# RDBMS File Access Guide

Release 8.1.3

November 2013


IKAN Solutions N.V.
Kardinaal Mercierplein 2
B-2800 Mechelen
BELGIUM

# Table of Contents

# CHAPTER 1
# About This Manual

MetaSuite file access for RDBMS is intended for users with some experience with MetaSuite. It provides the information you need to use the MetaSuite Relational Database File Access, including discussions on relational technology and defining a relational database to the MetaSuite MetaStore.

## 1.1. Prerequisites

Readers are expected to be familiar with their Relational Database Management System (RDBMS).

## 1.2. Related Publications

The MetaSuite User and Reference Guides describe the different MetaSuite components and provide examples for using MetaSuite. Those guides should be available for reference during the installation and test procedures described here.

The following table gives an overview of the complete MetaSuite documentation set.

| Release Information | Release Notes 8.1.3 |
|---|---|
| Installation Guides | • BS2000/OSD Runtime Component<br>• DOS/VSE Runtime Component<br>• Fujitsu Windows Runtime Component<br>• MicroFocus Windows Runtime Component<br>• MicroFocus UNIX Runtime Component<br>• OS/390 and Z/OS Runtime Component<br>• OS/400 Runtime Component<br>• VisualAge Windows Runtime Component<br>• VisualAge UNIX Runtime Component<br>• VMS Runtime Component |
| User Guides | • INI Manager User Guide<br>• Installation and Setup Guide<br>• Introduction Guide<br>• MetaStore Manager User Guide<br>• MetaMap Manager User Guide<br>• Generator Manager User Guide |

| | |
|---|---|
| Technical Guides | • ADABAS File Access Guide<br>• IDMS File Access Guide<br>• IMS DLI File Access Guide<br>• RDBMS File Access Guide<br>• XML File Access Guide<br>• Runtime Modules<br>• User-defined Functions User Guide |

If you are unfamiliar with MetaSuite, the following technical description provides you with a brief overview.

| | |
|---|---|
| **The MetaSuite System** | MetaSuite is designed for data retrieval, extraction, conversion and reporting. It includes a workstation-based graphical user interface and a mainframe runtime component. |
| **MetaSuite Database Interfaces** | MetaSuite can access data from a number of database management systems, using the same commands, program structure and retrieval techniques used for non-database files. Each database interface is available as an optional enhancement to the base product. |
| **MetaMap Manager** | MetaMap Manager is the MetaSuite tool used to define models. Such models are intuitively built by describing overall program specifications, input file definitions (data and process) and target file definitions (data and process). |
| **MetaStore Manager** | MetaStore Manager is a tool that provides metadata maintenance and documentation services. |
| **Generator Manager** | The Generator Manager is the system administration tool.All kinds of basic functionalities and customization possibilities are supported by this tool. |

CHAPTER 2

# MetaSuite File Access Overview

## 2.1. Overview

Because most MetaSuite commands are independent of the environment in which MetaSuite operates, only those commands that pertain to a relational database definition and access are described in this supplement. Additional chapters in this manual include:

# CHAPTER 3
# Relational Concepts and Terminology

## 3.1. Overview

This chapter presents a brief overview of the relational concepts and terms you should know before using MetaSuite to access a relational database. It is broken down as follows:

- Relational Data Structures discusses those relational entities that are pertinent to MetaSuite processing.
- Relational Data Retrieval provides an overview of relational data retrieval.
- Batch COBOL Processing provides an overview how to further process a generated program.

For more information on relational in general, refer to the specific RDBMS Documentation Set. Of particular use is the RDBMS Application Programmer's Guide.

## 3.2. Relational Data Structures

This section describes the basic data entities in relational and related topics. The topics covered are:

- Tables
- Views
- Data types
- Relational catalogue
- Creator
- Keys

Each topic is described separately below.

### Tables

In a relational database, data is available as a set of tables. A table is made up of columns and rows. Each column defines a specific piece of data, describing its data type, size, and other attributes. For example, in the EMPTAB table shown below, the columns are:

```
EMPNO, EMPNAME, DEPTCODE, SALARY and HIREDATE.
```

Each row contains one occurrence of data. For example, in the EMPTAB table shown below, there are three rows, one for each of the employees in this (very small) company.

The EMPTAB table illustrates a relational table, with its constituent rows and columns:

| EMPNO | EMPNAME | DEPTCODE | SALARY | HIREDATE |
|---|---|---|---|---|
| 01234 | ANNA SMITH | 003 | 29,333.00 | 19900104 |
| 12345 | GEORGE WHITE | 002 | 22,555.00 | 19880503 |
| 02538 | SUZANNE JONES | 003 | 19,000.00 | 19880504 |

In MetaSuite file access for RDBMS:

- A relational table is defined and referred to as a RECORD.
- A relational column is defined and referred to as a FIELD.

## Views

A view defines a set of rows and columns that may come from one or more tables. For data retrieval purposes, tables and views can be used interchangeably. The MetaSuite file access for RDBMS makes no distinction between tables and views. Both are defined and referred to as a RECORD.

## Column Data Types

Relational databases support column data types that may differ from those used in standard COBOL data processing. The following table summarizes the relational data types.

### Relational DATA TYPES

| Category | RDBMS Data Type | Description |
|---|---|---|
| Character | CHAR<br>BYTE | Character data |
| | VARCHAR<br>VARCHAR2 | Variable length character data |
| | LONG<br>RAW<br>LONGVARCHAR | Variable length character data |
| Graphic | GRAPHIC | Graphic data |
| | VARGRAPHIC | Variable length graphic data |
| Numeric | TINYINT | One byte binary integer data |
| | SMALLINT | half-word binary integer data |
| | INTEGER | full-word binary integer data |
| | BIGINT | double word integer data |
| | DECIMAL | Packed decimal data |
| | NUMBER | Packed decimal data |

| Category | RDBMS Data Type | Description |
|---|---|---|
| | FLOAT | Floating decimal point data |
| Date/Time | DATE<br>TIME<br>TIMESTAMP<br>DATETIME | Timestamp data |

The internal storage format for each data type is transparent to the user. MetaSuite file access for RDBMS supports the relational data types, and represents them in MetaSuite Definition Language (MDL) as equivalent standard data types. For a description of the MetaSuite file access for RDBMS relational data type correspondences, refer to the discussion of the ADD FIELD command in the chapter Defining A Relational Database To MetaSuite (page 12).

### Relational Catalogue

The relational catalogue contains definitions for all relational tables and views. The catalog itself is accessible as a set of tables or views, and can be read like any other table. Frequently, users work with views of the catalog rather than with the catalog itself.

The *Collect File* option in MetaStore Manager allows users to reference catalogue definitions directly rather than having to maintain the definitions (MDL) manually.

### Creator

The person who defines a table or view to the relational catalogue is the creator of that table or view. Before other users can access the table or view, the creator or the DBA must grant the authority to do so.

### Keys

Relational technology relies on the concept of a key to specify relationships between tables. Each table should have a column (known as a primary key) containing unique values. If a column has values that match values in a column of another table, the column of the other table is known as a foreign key. A foreign key may consist of one or more columns.

MetaSuite file access for RDBMS provides no formal mechanism to support the key concept. However, it does support the join operation with the SQL SELECT statement. This direct match capability allows a user to retrieve matching data from multiple tables or views. If shared values exist, application programmers can use that knowledge when constructing data retrieval requests.

## 3.3. Relational Data Retrieval

This section describes the basic elements of data retrieval in Relational that are relevant to the MetaSuite Relational Interface. The topics covered are:

- SQL (Structured Query Language)
- Join
- Access Method

Each topic is described separately below.

### SQL

Relational data is defined, updated, and accessed with SQL (Structured Query Language) commands. SQL commands can be entered online through SQL Online Query tools or embedded in programs written in languages such as COBOL.

Data retrieval is effected with the SQL SELECT command. A SELECT statement is used to retrieve one row at a time or sets of rows. The SELECT statement has a number of options, including the WHERE option to specify selection criteria and the ORDER BY option to specify the sequence in which the rows are retrieved.

The SELECT statement returns all rows that match the specified criteria. However, the retrieved rows can be presented to the program one at a time for processing, through the use of a cursor. SQL includes commands to declare and manage cursors.

MetaSuite file access for RDBMS includes a special version of the MetaSuite MetaStore SourceFile object that supports the SELECT statement, including the WHERE and ORDER BY options.

COBOL programs generated by the MetaSuite Generator using MetaSuite file access for RDBMS include embedded SQL statements. These SQL statements are static rather than dynamic, meaning that they are embedded in a MetaSuite application by the MetaSuite Generator, rather than having the user code them in-stream.

### Join

The SELECT statement can be used to retrieve matching data from multiple tables or views. Unlike a traditional file match in which all data is retrieved whether it matches or not, a join returns only matched data. The WHERE option of the SELECT statement determines the match criteria.

### Access Method

Relational itself selects the access method for any data retrieval, based on the program request. It may choose random retrieval based on an index, or it may choose sequential retrieval. The user's program can determine the sequence in which rows are returned, but it cannot control the actual access method used.

A MetaSuite application can request specific single rows or specific sets of rows.

## 3.4. Batch COBOL Processing

This section describes batch COBOL processing against a relational database. Relational data can be accessed online or in batch mode. In batch mode following steps need to be performed to access any relational data:

A batch COBOL program that includes embedded SQL statements to access Relational is processed in the following steps:

1. Pre-compile. The SQL statements in the COBOL program are syntax checked, modified, and prepared for compilation. The pre-compiler step produces a version of the original COBOL program with modified SQL statements.
2. Compile. The modified COBOL program produced by the pre-compiler goes through a standard COBOL compilation, producing a standard object module.
3. Link. The object module output by the compiler is linked to form a load module version of the program, which is stored as an executable.
4. Run. The executable module version of the program is executed.

The MetaSuite supplies the required scripts to execute all the above steps.

CHAPTER 4

# Using MetaSuite With A Relational Database

## 4.1. Overview

This chapter introduces the MetaSuite facilities that allow you to access a relational database from a MetaSuite application. You should be familiar with the relational concepts presented in chapter 2 before reading this chapter.

To access a relational database from a MetaSuite application, MetaSuite requires two things:

- Access to the relational data definitions necessary to process the data in the database. These definitions can come directly from the relational catalogue via the Collect File functionality in MetaStore Manager, prepared manually by creating a MDL (MetaSuite Definition Language) import – export text file or created manually using MetaStore Manager.
- Specific MetaMap Manager commands to define the actual data retrieval requests for the relational database. The generate option converts these commands into their SQL equivalents and produces a standard COBOL program with embedded SQL statements.

The remainder of this chapter presents a brief overview of the MetaSuite facilities for providing access to the relational data definitions and defining the data retrieval requests. This chapter also includes a description of the program generation process, as well as a summary of the differences between MetaSuite and relational terminology.

## 4.2. MetaSuite And Relational Terminology

All MetaSuite file access components use the same terminology when referring to data structures. The MetaSuite terminology, though, may differ from that of an individual DBMS. The relationships between the basic MetaSuite and relational terms are as follows:

### MetaSuite and Relational Terms

| MetaSuite | Relational |
|---|---|
| File | Arbitrary group of related tables |
| Record | Table/Row |
| Field | Column/Field |

The MetaSuite terms are described separately below, along with their relational correspondences. The MetaMap Manager term "SourceFilePath" is also described.

### File

In MetaSuite file access for RDBMS, an arbitrary group of Relational tables is defined as a set of records belonging to a particular file. There is no relational entity that corresponds directly to a MetaSuite file. For this reason, a file exists only as a definition in the MetaSuite MetaStore.

In MetaSuite file access for RDBMS, the file concept has several purposes, as follows:

- It provides a mechanism for joining tables.
- It is required in a MetaSuite application as the mechanism (via the SourceFile object) for controlling the processing for relational tables.
- It allows the retrieved data to be treated as a logical record.

### Record

In MetaSuite file access for RDBMS, a relational table is known as a record.

One occurrence of a row within a table is referred to as a record occurrence. Note, though, that the term "record" can be used to mean either a record type (that is, a table) or a record occurrence (that is, a row). The meaning in a given case depends on the context.

### Field

In MetaSuite file access for RDBMS, a relational column is known as a field. A field value is equivalent to a column value.

### SourceFilePath

In MetaSuite file access, related data is retrieved from the DBMS and presented to the MetaSuite application as a logical record. The logical record is then processed as though it where a single record occurrence from a simple sequential file. The SourceFilePath object controls the contents of the logical record, which is referred to as a path.

## 4.3. Data Definition Facilities

MetaSuite provides two separate data definition facilities for use in the relational environment:

- The Collect File functionality in MetaStore Manager provides MetaSuite with direct access to the data definitions stored in the relational catalogue to define database tables and columns to the MetaSuite MetaStore (as records and fields).
- Data definitions can be created manually in the MetaStore Manager.

Please refer to the chapter Defining A Relational Database To MetaSuite (page 12) for detailed instructions on how to define MetaSuite file definitions for relational databases and to the *MetaStore Manager User Guide* for more information on collecting files.

# 4.4. Programming Overview

MetaSuite programs have the same structure and report processing capabilities regardless of the file organization used. In other words, their structure and report processing capabilities are the same whether they are used to access DBMS or non-DBMS files.

Most of the MetaMap Manager commands with which you are already familiar can be used to process a relational database. In certain cases, these commands operate differently when processing a database than when processing a non-database file.

When used with a relational database, the MetaSuite Generator builds an SQL SELECT command from each MetaMap Manager SourceFile object that names a relational file in a MetaSuite application. Multiple SourceFile objects for relational files can be used in a single application program. The same matching, controlling, and buffering capabilities are used with a relational "file" as with a non-database file.

Please refer to the chapter Programming With MetaSuite File Access (RDBMS) (page 21) for detailed instructions on using the MetaMap Manager commands that access a Relational database.

## Embedded SQL Statements

In processing a program constructed with MetaSuite file access for RDBMS, the MetaSuite Generator converts MetaMap Manager data retrieval commands to equivalent SQL statements, which it embeds in the resulting COBOL source program.

The MetaSuite Generator produces an SQL SELECT statement to access the table(s) specified in the SourceFile object. The selection criteria can be defined by WHERE criteria. The sequence in which rows will be returned can be determined by the optional ORDER BY criteria. The WHERE and ORDER BY options of the MetaMap Manager SourceFile object are similar in use and coding to the SQL WHERE and ORDER BY options.

If the SourceFilePath object names multiple tables, than the generated SQL SELECT statement will cause the tables to be joined during execution. Additional SQL statements, such as cursor control commands, are generated as well.

## Extended MetaSuite Facilities

The SourceFile object offers the following expanded file-based retrieval and processing, beyond the joining capabilities described above:

- File matching. The MATCH option of the SourceFile object allows relational files (that is, data returned by relational) to be matched with relational or non-relational files.
- Controlled retrieval. The CONTROLLED option of the SourceFile object in conjunction with the GET command allows an individual row to be retrieved randomly. WHERE criteria must be specified to identify an individual row.
- Controlled by retrieval. The CONTROLLED BY option of the SourceFile object allows sets of rows to be retrieved randomly. WHERE criteria must be specified to identify an individual row.

## What the Program Sees

What a MetaSuite application sees is a logical record (buffer or path) consisting of one row returned by the Relational SELECT statement or columns from any tables joined by the SELECT statement. The relational SourceFilePath differs from that of any other SourceFilePath only in that the SELECT statement always returns matching data when tables are joined. In other words, you never have to worry about invalid data due to lack of a current occurrence of a named record.

## Multiple Databases

MetaSuite can access up to 100 files in one program. These files can include relational, and other non-DBMS files.

## Generated Programs

This section describes the operation of the MetaSuite Generator when used with MetaSuite file access for RDBMS.

Before generating a model which accesses a RDBMS, you must set up the MetaSuite Generator properly: The SQL dialect must be set. For more information, please refer to *Generator Manager User Guide*.

CHAPTER 5

# Defining A Relational Database To MetaSuite

## 5.1. Overview

This chapter describes how to provide MetaSuite with access to the definitions that it needs to process data in a relational database.

Note that the descriptions in this chapter use both the MetaSuite and relational terminology for data entities, as appropriate. The correspondences between the MetaSuite and relational terminology are discussed in detail in the chapter Using MetaSuite With A Relational Database (page 8). These correspondences are summarized in the following table.

### Comparing MetaSuite and Relational Data Structure

| MetaSuite | Relational |
|---|---|
| File | Arbitrary group of related tables |
| Record | Table/Row |
| Field | Column/Field |

Before you can access data in a relational database, you must provide MetaSuite with access to the definitions necessary to process the data. Relational versions of the ADD FILE, ADD RECORD, and ADD FIELD commands are provided for defining Relational files to the MetaSuite MetaStore. However, an easier approach is to use the *Collect File* option in MetaStore Manager to copy definitions out of the relational catalogue, and then load the copied definitions into the MetaSuite MetaStore.

## 5.2. Defining Databases Manually

This section describes how to define a relational database to the MetaSuite MetaStore manually using the MetaSuite Definition Language (MDL). To define a relational database to the MetaSuite MetaStore manually using the MetaStore Manager facilities, refer to the MetaSuite User and Reference Guides for more details.

Before defining a Relational database to the MetaSuite MetaStore, you should obtain the necessary table and column information from the Relational catalog.

**Note:** Use of the manual coding method is not recommended. It requires careful translation of the relational definitions into MetaSuite definitions. As a result, it is more subject to error than using the *Collect File* option in MetaStore Manager.

## Commands

The following table lists the commands used to manually code relational data definitions.

### MetaSuite Commands that Define Relational Data Structures

| Command | Used to define |
|---|---|
| ADD FILE | The file that logically groups one or more tables |
| ADD RECORD | A Relational table |
| ADD FIELD | A Relational column |

Each command is described separately below.

# 5.3. ADD FILE

The ADD FILE command defines a relational file to the MetaSuite MetaStore. The general syntax is described in the *Definition Language Reference Guide*. The options that refer to relational tables are described below.

The ADD FILE command is used simply to define a logical group of tables (records) that will be accessed together (joined) in a MetaSuite application that references the database.

## Format

```
ADD FILE File-name
[VERSION File-version]
TYPE SQL [SQL-dialect]
[DBNAME 'Database-name']
[RULE Text]
```

## File-name

Required. *File-name* is an arbitrary name of up to 32 characters. It can include alphabetic characters, numbers, the embedded characters #, @, $, embedded hyphens and embedded underscores. It must begin with an alphabetic character.

### File-version

Optional. The *VERSION* option specifies the version-number of a file.

### SQL-dialect

Optional. The database type can be specified here.

Following subtype values are available:

DB2_MVC, DB2_VSE, DB2_400, DB2_Luw, DEB2_2, Oracle, Sybase, Teradata, Oracle_RDB, Informix, Ingres, SQLServer, SESAM, Default, ODBC, MySQL

Teradata as SQL dialect should not be used. You should use DB2_MVC as SQL dialect instead.

For example:

ADD FILE datawarehouse VERSION 3 TYPE SQL SESAM DBNAME 'datawarehouse'

ADD FILE datawarehouse2 VERSION 3 TYPE SQL DB2_NT DBNAME 'datawarehouse'

### Database-name

Optional. *Database-name* is the name of the RDBMS to be accessed by the application program. The name must be enclosed in single quotes. This name may be used by the MetaSuite Generator to setup the database connection.

### Text

Optional. The *RULE* option is used to add descriptive text documenting your file.

## 5.4. ADD RECORD

The ADD RECORD command defines a relational table to the MetaSuite MetaStore. The general is described in the *Definition Language Reference Guide*. The options that refer to relational tables are described below.

### Format

```
ADD RECORD Table-name DBNAME 'Creator-name'
[RULE Text]
```

### Table-name

Required. *Table-name* is the name of the table or view, as defined in the relational catalogue. Its name is limited to 32 characters.

### Creator-name

Required. *Creator-name* is the name of the creator for the table, as defined in the relational catalogue. The name must be enclosed in single quotes.

When the *creator-name* is set to (Null), you indicate that you do not want to qualify the table with the creator of the table.

### Text

Optional. The *RULE* option is used to add descriptive text documenting your record.

## 5.5. ADD FIELD

The ADD FIELD command defines a table column as a field to the MetaSuite MetaStore. The general syntax is described in the *Definition Language Reference Guide*. Those options that are particular to relational are described below.

### Format

```
ADD FIELD Table-name.Column-name [OF Table-name]
POSITION position
SIZE characters
TYPE Datatype
[DBNAME {'NOTNULL' | 'DEFAULT'}]
[{TIMESTAMP | TIME | DATE 'YYYYMMDD'}]
[RULE Text]
```

### Table-name

Required. *Table-name* is the name of the table or view, as defined in the relational catalogue.

### Column-name

Required. *Column-name*, as defined in the relational catalogue, identifies the column that is being defined as a MetaSuite field. The name of the column is qualified with the name of its table. The composed name Table-name.Column-name is limited to 37 characters.

### Position

Required. *Position* defines the field's relative position in the record. *Position* must be an integer.

### Characters

Required. *Characters* defines the field's length. *Characters* must be an integer.

### Datatype

Required. Refer to "Specifying a Data Type" further in this chapter for more information on the different data types for SQL fields, and for more information on the Timestamp indications on the data type.

### Nulls allowed

Optional. Specify DBNAME 'NOTNULL' or DBN'DEFAULT' when null is NOT allowed. When no DBNAME is given, the field is considered to be Nulls Allowed.

### Text

Optional. The RULE option is used to add descriptive text documenting your field.

## Specifying a Data Type

MetaSuite data types fully support all relational data types. When programs are generated, MetaSuite data types are translated into the appropriate relational data types when appropriate. MetaSuite data types and sizes relate to the relational data types and sizes as follows:

### Oracle Data Types

| Oracle | MetaSuite |
|---|---|
| CHAR | CHARACTER SIZE c |
| VARCHAR2(mc) | VARCHAR SIZE mc |
| NUMBER(n,d) | PACKED SIZE n DECIMAL d |
| DECIMAL(n,d) | PACKED SIZE n DECIMAL d |
| INTEGER | BINARY SIZE 4 |
| SMALLINT | BINARY SIZE 2 |
| DATE (used as timestamp) | CHARACTER SIZE 14 TIMESTAMP |
| DATE (used as time) | CHARACTER SIZE 6 TIME |
| DATE (used as date) | ZONED UNSIGNED SIZE 8 DATE 'YYYYMMDD' |
| FLOAT | FLOAT SIZE f |

where: c = number of characters

mc = maximum number of characters

n = number of digits

d = number of decimal places

f = 4 or 8

### DB2/xxx Data Types

| DB2/xxx | MetaSuite |
|---|---|
| CHAR | CHARACTER SIZE c |
| VARCHAR(mc) | VARCHAR SIZE mc |
| DECIMAL(n,d) | PACKED SIZE n DECIMAL d |
| INTEGER | BINARY SIZE 4 |
| SMALLINT | BINARY SIZE 2 |

| DB2/xxx | MetaSuite |
|---|---|
| TIMESTAMP | CHARACTER SIZE 26 TIMESTAMP |
| TIME | CHARACTER SIZE 8 TIME |
| DATE | ZONED UNSIGNED SIZE 8 DATE 'YYYYMMDD' |
| FLOAT | FLOAT SIZE f |

where: c = number of characters
mc = maximum number of characters
n = number of digits
d = number of decimal places

## SQLServer Data Types

| SQLServer | MetaSuite |
|---|---|
| CHAR | CHARACTER SIZE c |
| VARCHAR(mc) | VARCHAR SIZE mc |
| NUMBER(n,d) | PACKED SIZE n DECIMAL d |
| DECIMAL(n,d) | PACKED SIZE n DECIMAL d |
| INTEGER | BINARY SIZE 4 |
| SMALLINT | BINARY SIZE 2 |
| DATETIME (used as timestamp) | CHARACTER SIZE 24 TIMESTAMP |
| DATETIME (used as time) | CHARACTER SIZE 12 TIME |
| DATETIME (used as date) | ZONED UNSIGNED SIZE 8 DATE 'YYYYMMDD' |
| FLOAT | FLOAT SIZE f |

where: c = number of characters
mc = maximum number of characters
n = number of digits
d = number of decimal places

## Sybase Data Types

| Sybase | MetaSuite |
|---|---|
| CHAR | CHARACTER SIZE c |
| VARCHAR(mc) | VARCHAR SIZE mc |
| DECIMAL(n,d) | PACKED SIZE n DECIMAL d |
| INTEGER | BINARY SIZE 4 |
| SMALLINT | BINARY SIZE 2 |
| DATETIME (used as timestamp) | CHARACTER SIZE 26 TIMESTAMP |
| DATETIME (used as time) | CHARACTER SIZE 26 TIME |
| DATETIME (used as date) | ZONED UNSIGNED SIZE 8 DATE 'YYYYMMDD' |
| FLOAT | FLOAT SIZE f |

where: c = number of characters

mc = maximum number of characters

n = number of digits

d = number of decimal places

## Informix Data Types

| Informix | MetaSuite |
|---|---|
| CHAR | CHARACTER SIZE c |
| VARCHAR(mc) | VARCHAR SIZE mc |
| DECIMAL(n,d) | PACKED SIZE n DECIMAL d |
| INTEGER | BINARY SIZE 4 |
| SMALLINT | BINARY SIZE 2 |
| DATETIME (used as timestamp) | CHARACTER SIZE 24 TIMESTAMP |
| DATETIME (used as time) | CHARACTER SIZE 6 TIME |
| DATETIME (used as date) | ZONED UNSIGNED SIZE 8 DATE 'YYYYMMDD' |
| FLOAT | FLOAT SIZE f |

where: c = number of characters
mc = maximum number of characters
n = number of digits
d = number of decimal places

## Oracle/RDB Data Types

| Oracle/RDB | MetaSuite |
|---|---|
| CHAR | CHARACTER SIZE c |
| VARCHAR(mc) | VARCHAR SIZE mc |
| NUMBER(n,d) | PACKED SIZE n DECIMAL d |
| DECIMAL(n,d) | PACKED SIZE n DECIMAL d |
| BIGINT | BINARY SIZE 8 |
| INTEGER | BINARY SIZE 4 |
| SMALLINT | BINARY SIZE 2 |
| TINYINT | BINARY SIZE 2 |
| TIMESTAMP (SQL_TIMESTAMP) | CHARACTER SIZE 26 TIMESTAMP |
| TIME (SQL_TIME) | CHARACTER SIZE 8 TIME |
| DATE (SQL_DATE_VMS) | ZONED UNSIGNED SIZE 8 DATE 'YYYYMMDD' |
| FLOAT | FLOAT SIZE f |

where: c = number of characters
mc = maximum number of characters
n = number of digits
d = number of decimal places

## CA/Ingres Data Types

| CA/Ingres | MetaSuite |
|---|---|
| CHAR | CHARACTER SIZE c |
| VARCHAR(mc) | VARCHAR SIZE mc |

| CA/Ingres | MetaSuite |
|---|---|
| NUMBER(n,d) | PACKED SIZE n DECIMAL d |
| DECIMAL(n,d) | PACKED SIZE n DECIMAL d |
| INTEGER | BINARY SIZE 4 |
| SMALLINT | BINARY SIZE 2 |
| DATETIME (used as timestamp) | CHARACTER SIZE 14 TIMESTAMP |
| DATETIME (used as date) | CHARACTER SIZE 8 TIME |
| DATETIME (used as time) | ZONED UNSIGNED SIZE 8 DATE 'YYYYMMDD' |
| FLOAT | FLOAT SIZE f |

where: c = number of characters

mc = maximum number of characters

n = number of digits

d = number of decimal places

CHAPTER 6

# Programming With MetaSuite File Access (RDBMS)

## 6.1. Overview

This chapter describes how to use the MetaMap Manager commands that access information stored in a relational database.

- Data source commands define the SourceFiles, SourceArrays and GlobalFields objects to be used during the program processing. For relational SourceFiles, the SourceFilePath can specify criteria to join tables.
- TargetFile objects define the output you want to generate.
- Procedural commands define the processing you want to occur, if any.

These program sections are described in detail in the MetaSuite User and Reference Guides.

The SourceFile objects may differ in use with a relational file than with a non-Relational file. Target objects are unaffected by access to a relational file.

The descriptions in this chapter use the MetaSuite terminology exclusively. The correspondences between the MetaSuite and the relational terminology are discussed in detail in the chapter Using MetaSuite With A Relational Database (page 8). Those correspondences are summarized in the following table.

### MetaSuite Versus Relational Data Structures

| MetaSuite | Relational |
|---|---|
| File | Arbitrary group of related tables |
| Record | Table Row |
| Field | Column/Field |

## 6.2. Programming considerations

When coding a program that accesses a relational database, you should be aware of the considerations below.

### Accessing the Database

In general, you access a relational file as you would access a non-relational file. You define each relational file you want to use with a SourceFile and a SourceFilePath object, whose options define the tables, selection criteria, and order criteria to be used. In addition, the SourceFile object specifies whether the tables are to be accessed automatically by MetaSuite or through procedural code.

### Program Commands

The MetaMap Manager commands and most procedural commands are unaffected by the use of relational database SourceFiles. The only exceptions are as follows:

- The options for a SourceFile object and a SourceFilePath are different for relational files. See SourceFile Object on page 22.
- Although you can use the EXCLUDE command to exclude retrieved relational data, it is more efficient to specify the record selection criteria in the SourceFile object WHERE option whenever possible. See WHERE Option on page 26.
- SourceRecord INPUT procedures are not allowed for relational records. See Procedures and Record Selection on page 28.
- Use the WHERE option on the SourceFile object to specify "start" criteria. See Positioning the SourceFile on page 29.
- The PUT command cannot be used to write to a relational file, use embedded SQL INSERT, UPDATE or DELETE.

## 6.3. SourceFile Object

The options of the SourceFile objects are different for controlled and non-controlled SourceFiles:

Non-controlled SourceFile object:

```
SOURCEFILE SourceFile-name [PREFIX 'prefix']
PATH
     (table-name correlation-name, …
     [WHERE (field-name operator value
          [{AND|OR} field-name operator value]…)]
     [ORDER BY field-name,…]
     )
[MATCH (match-key,…)]
```

Controlled By SourceFile object:

```
SOURCEFILE SourceFile-name [PREFIX 'prefix']
CONTROLLED BY controlling-SourceFile
PATH
     (table-name correlation-name, …
     [WHERE (field-name operator value
          [{AND|OR} field-name operator value]…)]
     )
```

Controlled SourceFile object:

```
SOURCEFILE SourceFile-name [PREFIX 'prefix']
CONTROLLED
PATH
     (table-name correlation-name,…
     [WHERE (field-name operator value
          [{AND|OR} field-name operator value]…)]
     )
```

Each option is described separately on the following pages.

## SourceFile-name

Names a file that has been defined directly to the MetaSuite MetaStore, which is used as SourceFile within a MetaMap Manager program.

## Prefix

Prefix is exactly four characters, including alphabetic characters, numbers, and embedded hyphens, beginning with an alphabetic character.

The *PREFIX* option allows the same definitions to be used in multiple SourceFile objects.

## Controlled SourceFile Object

The *CONTROLLED* option specifies that data will be retrieved from the SourceFile only through GET command processing.

### Unique occurrence retrieval

Use the WHERE clause of the PATH option to specify a unique unit of data to be retrieved. In other words, only one logical record should satisfy the WHERE criteria.

**Warning:** If more than one logical record is returned by the WHERE option when the GET command is issued, a processing error will occur. The MetaMap Manager syntax checker cannot detect errors of this kind.

### Count retrieval

When no fields of the controlled SourceFile are used within the program, the GET can be used to determine the number of rows within the SourceFile that match the WHERE criteria.

The *SourceFile* SYS-READ-COUNT will contain the count of rows that meet the WHERE criteria.

**Warning:** Once a field of the controlled SourceFile is used, the previous warning is valid. The WHERE criteria should be set in such a way that only one logical record is returned by the GET command.

### GET

The syntax of the GET command to be used with CONTROLLED processing is:

```
GET SourceFile-name
```

Whenever a GET command is issued, a status code is set in the SourceFile-name SYS-IO-STATUS system field to identify the success or failure of the retrieval. For a Relational CONTROLLED retrieval, there are two possible return codes, as follows:

#### CONTROLLED Retrieval Return Codes

| Return Code | Description |
|---|---|
| SYS-OK | Successful retrieval |
| SYS-ERROR | Record was not found |

Additionally, whenever a GET command is issued, the SQLCODE is set in the SourceFile-name SYS-INTERNAL-STATUS system field, and the SQLCODE will contain the value returned by a SELECT statement on the SourceFile.

In the following example, one row will be retrieved from the Relational-EMPLOYEE-MASTER file for each employee in the DRIVER. If the employee data is not retrieved (for example, if the employee's department is not included in the DEPTAB table), an error message will be produced.

```
SOURCEFILE DRIVER
SOURCEFILE Relational-EMPLOYEE-MASTER CONTROLLED
PATH (EMPTAB employee, DEPTAB department
   WHERE (department.DEPTCODE = employee.DEPTCODE
          AND employee.EMPNO = :DRIVER-EMPNO))
REPORT 1
DETAIL 1
( DEPTAB.DEPTCODE
, DEPTAB.DEPTNAME
, EMPTAB.EMPNO
)
DETAIL 2
( EMPTAB.DEPTCODE
, EMPTAB.EMPNO
)

BEGIN REPORT 1 INPUT
GET Relational-EMPLOYEE-MASTER
IF Relational-EMPLOYEE-MASTER SYS-IO-STATUS EQ SYS-ERROR
   PUT (TargetRecord1)
ELSE
   PUT (TargetRecord2).
```

## Controlled By SourceFile Object

The *CONTROLLED BY* option allows data from multiple SourceFiles to be combined automatically. The CONTROLLED BY option is used with Relational SourceFiles just as it is with non-Relational SourceFiles. The syntax, though, is somewhat different and is described here.

Refer to the *Specification Language Reference Guide* for more information on CONTROLLED BY retrieval.

In controlled by retrieval, there is a controlled SourceFile and a controlling SourceFile. Whenever a record is retrieved from the controlling SourceFile, a record (or view) of data is also retrieved from the controlled SourceFile. The controlled by condition is specified with the WHERE clause of the PATH option on the controlled SourceFile object, as described below.

For controlling-SourceFile, you can specify any SourceFile named in a prior SOURCEFILE statement. The controlling SourceFile itself may be controlled by another SourceFile. You can nest up to 100 SourceFiles in this way. (Note that 100 is the maximum number of SourceFiles in a program.) At least one SourceFile must be non-controlled.

In the WHERE clause of the PATH option for the controlled relational SourceFile, name a work field or a field from the controlling SourceFile. If the controlling SourceFile has a PATH option and is not a relational SourceFile, name a work field or a field from the last record named in the controlling SourceFile's PATH option.

Be sure to test the success or failure of the retrieval. As in the case of CONTROLLED retrieval, use the SourceFile-name SYS-IO-STATUS system field to determine this.

The controlled by retrieval itself does not occur until after SourceFile input processing is completed for the record or path from the controlling SourceFile.

In the following example, all deductions are listed for each employee selected through CONTROLLED BY retrieval. Only those employees are retrieved who are also listed in the (non-Relational) DRIVER-FILE. Employees in the DRIVER-FILE with any status but "A" are excluded prior to retrieval from Relational-EMPLOYEE-MASTER.

```
SOURCEFILE DRIVER-FILE
SOURCEFILE Relational-EMPLOYEE-MASTER
   CONTROLLED BY DRIVER-FILE
   PATH (EMPTAB employee, DEDUCTAB deduction
   WHERE (employee.EMPNO = deduction.EMPNO
   AND employee.EMPNO = :DRIVER-EMPNO))
        .
        .
        .
BEGIN SOURCEFILE DRIVER-FILE INPUT
IF DRIVER-STATUS NE 'A'
   EXCLUDE.

BEGIN REPORT 1 INPUT
IF Relational-EMPLOYEE-MASTER SYS-IO-STATUS EQ SYS-ERROR
   PUT (TargetRecord1)
ELSE
   PUT (TargetRecord2).
```

## Path Option

```
PATH (table-name correlation-name, ...
[WHERE (field-name operator value
      [{AND | OR} field-name operator value] ...)]
[ORDER BY table-name.field-name, ...] )
```

The PATH option is used to name the tables that are to be joined.

*Table-name* is the name of a table defined for the SourceFile. Up to 16 tables can be named in one PATH option list. However, you should check with your Relational Database Administrator before joining more than seven SourceFiles.

*Correlation-name* is a required alias for each table used. It is defined by the Path(Record)-name in your SourceFile definition in your MetaMap Manager program. This name should be used in the WHERE clause as a qualifier for each column name. The correlation name may not contain a hyphen ('-').

The following example joins two tables, DEPTAB and EMPTAB. In other words, it returns data for all employees who belong to departments that are listed in both the DEPTAB and EMPTAB tables.

```
SOURCEFILE Relational-EMPLOYEE-MASTER
PATH (DEPTAB department, EMPTAB employee
WHERE (department.DEPTCODE = employee.DEPTCODE))
```

The following example expands the previous example by including selection criteria that test whether the employees are from department 3 and were hired after November 2, 1990.

```
SOURCEFILE Relational-EMPLOYEE-MASTER
PATH (DEPTAB department, EMPTAB employee
WHERE (department.DEPTCODE = employee.DEPTCODE
         AND department.DEPTCODE = '003'
         AND employee.HIREDATE > 19901102
         )
      )
```

For 'table-name correlation-name … ' specify the tables as they appear hierarchically in the path list. For each table, all possible rows will be retrieved from its subordinate table.

For example, the following path list:

```
PATH (DEPTAB department, EMPTAB employee
   WHERE (department.DEPTCODE = employee.DEPTCODE))
```

results in all matching rows from the EMPTAB table being retrieved for each row in the DEPTAB table.

In the following example, the WHERE clause is omitted, and all rows are retrieved from the one table named in the PATH option list.

```
SOURCEFILE Relational-EMPLOYEE-MASTER
   PATH (EMPTAB employee)
```

## WHERE Option

```
WHERE (field-name operator value
      [ {AND | OR} field-name operator value] ...)
```

value is specified as:

{ 'alphanumeric-value' | numeric-value | field-name }

The WHERE clause of the PATH option contains the record selection criteria that determine what rows are retrieved for processing. Only those rows that satisfy all the criteria will be returned. The WHERE clause can include up to 16 criteria statements, joined with the AND or OR keywords.

*Field-name* is

- an input field, referenced as *correlation-name.field-name*.

The input field must be defined to a table in the PATH option.

*Operator* is used to compare field-name and a value. The supported operators are those of the RDBMS in use.

*Value* can be an explicit literal value, an explicit number, or a field name. If you specify an explicit literal value, the value should be contained within single quotation marks. If you specify a field name, it should be defined prior to be being referenced, and it should be defined as a variable to the RDBMS in use, by putting a colon (:) in front of its name (e.g., Field-name). If a field name is used, this may not be a subscripted field.

In the following example, rows will be returned for employees from department 3 only.

```
SOURCEFILE Relational-EMPLOYEE-MASTER
PATH (EMPTAB employee
WHERE (employee.DEPTCODE = '003'))
```

The {AND | OR} Boolean operators allow you to include multiple selection criteria in the WHERE clause.

In the following example, records will be retrieved for employees whose departments are listed in the DEPTAB table and which have deductions listed in the DEDUCTAB table.

```
SOURCEFILE Relational-EMPLOYEE-MASTER
PATH (DEPTAB dept, EMPTAB empl, DEDUCTAB deduc
WHERE (dept.DEPTCODE = empl.DEPTCODE
        AND empl.EMPNO = deduc.EMPNO))
```

When both the 'AND' and 'OR' Boolean operators are used, the AND statements take precedence over the 'OR' statements.

The following example selects only those employees that have departments listed in the DEPTAB table. From this group of employees, it further selects only those employees who were hired on or after WK-DATE or who have salaries greater than 30,000.

```
SOURCEFILE Relational-EMPLOYEE-MASTER
PATH (DEPTAB department, EMPTAB employee
   WHERE (department.DEPTCODE = employee.DEPTCODE
      AND employee.HIREDATE >= :WK-DATE
```

```
      OR department.DEPTCODE = employee.DEPTCODE
      AND employee.SALARY > 30000))
```

The WHERE clause is optional, but recommended for any PATH with a list containing more than one table. If no WHERE clause is included, all rows in each subordinate table will be retrieved for each row in the preceding table.

In the following example, all rows in the EMPTAB table will be returned for each row in the DEPTAB table (whether the department codes match or not) and all rows in the DEDUCTAB table will be returned for each row in the EMPTAB table.

```
SOURCEFILE Relational-EMPLOYEE-MASTER
PATH
(DEPTAB department, EMPTAB employee, DEDUCTAB deduction)
```

**Note:** Omitting the WHERE clause may cause significant performance problems due to possible and repeated retrieval of many unneeded rows.

## ORDER BY Option

`ORDER BY` *`table-name.field-name, …`*

The ORDER BY clause of the PATH option specifies the sequence in which the rows are returned to the program. Please check with the DBA to inform about the performance issues on the defined ORDER BY. Another option to specify a sequence for the row, is the usage of a SourceFile initial Sort procedure.

The ORDER BY option is not allowed for SourceFiles specified with the CONTROLLED or CONTROLLED BY options.

For *field-name*, specify a field from one of the tables named in the path list. You can specify up to 16 order by fields.

In the following example, the rows of the EMPTAB table are returned in department code order:

```
SOURCEFILE Relational-EMPLOYEE-MASTER
PATH (EMPTAB employee
ORDER BY EMPTAB.DEPTCODE)
```

In the following example, the rows are returned in department code order, in employee number order within each department:

```
SOURCEFILE Relational-EMPLOYEE-MASTER
PATH (DEPTAB department, EMPTAB employee
   WHERE (department.DEPTCODE = employee.DEPTCODE)
      ORDER BY DEPTAB.DEPTCODE, EMPTAB.EMPNO)
```

In the following example, the rows are returned in department code order, in order by hire date within each department:

```
SOURCEFILE Relational-EMPLOYEE-MASTER
PATH (DEPTAB department, EMPTAB employee
   WHERE (department.DEPTCODE = employee.DEPTCODE
      AND department.DEPTCODE = :WK-DEPT
      AND employee.HIREDATE > :WK-DATE)
      ORDER BY DEPTAB.DEPTCODE, EMPTAB.HIREDATE)
```

# 6.4. Procedural Commands

Procedural commands tell MetaSuite what, if any, special processing is to be performed. Procedural code for a program that accesses a relational database can include any of the procedural commands described in the *Specification Language Reference Guide*. In certain cases, these commands differ in their use with relational SourceFiles, as described below.

## Checking the Return Status

To check the status information returned by a MetaSuite relational request, reference the following system fields:

- SourceFile-name SYS-INTERNAL-STATUS contains the SQLCODE value returned from the last request to relational. SYS-INTERNAL-STATUS is defined as a character field with the edit-mask '---------9' (Leading blanks).
- SourceFile-name SYS-IO-STATUS contains an I/O status code that you can check using one of the following constants:
  - SYS-OK means that SQLCODE 0 (zero) was returned, indicating that the requested retrieval was successful.
  - SYS-EOF indicates that a HALT SOURCEFILE command has been executed for the named SourceFile, or that an SQLCODE of 100 has been returned for an *automatic* SourceFile.
  - SYS-ERROR means that an SQLCODE that was not 0 (zero) or 100 was returned from the *last* Relational request for the SourceFile. This indicates that the requested data was not retrieved. When the SourceFile is a CONTROLLED SourceFile, SYS-ERROR means that no record has been found for the Controlled SourceFile.

## Null values

Field-name SYS-STATUS is set to SYS-NULL-VALUE if the field contains a null value.

```
IF tablename.columnname SYS-SYS-STATUS EQ SYS-NULL-VALUE
   EXCLUDE.
```

Field-name SYS-STATUS should be assigned the SYS-NULL-VALUE value if the field is to be assigned a null value.

```
IF condition
tablename.columnname SYS-SYS-STATUS = SYS-NULL-VALUE
ELSE
tablename.columnname = Field.
```

## Checking the Length of a VARCHAR field

Field-name SYS-SQL-LENGTH contains the number of valid characters in a VARCHAR field. As stored in a table each VARCHAR field is preceded by a 2-byte indicator, that contains the number of valid characters in the field. When the field is retrieved by an SQL call, the RDBMS returns this number to the program.

```
IF tablename.columnname SYS-SQL-LENGTH GT value
   EXCLUDE.
```

## Procedures and Record Selection

With the exception of the record input procedure, the MetaSuite procedures can be used with relational SourceFiles just as they would be with non-relational SourceFiles. The SourceFile procedure considerations are summarized below.

Record selection for a relational SourceFile is controlled by the criteria in the WHERE clause of the SourceFile object and by use of the EXCLUDE command. The WHERE clause criteria control what records (rows) are presented to the program. The EXCLUDE command is used to exclude records that have been presented to the program but do not meet additional criteria.

A SourceFile initial procedure (ending with a SORT, EXTRACT, or PRE-PASS command) processes a logical record that contains one of the joined rows returned by relational.

**Note:** The ORDER BY clause of the SOURCEFILE statement can be used instead of a SourceFile initial procedure to pre-sort the data.

The SourceFile input procedure processes a logical record that contains one of the joined rows returned by relational, or it processes the record created by a SourceFile initial SORT or EXTRACT operation.

The EXCLUDE command in a SourceFile input procedure excludes the individual logical record from further processing. If excluded, the record is not seen by any TargetFile.

In a CONTROLLED BY retrieval, use the SourceFile input procedure to exclude records on the controlling SourceFile prior to retrieval of related records from the controlled SourceFile. This avoids unwanted retrievals and improves program efficiency.

The SourceRecord input procedure is not allowed for records from relational SourceFiles.

There are no special considerations for using any of the following procedure types with a relational file:

```
SourceFile EOF
TargetFile INITIAL
TargetFile INPUT
TargetFile GROUP
TargetFile EOF
TargetFile EOJ
Global Procedures
```

### Positioning the SourceFile

The START command is used for non-relational files to position the SourceFile so that processing begins with a specified record. The START command is not allowed for relational SourceFiles.

For a relational SourceFile, use the WHERE clause of the SourceFile object to position the SourceFile.

For example, to process all departments starting with department 3 (i.e., greater than or equal to 003), code:

```
SOURCEFILE Relational-EMPLOYEE-MASTER
PATH (DEPTAB department
   WHERE (department.DEPTCODE >= '003'))
```

## 6.5. External Array Object

It is possible to put an entire SQL table or a part of it into an external array. This enhances the program performance.

The method for defining an SQL external array is similar to the method for defining an SQL source file,except for the fact that a subpath (multiple record structure) is not possible in an external array.

The usage of this external array is equal to the usage of external arrays in general. (So a get on an SQL external array is not the same as a GET on an SQL source file.)

More information about the usage of external arrays can be found in the *MetaMap Manager User Guide*.