding file entities 16
 - describing files 15

E

- environment-dependent considerations 45
- external field, naming 25, 26
- external schema 14

F

- fields
 - multiple 26
 - naming 25
- file access method, defining 17
- file device assignments, defining 17
- file device type, defining 17
- file entities
 - adding 16
 - alternate indices 18
- file type, defining 17
- FKEY parameter 23
- function parameter 51

G

- generalized access syntax 31, 32
- generic key access 44
- GET command 39
- GIVING clause 38

I

- indices, alternate. See alternate indices
- INSERT command 41
- internal schema 14

K

- KEY parameter 22
- key-sequenced data sets
 - prerequisites for access 13
 - sample database 53
 - types of 13
- KSDS. *See* key-sequenced data sets

L

- load module CSVIBVRP 46
- logical record length, defining 17
- logical views 21

M

- multiple fields 26

N

- NONUNIQUE KEY parameter 23

O

- ONCE clause 34
- one-to-one relationships 34
- operating environments 13

P

- physical fields
 - adding 19
 - alternate indices 20
 - naming 25, 26
- Processing Program Table (PPT)
 - 47

R

- RDML
 - DELETE command 40
 - GET command 39
 - INSERT command 41
 - overview 39
 - UPDATE command 42
- Relational Data Manipulation Language. *See* RDML
- REQ parameter 23

S

- sample database 53
- specific access syntax 31, 32
- status parameter 52
- supported operating environments 13

U

- UNIQUE CONST parameter 24
- UPDATE command 42
- user exits 49
- USING clause, access syntax 35

V

- variable length records 13
- VIA clause 34
- views
 - base
 - column definition 22
 - defining 21
 - defining ACCESS 31
 - examples 56
 - logical 21
- VSAM
 - accessing files 14
 - control interval, defining 17
 - data sets 13
 - describing files 15
 - using with RDM 15

W

- WHERE clause, access syntax 33