

Cincom

SUPRA SERVER PDM

RDM PDM Support Supplement
(OS/390 & VSE)

P26-8221-63



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

Publication Number P26-8221-63

© 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This document contains unpublished, confidential, and proprietary information of Cincom.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 PDM Support Supplement (OS/390 & VSE)*, P26-8221-63, 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
Introduction to the PDM	13
Using RDM to access a PDM database.....	13
Conceptual system overview.....	14
Using the PDM with the RDM	15
Physical Data Manager files.....	16
PDM primary files	17
PDM related files	18
Defining PDM files on the Directory	19
Defining base views	20
Column definition in base views	21
ACCESS definition in base views.....	31
Accessing primary files	32
Accessing related files	38
Accessing PDM files using secondary keys.....	48
Primary file considerations	54
Processing considerations	54
Compound control keys	55

Related file considerations.....	56
Standard processing of related files	57
Using primary file extensions.....	57
Multiple related file usage	58
Inserting coded records	58
Retrieving coded records in occurrence order.....	59
Retrieving related records by record code.....	60
Retrieving related records with embedded one-to-one relationships	61
Retrieving related records with embedded one-to-many relationships.....	62
Retrieving related records with embedded many-to-one relationships.....	63
Switching linkpaths	64
Inserting related records with embedded relationships	64
Optimizing performance	65
Maintaining foreign keys	65
Secondary key considerations	66
Using generic secondary keys.....	67
Using the REVERSE keyword	68
Using the ONCE keyword.....	68
GET PRIOR/LAST command.....	68
User exits	69
Sample PDM database and base views	71
Sample database.....	71
Example view definitions	74
Index	81

About this book

Using this document

This manual provides the database administrator (DBA) with reference information, including how to define a base view, for using the SUPRA Relational Data Manager (RDM) to access a SUPRA Physical Data Manager (PDM). In order to use this manual effectively, you should have knowledge of PDM file structures and PDM access techniques.

Document organization

The information in this manual is organized as follows:

Chapter 1—Introduction to the PDM

Provides an introduction to RDM and its terminology and PDM and its terminology.

Chapter 2—Using the PDM with RDM

Provides a brief overview of PDM file structures and the definitions of PDM files on the Directory.

Chapter 3—User exits

Lists exits available to bypass database calls, perform your own database of user file calls, or satisfy special requirements.

Appendix A—Sample PDM database and base views

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

Index

Revisions to this manual

The following changes were made for this release:

- ◆ The NORMAL product is no longer distributed. If you use NORMAL, retain your files and previous documentation. References to NORMAL in this document have been deleted.

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>BEGNBbbbSERIAL</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
<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>	<pre>[(WAIT)] [(NOWAIT)]</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>STATISTICS</pre>
<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>	<pre>COPY MY_DATA.SEQ HOLD_DATA.SEQ</pre>
<p><i>Italics</i></p>	<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>
<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>	<pre>(<i>user-id</i>, <i>password</i>, <i>db-name</i>) INFILE 'Cust.Memo' CONTROL LEN4</pre>
<p>SMALL CAPS</p>	<p>Represent a required 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 Directory Maintenance, DBA utilities, DBAID, 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

Introduction to the PDM

The SUPRA Relational Data Manager (RDM) supports access to SUPRA Physical Data Manager (PDM) files. This manual explains how to use RDM to access PDM files. Before you can effectively use the information in this manual, you should:

- ◆ Understand RDM and its terminology (refer to the *SUPRA Server PDM RDM Administration Guide (OS/390 & VSE)*, P26-8220)
- ◆ Understand the PDM and its terminology (refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250)

Using RDM to access a PDM database

You can use RDM to access data in a PDM database with a primary file's control key or with a secondary key. You can further refine your access approaches with linkpath or record code information. You can also sweep a file. You cannot use RDM to access a PDM file record by its physical location.

If you are starting a new PDM database, you must define its files on the Directory using Directory Maintenance, and define the base views using DBAID, before you can access the database with RDM.

If you already have a PDM database, then you have already defined its files on the Directory. You need not change the files or the file definitions in any way. You must still define the base views with DBAID; then you can access the database with RDM.

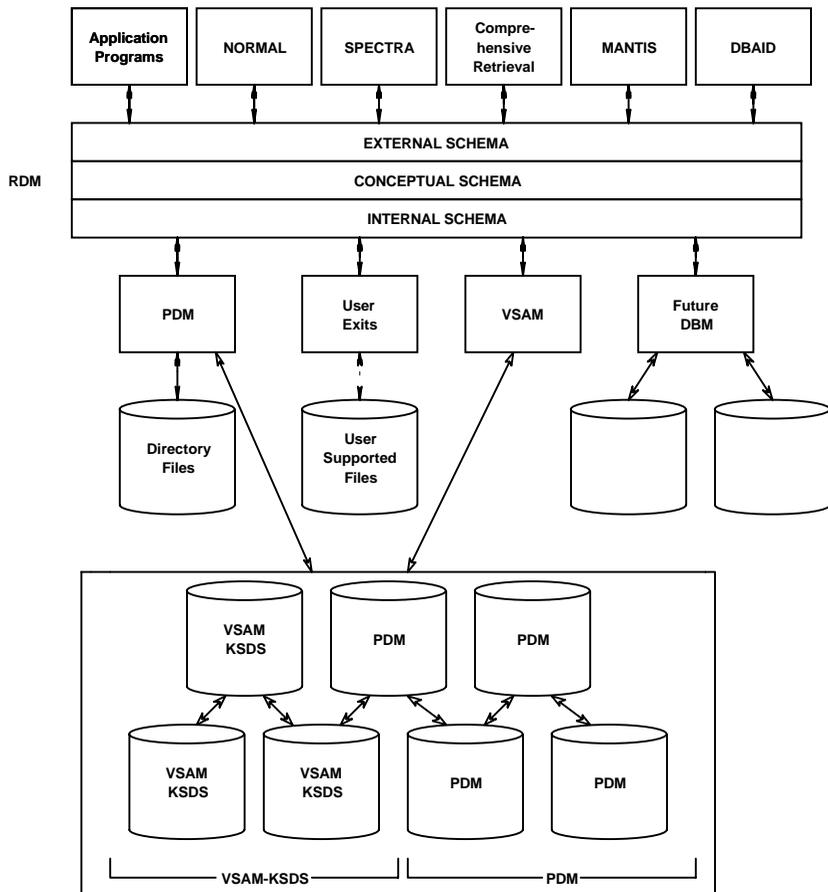
If you change your PDM files after you define your base views, you might need to change your base view definitions or your application programs. Refer to the *SUPRA Server PDM RDM Administration Guide (OS/390 & VSE)*, P26-8220, for more information.

After you define your base views, you can define derived views which access your base views. This manual does not discuss derived views. For information on derived views, refer to the *SUPRA Server PDM RDM Administration Guide (OS/390 & VSE)*, P26-8220.

RDM can access any combination of SUPRA's Physical Data Manager (PDM) files, native VSAM files, and files of any other supported type. RDM applications can use PDM files in much the same way they use any other files. Copy book libraries are unnecessary, because the record formats of the PDM files are stored in the Directory. With RDM, application programs and end users can manipulate and access data without any information about the data's physical location and structure.

Conceptual system overview

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

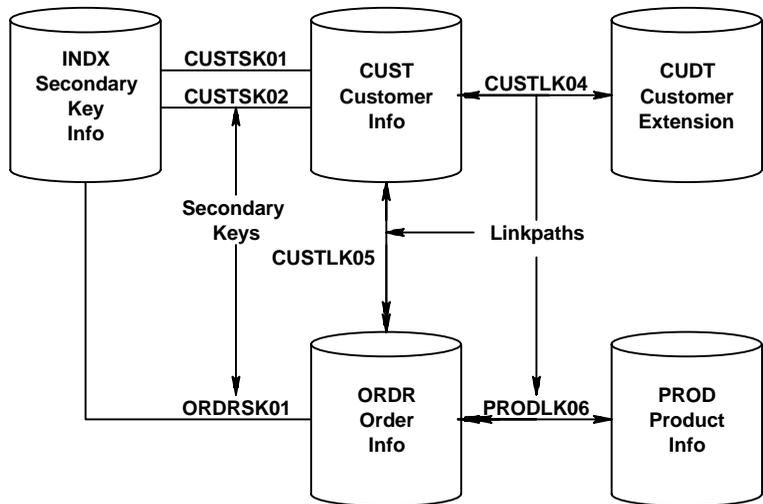
Using the PDM with the RDM

This chapter gives a brief overview of PDM file structures and the definition of PDM files on the Directory. This chapter gives a detailed explanation of the techniques for accessing data on PDM files with RDM. As part of this explanation, this chapter points out the implications of different access techniques for performance; your knowledge of these implications enables you to help the programmer design more efficient applications.

Physical Data Manager files

A Physical Data Manager (PDM) database can contain three types of files which are relevant to data access by RDM. A primary file contains data records, each of which includes a unique key, which is sufficient for accessing the record. A related file contains data records which can be accessed via pointers (linkpaths) in a primary file's data record. An index file contains no data records; it contains definitions and indexes for secondary keys. A secondary key consists of specified data in the data records of a primary or a related file, and it provides an alternate method of accessing those records.

The following figure shows a sample PDM database. "Sample PDM database and base views" on page 71 describes this sample database in detail and shows example base views.



PDM primary files

A PDM primary file is a keyed file. Each data record contains a physical field called the control key that uniquely identifies that record. Each data record can be accessed directly by its control key. In the example below, a primary file named CUST contains customer information. Its control key is the customer number, which corresponds to the external field named CUST-NO. The following view can be used to retrieve information on customer 100. (The view name is CUSTOMER.)

```
KEY CUST-NO
    CUST-NAME
ACCESS CUST WHERE CUST-NO = CUST-NO
```

The GET command would be:

```
GET CUSTOMER USING 100
```

You can also access primary files through the use of secondary keys. A secondary key consists of one or more data fields in the data record. You define, on the Directory, the composition of a secondary key, as well as the index file to contain its definition and indexing information. A secondary key need not be contiguous in the data record, and it need not be unique. You can specify a secondary key value and access the record(s) with that value. You can access data records sequentially, as if the records had been sorted in either ascending or descending order by their secondary key values. For information about using secondary keys with primary files, see [“Accessing PDM files using secondary keys”](#) on page 48 and [“Secondary key considerations”](#) on page 66.

Secondary keys, primary file control keys, and related file linkpath keys are all physical keys.

A primary file can, but need not, link to one or more related files. You can use linkpaths to navigate between primary and related files.

PDM related files

A related file is related to one or more primary files. In such a primary file, a data record in the primary file includes a pointer field (linkpath field) which points to a chain of data records in the related file. The logical relationship between a related file and a specified linkpath field in each data record of a primary file is called a linkpath. Each linkpath has a physical field name. You can also assign a logical name (External Field name) to a linkpath, a step we strongly encourage you to perform..

A chain of related file records that is related to a given primary file record via a given linkpath is called a linkpath set or linkpath chain. Each record in a linkpath chain includes a field that contains a copy of the primary file record's key; this field is called a referback field. In relational terms, a referback field is always a foreign key. Each record in a linkpath chain includes a pointer field (linkpath field) which points to the next and the previous records in the chain; the name of this physical field is the same as the name of the linkpath itself, and the same as the name of the primary file's physical linkpath field for this linkpath.

Every related file data record must be a member of at least one linkpath chain. A related file record can be a member of more than one linkpath chain, can have more than one referback field, and can relate to more than one primary file.

A data record in a coded related file is a coded record and consists of two portions: the base portion, which includes the record code field and other base fields which are present in every data record in the file; and the coded portion, which includes coded fields which may not be present in every record. All data records in the file contain the same set of base fields; two records with the same record code always contain the same set of coded fields; two records with different record codes can have partly or completely different sets of coded fields. Two records with different record codes can be members of the same linkpath chain. In such a case, we strongly recommend that the coded portions of all records be laid out "in parallel." Specifically, if the coded portion contains the referback on the linkpath connecting the records of the various record codes, the relative position of the referback field must be the same in the coded portion for all record codes on that linkpath. RDM cannot properly process such a linkpath chain if you do not follow this recommendation.

The linkpath provides an access path from the primary to the related file. For example, a related file containing transaction information might be related to a primary file containing customer information. Each customer record would link to a chain of transaction records for that customer.

You can also access related files through the use of secondary keys. The discussion of secondary keys in “[PDM primary files](#)” on page 17 applies to primary and to related files. For information on using secondary keys with related files, see “[Accessing PDM files using secondary keys](#)” on page 48 and “[Secondary key considerations](#)” on page 66.

Defining PDM files on the Directory

You must define the PDM files on the Directory in order to access them using RDM. For information on using Directory Maintenance to define PDM files, 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.

Defining base views

The Relational Data Manager (RDM) treats data as if it were arranged in tables (or relations). Each time an RDM 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. It is much easier to define views with DBAID than with Directory Maintenance.

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

A base view definition consists of the following parts:

- ◆ **Column definition.** Defines the columns of a view. You specify the name and logical properties of each column in the row. Refer to the *SUPRA Server PDM RDM Administration Guide (OS/390 & VSE)*, P26-8220, for a complete definition of the Column Definition syntax.
- ◆ **Access definition.** Defines the database access. ACCESS statements define the relationships between files, the sequence in which the files are used, and which database maintenance functions affect which files. The ACCESS statement varies depending on the types of files you are using. If your view uses files other than PDM files, consult the SUPRA RDM support supplement for that file type to determine the correct ACCESS statement syntax.

Any valid view always contains at least one column definition and one ACCESS definition.

Column definition 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. You specify the view name as part of the view definition.

KEY NONUNIQUE KEY REQ FKEY [UNIQUE] CONST	[column - name = [=]] field - name [[=] = field - name [...]] [= constant]
--	---

KEY

Description *Optional.* Indicates that this field 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 defines a required field. The [general considerations](#) on page 28 explain the effect of required fields on RDM's view processing.
- ◆ The maximum number of KEY and NONUNIQUE KEY fields which may be defined in a view is nine.
- ◆ If the field is a physical key, and you issue a GET USING command with a value for the field, RDM does a direct (or indexed) read.
- ◆ If the field is not a physical key, and you issue a GET USING command with a value for the field, RDM does a sequential search of the file for an equal condition on the field.
- ◆ If you define a view column as KEY but multiple occurrences of a given value exist in your database in the field supplying the column, RDM may not be able to read all the occurrences. To read all such occurrences, change KEY to NONUNIQUE KEY.

NONUNIQUE KEY

Description *Optional.* Indicates that this field 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

- ◆ This defines a required field. The [general considerations](#) on page 28 explain the effect of required fields on RDM's view processing.
- ◆ The maximum number of KEY and NONUNIQUE KEY fields which may be defined in a view is nine.
- ◆ If the field is a physical key, and you issue a GET USING command with a value for the field, RDM does a direct (or indexed) read.
- ◆ If the field is not a physical key, and you issue a GET USING command with a value for the field, RDM does a sequential scan of files for an equal condition on the field.
- ◆ 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.
- ◆ If a file's columns include a KEY column and a NONUNIQUE KEY column, the file is processed as if both columns were nonunique keys. This is because a nonunique key makes the entire combination of keys nonunique.

REQ

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

Consideration This defines a required field. The [general considerations](#) on page 28 explain the effect of required fields 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 field must be identified in the view either as a REQ 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

- ◆ This defines a required field. The [general considerations](#) on page 28 explain the effect of required fields 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.
- ◆ You must specify a CONST to supply a constant.
- ◆ 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 the following parameter) to identify the field.
- ◆ If you specify multiple field names for a column, you must specify a column name. 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 on the Directory has a validation option of E, the corresponding validation exit entry point must exist.

[=] = *field-name*[...]

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 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](#) on page 28 explain the effect of required fields on RDM's view processing.
- ◆ If a redundant 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 a redundant 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

nnnnnnnnn Numeric (Binary, Packed, or Zoned)

'cccc' Character

Considerations

- ◆ The length of the value depends on the length of the column being defined.
- ◆ If the column definition specifies CONST, you must code a constant value.

General considerations for column definition in base views

- ◆ Any columns defined as REQ, KEY, CONST, or NONUNIQUE KEY are required fields 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 fields 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 fields supplying data to required columns are present, valid, and non-null.
 - On any GET, a column identified with REQ or KEY or NONUNIQUE KEY in the base view need not be defined in a derived view.
 - On an INSERT or an UPDATE, RDM returns an error unless all required fields are present, valid, and non-null.
 - On a DELETE, required fields have no effect.
- ◆ If a file's columns include a KEY column and a NONUNIQUE KEY column, the file is processed as if both columns were nonunique keys. This is because a nonunique key makes the entire combination of keys nonunique.
- ◆ At the physical level, more than one data element in a record can be a view key; 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 view processing.
- ◆ A constant in a CONST column definition cannot be a null pattern.
- ◆ A constant must pass the validity checking if the column has validation criteria associated with it.

Examples for column definition in base views

- ◆ 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 database.
- With an UPDATE, a change in CUSTOMER-NUMBER updates both CUST-NO and AUDIT-CUST-NO.

- ◆ This example shows multiple field names making up a key column.

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

- ◆ RDM treats all 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 clause. If no key value is supplied on the GET command (you omit the USING clause), RDM does not guarantee that these values are equal as in the first part of the previous 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
```

ACCESS definition in base views

There are two types of ACCESS statements which you can use in your view to access PDM files:

- ◆ The generalized ACCESS statement includes the WHERE clause and leaves the access strategy for RDM to determine.
- ◆ The specific ACCESS statement includes the USING clause and/or the VIA clause to specify the control key, linkpath, or secondary key for accessing the file.

If you use a generalized ACCESS statement, RDM selects an access strategy when RDM opens the view. RDM determines whether to access the data through the use of a primary file's control key, through a linkpath, through a secondary key, or by scanning the file.

When you use a generalized ACCESS statement, you need to be aware of the possibility that RDM is scanning the file, even though this might not be what you intend or expect. RDM statistics were intended 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 view open, displays the ACCESS strategy selected for the view.

Accessing primary files

The ACCESS statements you use for PDM primary files can be either generalized or specific. You can use the WHERE clause without the USING clause so that RDM determines the access strategy (Format 1). You can use the USING clause so that RDM accesses the file directly with the file's control key (Format 2). You can use the WHERE clause together with the USING clause so that RDM accesses the file directly and applies additional criteria to the accessed record. You can also access primary files using a secondary key. See [“Accessing PDM files using secondary keys”](#) on page 48 for more information.

FORMAT 1—Generalized ACCESS statement

ACCESS *filename*

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

[GIVING *column₁* [...*column_n*]

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

FORMAT 2—ACCESS specifying a keyed read

ACCESS *filename*

USING *key-field*

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

[GIVING *column₁* [...*column_n*]

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

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

ACCESS

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

filename

Description *Required.* Identifies the file you are accessing.

Format 4-character logical file name as it exists on the Directory

Consideration The file definition must already exist on the Directory.

USING *key-field*

Description *Conditional.* Required for a keyed read, not used for a generalized statement. Indicates that RDM should access the primary file directly using the specified key field as the control key.

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

```
ACCESS MAS2 USING (FIELD1, '252', FIELD2)
```

Considerations

- ◆ Except for a base file (the first file accessed in the view), you must use a USING or a WHERE clause.
- ◆ If the control key is subdefined, you can specify those subdefinitions as the key fields. Each key field specified must correspond to the subdefined part of the control key as defined on the Directory. You must use parentheses in the USING to indicate the use of the subdefinition of the control key field.
- ◆ If the Directory definition of the control key field is:

```
01 MASTCTRL
02 SUB-PART-1
03 SUB-PART-1A
03 SUB-PART-1B
02 SUB-PART-2
```

you could specify the following in a USING clause:

```
USING MASTCTRL
USING (SUB-PART-1, SUB-PART-2)
USING ((SUB-PART-1A, SUB-PART-1B) SUB-PART-2)
```

- ◆ Key parts may be omitted, starting at the right-hand side of any group of key parts in parentheses. For example:

```
USING (SUB-PART-1)
```

- ◆ When specifying compound keys, key parts may be omitted starting at the right-hand side of any group of key parts in parentheses. For example:

```
USING (SUB-PART-1)  
USING ((SUB-PART-1A))
```

- ◆ RDM does not perform domain compatibility checking when you specify USING.

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

Description *Optional.* If you use the WHERE clause together with the USING clause, the WHERE clause 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 in the WHERE clause include the file's control key, RDM does a direct access on the file using the control key.
- ◆ If the fields do not include the control key, but they do include a secondary key, or the leftmost portion of a secondary key (that portion being a generic key), RDM does an indexed access on the file using the whole or partial secondary key.
- ◆ If the fields do not include the control key, a full secondary key, or a generic secondary key, RDM scans the file.

Format	WHERE	Specified as shown.
	<i>field-1</i>	Field-1 must be a field in the file. It may be a column, an external field, or a physical field.
	= [=]	Specifies an equal comparison between the field and the 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 value that field-1 must match. It can be a physical field, external field, column, or constant. It does not have to be a column in the view; it can be a field from a record supplied by a preceding ACCESS statement.
	AND	<i>Optional.</i> Allows specification of additional qualifications for the ACCESS statement.

Considerations

- ◆ Except for base files (the first file accessed in the view), you must use USING or WHERE clause.
- ◆ Field-1 and value must be the same length.
- ◆ If you wish to force the use of a secondary key, use the VIA clause.
- ◆ Use RDM statistics to measure the performance of the view when using the WHERE clause without the USING clause.
- ◆ Field values used in a join must be valid and non-null. That is, they must be equal, valid, and non-null for RDM to consider them equal .

GIVING column₁ [... column_n]

Description *Optional.* This clause overrides the normal data movement of physical fields to columns.

Format Keyword GIVING followed by one 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 columns during each access. For each file which occurs on more than one ACCESS statement and contains physical fields which correspond to needed columns, you can specify which columns should be filled on which 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 that have not been supplied by some previous ACCESS statement are filled with values using this ACCESS statement.

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

Description *Optional.* Specifies what physical actions are 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 SHARED Allows columns to be shared between views.

ALL Allows all three forms of database modifications.

INSERT Allows inserts on the database.

DELETE Allows deletes from the database.

UPDATE / REP Allows updates or replacements on the database.

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.
- ◆ RDM does not override any restrictions imposed by the PDM. For example, if the PDM does not allow an INSERT, then RDM does not allow you to insert even if you specify ALLOW INSERT.
- ◆ Use of SHARED causes RDM to skip the column comparison normally performed before you delete or replace records. This column comparison detects and reports changed column values. Use of SHARED has no impact on INSERT.

Accessing related files

The ACCESS statements you use for PDM related files can be either generalized or specific. You can use the WHERE clause without the USING clause so that RDM determines the access strategy (Format 1). You can use the VIA clause to specify the linkpath that RDM should use to access the file (Format 2). You can use the WHERE clause together with the VIA clause so that RDM accesses the file using the specified linkpath and applies additional criteria to the accessed record.

You can also access related files through the use of a secondary key. See [“Accessing PDM files using secondary keys”](#) on page 48 for more information.

FORMAT 1—Generalized ACCESS statement

```

ACCESS filename [(record-code1[...record-coden])]
[ONCE]
WHERE field-1 = [=] value [... AND field-n = [=] value]
[GIVING column1 [...columnn]]
[ ALLOW [SHARED][ALL][INSERT][DELETE][UPDATE] ]
[ REP ]

```

FORMAT 2—ACCESS statement specifying a linkpath

```

ACCESS filename [ ( [ record - code1 ... ] + ] record - coden )]
[FROM file [(record-code)]]
[ ONCE ] [REVERSE] VIA linkpath-field-name
[ SCAN ]
WHERE field-1 = [=] value [... AND field-n = [=] value]
[GIVING column1 [...columnn]]
[ALLOW [SHARED] [ALL] [INSERT][DELETE][UPDATE]]
[ ORDER column - name [DESCENDING] ]
[ FIRST ]
[ NEXT ]
[ PRIOR ]
[ LAST ]

```

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

ACCESS

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

filename

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

Format 4-character file name as defined on the Directory

$$\left[\left(\text{record - code}_1 \left[\dots \left[\left. \begin{array}{c} , \\ + \\ : \\ ; \end{array} \right\} \text{record - code}_n \right] \right) \right] \right]$$

Description *Optional.* Indicates that RDM should access records with the specified record code(s) in the specified related file.

Format Each record code must be a valid 2-character record code defined on the Directory for this related file. Each operator must be one of the special characters shown (comma, plus sign, colon, semicolon). The operator is a shorthand method available for specifying record code processing. You can use this method in place of the FROM clause or in conjunction with it.

◆ Operators:

, Specifies an "or" condition. For example: (HD,CM) causes RDM to search the file (or the specified linkpath chain in the file, if any) until either an HD or a CM record is found.

◆ If you use the VIA clause to determine the access strategy, you may use the following three operators:

+ 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.

: Specifies embedded one-to-many relationships. For example: (HD:IT) retrieves many IT coded records for each HD coded record retrieved.

; Specifies embedded many-to-one relationships. For example: (IT;HD) scans the related file linkpath chain for a record with the HD record code after finding an IT record.

Considerations

- ◆ If you use the WHERE clause without a VIA clause, you can only specify multiple record codes with commas (the "or" operator), indicating that any record with any of the listed record codes may be acceptable; for example, (HD,CU,OR).
- ◆ If you specify multiple record codes, then all fields in the WHERE clause must be base fields.
- ◆ You may specify combinations of record codes and operators. For example, (HD+CU:OR) indicates that for every header (HD), there is one associated customer (CU), and many orders (OR).
- ◆ Record code specification depends on file and record relationships. See "[Related file considerations](#)" on page 56 for all the possible processing combinations when accessing PDM related files.

FROM *file* [(*record-code*)]

Description *Optional.* Indicates that RDM should access the current file (whose name follows the ACCESS keyword in the current statement) using information obtained (by a preceding ACCESS statement) from a record in the file named in this clause. This clause can further specify the record code of the desired source record. This overrides the default navigation from primary to related files, and allows related-to-related file navigation.

Format	FROM	Specified as shown.
	<i>file</i>	4-character name of a PDM related file defined on the Directory.
	(<i>record-code</i>)	2-character record code valid for the file in this clause, as defined on the Directory.

Consideration You cannot use the FROM clause without the VIA clause.

ONCE **SCAN** [REVERSE] VIA *linkpath-field-name*

Description *Conditional.* Required for a statement specifying a linkpath; not used for a generalized statement. Specifies the linkpath RDM should use to access the specified related file.

Format The linkpath name must be the external field name for the appropriate linkpath field as defined on the Directory. The physical field name is always of the form '*ppppLKxx*' (where *pppp* is the name of the primary file), for example, CUSTLK05. Since a physical field of this name must be defined in both the primary and related files, which are linked by the linkpath, RDM requires that you define an external field name for the physical field in the related file. Use that external field name as the linkpath-field-name in the VIA clause or RDM will be unable to determine the correct navigation strategy.

Options	ONCE	Indicates that only the first record which meets the selection criteria is retrieved from the file. Indicates that a one-to-one relationship exists between the record obtained by the preceding ACCESS statement and this one.
	SCAN	Indicates that RDM should scan the linkpath chain until it finds a record with the specified record code. Indicates that a many-to-one relationship exists between the record obtained by the preceding ACCESS statement and this one.
	REVERSE	Indicates that RDM should follow the linkpath chain in reverse direction.

Considerations

- ◆ The normal, default situation in which you use the VIA clause is this. The record obtained by the previous ACCESS statement is a primary file record. A one-to-many relationship exists between the primary record and the related records on the linkpath chain. The current ACCESS statement accesses any records on the chain, in forward order, which meet the selection criteria. Each of the options (ONCE, SCAN, and REVERSE) indicates a situation that differs from the default situation.
- ◆ RDM does not perform domain checking when you use the VIA clause.

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

Description	<i>Conditional.</i> Required for a generalized statement, optional for a statement specifying a linkpath. If you use the WHERE clause together with the VIA clause, the WHERE clause provides additional selection criteria. If you use the WHERE clause without the VIA clause, RDM selects the optimum access strategy in the following manner:	
	<ul style="list-style-type: none"> ◆ If the fields in the WHERE clause include a referback for the related file, RDM does a direct access on the primary file for that linkpath using the referback as the control key, and then accesses the related file using the linkpath field in the primary record. ◆ If the fields do not include a referback, but do include a secondary key, or the leftmost portion of a secondary key (that portion being a generic key), RDM does an indexed access on the related file using the whole or partial secondary key. ◆ If the fields do not include a referback, a full secondary key, or a generic secondary key, RDM scans the entire related file. 	
Format	WHERE	Specified as shown.
	<i>field-1</i>	Field-1 must be a field in the file. It may be a column, an external field, or a physical field.
	= [=]	Specifies an equal comparison between the field and the value. If both fields have domains, the domains must be the same unless you override this restriction by using two equal signs. For example: ACCESS E\$CU WHERE CUST-NAME = = CUST-NO
	<i>value</i>	Specifies the value that field-1 must match. It can be a physical field, external field, column or constant. It does not have to be a column in the view; it can be a field from a record supplied by a preceding ACCESS statement.
	AND	<i>Optional.</i> Allows specification of additional qualifications for the ACCESS.

Considerations

- ◆ Field-1 and value must be the same length.
- ◆ If you wish to specify a linkpath or secondary key, use the VIA clause.
- ◆ Use RDM statistics to measure the performance of the view when using the WHERE clause without the VIA clause.
- ◆ Field values used in a join must be valid and non-null.

GIVING *column*₁ [... *column*_n]

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

Format The keyword GIVING followed by one 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 columns during each access. For each file which occurs on more than one ACCESS statement and contains physical fields which correspond to needed columns, you can specify which columns should be filled on which 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 that have not been supplied by some previous ACCESS statement are filled with values using this ACCESS statement.

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

Description	<i>Optional.</i> Specifies what physical actions are 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 not inserts and deletes.
Options	SHARED	Allows columns to be shared between views.
	ALL	Allows all three forms of database modifications.
	INSERT	Allows inserts on the database.
	DELETE	Allows deletes from the database.
	UPDATE / REP	Allows updates or replacements on the database.

Considerations

- ◆ You may place the ALLOW clause on as many ACCESS statements as are required to properly maintain the view.
- ◆ If you omit this clause, the file is only accessed for read-only processing.
- ◆ These options relate to physical I/O on the file, not to the application program's RDML.
- ◆ RDM does not override any restrictions imposed by the PDM. For example, if the PDM does not allow an INSERT, then RDM does not allow you to insert even if you specify ALLOW INSERT.
- ◆ Use of SHARED causes RDM to skip the column comparison normally performed before you delete or replace records. This column comparison detects and reports changed column values. Use of SHARED has no impact on INSERT.

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

Description *Optional.* Indicates that predetermined ordering criteria are used when operating on a related file named in the ACCESS statement.

Default All ordering defaults to ascending unless you specify DESCENDING. If you do not specify ordering in the ACCESS statement, the FIRST/NEXT/PRIOR/LAST option on the INSERT command can be used by the application program to control ordering of the physical insertion of records.

Format *Column-name* must be a column name defined in the view

Considerations

- ◆ You can use the ORDER clause only with a VIA clause that specifies linkpath navigation.
- ◆ In the case of record codes, you can change the linkpath and column to use for ordering from ACCESS statement to ACCESS statement within the same file. The ordering information is used for the GET USING and INSERT functions.
- ◆ With the GET USING, RDM uses ordering if the column in the ORDER clause is also a key (unique or nonunique) to the view.
- ◆ In the case of INSERT, the ordering criteria are always used. The chain is first positioned via the order criteria, and the row is inserted at that position. When inserting a row, the ORDER column is a required column.
- ◆ There are two levels of positioning available with the ORDER clause. In both cases, the ORDER clause overrides FIRST/NEXT/LAST/PRIOR used with the INSERT statement.
 - If you supply a column name in the ORDER clause, then positioning of FIRST/NEXT/LAST/PRIOR is applied only to multiple occurrences of the value for that clause.
 - If you do not supply a column name in the ORDER clause, then positioning FIRST/NEXT/LAST/PRIOR is applied to the entire chain for that file.
- ◆ If the view is currently not positioned in accordance with the order clause, 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 REVERSE option on the VIA clause also affects the order in which RDM accesses records on the linkpath chain.
- ◆ ORDER does not correct the sequencing of existing data on a linkpath chain. For example, suppose the data in an ORDER field on a linkpath chain happens to be as follows: 1, 3, 9, 4. (This isn't very clear. I'm trying to describe a linkpath chain of 4 records, and in this one particular field in each record, the data values you will find if you read along the chain are 1 then 3 then 9 then 4.) If you now issue an INSERT that supplies the value 5 in the ORDER field, the current position can affect your results.

First, suppose you're currently positioned on the record containing the 1. RDM sees your requested data value 5 is greater than the order data at the current position. So RDM scans forward on the chain, finds $3 < 5$ but $9 > 5$, and inserts the 5 between those 2 records. It does not scan further down the chain to discover the sequencing error.

Now instead suppose you're currently positioned on the record containing the 4. RDM again sees your requested data value 5 is greater than the order data at the current position. So RDM scans forward on the chain. Since it's already at the end of the chain, it appends the new record to the end of the chain and stops. It does not scan further back in the chain to discover that the initial data was out of sequence.

The moral of the story is this. If you code an ORDER clause, your data really needs to be in sequence. You should have created the data with ORDER in the first place. You can't suddenly use an ORDER clause to access data that was not created in sequence. If the data is out of sequence, supplying an ORDER clause does not cause RDM to correct the sequencing and you can get anomalous results.

Accessing PDM files using secondary keys

The following ACCESS statement allows you to access either PDM primary or related files using a secondary key. You must first define the secondary key on the Directory. The ACCESS statement for primary or related files using a secondary key is identical except that you may not specify a record code when accessing PDM primary files. After you specify the ACCESS statement and the file name, you may enter the optional clauses in any order. You can use the WHERE clause to provide additional selection criteria.

ACCESS filename [(record-code₁[...record-code_n])]

[ONCE][REVERSE] VIA secondary-key-name [USING key-field]

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

[GIVING column₁ [...column_n]]

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

ACCESS

Description *Required.* Identifies the statement as an Access Definition for the view.

filename

Description *Required.* Identifies the file that RDM should access.

Format Must be the 4-character name of the required file as defined on the Directory.

(record-code₁[...],record-code_n)

- Description** *Optional.* Indicates that RDM should access records with the specified record code(s) in the specified related file.
- Format** Each record code is two characters long and must be an actual record code defined on the Directory. The special character separating multiple record codes must be a comma; colon, semicolon, and plus sign are not valid in this phrase for secondary keys.

Considerations

- ◆ You can specify record codes for PDM related files only.
- ◆ If you specify multiple record codes, all fields that make up the secondary key must come from the base portion of the data record.
- ◆ You can specify multiple record codes only with commas (the "or" operator) as shown, indicating that any record with any of the listed record codes may be acceptable; for example, (HD,CU,OR).

[ONCE][REVERSE] VIA secondary-key-name

- Description** *Required.* Specifies the secondary key RDM should use to access the file.
- Format** The name specified must be the name of a valid secondary key as defined on the Directory. The format of the name is always 'fileSKnn', for example, CUSTSK01.
- Options**
- | | |
|---------|--|
| ONCE | Indicates that only the first physical record that meets the selection criteria is returned. See "Using the ONCE keyword" on page 68 for more information. |
| REVERSE | Indicates that RDM should access the records in order of descending secondary key values. See "Using the REVERSE keyword" on page 68 for more information. |

Considerations

- ◆ If you code both a VIA clause and a USING clause, the USING clause must immediately follow the VIA clause.
- ◆ RDM does not perform domain compatibility checking when you use a VIA or USING clause.

USING *key-field*

Description *Optional.* Indicates that RDM should do an indexed read on the file using the specified key field to supply the secondary key value.

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

```
ACCESS MAS2 VIA MAS2SK03 USING (FIELD1, '252', FIELD2)
```

Considerations

- ◆ You can specify key fields from parts of the secondary key as defined in the Directory. Depending on whether the secondary key is simple or compound and whether or not the field(s) are subdefined, determines if and how parentheses should be used.
- ◆ For a simple secondary key, each pair of parentheses indicates the sublevel, if any, at which the key fields specified in the USING clause, are defined.
- ◆ For example, consider a simple secondary key, built on a field named INDEXFIELD, which is subdefined as follows:

```
01 INDEXFIELD
  02 SUB-PART-1
    03 SUB-PART-1A
    03 SUB-PART-1B
  02 SUB-PART-2
```

you could specify the following in a USING clause:

```
USING INDEXFIELD
USING (SUB-PART-1, SUB-PART-2)
USING ((SUB-PART-1A, SUB-PART-1B) SUB-PART-2)
```

- ◆ Key parts may be omitted, starting at the right-hand side of any group of key parts in parentheses. For example:

```
USING (SUB-PART-1)
USING ((SUB-PART-1A))
```

- ◆ When specifying only the secondary key field, you must omit the parentheses.
- ◆ If you code both a VIA clause and a USING clause, the USING clause must immediately follow the VIA clause

- ◆ For a compound secondary key, if more than one of the fields, which comprise the key, are specified in the USING clause, then they must be enclosed in parentheses.
- ◆ If only the first field comprising the compound key is specified, then the parentheses are optional.
- ◆ If the fields specified are subdefinitions of those fields comprising the compound secondary key, then one pair of parentheses is required for each level of subdefinition plus one extra pair because the key is compound. For example, consider a compound secondary key built on the field defined above, INDXFLD and another field, INDXFLD01, then you could specify the following in a USING clause:


```

      USING INDEXFIELD           or           USING (INDEXFIELD)
      USING (INDEXFIELD,  INDXFLD01)
      USING ((SUB-PART-1,  SUB-PART-2),  INDXFLD01)
      USING (((SUB-PART-1A, SUB-PART-1B)), SUB-PART-2),  INDXFLD01)
      
```
- ◆ As with simple secondary keys, key parts may be omitted starting at the right-hand side of groups in parentheses.
- ◆ Constant values must be valid and non-null.

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

Description	<i>Optional.</i> The WHERE clause, used in conjunction with the VIA clause, provides additional selection criteria.	
Format	WHERE	Specified as shown.
	<i>field-1</i>	Field-1 must be a field in the file. It may be a column, an external field or a physical field.
	= [=]	Specifies an equal comparison between the field and the value. If both fields have domains, the domains must be the same unless you override this restriction by using two equal signs. For example: <pre style="margin-left: 40px;">ACCESS E\$CU WHERE CUST-NAME = = CUST-NO</pre>
	<i>value</i>	Specifies the value that field-1 must match. It can be a physical field, external field, column or constant. It does not have to be a column in the view; it can be a field from a record supplied by a preceding ACCESS statement.
	AND	<i>Optional.</i> Allows specification of additional qualifications for the ACCESS statement.

GIVING column₁ [...column_n]

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

Format Keyword GIVING followed by one 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 during each access. For each file which occurs on more than one ACCESS statement and contains physical fields which correspond to needed columns, you can specify which columns should be filled on which access of the file.
- ◆ If you omit column names on the GIVING clause, no column values are obtained on this access of the file. The file is used for navigation only.
- ◆ If you omit this clause, all columns that are derived from physical fields in the file, and that have not been supplied by some previous ACCESS statement, are filled with values using this ACCESS statement.

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

Description	<i>Optional.</i> Specifies what physical actions are 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 not inserts and deletes.
Options	SHARED Allows columns to be shared between views. ALL Allows all three forms of database modifications. INSERT Allows inserts on the database. DELETE Allows deletes from the database. UPDATE / REP Allows updates or replacements on the database.

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 only accessed for read-only processing.
- ◆ These options relate to physical I/O on the file, not to the application program's RDML.
- ◆ RDM does not override any restrictions imposed by the PDM. For example, if the PDM does not allow an INSERT, then RDM does not allow you to insert even if you specify ALLOW INS.
- ◆ Use of SHARED causes RDM to skip the column comparison normally performed before you delete or replace records. This column comparison detects and reports changed column values. Use of SHARED has no impact on INSERT.

Primary file considerations

This section discusses special considerations for using PDM primary files with RDM.

The most efficient way to retrieve records from a primary file is to use a logical key and have it correspond exactly to the control key in the record. For instance, the base file (the first file accessed) in the view is typically a keyed file. That ACCESS statement would be in the form:

```
ACCESS file USING column
```

or

```
ACCESS file WHERE field = value
```

In the first case, the column name would be designated as a key in the view. For example:

```
KEY CUST-NO  
ACCESS CUST USING CUST-NO
```

When you use a GET USING command and specify a value for CUST-NO, RDM performs a direct keyed read using the file's control key.

The view would be similarly efficient if the field in the WHERE clause corresponded exactly to the file's control key. For example,

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

Processing considerations

You cannot sweep a PDM primary file in a logical sequence, either forward or backwards, without the use of a secondary key; i.e., you may not issue a GET LAST or GET PRIOR command for a primary file which you are not accessing with a secondary key. If you try, RDM returns an error.

Insertion or deletion of some records in a PDM primary file can change the physical position and relative physical order of other records. Therefore, if you are serially processing a primary file while you or another user is inserting or deleting records in the file, you may retrieve some records multiple times or not at all.

Compound control keys

You may want to include compound control keys for primary files in your view. These keys are derived from a previous file access, constant value, or user-supplied field, which are concatenated to form the key's value.

To define a compound control key, use a USING phrase on the primary file ACCESS statement, for example, ACCESS PRI1 USING ('A',SUB-PART-2,SUB-PART-3). Parentheses indicate the beginning and ending of the compound control key or a subcomponent of the compound control key. Compound control keys may be nested to any level as long as the parts of the compound key correspond to the control key definition in the Directory. The following is an example of a compound control key.

Directory definition:	Length of field:
01 PRI1CTRL	20
02 SUB-PART-1	7
02 SUB-PART-2	9
03 SUB-PART-2A	5
03 SUB-PART-2B	4
02 SUB-PART-3	4

If you want a view which returns only those columns from the file PRI1 which have a first character of A in the control key, use the following ACCESS statement:

```
ACCESS PRI1 USING ('A',SUB-PART-2, SUB-PART-3)
```

If you also want to restrict access to only those columns with SUB-PART-2A equal to THURS, then the ACCESS statement should be:

```
ACCESS PRI1 USING ('A', ('THURS', SUB-PART-2B), SUB-PART-3)
```

Notice that each left parenthesis indicates a further subdefinition of the key column and a right parenthesis indicates the end of the subdefinition.

When you use compound control keys, remember that the fields used as sources for parts of the keys must be the same length as the Directory definition for the part, for example, any field used as a source for SUB-PART-3 must be a length of four. No part of a compound key can be invalid or null.

If you leave out part of a compound control key, it ceases to be a control key and remains only a logical key. For example,

```
ACCESS PRI1 USING ('A',('THURS'), SUB-PART-3)
```

leaves out SUB-PART-2B. RDM cannot access a PDM primary file directly without a complete control key, so it must sweep the file searching for a match for the fields in the compound logical key.

Related file considerations

This section discusses special considerations for using PDM related files with RDM.

In most cases, the relationship between a related file and a primary file represents a one-to-many relationship. For example, records in the ORDR file are associated to records in the CUST file. You can have customers with many orders, but you cannot have orders without them being linked to a customer.

You can create related files with either noncoded record formats or coded record formats. You can use coded records to build hierarchical structures within a file, such as header records, comment records, etc. For more information on PDM files, refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250.

Related file data records contain some special physical fields: linkpath fields, referback fields, and (in coded files only) record codes. You may not change these fields with an RDM update; RDM returns an error if you try.

This section describes the various methods you can use to navigate through related files. Which methods you choose depends on the logical relationships of the related records to each other and to primary file records.

Standard processing of related files

In the most common case, a primary file record has a one-to-many relationship with related file records. When retrieving occurrences of a view which accesses data in both the primary and related files, the program logic should retrieve data in a one-to-many relationship. In this case, your ACCESS statement should look like this:

```
ACCESS CUST USING CUST-NO
ACCESS ORDR VIA CUST-LINKPATH-05 (External field name of a
linkpath field)
```

or, to express the identical view using generalized access, like this:

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

Using primary file extensions

You can use a related file as an extension to a primary file. A primary file data record would relate via a linkpath to one (at most) data record in the extension related file. The related data record contains information which could just as well, logically, be contained in the primary record. The related records would have a one-to-one relationship with primary records. In other words, each linkpath chain in the extension file contains only one record.

For example, the customer data file (the related file CUDT) is used as an extension to the customer file (the primary file CUST). All the fields in CUST and CUDT could have been defined into CUST alone when the database was designed; but the extension arrangement can save space if there are a significant number of customers to whom the CUDT fields do not apply. To access data contained in the extension file, the ACCESS statement should look like this:

```
ACCESS CUST
ACCESS CUDT ONCE VIA CUST-EXTENSION-LINK (External field name of
a linkpath field)
```

A GET request on this view uses the linkpath defined by CUST-EXTENSION-LINK to access a record from CUDT, but only once for each occurrence of a record from CUST. An INSERT adds either the CUST record or the CUDT record, or both, as defined through the use of the ALLOW clauses.

Multiple related file usage

RDM can navigate from one primary file to multiple related files. If a primary record relates to more than one linkpath chain, and you specify the multiple linkpaths in your ACCESS statements, RDM scans the linkpath chains in parallel, with the occurrences in the view ending when the longest linkpath chain is exhausted.

The following example retrieves one record from CUST, and then scans the CUDT and ORDR linkpath chains in parallel in the second level of hierarchical search.

```
ACCESS CUST
ACCESS CUDT VIA CUST-EXTENSION-LINK
ACCESS ORDR VIA CUST-LINKPATH-05
```

If the linkpath chains are not the same length, RDM might reread previously read occurrences along the shorter linkpath as you continue down the longer one. For example, if the linkpath chain in CUDT above contains 10 records but the one in ORDR contains only 5, the first 5 GET NEXT commands read the first 5 records on each linkpath chain. The 6th GET reads the 6th record in CUDT but reports end-of-chain in ORDR, returning no data to columns supplied by ORDR. The 7th GET reads the 7th record in CUDT but starts over again with the first record in ORDR. In short, a view like this implies that the linkpath chains in the related files are, in fact, equal length. RDM detects no errors if this condition is not true, but your processing may not properly reflect your business model.

Inserting coded records

When RDM processes an INSERT RDML against a coded related file, the record code must be supplied in the physical record before RDM passes it to PDM. You can supply the record code as a column in the view, preferably as a KEY or NONUNIQUE KEY column. Alternatively, you can supply the record code on the ACCESS statement, in the form ACCESS file(rc). If you do neither, RDM selects one of the record codes defined for the file on the Directory. Since you have no control over this selection, it is essentially random and most unlikely to be the one you intended.

Retrieving coded records in occurrence order

You can retrieve data from several coded records in the order they appear on their linkpath chain. The resulting view may contain columns drawn from data records with different record codes, subject to the restriction described in the next paragraph. RDM reads related records in their order of occurrence by default unless you specify record code information in your ACCESS statement.

RDM navigates a linkpath by using the referback on the linkpath as the key. You can add records with different record codes to the same linkpath. If the referback on this linkpath is in the base portion of the coded records in the file, there is no problem. But if the referback is in the coded portion, the Directory and PDM permit you to define the referback at any offset within the coded portion. However, in such a case, RDM cannot properly determine the location of the referback key field for all record codes. RDM selects the first valid referback it can find in the Directory for that linkpath and uses it for all the record codes on that linkpath. (Since SUPRA does not define the concept of a "first," "second," "third," etc. record code, RDM's selection is essentially at random.) If the referback is in a different location in each coded portion, retrievals can be correct only for one of the record codes and are doomed to failure on all the others. If you intend to process a linkpath chain that connects related records from multiple record codes, you must ensure that the referback on that linkpath is at the same relative location in all the connected coded records. You do this by placing the referback in the base portion of the record, or by placing the referback at the same offset within each coded portion of the record. The only exception to this rule comes if you explicitly provide a single record code in the ACCESS statement (i.e., ACCESS file(rc)). In this case, since you are telling RDM to focus on just one of the record codes on the linkpath chain, the offset of the referback within the coded portions of the other record codes does not matter.

RDM reads and skips records on the linkpath chain until it finds a record that contributes a data value for some column in the view, includes any fields required by the view, and meets any selection criteria for the view; RDM then returns the necessary fields from that record.

Retrieving related records by record code

Another way to process a linkpath chain of coded related records is to skip through them searching for a record with a specific record code or with one of a list of record codes. You can specify this type of processing by putting a list of record codes (in parentheses) after the file name in the ACCESS statement, as shown:

```
ACCESS CUST  
ACCESS PROD (HD,CM) VIA CUST-TO-ORDERS
```

These statements cause RDM to read the PROD file until it finds either an HD or a CM record. At that point, the view retrieves whatever data is provided in that record. When inserting a record in the view, you must specify the record code in the view. See [“Retrieving coded records in occurrence order”](#) on page 59 for restrictions on multi-record code linkpaths.

If you use the WHERE clause on your ACCESS statement, you may implicitly access multiple record codes in this manner only if all of the fields in the WHERE clause are base fields. See [“Accessing related files”](#) on page 38 for more information.

Retrieving related records with embedded one-to-one relationships

A coded related file contains different types of records, the types being marked with different record codes. These different record types can have logical relationships with each other. Such relationships are called embedded because they exist between records in the same physical file.

In an embedded one-to-one relationship, a record of one type is used as an extension to a record of another type. This is analogous to the relationship between a primary record and its extension related record, as explained in “[Using primary file extensions](#)” on page 57. For example, suppose the coded file REL1’s record codes include HD and EX. An EX record is always an extension to an HD record, and it always follows its HD record on the PRI1LKXX linkpath chain. Following an HD record (the current record) on the chain, and before the next HD record (if any) or the chain’s end, there is either one EX record or none; if there is one, it is the current HD record’s extension. The following ACCESS statements access information from an HD record and from its extension (if any).

```
ACCESS PRI1
ACCESS REL1 (HD) VIA PRI1-LINKPATH-XX
ACCESS REL1 (EX) FROM REL1 (HD) ONCE VIA PRI1-LINKPATH-XX
```

or

```
ACCESS PRI1
ACCESS REL1 (HD+EX) VIA PRI1-LINKPATH-XX
```

The coded records are grouped in clusters with an HD record being the first in each cluster and an EX record optionally occurring after that record. You can think of each cluster as a single occurrence in which the retrieval of an HD record should trigger an attempt to retrieve one occurrence of an EX record.

You may access other records in the file at the same time you access these records with the one-to-one relationships. For example, if, in addition to HD and EX records, you want a random BR record retrieved when it is found, use the following syntax:

```
ACCESS CUST
ACCESS ORDR (HD,BR) VIA CUST-TO-ORDERS
ACCESS ORDR (EX) FROM PROD (HD) ONCE VIA CUST-TO-ORDERS
```

or

```
ACCESS CUST
ACCESS ORDR (HD+EX,BR) VIA CUST-TO-ORDERS
```

See “[Retrieving coded records in occurrence order](#)” on page 59 for restrictions on multi-record code linkpaths.

Retrieving related records with embedded one-to-many relationships

In an embedded one-to-many relationship, a record of one type relates to a group of records of another type. This is analogous to the one-to-many relationship between a primary record and the related records on its linkpath chain. For example, suppose the coded file ORDR's record codes include PO (purchase order) and IT (order item). PO records have a one-to-many relationship with IT records. An IT record always relates to a PO record, and it always follows its PO record on the CUSTLK05 linkpath chain. Following a PO record (the current record) on the chain, and before the next PO record (if any) or the chain's end, there are zero or more IT records; if there are any, they relate to the current PO record. The following ACCESS statements access information from each IT record and from its parent PO record.

```
ACCESS CUST
ACCESS ORDR (PO) VIA CUST-LINKPATH-05
ACCESS ORDR (IT) FROM ORDR (PO) VIA CUST-LINKPATH-05
```

or

```
ACCESS CUST
ACCESS ORDR (PO:IT) VIA CUST-LINKPATH-05
```

This reads the ORDR file along the CUSTLK05 chain, and starts with the specified customer's first record until a PO record is found. The chain is then read until either an IT or a PO record is found. If an IT record is found, the row containing PO and IT fields is returned. The search repeats, returning one row for each IT record, until all IT records for the particular PO have been accessed. The next occurrence of a PO record marks the end of that group of order records.

See [“Retrieving coded records in occurrence order”](#) on page 59 for restrictions on multi-record code linkpaths.

Retrieving related records with embedded many-to-one relationships

In an embedded many-to-one relationship, a group of records of one type relate to a record of another type. This is analogous to the many-to-one relationship between the related records on a linkpath chain and the primary record for that chain. For example, suppose the coded file ORDR's record codes include PO (purchase order) and IT (order item). IT records have a many-to-one relationship with PO records. An IT record always relates to the PO record most closely preceding it on the CUSTLK05 linkpath chain. More than one IT record can relate to the same PO record.

An embedded many-to-one relationship is a one-to-many relationship seen from the opposite perspective. “[Retrieving related records with embedded one-to-many relationships](#)” on page 62 shows the one-to-many relationship between purchase order records (records with the record code PO on the related file ORDR) and order items (IT records on ORDR). The example there showed access navigation from customer to customer's purchase order to order item. This section's example shows navigation in the opposite direction through the same relationship: from product to order item to purchase order.

```
ACCESS PROD
```

```
ACCESS ORDR (IT) VIA PROD-LINKPATH-06
```

```
ACCESS ORDR (PO) FROM ORDR (IT) SCAN REVERSE VIA CUST-LINKPATH-05
```

If the last two ACCESS statements above used the same linkpath, you could combine them into one statement using the ";" syntax, for example, (IT;PO).

This accesses the ORDR file using the linkpath between PROD and ORDR. Once an IT record is found on this linkpath chain, RDM switches linkpaths (see “[Switching linkpaths](#)” on page 64) and scans the CUSTLK05 linkpath chain for a PO record. The REVERSE keyword is included because the PO record precedes the IT record on the CUSTLK05 linkpath chain.

See “[Retrieving coded records in occurrence order](#)” on page 59 for restrictions on multi-record code linkpaths.

Switching linkpaths

A related file record can be a member of more than one linkpath chain. Views are not limited to accessing a single linkpath when processing coded related files. The example in “[Retrieving related records with embedded many-to-one relationships](#)” on page 63 shows where switching linkpaths can be useful. The only constraint on linkpath switching is that the record being switched from must be on both linkpath chains. In the previous example, the IT record type contains both the PRODLK06 and the CUSTLK05 linkpaths, so RDM could switch linkpaths at that record. However, the PO record type contains only the CUSTLK05 linkpath; therefore, switching cannot occur there.

Inserting related records with embedded relationships

When you insert a related record, and the record’s type is the destination of navigation through an embedded relationship, the values in the inserted record’s base fields default to those in the parent record’s base fields. For example, if you insert a record with record code IT, and the relevant ACCESS statement looks like this,

```
ACCESS ORDR (IT) FROM PROD (PO) ...
```

then RDM accesses the existing PO record (the parent record) first. If the parent record does not exist, your insert fails. By default, RDM gets the values for the base fields for the inserted IT record from the base fields of the PO record. If a base data field is provided for the inserted record in the view, the value from the row overrides the base data copied from the parent record.

Optimizing performance

The most efficient way to access a particular record on a linkpath chain is by a secondary key. If you use a logical key which is not a secondary key, RDM must search the linkpath chain to find that key value. However, you should let RDM do the search since it is more efficient than an application program.

Assigning unique keys to related file records has additional implications for performance. When you insert a record into a related file that has a unique logical key, RDM must search the entire linkpath chain to make sure that there are no duplicate keys. A nonunique key does not present this problem. Nevertheless, if you need to maintain your key's uniqueness on the linkpath chain, it is still more efficient to let RDM do it than to have the program check all the records on the linkpath chain.

When a search of a related file linkpath chain is required, ordering the linkpath chain (using the ORDER phrase in the ACCESS Definition - see ["Accessing related files"](#) on page 38) can improve performance. If you maintain the linkpath chain in a particular order, RDM can then optimize the search because it can determine where the record should be in the linkpath and does not have to search the entire linkpath chain.

It is less efficient for the application program to search the file than it is for RDM to do the search. Therefore, if the application program needs to search on values that are not physical keys, it is better to define them as logical keys and let RDM do the searching.

A generalized ACCESS statement (one with the WHERE clause but no VIA or USING clause) may result in a time-consuming sweep of a file. Use RDM statistics and the DBAID SHOW-NAVIGATION command to determine the efficiency of your view and to decide whether to define a new secondary key.

Maintaining foreign keys

A referback field in a related file is a foreign key and is treated like a foreign key by the PDM; the PDM automatically enforces referential integrity for referback fields. Nevertheless, 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.

Secondary key considerations

This section describes the use of secondary keys with RDM. The DBA can define any physical field or combination of fields within a PDM file as a secondary key. Secondary keys:

- ◆ Need not be unique
- ◆ Need not be contiguous in the data record
- ◆ May be accessed in either ascending or descending order
- ◆ Are fully recoverable under all recovery methods supported by the PDM
- ◆ May be used as an alternative to accessing a file using a control key or through a linkpath
- ◆ Allow you to build views which present data in the order of the secondary key
- ◆ Are defined on the Directory using the SK category
- ◆ Are stored in an index file

To use a secondary key, define it in the Directory, populate it, make it eligible to RDM, then include it in the ACCESS statement (see “[Accessing PDM files using secondary keys](#)” on page 48). Refer to the [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, and [SUPRA Server PDM & Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250, for information on defining a secondary key.

The ACCESS statement for secondary keys allows you to use the REVERSE keyword (see “Using the REVERSE keyword” on page 68) and the ONCE keyword (see “Using the ONCE keyword” on page 68). You may also use GET PRIOR and GET LAST to change directions (see “GET PRIOR/LAST command” on page 68).

The following example shows a simple unique secondary key. This view allows access to the CUST file with records ordered by the CUST-CITY value. CUSTSK01 is a secondary key based on the CUST-CITY field.

```
KEY CUST-CITY
CUST-NO
CUST-NAME
CUST-ADDRESS
ACCESS CUST VIA CUSTSK01 USING CUST-CITY
```

The following GET command would retrieve data from the CUST file using the secondary key.

```
GET FIRST CUST-VIEW USING 'CINCINNATI'
```

For more examples of views using secondary keys, see “Sample PDM database and base views” on page 71.

Using generic secondary keys

You can use generic secondary keys instead of full secondary keys to access PDM files. Any combination of fields that are contiguous in the key, and that include the leftmost character in the key, is a generic key. A generic key need not be unique. The use of generic secondary keys does not require a sweep of the file. In the example below, we use a generic key derived from the secondary key ORDRSK01. We have defined ORDRSK01 itself as consisting of the fields ORDR-CUST-NO (customer number) and ORDR-PO-NUMBER (purchase order number) in that order.

```
KEY CUST-NO
KEY ORDR-PO-NUMBER
ORDR-NAME
ORDR-ADDRESS
ACCESS CUST USING CUST-NO
ACCESS ORDR VIA ORDRSK01 USING CUST-NO
```

This view performs a keyed read on the CUST file, but a generic read is performed on the ORDR file through the secondary key, ORDRSK01, since ORDR-PO-NUMBER was not specified in the USING phrase.

Using the REVERSE keyword

You can specify that RDM access the secondary key in reverse order by including the REVERSE keyword on the ACCESS statement (see “Accessing PDM files using secondary keys” on page 48).

The following view returns the customer numbers in random order and the purchase order numbers in reverse order. Since the index is stored in ascending order by default, the purchase order numbers are returned in descending order.

```
KEY CUST-NO
KEY ORDR-PO-NUMBER
ORDR-NAME
ORDR-ADDRESS
ACCESS CUST USING CUST-NO
ACCESS ORDR REVERSE VIA ORDRSK01
USING (CUST-NO, ORDR-PO-NUMBER)
```

Using the ONCE keyword

You can establish a one-to-one relationship with the previous record by using the ONCE keyword on the ACCESS statement. The following view returns only the first ORDR record found since it accesses the ORDR file only once through the secondary key. A subsequent GET would move to the next CUST record and then get only the first ORDR record based on the new secondary key value.

```
KEY CUST-NO
KEY ORDR-PO-NUMBER
ORDR-NAME
ORDR-ADDRESS
ACCESS CUST USING CUST-NO
ACCESS ORDR ONCE VIA ORDRSK01
USING (CUST-NO, ORDR-PO-NUMBER)
```

GET PRIOR/LAST command

You can use the GET command in application programs or with DBAID to retrieve rows. Use the GET command in conjunction with the LAST, FIRST, PRIOR, and NEXT clauses to penetrate and position within a view. For example,

```
GET LAST CUST USING 'CINCINNATI'

or

GET PRIOR CUST USING 'CINCINNATI'
```

The PDM does not support GET PRIOR or LAST with primary files unless you specify a secondary key on the ACCESS statement.

3

User exits

Several exits are available to you which allow you to insert processing routines before and after Physical Data Manager (PDM) calls. You can use the exits to:

- ◆ Bypass database calls
- ◆ Perform your own database or user file calls
- ◆ Satisfy any special requirements for your system

There are two levels in your processing where you can insert routines:

- ◆ The RDM processing level which is environment independent
- ◆ The PDM call processing level which is environment dependent

Values for the PDM function parameter include the following:

ADD-M	Add primary record
ADDVA	Add related record next on chain
ADDVB	Add related record previous on chain
ADDVC	Add related record at end of chain
ADDVR	Replace related record
DEL-M	Delete primary record
DELVD	Delete related record
FINDX	Read next qualified record (either primary or related)
RDNXT	Read next record (either primary or related)
READD	Read related record directly
READR	Read related record previous on chain
READV	Read related record next on chain
READX	Read record (either primary or related) via secondary key
WRITM	Update primary record
WRITV	Update related record

For more information on all of the available RDM user exits, and on using the function parameter value, refer to the *SUPRA Server PDM RDM Administration Guide (OS/390 & VSE)*, P26-8220, and the *SUPRA Server PDM & Directory Administration Guide (OS/390 & VSE)*, P26-2250.

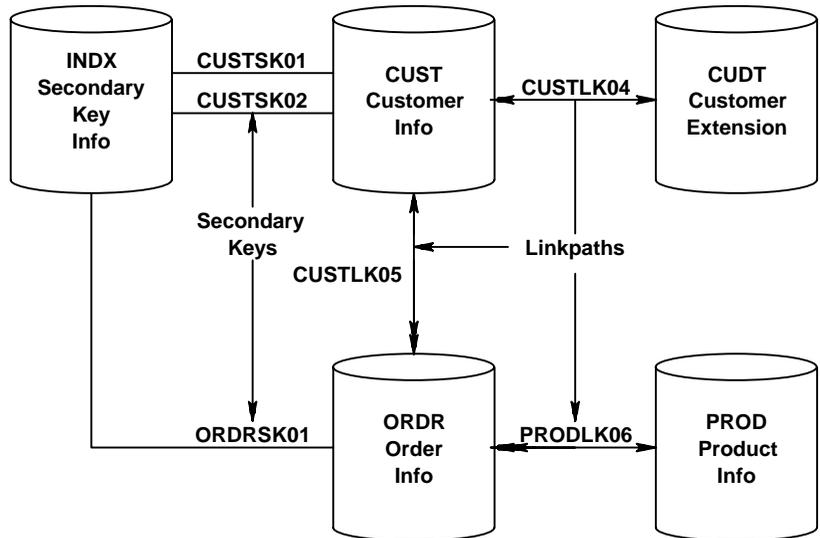
A

Sample PDM database and base views

This Appendix describes a sample database and shows some base views which access the database.

Sample database

The following figure illustrates the sample database. The double arrows drawn between the files indicate many-to-one relationships. Following the diagram are descriptions of the files.



The sample database consists of the following:

- ◆ **CUST file.** A PDM primary file which contains customer information such as addresses, city, phone number, etc. The control key to this file is the customer number (CUST-NO). The secondary key CUSTSK01 is defined as the field CUST-CITY. The secondary key CUSTSK02 is defined as the field CUST-NO (same field as the control key). The file's data records contain the following 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
CUSTPHNE	CUST-PHONE	PHONE
CUSTZIPC	CUST-ZIP	ZIP
CUSTLK04	CUST-EXTENSION-LINK	
CUSTLK05	CUST-TO-ORDERS	

- ◆ **ORDR file.** A PDM coded related file which contains order information such as customer number and list price. The linkpath referback fields for this file are ORDR-CUST-NO for linkpath CUSTLK05, and ORDR-PROD-NUMB for linkpath PRODLK06. The secondary key ORDRSK01 is defined as the concatenation of ORDR-CUST-NO and ORDR-PO-NUMBER. Which linkpath keys and other fields are present in a record depend on the record code for that record. The record codes for this file are:

- PO—Purchase Order
- CN—Contact
- IT—Item

The file's data records contain the following fields:

Physical field	External field	Column name	RC
ORDRCODE			ALL
ORDRCUNO	ORDR-CUST-NO	CUST-NUMBER	ALL
CUSTLK05	CUST-LINKPATH-05		ALL
ORDRCONT	ORDR-NAME	CONTACT-NAME	CN
ORDRADDR	ORDR-ADDRESS	CONTACT-ADDR	CN
ORDRTITLE	ORDR-TITLE	CONTACT-TITLE	CN
ORDRPHON	ORDR-PHONE	CONTACT-PHONE	CN
ORDRPONO	ORDR-PO-NUMBER	PO-NO	PO
ORDRITEM	ORDR-ITEM	ITEM	IT
ORDRPROD	ORDR-PROD-NUMB	PROD-NO	IT
PRODLK06	PROD-LINKPATH-06		IT

- ◆ **PROD file.** A PDM primary file which contains product information such as the product number and sales price for an item. The key for this file is the product number (PRODCTRL). The file's data records contain the following fields:

Physical field	External field	Column name
PRODNUMB	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
PRODLK06		

- ◆ **CUDT file.** A PDM related file that is a one-to-one extension of the CUST file and contains customer credit information. The linkpath referback field for this file is CUDT-CUST-NO for linkpath CUSTLK04.

Physical field	External field	Column name
CUDTCUNO	CUDT-CUST-NO	CUST-NO
CUDTCRED	CUDT-CREDIT-RATING	CREDIT-RATING
CUDTCREL	CUDT-CREDIT-LIMIT	CREDIT-LIMIT
CUDTCREU	CUDT-CREDIT-USED	CREDIT-USED
CUSTLK04	CUST-LINKPATH-04	(none)

- ◆ **INDX file.** An index file which contains secondary key information for the secondary keys CUSTSK01, CUSTSK02, and ORDRSK01.

Example view definitions

The following examples describe views which access the files shown in “[Sample database](#)” on page 71:

Example view 1. This view shows a direct keyed read using the customer number as the key. The ALLOW clause allows the replacement, deletion, and insertion of new customers.

```

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

Example view 2. This view is identical to the view shown above but it uses the generalized access syntax (the WHERE clause) on the access statement. RDM determines the most efficient access strategy and performs a direct-keyed read as in the above view.

```
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 3. This view retrieves every customer who resides in California. In this view, 'CA' is used as a constant so you must place it in quotes. This view results in a scan of the customer file; RDM must look at every record to see if the state field contains CA. It would be more efficient to define an index on the state field; then RDM could use a secondary key to retrieve only the records you need without scanning.

```
KEY CUST-NO
  CUST-NAME
  CUST-CITY
ACCESS CUST WHERE CUST-STATE = 'CA'
ALLOW ALL
```

Example view 4. This view is the same as the above view except that it specifies an additional condition which must be met before a record is retrieved: The customer must reside in Los Angeles, California. This view would be more efficient if you defined a secondary key on the concatenation of the city and state fields in either order because then RDM could access the records directly without scanning the database. If an index is not available for the city/state concatenation, RDM attempts to use an index on city or state, if available.

```
KEY CUST-NO
  CUST-NAME
ACCESS CUST WHERE CUST-STATE = 'CA'
  AND CUST-CITY = 'LOS ANGELES'
ALLOW ALL
```

Example view 5. This view shows the WHERE clause used to specify additional selection criteria (state and city) when accessing a file with the USING clause to specify a direct keyed read. If you don't specify a logical key value on the 'GET' for the view, RDM scans the file while searching for a city/state match.

```
KEY CUST-NO
ACCESS CUST USING CUST-NO
WHERE CUST-STATE = 'CA'
AND CUST-CITY = 'LOS ANGELES'
```

Example view 6. This view shows the use of the coded record ORDR(IT). Note that although there are no columns from the ORDR file, a DELETE causes the PROD file record and all records on the ORDR file on the linkpath PRODLK06 that are chained to the PROD record to be deleted.

```
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(IT) VIA PROD-LINKPATH-06
ALLOW DELETE
```

Example view 7. This view is identical to the above view, but it uses the generalized form of the access syntax which enables RDM to determine the access strategy.

```
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(IT) WHERE ORDR-PROD-NUMB = PROD-NUMB
    ALLOW DELETE
```

Example view 8. This view shows the use of two keys. It uses the customer number to scan through the order list to get all orders associated with a given customer. An INSERT can succeed only if the customer exists.

```
KEY CUST-NO
KEY CONTACT-NAME = ORDR-NAME
    CONTACT-TITLE = ORDR-TITLE
    CONTACT-PHONE = ORDR-PHONE
ACCESS CUST USING CUST-NO
ACCESS ORDR(CN) VIA CUST-LINKPATH-05
    ALLOW ALL
```

Example view 9. This view shows a more complicated view than the preceding three. It shows which customers ordered each of the products and shows the purchase order numbers. The view uses a physical field instead of an external field (ITEM column). Once the IT record is located on the ORDR file, the file is scanned in REVERSE order along the CUSTLK05 linkpath. The two ACCESS statements to ORDR show an example of switching linkpaths.

```
KEY PROD-NUMB
    DESCRIPTION = PROD-DESC
KEY CUST-NUMBER = ORDR-CUST-NO
    NAME = CUST-NAME
    PO-NO = ORDR-PO-NUMBER
    ITEM = ORDRITEM
ACCESS PROD USING PROD-NO
ACCESS ORDR(IT) VIA PROD-LINKPATH-06
ACCESS ORDR(PO) FROM ORDR(IT) SCAN REVERSE VIA CUST-LINKPATH-05
ACCESS CUST USING CUST-NUMBER
```

Example view 10. This view allows the insertion of items into a purchase order. Note the ordering of purchase orders by purchase order number, and the ordering of items within a purchase order by item number.

```
KEY CUST-NO
KEY PO-NO = ORDR-PO-NUMBER
KEY ITEM = ORDRITEM
    PROD-NUMBER = ORDR-PROD-NUMB
    NAME = CUST-NAME
    DESCRIPTION = PROD-DESC
ACCESS CUST USING CUST-NO
ACCESS ORDR(PO) VIA CUST-LINKPATH-05 ORDER PO-NO
    ALLOW ALL
ACCESS ORDR(IT) FROM ORDR(PO) VIA CUST-LINKPATH-05 ORDER ITEM
    ALLOW ALL
ACCESS PROD USING PROD-NUMBER
```

Example view 11. This view shows the use of a compound logical key. In this example, the view is accessed by city and customer name within the city. CUSTSK01 is a secondary key based on the CUST-CITY field. The application has two logical keys available, CUST-CITY and CUST-NAME. Either of the keys may be used for selection.

```
KEY CUST-CITY
    CUST-NO
KEY CUST-NAME
    CUST-ADDRESS
ACCESS CUST VIA CUSTSK01
    USING CUST-CITY
```

Example view 12. This view shows the navigation between files through secondary keys. The value to be used for the index is supplied by the previous file (CUST supplies CUST-NO) and a logical key (ORDR-PO-NUMBER). The view returns customer numbers in random order, but the purchase order numbers for the CUST-NO in question are returned in ascending order.

```
KEY CUST-NO
KEY ORDR-PO-NUMBER
    ORDR-NAME
    ORDR-ADDRESS
ACCESS CUST
ACCESS ORDR VIA ORDRSK01
    USING (CUST-NO, ORDR-PO-NUMBER)
```

Example view 13. If you want the customer numbers in order, your view should look like this:

```
KEY CUST-NO
KEY ORDR-PO-NUMBER
    ORDR-NAME
    ORDR-ADDRESS
ACCESS CUST VIA CUSTSK02 USING CUST-NO
ACCESS ORDR VIA ORDRSK01
    USING (CUST-NO, ORDR-PO-NUMBER)
```

CUSTSK02 is the secondary key on the CUST-NO physical key field.

Example view 14. The following view shows the use of a customer extension file. The CUDT file is a one-to-one extension of the customer file.

```
KEY CUST-NO
    CREDIT-RATING = CUDT-CREDIT-RATING
    CREDIT-LIMIT = CUDT-CREDIT-LIMIT
    CREDIT-USED = CUDT-CREDIT-USED
ACCESS CUST USING CUST-NO
ACCESS CUDT ONCE VIA CUST-LINKPATH-04
```

Index

A

ACCESS statement
described 20
for related files 38
generalized, considerations 31
generalized, for related files 38,
59
generated by NORMAL 31, 32,
38
using secondary keys 48
ACCESS Statement
for primary files 32
ALLOW clause 37, 53
ALLOW option 45

B

Base fields 18, 64
Base files 33, 54
Base views 13, 20

C

Coded fields 18, 60
Coded records 18
Column definition
described 20
format 21
Column definitions
considerations 28
examples 29
Columns, redundant. *See*
Field(s), redundant
Compound keys 34, 51, 55
CONST option 23, 26

D

Defining columns. *See* Column
definition
Derived views 13, 28
Direct read example 74

E

Embedded many-to-one
relationships 63
Embedded one-to-many
relationships 62
Embedded related records,
insertion and replacement
62
Environment independent exits
69
Example view definitions 74
Exits 69

F

Fields
base 18, 64
coded 18, 60
linkpath 18, 42
redundant 26
related 18
File access. *See* ACCESS
statement
File extension 62
Files
base 54
index 16, 17
PDM 13, 16, 19
primary 17, 54
primary, ACCESS 32
primary, extensions 57
related 56
related, ACCESS 38
related, as extensions 57
FKEY option 23
Foreign keys 18, 23, 65
FROM clause 41, 62

G

Generalized ACCESS statement.
See ACCESS statement,
generalized
Generate. 80. *See* NORMAL
GET PRIOR/LAST 54, 67, 68
GIVING clause 36, 52
GIVING option 44

I

Integrity 23
Intergrity 65

K

KEY option 21, 26, 28

Keys

- compound 34, 51, 55
- control 17
- foreign 18, 23, 65
- generic 67
- linkpath 17, 18
- logical 65
- nonunique 65
- physical 17, 26
- secondary 17, 66

L

Linkpath

- keys 18

Linkpath chains 18

Linkpath example 77

Linkpath fields 18, 42

Linkpath switching 64

Linkpaths 18

Logical key(s) 65

Logical views. *See* Views

M

Multiple related file usage 58

N

Navigation definition. *See*

ACCESS statement

NONUNIQUE KEY option 22

Nonunique key(s) 65

NORMAL

- general 13, 19
- generated ACCESS statement 38
- generated ACCESS statements 31, 32

O

ONCE keyword 68

ONCE option 42, 49

One-to-many relationships 42, 56

Optimizing performance 65

ORDER clause 46

P

PDM files 13, 16, 19

Performance optimization 31, 65

Physical keys 17, 26

Primary file extensions 57

Primary files. *See* Files, primary

R

RDM 13

Record code phrase 40, 41, 49

Records

- coded 46
- primary 17, 54
- related 18, 56
- related, relationships between 62

Redundant columns. *See*

Redundant fields

Redundant fields 26

Referential integrity 23, 65

Related files. *See* Files, related

Related records. *See* Records, related

Relational Data Manager (RDM) 13

Relationships

- between related records 62
- embedded 62
- one-to-many 42, 56

REVERSE keyword 68

REVERSE option 42

S

SCAN option 42

Secondary keys

- ACCESS statement 46
- defined 16, 17

Serial processing (file sweep) 31, 65

SSsecondary keys

- considerations 66

Statistics 31

Switching linkpaths 64

U

USING clause 34, 50

V

- VIA option 42
- Views
 - base 13, 20
 - base, example 74
 - derived 13, 28

W

- WHERE clause
 - general 31
 - primary files 32, 35
 - related files 38, 43
 - secondary keys 51

