

WebFOCUS

WebFOCUS Relational Interface Manual
Version 4 Release 3.5

WebFOCUS Relational Interface Manual

Copyright © 2001 Information Builders, Inc.

Table of Contents

Relational Interfaces	6
Ease of Use	7
Efficiency	7
Security	8
Describing Relational Tables	9
File Descriptions	10
File Attributes	10
FILENAME	10
SUFFIX	10
Segment Attributes	10
SEGNAME	11
SEGTYPE	11
CRFILE	11
Field Attributes	11
The Primary Key	12
FIELDNAME	12
ALIAS	13
USAGE/FORMAT	13
ACTUAL	13
ACTUAL = DATE	13
MISSING	14
Optional Field Attributes	14
Access Files	14
Segment Declarations	14
SEGNAME (Segment Declarations)	15
TABLENAME	15
WRITE	16
KEYS	16
KEYORDER	17
FALLBACK (Teradata Only)	17
The Long Fieldname Alternative	17
The FOCUS OCCURS Segment	18
Creating an OCCURS Segment	18
The ORDER Field	19

Describing Multi-Table Relational Structures.....	21
Advantages of Multi-Table Structures	21
Creating a Multi-Table Structure	21
Multi-Table File Descriptions.....	21
SEGNAME (Multi-Table).....	22
SEGTYPE (Multi-Table).....	23
PARENT (Multi-Table)	23
CRFILE (Multi-Table)	23
FIELD (Multi-Table).....	23
ALIAS (Multi-Table).....	23
ECOURSE File Description.....	24
Multi-Table Access Files	24
KEYFLD and IXFLD.....	25
Multi-Field Embedded Join	25
Remote Segment Descriptions	26
The Interface Optimizer	29
The SET OPTIMIZATION Command.....	29
Optimization Example	30
Optimization Logic	31
Record Selection and Projection.....	31
Record Selection.....	31
Projection	31
Joins	32
Sorting	33
Aggregation	33
DEFINE Fields.....	34
Valued Expressions.....	34
Arithmetic Expressions.....	34
Character String Expressions	34
Logical Expressions	35
SQL Limitations	35
Advanced Reporting Techniques	36
The TABLEF Command.....	36
Creating Extract Files on the RDBMS.....	37
Generated File Descriptions.....	38
Generated Access Files	38
Other Generated Files.....	39
HOLD FORMAT SQL Usage Restrictions.....	39
Extract File Conversion Chart	39
The Dynamic JOIN Command	40
Single-Field Dynamic JOIN	41
Dynamic JOIN Examples	42
Multi-Field Dynamic JOIN	43

Multi-Field Joins Summary Chart	45
Missing Rows of Data in Cross-Referenced Tables	46
The SET ALL Command	46
SET ALL=ON and Join Optimization	47
Missing Rows in Unique Descendants	47
OPTIMIZATION Disabled With a Unique JOIN	47
OPTIMIZATION Enabled With a Unique JOIN	48
Missing Rows in Non-Unique Descendants	48
SET ALL=OFF With a Non-Unique JOIN	48
SET ALL=ON With a Non-Unique JOIN	49
SET ALL=ON With Screening Conditions	50
Selective ALL. Prefix	51
Dynamic JOIN Summary Chart	52
Search Limits	52
Using READLIMIT With JOIN	53
Multiple Retrieval Paths	53
Multiple Retrieval Paths With Sort Phrases and Screening Tests	54
Direct SQL Passthru	55
Invoking Direct SQL Passthru	55
Issuing Commands and Requests	56
Example: Issuing Interface Environmental Commands	57
Example: Issuing Native SQL Commands (Non-SELECT)	57
Displaying the Effects of UPDATE and DELETE Commands	58
Example: Issuing SQL SELECT Commands	58
The SQLOUT File Description	59
Example: Customizing a Default Report	61
Creating FOCUS Views With Direct SQL Passthru	62
Controlling SQL Passthru Transactions	64
BEGIN/END SESSION	64
COMMIT WORK (Oracle, DB2/2 only)	65
ROLLBACK WORK (Oracle, DB2/2 only)	65
Parameterized SQL Passthru	65
Parameterized SQL Command Summary	66
PREPARE	67
EXECUTE	68
PURGE	69
BIND	70
Sample Session: Oracle	71
Sample Session: Teradata	72

Transaction Control Commands.....	73
The SET AUTOAction ON Event Command.....	73
Actions.....	74
AUTOCOMMIT.....	74
AUTODISCONNECT	74
Action and Event Combinations.....	74
Effects of Action and Event Combinations.....	75
SET AUTOCOMMIT ON CRTFORM	75
SET AUTOCOMMIT ON FIN	75
SET AUTODISCONNECT ON COMMIT.....	76
SET AUTODISCONNECT ON FIN	76
Combinations of SET AUTOAction Commands.....	76
Types of FOCUS Sessions	77
The Default Interface Session.....	77
The User-Controlled Session	78
Example: Explicit Control of a Logical Unit of Work (LUW)	79
The Interface Environment.....	80
Interface Environment SET Commands.....	80
? Query (All Interfaces)	80
FETCHSIZE and INSERTSIZE (Oracle Only)	81
OPTIMIZATION (All Interfaces)	82
PASSRECS.....	83
ERRORTYPE (All Interfaces).....	84
ANSITOOEM (All Interfaces)	84
AUTOLOGON (All Interfaces).....	85
Interacting With the Interfaces	85
Default DATE Considerations	85
The Default DATE Value.....	85
Status Return Variable: &RETCODE	86
Standard FOCUS and Interface Differences.....	86
Maintaining Relational Tables.....	88
Types of Relational Transaction Processing	89
The Role of the Primary Key.....	89
Index Considerations	89
Modifying Data.....	90
The MATCH Statement.....	90
Interface MATCH Behavior	91
NEXT	91
NEXT Processing Without MATCH	91
NEXT Processing After MATCH on a Full Key or on a Superset	92
NEXT Processing After MATCH on a Non-Key Field or Partial Key.....	93
INCLUDE, UPDATE, and DELETE Processing.....	95
RDBMS Transaction Control.....	96

RDBMS Transaction Termination (COMMIT WORK).....	97
RDBMS Transaction Termination (ROLLBACK WORK) (Oracle, ODBC, DB2/2 Only).....	97
RDBMS Transaction Control Example	98
DBC/1012 Transaction Control Within MODIFY (Teradata only)	98
Transaction Control Syntax.....	99
Examples of Transaction Control (Sample Code)	99
Teradata DBC/1012 Transaction Termination (ROLLBACK WORK)	100
Using the Return Code Variable: FOCERROR.....	101
Using the Interface SET ERRORRUN Command	101
Modifying Tables Without Primary Keys	101
Referential Integrity.....	102
RDBMS Referential Integrity	102
FOCUS Referential Integrity	103
FOCUS INCLUDE Referential Integrity.....	103
FOCUS DELETE Referential Integrity.....	104
Inhibiting FOCUS Referential Integrity	105
The COMBINE Facility.....	105
The LOOKUP Function.....	108
The FIND Function	108
Issuing SQL Commands in MODIFY.....	109
LOADONLY	109
AUTOCOMMIT ON CRTFORM	109
The FOCURRENT Variable	111
Rejected Transactions and T. Fields.....	111
Index.....	113

Relational Interfaces

Topics:

- Ease of use.
- Efficiency.
- Security.

The FOCUS Interfaces allow you to access relational data using all the powerful data maintenance, reporting, and analysis facilities of FOCUS. Relational interfaces are provided to Sybase SQL Server, IBM DB2/2, Oracle, Teradata, Informix, and any ODBC enabled data source. The relational interfaces allow transparent access to each relational database management system (RDBMS) through the FOCUS file description and access file. You do not need to know SQL (the data access language for relational systems) in order to access the information. In addition, FOCUS provides many facilities that are not available in SQL. These include complex IF... THEN... ELSE calculated columns and cross-tab reports.

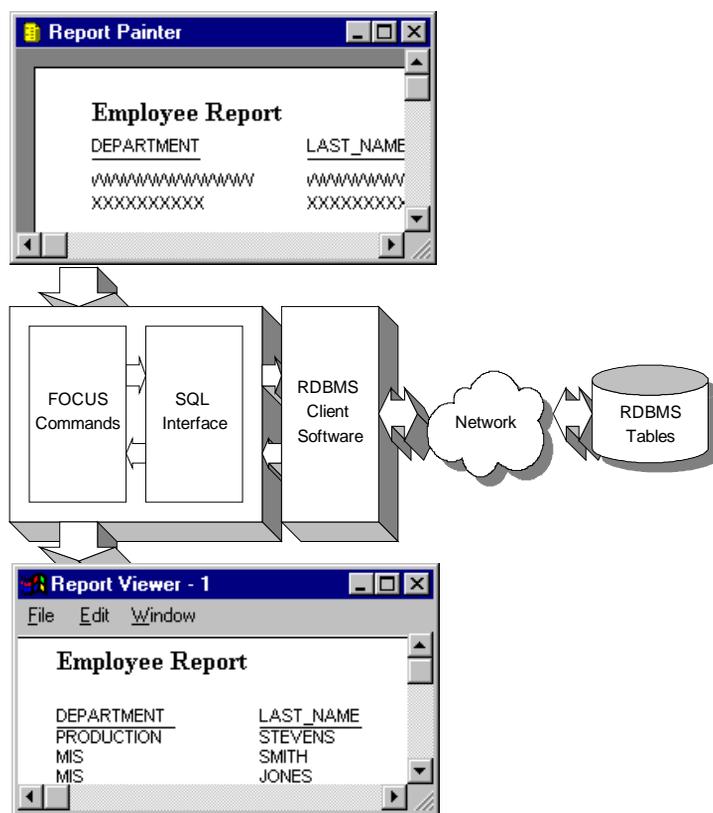
FOCUS provides straightforward transaction processing and application development tools for updating and maintaining the tables in a relational database management system (RDBMS). The FOCUS transaction processors, MAINTAIN and MODIFY, support both interactive maintenance procedures and batch maintenance using external input files.

The Interfaces translate FOCUS retrieval and update requests into an equivalent set of SQL statements. It also initiates and monitors communication between itself and the RDBMS, and provides descriptive error message and recovery support when necessary.

FOCUS and the RDBMS interact as follows:

1. Given a Report, Graph, MAINTAIN, or MODIFY request, the Interface builds SQL statements that define the request in terms the RDBMS can understand.
2. Having received these SQL statements from the Interface, the RDBMS retrieves or updates data targeted by the request.
3. In response, the RDBMS sends the data (answer set) and/or return code back to the Interface.
4. The Interface passes it to FOCUS for further processing (for example, to display a report or show the information on a form).

The following diagram illustrates how FOCUS interacts with an interface.



See the interface topics for more specific illustrations of the flow between the FOCUS Interface and each of the relational data sources it supports.

Advanced users can embed SQL statements directly into FOCUS application code. With the Direct SQL Passthru facility, you can include SQL SELECT statements in report requests and retrieve answer sets from the RDBMS. You can create new tables with SQL commands or with the FOCUS CREATE FILE command.

Ease of Use

The Interface uses the FOCUS language to request access to tables and views. There is no need for specialized functions or embedded SQL commands, although with the Direct SQL Passthru facility you can include all SQL commands in report requests.

To make an RDBMS table intelligible to FOCUS, each table or view is described once in FOCUS terminology. This description is stored as a file description and an associated access file. These descriptions make it possible to refer to the individual columns of the table via the FOCUS fieldname or the RDBMS column name. The Interface provides an AutoBuild facility that can create a file description and access file for each relational data source. (Issuing requests through the Direct SQL Passthru facility eliminates the need for FOCUS file descriptions and access files, but retains all FOCUS reporting capabilities.)

Once you have a FOCUS file description and access file for a table, you can use all the FOCUS facilities: the database maintenance language, graphics, statistics, etc. You do not need to know SQL.

Efficiency

The Interface retrieves from the RDBMS only those rows or columns referenced in the FOCUS report request. Additionally, the Interface enables the RDBMS to perform all the work required in the joining, sorting, and aggregation of data. This reduces the volume of RDBMS-to-FOCUS communication, resulting in faster response times for interface users.

Security

FOCUS respects all existing RDBMS security. That is, an Interface user must be authorized to retrieve data or update tables. This authorization must come from the RDBMS database administrator or another authorized user.

FOCUS provides its own security facilities that you can use as a complement to RDBMS security. FOCUS security can enforce the following levels of restriction:

- File-level security to prevent access to a table.
- Field-level security to limit the fields within a table that are accessible to a user.
- Field-value security to limit the rows within a table that are accessible to a user based on a specified field's values.

For more information on FOCUS DBA., refer to *Creating File Descriptions* in online help and *Providing File Security: DBA* in the online *Language Reference*.

Describing Relational Tables

Topics:

- File descriptions.
- Access files.
- The long fieldname alternative.
- The FOCUS OCCURS segment.

In order to access a table or view using FOCUS, you must first describe it to FOCUS in two files: a file description, and an associated access file.

Note: The term *table* in this manual refers to both RDBMS base tables and views.

The file description describes the columns of the RDBMS table using keywords in comma-delimited format. The access file includes additional parameters that complete the FOCUS definition of the RDBMS table. The Interface requires both a file description and an access file to generate SQL queries.

There are two methods for creating file descriptions and access files:

- Execute an automated procedure—AutoBuild—that creates file descriptions and access files for existing RDBMS tables. See the appropriate topics for your Interface for details about these facilities.
- Use the File Painter or an editor to manually create the descriptions of the table. Refer to a copy of the native SQL CREATE TABLE statement or a detailed report from the system catalog tables for column names, datatypes, lengths, and other descriptive information.

FOCUS file descriptions can represent an entire table or part of a table. Also, several pairs of file descriptions can define different subsets of columns for the same table, or one pair consisting of a file description and an access file can describe several tables. These topics present single-table file descriptions and access files. For multi-table file descriptions, see Describing Multi-Table Relational Structures.

You can represent tables with repeating columns as FOCUS OCCURS segments.

FOCUS processes a request with the following steps.

1. It locates the file description for the table. The filename specified in the report request identifies the file description.
2. It detects the SUFFIX value in the file description. If this value indicates that the data is in an RDBMS table, FOCUS passes control to the Interface.
3. The Interface locates the corresponding access file, uses the information contained in both descriptions to generate the SQL statements required by the report request, and passes the SQL statements to the RDBMS.
4. The Interface retrieves the answer sets generated by the RDBMS and returns control to FOCUS. Depending on the requirements of the request, FOCUS may perform additional processing on the returned data.

This example is a FOCUS file description for the table EMPINFO:

```
FILE=EMPINFO , SUFFIX=SQLDBM , $
SEGNAME=EMPINFO , SEGTYPE=S0 , $
FIELD=EMP_ID , ALIAS=EMP_ID , USAGE=A9 , ACTUAL=A9 , MISSING=OFF , $
FIELD=LAST_NAME , ALIAS=LAST_NAME , USAGE=A15 , ACTUAL=A15 , MISSING=OFF , $
FIELD=FIRST_NAME , ALIAS=FIRST_NAME , USAGE=A10 , ACTUAL=A10 , MISSING=ON , $
FIELD=HIRE_DATE , ALIAS=HIRE_DATE , USAGE=MDY , ACTUAL=DATE , MISSING=ON , $
FIELD=DEPARTMENT_CODE , ALIAS=DEPARTMENT_CODE ,
USAGE=A10 , ACTUAL=A10 , MISSING=ON , $
FIELD=CURRENT_SALARY , ALIAS=CURRENT_SALARY ,
USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=ED_HRS , ALIAS=ED_HRS , USAGE=D6.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=JOBCODE , ALIAS=JOBCODE , USAGE=A3 , ACTUAL=A3 , MISSING=ON , $
```

The following is an access file for the table EMPINFO:

```
SEGNAME=EMPINFO, TABLENAME=USERID.EMPINFO , KEYS=0, WRITE=YES,$
```

File Descriptions

A table is an RDBMS object consisting of rows and columns. A FOCUS file description represents a table as a single segment. The syntax for describing an RDBMS table in a FOCUS file description is similar to that for any external (non-FOCUS DBMS) file.

A file description contains three types of declarations:

- The file declaration indicates that the data is stored in an RDBMS table or view.
- The segment declaration identifies a table.
- Field declarations describe the columns of the table.

Each declaration must begin on a separate line. A declaration consists of keyword-value pairs (called attributes) separated by commas. A declaration can span as many lines as necessary, as long as no single keyword-value pair spans two lines. Certain keywords are required; the rest are optional (see Optional Field Attributes).

Do not use system or SQL reserved words as names for files, segments, fields, or aliases.

File Attributes

Each file description begins with a file declaration that names the file and describes the type of data source, in this case an RDBMS table or view. The file declaration has two attributes, FILENAME and SUFFIX. The syntax is

```
FILE[NAME]=name, SUFFIX=suffix, [$]
```

where:

`name` Is a one- to eight-character filename.

`suffix` Is the SUFFIX value for the relational database Interface. Following is a list of supported suffixes:

<code>SQLSYB</code>	Sybase SQL Server
<code>SQLORA</code>	Oracle
<code>SQLDBM</code>	IBM DB2/2
<code>SQLDBC</code>	Teradata DBC/1012
<code>SQLODBC</code>	Microsoft ODBC
<code>SQLINF</code>	Informix
<code>EDA</code>	Enterprise Data Access/SQL

FILENAME

The FILENAME (or FILE) keyword names the file description.

The file name consists of up to eight alphanumeric characters and must contain at least one letter. You should make the name representative of the table or view contents. It can have the same name as the RDBMS table if the table name complies with FOCUS naming conventions. It must have the same name as its corresponding access file.

SUFFIX

The SUFFIX keyword indicates the Interface required for interpreting requests.

Segment Attributes

Each table described in a FOCUS file description requires a segment declaration that consists of at least two attributes, SEGNAME and SEGTYPE.

If several file descriptions (used only with TABLE requests) include the same table, you can avoid repeating the same description multiple times. Describe the table in *one* of the file descriptions, and use the CRFILE attribute in the other file descriptions to access the existing description. For a full explanation of remote segment descriptions, see Describing Multi-Table Relational Structures.

The syntax for a segment declaration is

```
SEGNAME=segname, SEGTYPE= {S0  
                           {KL} [,CRFILE=crfile] [,$]  
                           ( )
```

where:

segname Is a one- to eight-character name that identifies the segment. If this segment references a remote segment description, segname must be identical to the SEGNAME from the file description that contains the full definition of the RDBMS table's columns (see Describing Multi-Table Relational Structures).

S0 Indicates to the Interface that the RDBMS handles the storage order of the data.

KL References a remote segment description (see Describing Multi-Table Relational Structures).

crfile Indicates the name of the remote file description that contains the full definition of the RDBMS table's columns (see Describing Multi-Table Relational Structures). This parameter is required only to reference a remote segment description.

SEGNAME

The SEGNAME attribute identifies one table or view. The one- to eight-character SEGNAME value may be the same as the name chosen for FILENAME, the actual table name, or an arbitrary name. To reference a remote segment description, the SEGNAME value must be identical to the SEGNAME in the file description that contains the full definition of the RDBMS table's columns.

The corresponding access file must contain a segment declaration with the same SEGNAME value as the file description. The segment declaration in the access file specifies the name of the RDBMS table. In this manner, the SEGNAME value serves as a link to the actual table name.

SEGTYPE

In a single table file description, SEGTYPE always has a value of S0 (or KL for a remote segment description). The RDBMS assumes responsibility for both physical storage of rows and the uniqueness of column values (if a unique index exists).

SEGTYPE values for multi-table file descriptions are discussed in Describing Multi-Table Relational Structures.

CRFILE

Include the CRFILE attribute in a segment declaration only if the actual description of the table's columns is stored in another (remote) file description. The CRFILE value must be the name of the file description that contains the full definition of the RDBMS table's columns. For a complete discussion of remote segment descriptions, see Describing Multi-Table Relational Structures.

Field Attributes

Each row in a table consists of one or more columns. In the file description, you define each column as a FOCUS field with the primary attributes FIELDNAME, ALIAS, USAGE, ACTUAL, and MISSING. The *Language Reference* manual explains additional attributes.

You can get values for these attributes from the RDBMS system catalog table.

The syntax for a field declaration is

```
FIELD[NAME]=name, [ALIAS=]sqlcolumn, [ { [USAGE ] } ]
[ { [FORMAT ] } ] display [,ACTUAL=]sqlfmt
[ { [MISSING= {OFF} ] } ]
[ { [ON ] } ] , $
```

where:

<code>name</code>	Is a 1- to 66-character unqualified name. In requests, you can qualify a fieldname with its file and/or segment name. Although the qualifiers and qualification characters do not appear in the file description, they count toward the 66-character maximum.
<code>sqlcolumn</code>	Is the RDBMS column name, up to 66 characters long. If the column name contains reserved words, characters, or spaces, enclose the column name in double quotes ("). Optionally, the alias name can be blank and the column name can be described in the access file (see The Long Fieldname Alternative).
<code>display</code>	Is the FOCUS display format for the field.
<code>sqlfmt</code>	Is the FOCUS definition of the RDBMS datatype and length, in bytes, for the field. (See the datatype conversion chart in the appropriate topics for your RDBMS.)
<code>OFF/ON</code>	Indicates whether the field can contain null values. OFF, the default, does not permit null values.

The Primary Key

A table's primary key is the column or combination of columns whose values uniquely identify each row of the table. In the EMPINFO table, every employee is assigned a unique employee identification number. This identification number, and its corresponding employee, are represented by one (and only one) row of the table.

Note: The terms *primary key* and *foreign key* refer to columns that relate two tables. In this manual, they do not refer to primary and foreign keys defined in SQL CREATE TABLE statements (RDBMS referential integrity) unless explicitly stated.

The Interface uses information from both the file description and the access file to identify the primary key. In the access file, the KEYS=*n* attribute specifies the *number* of key fields, *n*. In the file description, the *first* *n* fields described immediately after the segment declaration constitute the primary key. Therefore, the order of field declarations in the file description is significant.

To define the primary key in a file, describe its component fields first after the segment declaration. You can specify the remaining fields (those that do not participate in the primary key) in any order.

The KEYS attribute in the access file completes the process of defining the primary key.

Typically, the primary key is supported by an SQL unique index to prevent the insertion of duplicate key values. The Interface itself does not require any index on columns comprising the primary key (although a unique index is certainly desirable for both data integrity and performance reasons).

FIELDNAME

FOCUS fieldnames must be unique in a single-table file description and can consist of up to 66 alphanumeric characters (including any filename and segname qualifiers and qualification characters you may later prefix to them in requests). In the file description, fieldnames cannot include qualifiers. Column names are acceptable values if they meet the following FOCUS naming conventions:

- A name can consist of letters, digits, and underscore characters. Special characters and embedded blanks are not advised.
- The name must contain at least one letter.

Since the fieldname appears as the default column title for reports, select a name that is representative of the data. In requests, you can specify fieldnames and you can qualify a fieldname with its filename and/or segment name.

ALIAS

The ALIAS value for each field must either be the full SQL column name (the Interface uses it to generate SQL statements), or blank if the column name is described in the access file (see The Long Fieldname Alternative). The ALIAS name must be unique in the segment. The ALIAS name must comply with the same naming conventions described for fieldnames.

USAGE/FORMAT

The USAGE keyword indicates the display format of the field. An acceptable value must include the field type and length and may contain edit options. FOCUS uses the USAGE format for data display on reports, CRTFORMs, and MAINTAIN forms. All standard FOCUS USAGE formats are available. For a complete list of USAGE formats appropriate for external files, see the *Language Reference* manual.

Note:

- The field *type* described by the USAGE format must be identical to that of the ACTUAL format. For example, a field with an alphanumeric USAGE field type must have an alphanumeric ACTUAL field type, or a field with a double precision (D) USAGE fieldtype must have a double precision (D) ACTUAL field type.
- Fields with decimal and floating point datatypes must be described with the correct scale and precision. Scale (s) is the number of positions to the right of the decimal point. Precision (p) is the total length of the field. For FOCUS field formats, the field length includes the decimal point and negative sign. RDBMS datatypes exclude positions for the decimal point and negative sign.

For example, a column defined as DECIMAL(5,2) in DB2/2 would have a USAGE attribute of D7.2 to allow for the decimal point and a possible negative sign.

ACTUAL

The ACTUAL keyword indicates the FOCUS representation of RDBMS datatypes. For information on the datatypes that are supported for your RDBMS, refer to the appropriate topics.

ACTUAL = DATE

The Interface fully supports a comprehensive set of date formats. ACTUAL = DATE, in conjunction with USAGE date formats (such as YDM, DMY, MDY), describes columns with DATE datatypes. You can specify FOCUS date formats containing any combination of the components year, month, and day to display dates stored with the RDBMS DATE datatype. This feature makes it easier to manipulate dates, and the storage format is compatible with both FOCUS and non-FOCUS programs.

ACTUAL = DATE makes it possible to:

- Sort by date in date sequence, regardless of the USAGE display format.
- Define date components such as year, month, and day, and extract them from the date field.
- Perform date arithmetic and date comparisons without using special date-handling functions.
- Refer to dates in a natural way (JAN 1 1993, for example) regardless of formats.
- Automatically validate dates in transactions.
- Easily convert dates from one USAGE format to another (YMD to DMY, for example).

Note:

- In report requests, enclose in single quotation marks a DATE literal in a WHERE clause so it conforms to its USAGE format. For example, specify the field DATE_FLD, with a display format of YMD and an ACTUAL format of DATE, as:

```
WHERE DATE_FLD EQ '930224'
```

- When you use MODIFY and MAINTAIN to update a field with ACTUAL = DATE, be aware that if the USAGE format lacks a day and/or month component (YY, for example), the Interface assigns a default day and/or month of '01' to the incoming transaction value (for example, 1993 becomes 19930101).

MISSING

The Interface supports RDBMS null data. In a table, a null value represents a missing or unknown value; it is not the same as a blank or zero. For example, you can use a column specification that allows nulls for a column that does not have to contain a value in every row (such as a raise amount in a table containing payroll data).

When a FOCUS MODIFY or MAINTAIN procedure enters data in a table, it represents missing data in the table as RDBMS standard null data for columns that allow nulls. At retrieval, null data is translated to the FOCUS missing data display value. The default FOCUS NODATA display value is the period (.).

When a field declaration omits the MISSING keyword, it defaults to OFF. The syntax is

```
MISSING = { OFF } , $  
          { ON  } , $
```

where:

OFF Is the default. FOCUS treats null data as blank or zero.

ON Displays the FOCUS NODATA value for missing data.

For null data support:

- The RDBMS table definition must describe a column that allows null data.
- The FOCUS file description for that table must describe a column that allows null data as a field with the attribute MISSING=ON.

Note: If the column allows null data but the corresponding field in the FOCUS file description uses the attribute MISSING=OFF, null data appears as a zero or blank. In MODIFY or MAINTAIN, incoming null values for these fields are stored as zero or blank. This practice is not recommended since it can affect the results of SUM or COUNT aggregate operations, as well as allowing the (perhaps unintentional) storage of real values for fields that, in fact, should be null.

Optional Field Attributes

The Interface supports the use of the optional field attributes DESCRIPTION, TITLE, DEFINE, and ACCEPT. Refer to *Describing Data Files* in the online *Language Reference* for information about these attributes.

Note: The Interface does not support the use of FIND in the ACCEPT attribute, nor does it support the keywords GROUP and FIELDTYPE.

Access Files

Each file description has a corresponding access file. The name of the access file must be identical to that of the file description. The access file associates a segment in the file with the table it describes.

The access file must identify the table and primary key (if there is one). It may also indicate the logical sort order of data and identify storage areas for the table. Access file field declarations can define fieldnames and aliases (Describing Multi-Table Relational Structures discusses fieldnames and aliases in the access file).

For multi-table structures, the access file also contains KEYFLD and IXFLD keywords that implement embedded JOINS. See Describing Multi-Table Relational Structures for details.

The following is an access file for the table EMPINFO:

```
SEGNAME = EMPINFO, TABLENAME = USER1.EMPINFO, KEYS = 1, WRITE = YES , $
```

Segment Declarations

The segment declaration in the access file establishes the link between the FOCUS file description and the RDBMS table or view. Attributes that constitute the segment declaration are: SEGNAME, TABLENAME, DBSPACE, WRITE, KEYS, KEYORDER, FALLBACK, SERVER, HOST, and HOME. Values for SEGNAME and TABLENAME are required; the remaining parameters acquire default values if they are omitted.

Note: If any part of the TABLENAME contains a dollar sign (\$) or a space(), enclose that part in double quotation marks, and enclose the entire TABLENAME value in single quotation marks. For example:

```
TABLENAME = 'USER1."TABLE$1"'
```

or

```
TABLENAME = 'USER1."TABLE 1"'
```

WRITE

The read/write security parameter, WRITE, determines whether or not the Interface allows FOCUS MODIFY and MAINTAIN operations (INCLUDE, UPDATE, or DELETE) on the table. The syntax is

$$\text{WRITE} = \left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\} ,$$

where:

YES Specifies read and write access. YES is the default.

NO Specifies read-only access. You can use read-only functions for displaying rows, such as MATCH and NEXT.

Note:

- The WRITE attribute has no effect on FOCUS reporting operations.
- Regardless of the WRITE value, RDBMS security must approve all operations and activities.

KEYS

The KEYS attribute indicates how many columns constitute the primary key for the table. Acceptable values range from 0 to 16. Zero, the default, indicates that the table does not have a primary key. In the corresponding file description, primary key columns must correspond to the first n fields described.

The syntax is

$$\text{KEYS} = \left\{ \begin{array}{l} 0 \\ n \\ \end{array} \right\} ,$$

where:

n Is a value from 0 to 16.

The KEYS value has the following significance in reporting operations:

- When you use FOCUS FST. or LST. direct operators, the Interface instructs the RDBMS to sort the answer set by the primary key in KEYORDER sequence.

Note: LST. processing is automatically invoked if you request SUM or WRITE of an alphanumeric field or use one in a report heading or footing.

- The Interface uses the KEYS value to determine the relationship between two joined tables. For example, if the primary key of one table is joined to the primary key of another table, the Interface can assume that a one-to-one relationship exists between the two tables. It then uses this assumption in conjunction with the JOIN specification and the current optimization setting to produce SQL statements. See The Interface Optimizer and Advanced Reporting Techniques for a detailed explanation.

To provide consistent access to tables, you should specify the KEYS attribute whenever a primary key exists.

The KEYS value also has significance in MODIFY and MAINTAIN operations (see Maintaining Relational Tables for a detailed explanation).

KEYORDER

The KEYORDER attribute is optional. It specifies the logical sort sequence of data by the primary key; it does not affect the physical storage of data. The Interface uses the KEYORDER value when you specify FST. and LST. direct operators in report requests. The syntax is

```
KEYORDER= { LOW }  
           { ASC } ,  
           { HIGH }  
           { DESC }
```

where:

LOW Sorts the rows in ascending primary key sequence. LOW is the default.

ASC Is a synonym for LOW.

HIGH Sorts the rows in descending primary key sequence.

DESC Is a synonym for HIGH.

For example, in DB2/2, to retrieve the most recent pay dates first, specify KEYORDER = HIGH for the SALINFO table:

```
SEGNAME = SALINFO, TABLENAME = USER1.SALINFO, KEYS = 2, WRITE = YES,  
KEYORDER = HIGH, $
```

The Interface requests rows ordered by SALEID and PAY_DATE in descending order. FST.PAY_DATE retrieves the most recent salary data for each employee.

KEYORDER also determines the logical sort order for MODIFY and MAINTAIN subcommands (see Maintaining Relational Tables).

FALLBACK (Teradata Only)

The FALLBACK keyword indicates whether a secondary copy of data is maintained on the Teradata DBC/1012, in addition to the primary table. The Interface incorporates this backup operation during table creation when the FOCUS CREATE FILE command is specified. You may also specify the FALLBACK keyword in native DBC/SQL CREATE TABLE statements.

The syntax is:

```
FALLBACK= { YES }  
           { NO } ,
```

where:

YES Establishes a backup copy.

NO Does not establish a backup copy. NO is the default setting.

After table creation, the backup copy is used to process requests when the primary data is unavailable. This mode of processing implies a tradeoff between system resources and performance for the sake of data availability. Consult the *Teradata System Manual* about the effects of FALLBACK.

The Long Fieldname Alternative

It is also possible to describe fieldnames greater than 12 characters using a combination of the file description and access file. However, since FOCUS now supports long fieldnames in the file description, this is no longer necessary or recommended. This method is documented here solely for upward compatibility with previous FOCUS releases.

For this method, specify the fieldname declaration with a 12-character (or less) fieldname in the file description; leave the ALIAS value blank. In the corresponding access file, specify a long fieldname declaration after the appropriate segment declaration.

A long fieldname declaration consists of two attributes, FIELD and ALIAS. The FIELD value is the 12-character fieldname from the file description. The ALIAS value identifies the full column name. The long fieldname declarations must follow their associated segment declarations and correspond to the field order of the file description.

For example, this EMPINFO file description contains blank ALIAS values for the DEPARTMENT and SALARY fields. The column names DEPARTMENT_CODE and CURRENT_SALARY are longer than 12 characters.

```
FILENAME=EMPINFO          , SUFFIX=SQLDBM, $
SEGNAME=EMPINFO          , SEGTYPE=S0, $
FIELD=EMP_ID             , ALIAS=EID    , USAGE=A9    , ACTUAL=A9    , $
FIELD=LAST_NAME          , ALIAS=LN    , USAGE=A15   , ACTUAL=A15   , $
FIELD=FIRST_NAME         , ALIAS=FN    , USAGE=A10   , ACTUAL=A10   , $
FIELD=HIRE_DATE          , ALIAS=HDT   , USAGE=YMD   , ACTUAL=DATE  , $
FIELD=DEPARTMENT         , ALIAS=      , USAGE=A10   , ACTUAL=A10   , MISSING=ON  , $
FIELD=SALARY              , ALIAS=      , USAGE=D9.2  , ACTUAL=D8    , $
FIELD=CURR_JOBCODE       , ALIAS=CJC   , USAGE=A3    , ACTUAL=A3    , $
FIELD=ED_HRS             , ALIAS=OJT   , USAGE=D6    , ACTUAL=D8    , MISSING=ON  , $
```

The long fieldname declarations in the access file are specified:

```
SEGNAME=EMPINFO, TABLENAME=USER1.EMPINFO, KEYS=1, WRITE=YES, , $
FIELD=DEPARTMENT , ALIAS=DEPARTMENT_CODE , $
FIELD=SALARY      , ALIAS=CURRENT_SALARY , $
```

The FOCUS OCCURS Segment

For use with FOCUS TABLE requests, you can describe tables that contain repeating columns as FOCUS OCCURS segments. Repeating data is not characteristic of normalized tables and views; however, denormalized tables may exist for some situations.

The FOCUS OCCURS segment, a virtual construct, eliminates the need to specify the names of every column in a TABLE request if a group of columns contains similar data. In the OCCURS segment definition, you redefine all the separate columns as one group that shares the same name. You can then define the order field, an internal FOCUS counter that enables you to access specific columns by their sequence numbers in the group instead of by separate names.

To define a FOCUS OCCURS segment, you need:

- An additional segment declaration in the file description to define the OCCURS segment.
- One field declaration to represent the repeating fields.
- Optionally, a field declaration for the ORDER field, a counter that is internal to FOCUS and not contained in the RDBMS.

Creating an OCCURS Segment

To create an OCCURS segment:

1. Describe the entire table as a single-table file description. Include a field declaration for each repeating column.
2. Add a segment description for the related OCCURS segment. In it, include a field declaration that redefines the repeating portion of the table.

The syntax for an OCCURS segment declaration is

```
SEGNAME=segname, PARENT=name, POSITION=field, OCCURS=nnnn,$
```

where:

<code>segname</code>	Is the name of the OCCURS segment, up to eight characters.
<code>parent</code>	Is the SEGNAME value of the table that contains declarations for the repeating fields.
<code>field</code>	Is the name of the first repeating field in the parent table.
<code>nnnn</code>	Is the number of repeating fields. Acceptable values range from 1 to 4095.

Note:

- The OCCURS segment declaration in the file description must not include the SEGTYPE keyword.
- The access file must not contain a corresponding segment declaration for the OCCURS segment.
- OCCURS segments are available for TABLE operations. To change data, you must specify the individual repeating fields from the parent segment.
- Using an OCCURS segment disables optimization.

The following SALARY table contains monthly payroll tax deductions for an employee, and the file description describes the repeating columns as 12 separate deduction fields. It also includes an OCCURS segment, OCC:

```
FILE=SALARY, SUFFIX=SQLDBM , $
SEGNAME=SALARY, SEGTYPE=S0 , $
FIELD=EMPID , ALIAS=EMPID , USAGE=A9 , ACTUAL=A9 , MISSING=OFF , $
FIELD=EMPNAME , ALIAS=EMPNAME , USAGE=A10 , ACTUAL=A10 , MISSING=OFF , $
FIELD=SALARY , ALIAS=SALARY , USAGE=D9.2 , ACTUAL=D8 , MISSING=OFF , $
FIELD=DEDUCT_JAN , ALIAS=DEDUCT_JAN , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_FEB , ALIAS=DEDUCT_FEB , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_MAR , ALIAS=DEDUCT_MAR , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_APR , ALIAS=DEDUCT_APR , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_MAY , ALIAS=DEDUCT_MAY , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_JUN , ALIAS=DEDUCT_JUN , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_JUL , ALIAS=DEDUCT_JUL , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_AUG , ALIAS=DEDUCT_AUG , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_SEP , ALIAS=DEDUCT_SEP , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_OCT , ALIAS=DEDUCT_OCT , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_NOV , ALIAS=DEDUCT_NOV , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_DEC , ALIAS=DEDUCT_DEC , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
SEGNAME=DEDUCT , PARENT=SALARY , POSITION=DEDUCT_JAN , OCCURS=12 , $
FIELD=TAX , ALIAS=TAX_DEDUCT , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
```

The OCCURS segment redefines the 12 deduction fields in the SALARY segment, beginning with DEDUCT1. The TAX field in the OCCURS segment represents the 12 repeating fields.

The corresponding access file does not contain a declaration for the OCCURS segment:

```
SEGNAME=SALARY, TABLENAME=USERID.DBMOCCUR, KEYS=1, WRITE=YES, $
```

The ORDER Field

The ORDER field is a FOCUS counter that assigns a sequence number to each field in a group of repeating fields. Specify this optional field when the order of data is significant. The ORDER field does not represent an existing column; it is used only for internal processing.

The ORDER field must be the last field described in the OCCURS segment. The syntax is

```
FIELD=name, ALIAS=ORDER, USAGE= In, ACTUAL=I4, $
```

where:

name Is any meaningful name.

In Is an integer (In) format.

Note:

- The value of ALIAS must be ORDER.
- The value of ACTUAL must be I4.

In the previous SALARY table example, no column explicitly specifies the month for each TAX field. To associate the month, the next example adds the ORDER field as the last field in the OCCURS segment:

```
FILE=SALARY, SUFFIX=SQLDBM , $
SEGNAME=SALARY, SEGTYPE=S0, $
FIELD=EMPID , ALIAS=EMPID , USAGE=A9 , ACTUAL=A9 , MISSING=OFF, $
FIELD=EMPNAME , ALIAS=EMPNAME , USAGE=A10 , ACTUAL=A10 , MISSING=OFF, $
FIELD=SALARY , ALIAS=SALARY , USAGE=D9.2 , ACTUAL=D8 , MISSING=OFF, $
FIELD=DEDUCT_JAN , ALIAS=DEDUCT_JAN , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_FEB , ALIAS=DEDUCT_FEB , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_MAR , ALIAS=DEDUCT_MAR , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_APR , ALIAS=DEDUCT_APR , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_MAY , ALIAS=DEDUCT_MAY , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_JUN , ALIAS=DEDUCT_JUN , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_JUL , ALIAS=DEDUCT_JUL , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_AUG , ALIAS=DEDUCT_AUG , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_SEP , ALIAS=DEDUCT_SEP , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_OCT , ALIAS=DEDUCT_OCT , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_NOV , ALIAS=DEDUCT_NOV , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=DEDUCT_DEC , ALIAS=DEDUCT_DEC , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
SEGNAME=DEDUCT , PARENT=SALARY , POSITION=DEDUCT_JAN , OCCURS=12 , $
FIELD=TAX , ALIAS=TAX_DEDUCT , USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=MONTH_NUMBER , ALIAS=ORDER , USAGE=I4 , ACTUAL=I4 , $
```

In subsequent report requests, you can use the DECODE function to translate the ORDER field to monthly values.

In this example, a DEFINE statement assigns the month to each counter value. You can specify a temporary field before the report request or in the file description:

```
DEFINE FILE SALARY
  MONTH/A3 = DECODE MONTH_NUMBER ( 1 'Jan' 2 'Feb' 3 'Mar' 4 'Apr'
                                     5 'May' 6 'Jun' 7 'Jul' 8 'Aug'
                                     9 'Sep' 10 'Oct' 11 'Nov' 12 'Dec' );
END

TABLE FILE SALARY
PRINT EMPNAME TAX
WHERE MONTH EQ 'Jan'
END
```

You can also use the ORDER field in selection tests. For example:

```
TABLE FILE SALARY
PRINT EMPNAME TAX
WHERE MONTH_NUMBER EQ 12
END
```

Describing Multi-Table Relational Structures

Topics:

- [Advantages of multi-table structures.](#)
- [Creating a multi-table structure.](#)
- [Multi-field embedded join.](#)
- [Remote segment descriptions.](#)

With the Interface, you can describe two or more related tables in a single file description and access file pair. This type of multi-table structure is called an *embedded join*.

In a multi-table structure, each participating table must have at least one field in common with at least one other table in the structure. Typically, this common field is the primary key of one table and the foreign key of the other. A single file description and access file pair can relate up to 64 separate tables in this manner.

Multi-table file descriptions and access files describe the relationships between tables. The Interface implements these relationships at run time by matching values in fields common to two or more tables. A report or MODIFY procedure can refer to any or all of the tables included in the multi-table description.

In these topics, the terms *primary key* and *foreign key* refer to the common fields in two related tables. These may or may not have been described as primary and foreign keys in SQL CREATE TABLE statements (RDBMS referential integrity). In practice, FOCUS can use any two fields that share a common format to relate tables in multi-table file descriptions.

Advantages of Multi-Table Structures

The advantages of defining multiple tables in a single FOCUS file description are:

- For reporting, a multi-table structure automatically joins tables referenced in the report request, so relationships between tables can be pre-defined for a user without creating an additional RDBMS view.
- A multi-table structure creates a FOCUS logical view of the data tailored to contain only those columns that should be seen by a user. The FOCUS DBA security feature can define additional levels of security.
- Tables described in a multi-table file description can be maintained together using the FOCUS MAINTAIN or MODIFY facilities and can take advantage of FOCUS referential integrity provided by the Interface. (Refer to Maintaining Relational Tables for a description of MAINTAIN and MODIFY with FOCUS referential integrity.)
- TABLE requests access only those RDBMS tables that contain columns referenced (either explicitly or implicitly) in the request.

A FOCUS TABLE request that references a dynamically joined structure generates SQL join predicates for all segments in the subtree above the highest segment referenced in the request. Multi-table file descriptions do not generate these predicates; in a multi-table structure, the subtree effectively begins with the highest referenced segment. This effect may cause identical TABLE requests to produce different reports when run against a dynamic JOIN structure and a multi-table file description that represent the same tree structure. See Advanced Reporting Techniques for a discussion of dynamic joins.

Creating a Multi-Table Structure

To create a multi-table structure, describe the tables, and the relationships between them, in a single file description and access file pair.

Multi-Table File Descriptions

All segment declarations in a multi-table file description must describe RDBMS tables or views. Each segment represents one table or view, up to a total of 64 segments. An RDBMS view counts as one segment toward the total, even if the view represents a join of two or more tables.

If several file descriptions (used only with TABLE requests) include the same table, you can avoid repeating the same description multiple times. Describe the table in *one* of the file descriptions, and use the CRFILE attribute in the other file descriptions to access the existing description. For a full explanation of remote segment descriptions, see Remote Segment Descriptions.

The embedded join facility uses existing FOCUS file description syntax (specifically the PARENT segment attribute) to describe the relationships between tables

```
FILENAME=mtname, SUFFIX=SQLDS    [,$]

SEGNAME=table1, SEGTYPE=      {S0}
                             {KL}    [,CRFILE=crfile1]    [,$]
                             {    }

FIELD=name, ..., $
.
.
.

SEGNAME=table2, SEGTYPE=      {S0 }
                             {U  } , PARENT=table1 [,CRFILE=crfile2] [,$]
                             {KL }
                             {KLU}

FIELD=name, ..., $
.
.
.
```

where:

- mtname** Is the one- to eight-character name of the multi-table file description.
- table1** Is the SEGNAME value for the parent table. If this segment references a remote segment description, table1 must be identical to the SEGNAME from the file description that contains the full definition of the RDBMS table's columns.
- name** Is any fieldname.
- table2** Is the SEGNAME value for the related table. If this segment references a remote segment description, table2 must be identical to the SEGNAME from the file description that contains the full definition of the RDBMS table's columns.
- S0** Indicates that the related table is in a one-to-many or many-to-many (non-unique) relationship with the table named as its parent.
- U** Indicates that the related table is in a one-to-one or a many-to-one (unique) relationship with the table named as its parent.
- KL** References a remote segment description. Indicates that the related table is in a one-to-many or many-to-many (non-unique) relationship with the table named as its parent.
- KLU** References a remote segment description. Indicates that the related table is in a one-to-one or a many-to-one (unique) relationship with the table named as its parent.
- crfile1** References a remote segment description. Indicates the name of the file description that contains the full definition of table1's columns.
- crfile2** References a remote segment description. Indicates the name of the file description that contains the full definition of table2's columns.

SEGNAME (Multi-Table)

The SEGNAME attribute names the segment. SEGNAME values must be unique in the file description. If the segment references a remote file description, its SEGNAME value must be identical to the SEGNAME from the file description that contains the full definition of the RDBMS table's columns.

SEGTYPE (Multi-Table)

The SEGTYPE attribute indicates how a table participates in a relationship. The SEGTYPE of the first segment in a multi-table file description is always S0 (or KL for a remote segment description). Thereafter, related tables (additional segments) are described as follows:

- SEGTYPE=S0 (zero) indicates that the related table is in a one-to-many or many-to-many (non-unique) relationship with the table named as its parent. (For every row of the parent table there may be more than one matching row in the related table.)
- SEGTYPE=U indicates that the related table is in a one-to-one or a many-to-one (unique) relationship with the table named as its parent. (For every row of the parent table, there is at most one matching row in the related table. For a one-to-one relationship to exist, both tables must share the same primary key. For a many-to-one relationship to exist, the primary key of the related table must be a subset of the primary key of the parent table.)
- SEGTYPE=KL references a remote segment description. It indicates that the related table is in a one-to-many or many-to-many (non-unique) relationship with the table named as its parent. (For every row of the parent table there may be more than one matching row in the related table.)
- SEGTYPE=KLU references a remote segment description. It indicates that the related table is in a one-to-one or a many-to-one (unique) relationship with the table named as its parent. (For every row of the parent table, there is at most one matching row in the related table. For a one-to-one relationship to exist, both tables must share the same primary key. For a many-to-one relationship to exist, the primary key of the related table must be a subset of the primary key of the parent table.)

PARENT (Multi-Table)

All segment declarations other than the first require the PARENT attribute. The PARENT value for a segment is the SEGNAME of the table to which it will be related at run time.

CRFILE (Multi-Table)

Specify the CRFILE attribute only if the actual description of the table's columns is stored in another (remote) file description. The CRFILE value must be the name of the file description that contains the full definition of the RDBMS table's columns. For a complete discussion of remote segment descriptions, see Remote Segment Descriptions.

FIELD (Multi-Table)

FOCUS fieldnames can consist of up to 66 alphanumeric characters (including any filename and segname qualifiers and qualification characters you may later prefix to them in requests). In the file description, fieldnames cannot include qualifiers. Column names are acceptable values if they meet the following FOCUS naming conventions:

- A name can consist of letters, digits, and underscore characters. Special characters and embedded blanks are not advised.
- The name must contain at least one letter.

Since the fieldname appears as the default column title for reports, select a name that is representative of the data. In requests, you can specify fieldnames, aliases, or a unique truncation of either, and you can qualify a fieldname with its filename and/or segment name.

Fieldnames must be unique in a single segment. If fieldnames are duplicated across segments, use the segment name as a high level qualifier when referencing them in requests.

ALIAS (Multi-Table)

The ALIAS value for each field must either be the full SQL column name (the Interface uses it to generate SQL statements), or blank if the column name is described in the access file. The ALIAS name must be unique in the segment. The ALIAS name must comply with the naming conventions described for fieldnames.

ALIAS names may be duplicated in the file description if they are defined for different tables.

ECOURSE File Description

The following file description relates the EMPINFO and COURSE tables. The EMPINFO table is described first; therefore, its segment declaration does not include the PARENT attribute. In the COURSE segment declaration, the PARENT keyword identifies EMPINFO as the parent and notifies the Interface that the two tables may be joined at run time for reporting purposes. The FILENAME ECOURSE identifies this relationship (for an example of a multi-table file description with a remote segment, refer to Remote Segment Descriptions):

```
FILE=ECOURSE , SUFFIX=SQLDBM, $
SEGNAME=EMPINFO , SEGTYPE=S0, $
FIELD=EMP_ID , ALIAS=EMP_ID , USAGE=A9 , ACTUAL=A9 , MISSING=OFF, $
FIELD=LAST_NAME , ALIAS=LAST_NAME , USAGE=A15 , ACTUAL=A15 , MISSING=OFF, $
FIELD=FIRST_NAME , ALIAS=FIRST_NAME , USAGE=A10 , ACTUAL=A10 , MISSING=ON , $
FIELD=HIRE_DATE , ALIAS=HIRE_DATE , USAGE=MDY , ACTUAL=DATE, MISSING=ON , $
FIELD=DEPARTMENT_CODE , ALIAS=DEPARTMENT_CODE ,
USAGE=A10 , ACTUAL=A10 , MISSING=ON , $
FIELD=CURRENT_SALARY , ALIAS=CURRENT_SALARY ,
USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=ED_HRS , ALIAS=ED_HRS , USAGE=D6.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=JOBCODE , ALIAS=JOBCODE , USAGE=A3 , ACTUAL=A3 , MISSING=ON , $
SEGNAME=COURSE , SEGTYPE=S0, PARENT=EMPINFO, $
FIELD=COURSE_NAME , ALIAS=COURSE_NAME, USAGE=A30 , ACTUAL=A30 , MISSING=OFF, $
FIELD=EMP_ID , ALIAS=EMP_ID , USAGE=A9 , ACTUAL=A9 , MISSING=OFF, $
FIELD=GRADE , ALIAS=GRADE , USAGE=A1 , ACTUAL=A1 , MISSING=OFF, $
FIELD=QUARTER , ALIAS=QUARTER , USAGE=A1 , ACTUAL=A1 , MISSING=OFF, $
FIELD=YR_TAKEN , ALIAS=YR_TAKEN , USAGE=A2 , ACTUAL=A2 , MISSING=OFF, $
```

Note: Multi-table file descriptions used in MAINTAIN and MODIFY procedures invoke FOCUS referential integrity. Refer to Maintaining Relational Tables for a discussion of referential integrity before deciding whether to use the same file descriptions for both reporting and maintenance procedures.

Multi-Table Access Files

A multi-table file description indicates that tables are related. However, to implement a join (TABLE) or FOCUS referential integrity, you must identify their common fields to FOCUS in the corresponding access file.

The multi-table access file includes a segment declaration for each table described in the file description, even if the segment in the file description was referenced remotely. The order of segment declarations in the access file does not have to match the order in the file description, but maintaining the same order enhances readability.

Each segment declaration in the access file must contain the required keywords described in Describing Relational Tables.

In addition, each segment, except the first, must identify the fields that it shares with its parent segment. In each access file segment declaration other than the first, the KEYFLD and IXFLD attributes supply the names of the primary key and foreign key fields that implement the relationships established by the multi-table file description.

The syntax for a multi-table access file is

```
SEGNAME=table1, TABLENAME=tname1, ..., $
SEGNAME=table2, TABLENAME=tname2, ...,
KEYFLD=pkfield, IXFLD=fkfield, $
```

where:

- `table1` Is the SEGNAME of the parent table from the multi-table file description.
- `table2` Is the SEGNAME of the related table from the multi-table file description.
- `tname1` Is the fully qualified table name for the parent table.
- `tname2` Is the fully qualified table name for the related table.
- `pkfield` Is the fieldname of the primary key column in the parent (table1) table.

`fkfield` Is the fieldname of the foreign key column in the related (table2) table.

KEYFLD and IXFLD

The KEYFLD and IXFLD keywords identify the field shared by a related table pair. KEYFLD is the FIELDNAME of the common column from the parent table. IXFLD is the FIELDNAME of the common column from the related table. KEYFLD and IXFLD must have the same datatype. It is recommended, but not required, that their lengths also be the same.

Note: An RDBMS index on both the KEYFLD and IXFLD columns provides the RDBMS with a greater opportunity to produce efficient joins. The columns must have the same datatype. If their length is the same, the RDBMS handles the JOIN more efficiently.

In the ECOURSE example from ECOURSE File Description, the fields EMP_ID in EMPINFO and EMP_ID in COURSE both contain employee identification numbers; they represent the common field. Since the COURSE table is the related table, its segment declaration in the ECOURSE access file identifies these common columns from EMPINFO and COURSE:

```
SEGNAME=EMPINFO ,TABLENAME=USERID.EMPINFO, KEYS=1 ,WRITE=YES,$
SEGNAME=COURSE ,TABLENAME=USERID.COURSE , KEYS=2 ,WRITE=YES,
KEYFLD=EMP_ID, IXFLD=EMP_ID,$
```

Multi-Field Embedded Join

In relational systems, a relationship or link between tables can depend on multiple columns. Embedded joins defined in multi-table file descriptions and access files also provide this ability. (The dynamic JOIN command also supports this feature; see Advanced Reporting Techniques.)

To describe a multi-field join for a multi-table structure, specify multiple fieldnames for the KEYFLD and IXFLD attributes in the access file. Separate the component fields participating in the multi-field join with slash (/) symbols.

The syntax for a multi-table access file with a multi-field link is

```
SEGNAME=name1, TABLENAME=table1,...,$
SEGNAME=name2, TABLENAME=table2,...,
KEYFLD=pkfield1/pkfield2,
IXFLD=fkfield1/fkfield2,$
```

where:

`name1` Is the SEGNAME of the parent table from the multi-table file description.

`name2` Is the SEGNAME of the related table from the multi-table file description.

`table1` Is the fully qualified table name for the parent table.

`table2` Is the fully qualified table name for the related table.

`pkfield1/pkfield2` Are the fieldnames that compose the primary key in the parent (or host) table.

`fkfield1/fkfield2` Are the fieldnames that compose the foreign key in the related table.

Up to 16 fields can participate in a link between two tables. The fields that constitute this multi-field relationship do not have to be contiguous either in the table or the FOCUS file description.

The number and order of fields for the KEYFLD value must correspond to those for the IXFLD value.

If the list of fields exceeds one line (80 characters), continue it on a second line. You can use as many lines as necessary, provided that each line is filled up to and including the 80th position. The 80th position cannot contain a slash (/) character.

To illustrate the multi-field join, suppose that the fields LAST_NAME and FIRST_NAME compose the primary key for the EMPINFO1 table. Also, assume that the fields LNAME and FNAME serve as the common fields (foreign key) in the COURSE1 table.

The ECOURSE1 file description (defined solely for the purpose of this example) reflects the new fields:

```
FILE=ECOURSE1 , SUFFIX=SQLDBM, $
SEGNAME=EMPINFO1 , SEGTYPE=S0, $
FIELD=LAST_NAME , ALIAS=LAST_NAME , USAGE=A15 , ACTUAL=A15 , MISSING=OFF, $
FIELD=FIRST_NAME , ALIAS=FIRST_NAME , USAGE=A10 , ACTUAL=A10 , MISSING=ON , $
FIELD=HIRE_DATE , ALIAS=HIRE_DATE , USAGE=MDY , ACTUAL=DATE, MISSING=ON , $
FIELD=DEPARTMENT_CODE , ALIAS=DEPARTMENT_CODE ,
USAGE=A10 , ACTUAL=A10 , MISSING=ON , $
FIELD=CURRENT_SALARY , ALIAS=CURRENT_SALARY ,
USAGE=D9.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=ED_HRS , ALIAS=ED_HRS , USAGE=D6.2 , ACTUAL=D8 , MISSING=ON , $
FIELD=JOBCODE , ALIAS=JOBCODE , USAGE=A3 , ACTUAL=A3 , MISSING=ON , $
SEGNAME=COURSE1 , SEGTYPE=S0 , PARENT=EMPINFO1, $
FIELD=COURSE_NAME , ALIAS=COURSE_NAME, USAGE=A30 , ACTUAL=A30 , MISSING=OFF, $
FIELD=LNAME , ALIAS=LNAME , USAGE=A15 , ACTUAL=A15 , MISSING=OFF, $
FIELD=FNAME , ALIAS=FNAME , USAGE=A10 , ACTUAL=A10 , MISSING=ON , $
FIELD=GRADE , ALIAS=GRADE , USAGE=A1 , ACTUAL=A1 , MISSING=OFF, $
FIELD=QUARTER , ALIAS=QUARTER , USAGE=A1 , ACTUAL=A1 , MISSING=OFF, $
FIELD=YR_TAKEN , ALIAS=YR_TAKEN , USAGE=A2 , ACTUAL=A2 , MISSING=OFF, $
```

In the ECOURSE1 access file, the KEYFLD and IXFLD values consist of fieldnames separated by a slash (/):

```
SEGNAME=EMPINFO1 , TABLENAME=USERID.EMPINFO1 , KEYS=2 , WRITE=YES, $
SEGNAME=COURSE1 , TABLENAME=USERID.COURSE1 , KEYS=3 , WRITE=YES,
KEYFLD=LAST_NAME/FIRST_NAME , IXFLD=LNAME/FNAME, $
```

Remote Segment Descriptions

Remote segment descriptions simplify the process of describing hierarchies of RDBMS tables to FOCUS. You can use them for file descriptions and access files that provide read-only access to RDBMS tables; you cannot use the same descriptions for MODIFY or MAINTAIN procedures.

The Interface allows you to create multi-table file descriptions and access files that define RDBMS tables or views as segments in the description. Each file segment description consists of a segment declaration followed by descriptions of all of the fields in the segment (columns of the corresponding table).

It is not unusual for several file descriptions to contain a segment description for the same RDBMS table. If a table's description is detailed in one file description, you can automatically incorporate that description in other file descriptions. The syntax is

```
SEGNAME=segname , PARENT=parent , SEGTYPE= { KL } , CRFILE=filename, $
{ KLU }
```

where:

<code>segname</code>	Is identical to the SEGNAME in the file description that contains the full description of the RDBMS table's columns (the remote file description).
<code>parent</code>	Is the parent of the segment.
<code>KL</code>	Describes one-to-many relationships.
<code>KLU</code>	Describes unique relationships.
<code>filename</code>	Is the name of the remote file description that contains the full description of the segment's fields.

Note:

- You may not use the attributes CRKEY and CRSEGNAME.
- If a file description that contains a CRFILE cross-reference to a segment in another file description does not contain a declaration for that segment in its own access file, the Interface issues a FOC1351 message as a warning. The Interface then attempts to use the corresponding segment reference from the cross-referenced access file. If the information in that access file (such as the KEYFLD/IXFLD pair) can function correctly with the local file description, the Interface continues processing. If not, it displays additional error messages, and processing terminates. The FOC1351 message should be considered only a warning unless accompanied by additional messages.
- You may not use file descriptions containing remotely-described segments in CREATE FILE commands, or in MODIFY or MAINTAIN procedures.
- SEGTYPEs KL and KLU describe segments whose field attributes are described in other file descriptions that FOCUS may read at run-time. You can use remote segment descriptions for situations in which several file descriptions introduce different views on the same collection of RDBMS tables. You describe the fields of one or several tables in one file description, and refer to this first file from other file descriptions without including all the field descriptions again.
- The separately-described segment must have the same name in the file in which it is defined and in the file that references it. The CRFILE attribute identifies the file description that contains the complete segment definition. For example:

```
SEGNAME=DEPT, PARENT=EMP, SEGTYPE=KL, CRFILE=MAINFILE, $
```
- The Interface obtains the descriptions of fields in the DEPT segment in this file from the DEPT segment in the MAINFILE file description, where they must physically reside; DEPT cannot be a KL or KLU segment in MAINFILE. Similarly, a dynamic JOIN may not specify a KL or KLU segment as the cross-referenced segment in the target file.
- Once you specify the CRFILE attribute in a file description, that specification becomes the default for subsequently described segments; if you later wish to describe a segment locally (using the traditional method), you must respecify the local filename using the CRFILE attribute, even though this is not technically a cross-reference. Obviously, the same holds true if you wish to change cross-referenced files from one segment to the next.
- FOCUS reads only the field attributes from the segment, not the segment attributes. The MAINFILE file description does not have to be the description of a real file. It does not need an access file; it just needs the description of the named segment. Thus, you can set up one large file description that contains field descriptions of all RDBMS tables or views you may use in reporting, even if you only use the large description for reference, not for reporting.
- If you describe the root segment remotely, specify its SEGTYPE as KL.
- If two or more segments in a FOCUS file description represent the same RDBMS table, only one of these segments can have a remote segment description. This requirement is necessary in order to preserve unique SEGNAMEs (because a remote SEGNAME must be identical to its corresponding local SEGNAME).
- Remote segment descriptions exist only for convenience. They save typing effort but offer no logical implications regarding parent-child relationships and their implementation.

- A file description that contains remote segment declarations must have its own access file for defining the relationships between all its segments (including the remote segments). This local access file overrides the access file corresponding to the CRFILE file description, if one exists.

The following example shows the ECOURSE file description with the COURSE segment described remotely:

```
FILE=EMPINFO      ,SUFFIX=SQLDBM,$
SEGNAME=EMPINFO  ,SEGTYPE=S0,$
FIELD=EMP_ID     ,ALIAS=EMP_ID     ,USAGE=A9   ,ACTUAL=A9   ,MISSING=OFF,$
FIELD=LAST_NAME  ,ALIAS=LAST_NAME  ,USAGE=A15  ,ACTUAL=A15  ,MISSING=OFF,$
FIELD=FIRST_NAME ,ALIAS=FIRST_NAME ,USAGE=A10  ,ACTUAL=A10  ,MISSING=ON  ,$
FIELD=HIRE_DATE  ,ALIAS=HIRE_DATE  ,USAGE=MDY  ,ACTUAL=DATE ,MISSING=ON  ,$
FIELD=DEPARTMENT_CODE ,ALIAS=DEPARTMENT_CODE ,
  USAGE=A10  ,ACTUAL=A10  ,MISSING=ON  ,$
FIELD=CURRENT_SALARY ,ALIAS=CURRENT_SALARY ,
  USAGE=D9.2 ,ACTUAL=D8   ,MISSING=ON  ,$
FIELD=ED_HRS      ,ALIAS=ED_HRS      ,USAGE=D6.2 ,ACTUAL=D8   ,MISSING=ON  ,$
FIELD=JOBCODE     ,ALIAS=JOBCODE     ,USAGE=A3   ,ACTUAL=A3   ,MISSING=ON  ,$
SEGNAME=COURSE   ,PARENT=EMPINFO   ,SEGTYPE=KL,CRFILE=COURSE,$
```

The corresponding access file is:

```
SEGNAME=EMPINFO ,TABLENAME=USERID.EMPINFO ,KEYS=1 ,WRITE=YES,$
SEGNAME=COURSE  ,TABLENAME=USERID.COURSE  ,KEYS=2 ,WRITE=YES,
  KEYFLD=EMP_ID ,IXFLD=EMP_ID,$
```

The Interface Optimizer

Topics:

- [The set optimization command.](#)
- [Optimization logic.](#)
- [DEFINE fields.](#)

Optimization is the process in which the Interface translates the projection, selection, join, sort, and aggregation operations of a FOCUS report request to their SQL equivalents and passes these SQL statements to the RDBMS for processing. Interface optimization allows the RDBMS to perform the work for which it is best suited and reduces the volume of RDBMS-to-FOCUS communication, improving response time. It also enables the RDBMS to make use of its own internal optimization techniques.

The SET OPTIMIZATION Command

To invoke the optimization process, enter the following Interface command at the FOCUS command level

```
SQL [target_db] SET {OPTIMIZATION} {OFF }  
                   {SQLJOIN } {ON }  
                   { } {FOCUS}  
                   { } {SQL }
```

where:

[target_db](#) Is the target RDBMS. Valid values are:

SQLSYB	Sybase SQL Server.
SQLORA	Oracle.
SQLDBM	IBM DB2/2.
SQLDBC	Teradata DBC/1012.
SQLODBC	Microsoft ODBC.
SQLINF	Informix
EDA	Enterprise Data Access.

Note: Omit if you previously issued the SET SQLENGINE command.

[SQLJOIN](#) Is a synonym for OPTIMIZATION.

[OFF](#) Instructs the Interface to create SQL statements for simple data retrieval from each table. FOCUS handles all aggregation, sorting, and joining on your workstation to produce the report.

[ON](#) Instructs the Interface to create SQL statements that take advantage of RDBMS join, sort, and aggregation capabilities. Is compatible with previous releases in regard to the multiplicative effect: misjoined unique segments and multiplied lines in PRINT and LIST based report requests do not disable optimization. Other cases of the multiplicative effect invoke the Interface-managed native join logic. ON is the default.

[FOCUS](#) Passes join logic to the RDBMS only when the results will be the same as from a FOCUS-managed request. Misjoined unique segments, the multiplicative effect, and multiplied lines in PRINT and LIST based report requests invoke the Interface-managed native join logic.

[SQL](#) Passes join logic to the RDBMS in all possible cases. The multiplicative effect does not disable optimization, even in cases involving aggregation (SUM, COUNT). Does not pass join logic to the RDBMS for tables residing on multiple subsystems, for tables residing on multiple DBMS platforms, and, depending on the target RDBMS, for SET ALL=ON. See Advanced Reporting Techniques for information about SET ALL=ON and join optimization.

Optimization Example

The example in this topic demonstrates the differences between SQL statements generated with and without optimization; the report request joins tables EMPINFO and FUNDTRAN with traces FSTRACE3 and FSTRACE4 allocated.

When optimization is disabled, the Interface generates two SELECT statements. The first SELECT retrieves any rows from the EMPINFO table that have the value MIS in the DEPARTMENT column. For each EMPINFO row, the second SELECT retrieves rows from the cross-referenced FUNDTRAN table, resolving the parameter marker, ?, with the value of the host field (EMP_ID). Both SELECT statements retrieve answer sets, but FOCUS performs the join, sort, and aggregation operations. The sample procedure below

```
SQL SQLDBM SET OPTIMIZATION OFF
```

```
JOIN EMP_ID IN EMPINFO TO ALL EMP_ID IN FUNDTRAN AS J1
```

```
TABLE FILE EMPINFO
SUM AVE.CURRENT_SALARY
BY BANK_CODE
BY BANK_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

produces the following messages:

```
(FOC2510) JOIN PROCESSING BY SQL DISABLED FOR THE FOLLOWING REASON:
(FOC2511) DISABLED BY USER
(FOC2590) AGGREGATION NOT DONE FOR THE FOLLOWING REASON:
(FOC2592) OPTIMIZATION/JOIN HAS BEEN DISABLED
SELECT T1.EMP_ID,T1.DEPARTMENT_CODE,T1.CURRENT_SALARY FROM
USERID.EMPINFO T1 WHERE (T1.DEPARTMENT_CODE = 'MIS') FOR FETCH
ONLY;
SELECT T2.BANK_CODE,T2.BANK_NAME FROM USERID.FUNDTRAN T2 WHERE
(T2.EMP_ID = ?) FOR FETCH ONLY;
```

With optimization enabled, the Interface generates one SELECT statement that incorporates the join, sort, and aggregation operations. The RDBMS manages and processes the request; FOCUS only formats the report. The sample procedure below

```
SQL SQLDBM SET OPTIMIZATION ON
```

```
JOIN EMP_ID IN EMPINFO TO ALL EMP_ID IN FUNDTRAN AS J1
```

```
TABLE FILE EMPINFO
SUM AVE.CURRENT_SALARY
BY BANK_CODE
BY BANK_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

produces the following message:

```
AGGREGATION DONE...
SELECT T2.BANK_CODE,T2.BANK_NAME, AVG(T1.CURRENT_SALARY) FROM
USERID.EMPINFO T1,USERID.FUNDTRAN T2 WHERE (T2.EMP_ID =
T1.EMP_ID) AND (T1.DEPARTMENT_CODE = 'MIS') GROUP BY
T2.BANK_CODE,T2.BANK_NAME ORDER BY T2.BANK_CODE,T2.BANK_NAME
FOR FETCH ONLY;
```

Both situations produce the same report.

Optimization Logic

Optimization means passing the following tasks to the RDBMS:

1. Record selection and projection.
2. Joins.
3. Sorting.
4. Aggregation.

Aside from record selection and projection, the Interface does not offload these functions to the RDBMS if you set `OPTIMIZATION OFF`. When `OPTIMIZATION` is not `OFF`, the Interface evaluates each report request for the existence of joins, sort operations, and finally aggregation operations. It automatically invokes optimization for record selection and projection operations.

Record Selection and Projection

Regardless of the `OPTIMIZATION` setting, the Interface may pass record selection and projection to the RDBMS; it is always more efficient to do so.

Record Selection

The Interface can translate all forms of FOCUS record selection to predicates of an SQL `WHERE` clause, except those that contain:

- Certain FOCUS DEFINE fields.
- EDIT to perform datatype conversions, EDIT of an alphanumeric field that was the result of such a conversion, or EDIT of a DEFINE field.
- LIKE with fields created using DEFINE or COMPUTE.
- DATE fields with formats other than YMD or YYMD.

Starting with this release, the Interface optimizes screening conditions based on DEFINE fields that derive their values from a single segment of the join structure. The Interface optimizes these screening conditions as long as you do not set `OPTIMIZATION OFF`, even when join optimization in your request is disabled by the Interface.

To use LIKE in a WHERE clause, make sure any constant in the LIKE predicate either has the same length as the field used in the comparison, is padded with blanks or underscore characters (`_`) to maintain the appropriate length, or contains the `%` wildcard character to denote any sequence of characters (see the *Language Reference* manual for a discussion of LIKE).

If you fail to specify a wildcard pattern in the LIKE predicate, the WHERE clause passes an equality predicate to the RDBMS instead of the LIKE predicate. For example

```
WHERE EID LIKE 'A'
```

will generate

```
WHERE (T1.EID = 'A')
```

not:

```
WHERE (T1.EID LIKE 'A')
```

In addition, because of unpredictable comparisons between VARCHAR datatypes, if the specified pattern is not equal in length to the column used in the comparison, and the pattern contains one or more wildcard characters but does not terminate with the percent character (`%`), the WHERE clause is not passed to the RDBMS. FOCUS performs the screening condition on all rows returned from the RDBMS.

Projection

In relational terminology, projection is the act of retrieving any subset of the available columns from a table. The Interface implements projection by retrieving only those columns referenced in a TABLE request. Projection reduces the volume of data returned from the RDBMS, which, in turn, improves application response time.

Joins

This topic applies to joins invoked either by the FOCUS dynamic JOIN command (see Advanced Reporting Techniques) or by the embedded join facility (see Describing Multi-Table Relational Structures).

When you invoke OPTIMIZATION, the Interface attempts to build a single SQL statement. In order for the Interface to create one SQL statement that incorporates an RDBMS join on primary and foreign keys:

- The tables must share a single retrieval path in the joined structure; multiple paths are not permitted.
- Except for the lowest level segment referenced in the request, primary keys must exist (KEYS > 0 in the access file) for all segments referenced in a report request that includes:
 - Aggregation operators such as SUM or COUNT, when the OPTIMIZATION setting is ON or FOCUS.
 - Multiple FST. or LST. operators on a segment that could return more than one record per sort break, regardless of the OPTIMIZATION setting.

In prior releases, SUM or COUNT operations at any level other than the lowest level in the join structure caused the RDBMS to duplicate data. In effect, each host row was replicated for each associated row in the cross-referenced table; this duplication of data is known as the *multiplicative effect*. In prior releases, detection of the multiplicative effect disabled optimization unless the user set OPTIMIZATION to SQL (see The SET OPTIMIZATION Command).

Starting with this release, when all segments other than the lowest level segment in the request have primary keys, the Interface passes such joins to the RDBMS with an SQL ORDER BY clause on all columns of each segment's primary key, in top to bottom order. The resulting sort on the returned answer set enables the Interface to eliminate duplicate rows before passing the data to FOCUS. This technique is called Interface-managed native join optimization.

In some cases, Interface-managed joins may be less efficient than FOCUS-managed joins from prior releases. To invoke a FOCUS-managed join, set OPTIMIZATION to OFF.

When the Interface manages join optimization, it does not optimize aggregation or sorting, as described in the next two topics. This behavior is consistent with that of prior releases when the join was not passed. However, expressions that are based on single segments in the structure are passed (see Record Selection and Projection).

- The tables must reside at the same location (subsystem).
- The file descriptions for the tables must not include any OCCURS segments.
- You must not join more than 15 tables or views.

If these conditions are satisfied, the Interface generates one SQL SELECT statement that joins all the referenced tables in a single request.

If you set Interface optimization OFF, or if any condition is not satisfied, the Interface generates an individual SQL statement for each table; FOCUS performs the required processing on the returned answer sets.

Note:

- The SET ALL command (or the ALL. prefix) controls the type of join passed to the RDBMS. See Advanced Reporting Techniques for information about the SET ALL command and the retrieval of "missing" rows.
- A FOCUS TABLE request that references a dynamic JOIN generates SQL join predicates for all segments in the subtree above the highest segment referenced in the request. The Interface does not generate these predicates for multi-table file descriptions (see Describing Multi-Table Relational Structures); in a multi-table structure the subtree effectively begins with the highest referenced segment. This effect may cause identical TABLE requests to produce different reports when run against a dynamic JOIN structure and a multi-table file description that represent the same tree structure.
- When the Interface does not pass a join to the RDBMS, the order in which you join the files becomes critical. To minimize the number of cursors opened, order the files from left to right by increasing number of records in the returned answer set.

Join order should not matter in an optimized join; however, performance may degrade when you join more than three tables.

Sorting

Next, if the request specifies a sort operation, the Interface must determine how to sort the data. The Interface transfers the sort operation to the RDBMS only when:

- It can generate a single SQL statement (see Joins).
- No FOCUS sort phrase uses a DEFINE field as a sort field.
- Interface-managed join optimization is not in effect (see Joins).

Under these conditions, the Interface invokes one of the following types of RDBMS-managed sorting:

- If the request does not use the direct operators FST. and LST., the Interface translates sort phrases (BY or ACROSS) into SQL ORDER BY clauses, thus passing responsibility for the primary sorting of data to the RDBMS.
- Starting with this release, when the Interface determines that it will retrieve at most one segment instance per sort break, it translates FOCUS FST. and LST. operators to SQL MIN and MAX operators and translates sort phrases (BY and ACROSS) into SQL ORDER BY clauses. The Interface applies this technique both in requests that explicitly specify the FST. or LST. operators and requests that implicitly use them by referencing a non-numeric field in a report heading or footing.

FOCUS FST. and LST. operators retrieve the first or last segment instance per sort break. By definition, if a FOCUS request includes FST. or LST. operators on multiple fields from one segment, each field value displayed on the resulting report must come from the same instance of that segment. SQL MIN and MAX operators do not dictate that the resulting fields all come from the same segment instance.

Therefore, before translating the FST. and LST. operators in a FOCUS request to SQL MIN and MAX operators, the Interface must ascertain that it will retrieve at most one segment instance per sort break. It makes this determination by analyzing the sort fields in the request, the primary key of each segment in the join structure, and any join fields that are components of the primary keys.

One segment instance is returned per sort break in any of the following situations:

- The sort field includes a segment's entire primary key.
- The sort field is a partial key joined to another sort field consisting of the remaining primary key component.
- The request includes only one FST. or LST. operator on a segment.
- If the report request contains FST. or LST. operators on a segment that may return multiple instances, the Interface directs the RDBMS to sort the data by primary key in the sequence specified by the access file KEYORDER attribute. From this, the Interface retrieves column values for the logically first (FST.) or last (LST.) key values of the returned data. After the RDBMS sort, FOCUS re-sorts the data according to the sort phrases.

Note: LST. processing is invoked for SUM or WRITE operations involving alphanumeric or date fields and for alphanumeric or date fields in a report heading or footing.

From a performance standpoint, consider using the FOCUS TABLEF command in conjunction with RDBMS-managed sorting to free FOCUS from having to verify the sort order. Refer to Advanced Reporting Techniques for the TABLEF command.

Aggregation

FOCUS aggregation verbs SUM, COUNT, and WRITE, and direct operators MIN., MAX., and AVE., retrieve a final aggregated answer set rather than individual values. Since the RDBMS handles aggregation efficiently, the Interface structures its retrieval request so that the RDBMS performs the aggregation. The Interface passes aggregation to the RDBMS when:

- The IF or WHERE tests in the FOCUS request translate to SQL WHERE clauses.
- FOCUS generates a single SQL statement (see Joins).
- Interface-managed join optimization is not in effect (see Joins).

- The request specifies only the FOCUS SUM, COUNT, or WRITE aggregate verbs and the MIN., MAX., AVE., SUM., DST., and CNT. direct operators. Under certain conditions (described in Sorting), the request can include the FOCUS FST. and LST. direct operators.
- The request does not reference DEFINE fields that do not comply with the definition of "valued expressions" or that are otherwise restricted (see SQL Limitations).

Note: FOCUS calculates COMPUTE fields on its internal matrix; they do not affect the Interface's ability to pass requests for aggregation to the RDBMS.

The Interface translates IF TOTAL and WHERE TOTAL tests to the SQL HAVING clause. It translates IF TOTAL and WHERE TOTAL tests on DEFINE and COMPUTE fields subject to the general limitations on the use of DEFINE in aggregation (see Valued Expressions).

DEFINE Fields

The RDBMS can process certain types of DEFINE expressions *as part of aggregation or record selection operations only*. DEFINE expressions that are not translated for the RDBMS are listed in SQL Limitations.

- For RDBMS-managed record selection, the DEFINE expression must be an arithmetic valued expression, a character string valued expression, or a logical expression.
- For RDBMS-managed aggregation, the DEFINE expression must be an arithmetic or character string valued expression.

For more information about DEFINE statements and examples of DEFINE expressions, refer to Creating Temporary Fields in online help.

Valued Expressions

There are two types of valued expressions: arithmetic and character string.

Arithmetic Expressions

Character String Expressions

Arithmetic Expressions

Arithmetic expressions return a single number. DEFINE fields are classified as arithmetic if the expression on the right includes one or more of the following elements:

- Real-field operands of numeric datatype (I, P, D, F).
- Numeric constants.
- Arithmetic operators (**, *, /, +, -).
- The subtraction of one DATE field from another.
- DEFINE-field operands satisfying any of the preceding items.

For example, the arithmetic expression in the following DEFINE uses a numeric real-field operand (CURR_SAL), a previously-specified DEFINE field (OTIME_SAL), and two numeric constants (1.1 and 100):

```
NEW_SAL/D12.2= ((CURR_SAL + OTIME_SAL) * 1.1) - 100;
```

Character String Expressions

Character string expressions return a character string. DEFINE fields are classified as character strings if the expression on the right includes one or more of the following elements:

- Real-field operands of alphanumeric (A) datatype. (TX field formats are not supported in DEFINE expressions.)
- String constants.
- String concatenation operators (||).
- DEFINE-field operands satisfying any of the preceding items.

For example, the character string expression in the following DEFINE uses two alphanumeric field operands (LAST_NAME and FIRST_NAME), a string constant (' '), and string concatenation operators (||):

```
FORMAL_NAME/A36= LAST_NAME || ' ' || FIRST_NAME;
```

Logical Expressions

Logical expressions return one of two values: True (1) or False (0). DEFINE fields are classified as logical if the expression on the right includes any of the following elements:

- Real-field operands of any FOCUS-supported datatype (including DATE fields).
- Constants of a datatype consistent with fields in the predicate.
- Relational operators (EQ, NE, GT, LT, GE, LE).
- Logical operators (AND, OR, NOT).
- Arithmetic or character string expression operands (previously described).
- DEFINE-field operands satisfying any combination of the preceding items.

For example, the logical expression in the following DEFINE is composed of two expressions connected by the logical operator OR; each part is itself a logical expression:

```
SALES_FLAG/I1= (DIV_CODE EQ 'SALES') OR (COMMISSION GT 0);
```

In the next example, the DEFINE field, QUOTA_CLUB, is the value of a logical expression composed of two other expressions connected by the logical operator AND. Note that the first expression is the previously-specified DEFINE field, SALES_FLAG:

```
QUOTA_CLUB/I1= (SALES_FLAG) AND (UNITS_SOLD GT 100);
```

SQL Limitations

Since the FOCUS reporting language is more extensive than native SQL, the Interface cannot pass certain DEFINE expressions to the RDBMS for processing. The Interface does not offload DEFINE-based aggregation and record selection if the DEFINE includes:

- User-written subroutines.
- IF-THEN-ELSE compound expressions.
- Self-referential expressions such as:

```
X=X+1;
```

- EDIT functions for numeric-to-alpha or alpha-to-numeric field conversions.
- DECODE functions for field value conversions.
- Relational operators CONTAINS, OMTS, IS-MISSING, INCLUDES, and EXCLUDES.
- FOCUS subroutines ABS, INT, MAX, MIN, LOG, and SQRT.

Note: Do not confuse the FOCUS user-written subroutines MAX and MIN with the MAX. and MIN. prefix operators. DEFINE fields cannot include prefix operators.

- Expressions involving fields with ACTUAL=DATE, except for the subtraction of one DATE field from another and all logical expressions on DATE fields.
- Extended Matrix Reporting (EMR) cell calculations. EMR is also referred to as Financial Reporting Language (FRL).

Note: EMR report requests are extended TABLE requests. The Extended Matrix Reporting Language provides special functions for detailed reporting; consult the online *Language Reference* for more information.

Advanced Reporting Techniques

Topics:

- [The TABLEF command.](#)
- [Creating extract files on the RDBMS.](#)
- [The dynamic JOIN command.](#)
- [Missing rows of data in cross-referenced tables.](#)
- [Search limits.](#)
- [Multiple retrieval paths.](#)

Using the Interface, you can create graphs with FOCUS GRAPH requests, design interactive procedures with Dialogue Manager and forms or update data using MAINTAIN. However, of all the facilities the Interface makes available to you for reading, retrieving, and organizing data from RDBMS tables, the most widely used are the reporting facilities, invoked by the FOCUS TABLE, TABLEF, and MATCH commands. Some reporting features for the Interface vary from standard FOCUS; these minor variations are discussed in these topics.

If you are not familiar with FOCUS syntax, you can create report requests with Report Painter, a graphical report generator. Consult online help for more information about FOCUS reporting.

Some FOCUS and SQL query functions are similar:

- SQL SELECT is equivalent to the FOCUS verb PRINT. Both display individual values for a given column or field.
- SQL aggregate functions SUM, COUNT *, MAX, MIN, and AVG are comparable to FOCUS SUM., CNT., MAX., MIN., and AVE. field prefix operators.
- The SQL WHERE predicate is similar to FOCUS IF or WHERE screening tests.
- The SQL ORDER BY sort phrase is equivalent to the FOCUS BY sort phrase.
- SQL set operations such as UNION, intersection, and difference are available as FOCUS MATCH FILE options. The FOCUS MATCH FILE command generates a subset of data from tables or views. The subset is stored in a HOLD extract file and is available as a source for further report processing. Refer to the *Language Reference* manual for more information about the MATCH FILE command.

You can issue an SQL SELECT request from within FOCUS using the Direct SQL Passthru facility (see Direct SQL Passthru).

The TABLEF Command

TABLEF FILE is an alternate for the TABLE FILE command when the answer set returned from the RDBMS is already in the sort order required for the report. TABLEF is faster and more efficient because FOCUS does not build (and sort) an internal matrix, but works directly on the data returned by the RDBMS.

To increase efficiency even more, create an SQL clustered index on the sort field used in the TABLEF request. The RDBMS physically stores the data in the same order as the clustered index, so satisfying the request requires fewer I/Os to data pages.

Most reporting syntax is acceptable in TABLEF requests, but you must be aware of the following:

- If BY phrases are passed to the RDBMS as SQL ORDER BY phrases, the returned answer sets are in sorted order and FOCUS does not have to sort them again or verify the sort order.

In some circumstances FOCUS does not pass an ORDER BY phrase to the RDBMS (see The Interface Optimizer), so you must verify that the ORDER BY is passed before relying on the sorted order of the returned answer set.
- You may not use ACROSS phrases.
- You may use only one verb; no multiple verb sets are allowed.
- The RETYPE command is not available.

- You can include the ON TABLE HOLD statement in the TABLEF request to produce an extract file.

Note: In a request that generates report output to the terminal, TABLEF holds locks on data pages until the Interface issues a COMMIT (usually when the report display is terminated). If the Isolation Level (see Maintaining Relational Tables) is set to Repeatable Read (RR), all pages accessed to produce the report are locked; if it is set to Cursor Stability (CS), the last page accessed is locked. These locks may prevent access to the data by other applications; if this presents a problem, reports generated to the terminal using TABLEF should be processed and terminated as quickly as is feasible. This consideration does not apply to regular TABLE requests, because the Interface normally issues the COMMIT prior to displaying the report. Nor is it a problem for requests that generate only offline reports or extract (HOLD) files, as the answer set is exhausted automatically.

Creating Extract Files on the RDBMS

Using FOCUS TABLE syntax, you can create extract files (tables) in the RDBMS. You can then use these tables, like any other RDBMS table, for both read-only and read-write operations. In fact, with the FOCUS HOLD FORMAT `target-db` option, you can create RDBMS tables from *any* FOCUS-readable file. This feature facilitates data migration and leaves the original source unaffected.

In order to create RDBMS tables and indexes, you must have an adequate level of RDBMS authority. Contact your site DBA for more information.

To extract data and convert it to an RDBMS table, issue the HOLD command with the FORMAT `target-db` option either in the report request or after the report has printed. The Interface generates single-table file descriptions and access files, and it creates and loads one RDBMS table. If the report request uses the verbs PRINT or LIST, it also creates both a FOCLIST field with internal list values and a unique index on all BY fields and the FOCLIST field. If the request uses the verb SUM, the Interface creates a unique index on any BY fields.

Within the report request, the syntax is:

```
ON TABLE HOLD [ AS name ] FORMAT {target-db}
```

At the command level, the syntax is

```
HOLD [ AS name ] FORMAT {target-db}
```

where:

`name` Is a name for the extract file; the default name is HOLD. Maximum 27 characters (including a period to separate the creator ID from the table name). The first 8 characters become the resulting file description name and the FILENAME value in the file description. All 27 characters become the TABLENAME value in the access file.

Note: If the name contains a period (.), the characters preceding it are treated as the creator ID; characters following the period become the file description name and the FILENAME value in the file description. Consult Describing Multi-Table Relational Structures for information about creator IDs and TABLENAME values.

`target-db` Is the target RDBMS for the extract data file. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server
<code>SQLORA</code>	Oracle
<code>SQLDBM</code>	IBM DB2/2
<code>SQLDBC</code>	Teradata DBC/1012
<code>SQLODBC</code>	Microsoft ODBC
<code>SQLINF</code>	Informix

All HOLD tables are permanent and must be explicitly dropped.

The Interface automatically creates the file description and access file for the HOLD table.

Generated File Descriptions

Even if the original file description describes several segments, the file description resulting from the ON TABLE HOLD extract is a single-table description; it contains the attributes described in Describing Multi-Table Relational Structures. Following is a list of the generated keyword/value pairs:

- The default FILENAME value is HOLD. If you specified a name with the AS phrase, the FILENAME value is the first eight characters of that name. If the name contains a period (.), the characters preceding the period are treated as the creator id; characters following the period become the resulting file description name and FILENAME value in the file description.
- The SUFFIX value is the same as the value used for the target-db.
- The SEGNAME value is SEG01 and the SEGTYPE value is S0.
- FIELDNAME and ALIAS values from the original file description are retained. If the original file description does not define aliases, the original fieldnames are also the new ALIAS values.

If the report request contains an AS phrase to rename a field, the AS phrase name becomes the new value for FIELDNAME and the original fieldname becomes the new value for ALIAS. The AS phrase name is also included as a TITLE value.

The FOCUS verbs PRINT and LIST create an additional field named FOCLIST that contains internal list values.

- The resulting USAGE field formats are generally the same as the original USAGE formats. The new ACTUAL formats are converted based on original USAGE formats and certain conditions (see Extract File Conversion Chart). If the original file description contains ACTUAL formats, they are ignored.
- MISSING parameter settings are converted to SQL NULL statements.

The following example is the generated file description from the ON TABLE HOLD AS PRODSQL statement in the previous TABLE request:

```
FILE=PRODSQL          ,SUFFIX=SQLDBM,$
SEGNAME=SEG01        ,SEGTYPE=S0,$
FIELDNAME   =FOCLIST    ,FOCLIST      ,I5      ,I4      ,,$
FIELDNAME   =PROD_CODE  ,PCODE        ,A3      ,A3      ,,$
FIELDNAME   =PROD_NAME  ,ITEM         ,A15     ,A15     ,,$
FIELDNAME   =PACKAGE    ,SIZE         ,A12     ,A12     ,,$
FIELDNAME   =UNIT_COST  ,COST         ,D5.2M   ,D8      ,,$
```

Generated Access Files

The access file resulting from the ON TABLE HOLD extract contains the declarations described in Segment Declarations. For PRINT and LIST based reports, the FOCLIST field and the BY phrases determine the KEYS value and how the index is created. Following is a list of the generated keyword/value pairs:

- The SEGNAME value is SEG01.
- The TABLENAME value may be up to 64 characters long (including a period), and should conform to the naming conventions for the target RDBMS. The default is HOLD. If you specified a name with the AS phrase, that name becomes the TABLENAME value. If the name contains a period (.), the characters preceding it are considered the creator name; characters following the period become the table name.
Note: If you do not specify a creator name, your userid becomes the creator by default.
- The KEYS value and the fields that are indexed depend on the verb and whether BY sort phrases are used in the request:
 - If the request specifies the NOPRINT option for a BY phrase, that sort field is not included in the table or the calculation of the KEYS value, and it is not indexed.
 - All PRINT (or LIST) requests generate a FOCLIST field. SUM (or COUNT) requests do not.
 - If the request specifies an aggregate verb such as SUM (or COUNT) with printed BY phrases, the KEYS value equals the number of printed sort fields.

- If the request specifies the verb PRINT (or LIST) with printed BY phrases, the KEYS value equals the number of printed sort fields plus one for the generated FOCLIST field.
 - If there are no BY phrases in the request, the KEYS value is 01 regardless of the verb.
 - The Interface creates a unique index on the printed BY fields plus the FOCLIST field (if it exists). If there are no BY fields, for a PRINT (or LIST) request, the Interface creates a unique index on FOCLIST alone; for a SUM (or COUNT) request, it creates a unique index on the first column referenced in the request.
- The WRITE value is YES.

For example, the generated access file from the ON TABLE HOLD AS PRODSQL statement in the previous TABLE request is:

```
SEGNAME=SEG01 ,
TABLENAME=PRODSQL ,
KEYS=01 , WRITE=YES, $
```

Other Generated Files

Three work files, FOC\$HOLD.MAS, FOC\$HOLD.FTM, and FOCSORT.FTM are used with an internal MODIFY procedure to create and load the extract table. The FOC\$HOLD file description is a fixed-format file with a corresponding sequential data file.

HOLD FORMAT SQL Usage Restrictions

The following restrictions apply to the HOLD FORMAT SQL options:

- You may not use the HOLD FORMAT SQL option in multi-verb TABLE requests; only one verb is allowed.
- You may not use the ACROSS sort phrase.
- Names used in AS phrases to rename fields should conform to column naming conventions for the target RDBMS.

Extract File Conversion Chart

The following chart shows original USAGE formats, conditions, and the resulting USAGE and ACTUAL formats for ON TABLE HOLD; the field length is represented by n:

Non-SQL USAGE	Conditions	HOLD USAGE	HOLD ACTUAL
An	none	An	An
Dn	none	Dn	D8
Fn	none	Fn	F4
In	n EQ 1 or 2 n GT 2 MISSING=ON	In In In	I2 I4 I4
Pn	MISSING=ON	Pn	D8
date	Date format	date	DATE
TXnn	TEXT field	TXnn	TX

Note:

- USAGE field types A, D, F, and TX from the original file description remain the same; their generated ACTUAL formats vary slightly.
- USAGE field types I and P from the original file description are converted based on the original length and on whether null values are permitted (MISSING=ON).

- USAGE values for FOCUS date formats remain the same; they are converted to the ACTUAL format DATE.

The Dynamic JOIN Command

With the FOCUS dynamic JOIN command, you can reference two or more related tables (or external files) in a single FOCUS report request. The data structures remain physically separate, but FOCUS treats them as a single logical structure. The terms *primary key* and *foreign key* refer to the common columns that relate host and cross-referenced tables.

Although the run-time effect of the dynamic JOIN is very similar to that of the embedded JOIN discussed in Describing Multi-Table Relational Structures, the dynamic JOIN is easier to construct since it does not require a separate file description and access file. You can create a dynamic JOIN on an as-needed basis.

Note: A FOCUS TABLE request that references a dynamic JOIN generates SQL join predicates for all segments in the subtree *above* the highest segment referenced in the request. These predicates are not generated for multi-table file descriptions; in a multi-table structure, the subtree effectively begins with the highest referenced segment. This effect may cause identical TABLE requests to produce different reports when run against a dynamic JOIN structure and a multi-table file description that represent the same tree structure.

The dynamic JOIN is limited to FOCUS read-only operations (for example, TABLE, GRAPH, and the MODIFY LOOKUP facility). Embedded JOINS in a file description support both read-only and read-write operations.

FOCUS provides two types of dynamic JOIN. The difference between the two depends on the cross-referenced relation and its foreign key values:

- The multiple or non-unique JOIN defines a one-to-many or many-to-many correlation between the records of the host file and the records of the cross-referenced file. That is, for any row in the host file, there may be more than one corresponding row in the cross-referenced file.
- The unique JOIN defines a one-to-one correlation between a record in the host file and one record in the cross-referenced file. For any row in the host file, there is, at most, one corresponding row in the cross-referenced file. When FOCUS executes a unique JOIN, it retrieves only one row from the cross-referenced file; be careful not to identify a join as unique to FOCUS if it is really non-unique.

The unique join is a FOCUS concept. The RDBMS makes no distinction between unique and non-unique situations; it always retrieves all matching rows from the cross-referenced file.

Note: If the RDBMS processes a join that the FOCUS request specifies as unique, and if there are, in fact, multiple corresponding rows in the cross-referenced file, the RDBMS inner join returns all matching rows. If, instead, optimization is disabled so that FOCUS processes the join, a different report results because FOCUS, respecting the unique join concept, returns only one cross-referenced row for each host row.

With either type of join, some rows in the host table may lack corresponding rows in the cross-referenced table. In a report request, such a retrieval path is called a *short path*.

When a report omits host rows that lack corresponding cross-referenced rows, the join is called an *inner join*. When a report displays all matching rows plus all rows from the host file that lack corresponding cross-referenced rows, the join is called a left *outer join*.

Sometimes FOCUS passes responsibility for a join to the RDBMS. The OPTIMIZATION setting is one factor in determining whether a join is optimized; other factors depend on the specific elements in the report request (see The Interface Optimizer). In order to understand join behavior, you must know whether optimization is enabled or disabled for a particular report request.

For both embedded and dynamic joins, factors that determine whether a report includes short path rows are the type of join, the FOCUS SET ALL command, and whether FOCUS or the RDBMS is handling the join. Subsequent topics describe how these factors interact.

Single-Field Dynamic JOIN

The syntax for a JOIN based on relating one column of the host and cross-referenced tables is

```
JOIN fielda1 [WITH rfield] IN hostfile [TAG tag1] TO [ALL]
fieldb1 IN crfile [TAG tag2] AS joinname
[END]
```

where:

<code>fielda1</code>	Is a field in the host file or a DEFINE field that shares values and format with fieldb1 in the cross-referenced file.
<code>WITH rfield</code>	Use only if fielda1 is a DEFINE field; associates the DEFINE field with the segment location of a real field (rfield) in the host file.
<code>hostfile</code>	Is the name of the host file. Use this name in subsequent TABLE requests on the joined structure.
<code>tag1</code>	Is the tag name for the host file, a one- to eight-character tablename qualifier for field and alias names in the host file. The tag name for the host must be the same in all the JOIN commands for a JOIN structure. The tag name may not be the same as any tablename in the structure.
<code>ALL</code>	Indicates that the host and cross-referenced files participate in a one-to-many or many-to-many relationship. That is, for any value of the join field in the host file (fielda1), there may be more than one corresponding instance of that value for the join field in the cross-referenced file (fieldb1). In FOCUS terminology, this is known as a non-unique or multiple JOIN. Note: The use of the ALL parameter does not disable optimization. Do not confuse this ALL parameter with the SET ALL command. Omitting the ALL parameter indicates a unique JOIN. Omit the ALL parameter only when each row in the host file has, at most, one corresponding row in the cross-referenced file. The unique JOIN is a FOCUS concept; the RDBMS makes no distinction between unique and non-unique situations when it processes a JOIN.
<code>fieldb1</code>	Is a field in the cross-referenced file whose values and format can match that of fielda1.
<code>crfile</code>	Is the name of the cross-referenced file.
<code>tag2</code>	Is the tag name for the cross-referenced file, a one- to eight-character tablename qualifier for field and alias names in the cross-referenced file. The tag name may not be the same as any tablename in the structure. In a recursive JOIN structure, if no tagname is provided, FOCUS prefixes all fieldnames and aliases with the first four characters of the joinname.
<code>joinname</code>	Assigns an internal name to the JOIN structure, up to eight characters in length. Joinname also provides a unique prefix for fieldnames participating in a recursive JOIN. You can clear the joinname when you no longer need this JOIN; for more information see the <i>Creating Reports</i> manual. Do not specify joinname as the filename in subsequent TABLE FILE requests; use hostfile.
<code>END</code>	Required when the JOIN statement is longer than one line; terminates the statement.

The following is an example of a single-field dynamic JOIN:

```
JOIN EID IN EMPINFO TAG FILE1 TO ALL PAYEID IN PAYINFO TAG FILE2 AS JOIN1
```

You can join up to 16 structures to create a FOCUS view of the data. The joined structure remains in effect for the duration of the FOCUS session or until you clear the joinname using the JOIN CLEAR command; for more information see the *Creating Reports* manual. The Interface instructs the RDBMS to perform a JOIN based on the smallest subtree of referenced tables (from the root) required to satisfy the request.

Note: For a JOIN to be optimized, it cannot involve more than 15 tables or views, the RDBMS limit for an SQL statement.

Each cross-referenced table must have at least one data field in common with its host file. Fixed sequential, EDA/SQL, Teradata, SQL Server, DB2/2, Oracle, and ODBC tables and views can be both host files and cross-referenced files in any combination. You can also join these files to all segments of a FOCUS database by using FOCUS database indexed fields.

If the tables are located on two different servers or subsystems:

- You can use the dynamic JOIN command to join tables from two different locations. FOCUS processes the join since the RDBMS does not allow a single SQL statement to reference tables at more than one location.
- When it detects the implicit or explicit presence of multiple LOCATION or SERVER attributes in the access file for the tables referenced in the report request, the Interface temporarily disables optimization so that FOCUS can manage the join. If you are using local RDBMS aliases for remote tables, you must SET OPTIMIZATION OFF for each request that joins tables from more than one location.

The following chart lists some of the more common combinations available with the FOCUS dynamic JOIN command and restrictions on their use. Consult the *Creating Reports* manual for additional file combinations and examples of dynamic JOINS. In the chart, SQL refers to EDA/SQL, DB2/2, SQL Server, ODBC, Teradata, or Oracle tables and views:

From	To	Special Rules That May Apply
SQL	SQL	Joined fields must be of the same datatype. For efficient retrieval, their lengths should also be equivalent. Use of indexed fields for the host and cross-referenced fields is recommended, but the RDBMS uses the indexes only if both the datatypes and lengths are equal.
SQL	FOCUS	Must join to an indexed field (FIELDTYPE=I). Joined fields must have common datatype and length.
SQL	Btrieve	For a unique join, the Btrieve join field must be a full primary key. For a non-unique join, the join field can be an initial subset of the primary key. Joined fields must have a common datatype and length.
SQL	sequential	A sequential file must be sorted by its join field. Joined fields must have a common datatype and length.

Note:

- One common example of a sequential file is a FOCUS HOLD file.
- When joining to a Btrieve or sequential data structure, the field in the host file can be shorter in length than the field in the cross-referenced file, but performance is adversely affected. This is the only exception to the "same datatype, same length" rule when joining non-relational files.
- For efficiency, create RDBMS indexes on host and cross-referenced fields. Check with the database administrator or query the index catalog table to see which columns are indexed.
- Fields with text field type (TX) may not participate as host or cross-referenced fields.
- In report requests, specify the name of the host structure in the TABLE FILE statement, not the joinname specified with the AS phrase.
- You can construct a dynamic JOIN based on more than one field. See Multi-Field Dynamic JOIN for multi-field JOINS.

Dynamic JOIN Examples

The following examples illustrate the use of the FOCUS dynamic JOIN command. Each example specifies a FOCUS non-unique join and presents an equivalent SQL request for comparison purposes. Both forms of the request, FOCUS and SQL, return the same result.

The first example (using the DB2/2 Interface) prints each employee's full name and courses taken. Since the employee may have taken more than one course, the example specifies the FOCUS non-unique join.

Employee information is stored in the EMPINFO table and is represented by the fields LAST_NAME and FIRST_NAME. Course information is stored in the COURSE table and is represented by the COURSE_NAME field.

The following

```
FILEDEF FSTRACE4 TERM
JOIN EMP_ID IN EMPINFO TAG FILE1 TO ALL EMP_ID IN COURSE TAG FILE2 AS J1
TABLE FILE EMPINFO
PRINT COURSE_NAME
BY LAST_NAME
BY FIRST_NAME
END
```

produces the following trace:

```
SELECT T1.EMP_ID,T1.LAST_NAME,T1.FIRST_NAME,T2.COURSE_NAME
FROM USERID.EMPINFO T1,USERID.COURSE T2 WHERE (T2.EMP_ID =
T1.EMP_ID) ORDER BY T1.LAST_NAME,T1.FIRST_NAME FOR FETCH ONLY;
```

The SQL request references both tables in the FROM clause and as a condition (or JOIN predicate) of the WHERE clause. The FOCUS BY phrases translate to the ORDER BY phrase. The PRINT verb translates as SELECT.

The next example (using the DB2/2 Interface) illustrates a screening test that lists the employees who have taken either the Advanced or Internals course.

The following

```
FILEDEF FSTRACE4 TERM
JOIN EMP_ID IN EMPINFO TAG FILE1 TO ALL EMP_ID IN COURSE TAG FILE2 AS J1
TABLE FILE EMPINFO
PRINT COURSE_NAME
BY LAST_NAME
BY FIRST_NAME
WHERE COURSE_NAME EQ 'ADVANCED' OR 'INTERNALS';
END
```

would generate the following trace:

```
SELECT T1.EMP_ID,T1.LAST_NAME,T1.FIRST_NAME,T2.COURSE_NAME
FROM USERID.EMPINFO T1,USERID.COURSE T2 WHERE (T2.EMP_ID =
T1.EMP_ID) AND (T2.COURSE_NAME IN('ADVANCED','INTERNALS'))
ORDER BY T1.LAST_NAME,T1.FIRST_NAME FOR FETCH ONLY;
```

The SQL request references both tables in the FROM clause and in a JOIN predicate in the WHERE clause (along with additional screening conditions). The FOCUS BY phrases translate to the ORDER BY phrase.

Multi-Field Dynamic JOIN

You can construct a dynamic JOIN based on multiple fields from both the host file and the cross-referenced file. Separate the participating fieldnames with the keyword AND.

The syntax for the multi-field JOIN is

```
JOIN fielda1 AND fielda2 IN hostfile [TAG tag1] TO [ALL]
fieldb1 AND fieldb2 IN crfile [TAG tag2] AS joinname
[END]
```

where:

<code>fielda1</code>	Is a field in the host file that shares values and format with fieldb1 in the cross-referenced file.
<code>AND fielda2</code>	Is a field in the host file that shares values and format with fieldb2 in the cross-referenced file.

<code>hostfile</code>	Is the name of the host file. Use this name in subsequent TABLE requests on the joined structure. The host file can be any type of database or table. See Single-Field Dynamic JOIN for possible JOIN combinations.
<code>tag1</code>	Is the tag name for the host file, a one- to eight-character tablename qualifier for field and alias names in the host file. The tag name for the host must be the same in all the JOIN commands for a JOIN structure. The tag name may not be the same as any tablename in the structure.
<code>ALL</code>	Indicates that the host and cross-referenced files participate in a one-to-many or many-to-many relationship. That is, for any instance of the JOIN fields in the host file (fielda1, fielda2), there may be more than one corresponding instance of the JOIN fields in the cross-referenced file (fieldb1, fieldb2). In FOCUS terminology, this is known as a non-unique or multiple JOIN. Note: The use of the ALL parameter does not disable optimization. Do not confuse this ALL parameter with the SET ALL command. Omitting the ALL parameter indicates a unique JOIN. Omit the ALL parameter only when each row in the host file has, at most, one corresponding row in the cross-referenced file. The unique JOIN is a FOCUS concept; the RDBMS makes no distinction between unique and non-unique situations when it processes a JOIN.
<code>fieldb1</code>	Is a field in the cross-referenced file whose values and format can match that of fielda1.
<code>AND fieldb2</code>	Is a field in the cross-referenced file whose values and format can match that of fielda2.
<code>crfile</code>	Is the name of the cross-referenced file. The cross-referenced file may be an EDA/SQL, Teradata, DB2/2, SQL Server, or Oracle table, or a Btrieve file.
<code>tag2</code>	Is the tag name for the cross-referenced file, a one- to eight-character tablename qualifier for field and alias names in the cross-referenced file. The tag name may not be the same as any tablename in the structure. In a recursive JOIN structure, if no tagname is provided, FOCUS prefixes all fieldnames and aliases with the first four characters of the joinname.
<code>joinname</code>	Assigns an internal name to the JOIN structure, up to eight characters in length. It also provides a unique prefix for fieldnames participating in a recursive JOIN. You can clear the joinname when you no longer need this JOIN (see the <i>Creating Reports</i> manual). Do not specify joinname as the filename in subsequent TABLE FILE requests; use hostfile.
<code>END</code>	Required when the JOIN statement is longer than one line; terminates the statement.

For a multi-field JOIN, the joined fields from each table must be equal in number, datatype, and field length. The full set of common fields must reside in both the host and cross-referenced tables.

You can join a maximum of 16 fields from a host file to 16 fields in a cross-referenced file. Each multi-field JOIN command counts as one JOIN toward the FOCUS maximum of 16 concurrent JOINS.

Note: For a JOIN to be optimized, it cannot involve more than 15 tables or views, the RDBMS limit for an SQL statement.

For example, two RDBMS tables, EMPINFO and COURSE1, have first and last name fields. In the EMPINFO file description, LAST_NAME and FIRST_NAME have field formats A15 and A10:

```
FILE=EMPINFO , SUFFIX=SQLDBM,$
  SEGNAME=EMPINFO , SEGTYPE=S0,$
    FIELD=EMP_ID ,ALIAS=EMP_ID ,USAGE=A9 ,ACTUAL=A9 ,MISSING=OFF,$
    FIELD=LAST_NAME ,ALIAS=LAST_NAME ,USAGE=A15 ,ACTUAL=A15 ,MISSING=OFF,$
    FIELD=FIRST_NAME ,ALIAS=FIRST_NAME ,USAGE=A10 ,ACTUAL=A10 ,MISSING=ON ,,$
    FIELD=HIRE_DATE ,ALIAS=HIRE_DATE ,USAGE=MDY ,ACTUAL=DATE ,MISSING=ON ,,$
    FIELD=DEPARTMENT_CODE ,ALIAS=DEPARTMENT_CODE ,USAGE=A10 ,ACTUAL=A10 ,MISSING=ON
,$
  FIELD=CURRENT_SALARY ,ALIAS=CURRENT_SALARY ,USAGE=D9.2 ,ACTUAL=D8 ,MISSING=ON
,$
  FIELD=ED_HRS ,ALIAS=ED_HRS ,USAGE=D6.2 ,ACTUAL=D8 ,MISSING=ON ,,$
  FIELD=JOBCODE ,ALIAS=JOBCODE ,USAGE=A3 ,ACTUAL=A3 ,MISSING=ON ,,$
```

In the COURSE1 file description, LAST_NAME and FIRST_NAME have the same field formats, A15 and A10:

```
FILE=COURSE , SUFFIX=SQLDBM, $
SEGNAME=COURSE , SEGTYPE=S0, $
FIELD=COURSE_NAME , ALIAS=COURSE_NAME , USAGE=A30 , ACTUAL=A30 , MISSING=OFF, $
FIELD=EMP_ID , ALIAS=EMP_ID , USAGE=A9 , ACTUAL=A9 , MISSING=OFF, $
FIELD=GRADE , ALIAS=GRADE , USAGE=A1 , ACTUAL=A1 , MISSING=OFF, $
FIELD=QUARTER , ALIAS=QUARTER , USAGE=A1 , ACTUAL=A1 , MISSING=OFF, $
FIELD=YR_TAKEN , ALIAS=YR_TAKEN , USAGE=A2 , ACTUAL=A2 , MISSING=OFF, $
```

To create the multi-field JOIN, list both fields for each table with the keyword AND:

```
JOIN LN AND FN IN EMPINFO TAG FILE1
TO ALL LAST_NAME AND FIRST_NAME IN COURSE1 TAG FILE2 AS J1
END
```

When a report request references the multi-field dynamic JOIN, the Interface generates an SQL SELECT statement to satisfy the request. For example, the following request using the DB2/2 Interface

```
JOIN LAST_NAME AND FIRST_NAME IN EMPINFO TAG FILE1
TO ALL LAST_NAME AND FIRST_NAME IN COURSE1 TAG FILE2 AS J1
END
```

```
TABLE FILE EMPINFO
PRINT LAST_NAME FIRST_NAME COURSE_NAME
END
```

would produce the following test:

```
SELECT T1.LAST_NAME, T1.FIRST_NAME, T2.COURSE_NAME FROM
USERID.EMPINFO T1, COURSE1 T2 WHERE (T2.LAST_NAME =
T1.LAST_NAME) AND (T2.FIRST_NAME = T1.FIRST_NAME) FOR FETCH
ONLY;
```

The SQL SELECT statement implements two joins as conditions (or predicates) of the WHERE clause.

Multi-Field Joins Summary Chart

In some cases, you may want to join more than one host field to a single cross-referenced field. The procedure, as well as the relationship, differs for joins between two relational database files and joins between relational and FOCUS database files.

There are two types of multi-field joins:

- Multi-field join; when you join multiple host fields to multiple cross-referenced fields.
- Concatenated join; the field resulting from the concatenation of two host fields joined to a single cross-referenced field.

The following table lists the types of joins FOCUS allows between files:

Host	Cross-referenced	Supported Multi-field Join Constructions
SQL*	SQL	FOCUS allows you to join two SQL files with a multi-field construction. You cannot directly join two SQL files with a concatenated join. You can, however, use a Define-based join to achieve the same result that a concatenated join would produce between two SQL files. For more information on using a Define-based join, see Using a Virtual Field as the Join Field under Joining Databases in online Help.
SQL	FOCUS	You can use only the concatenated join construction to join two SQL host fields to a single FOCUS cross-referenced field.
FOCUS	SQL	FOCUS allows you to join a FOCUS file and an SQL file with multi-field join construction. You cannot join a FOCUS file and an SQL file with a concatenated join. You can, however, use a Define-based join to achieve the same result that a concatenated join would produce between these files. For more information on using a Define-based join, see Using a Virtual Field as the Join Field under Joining Databases in online Help.

* SQL represents all relational databases.

Missing Rows of Data in Cross-Referenced Tables

This topic describes factors that affect report results when a host row has no corresponding cross-referenced row. The discussion applies to both dynamic and embedded joins.

Normally, when a row from the host table or view is retrieved, a corresponding row from the cross-referenced table can also be retrieved. If a host row lacks a corresponding cross-referenced row, the retrieval path is called a *short path*. When there are short paths, the processing of the host row and the report results depend on:

- The FOCUS SET ALL command (or the use of the ALL. prefix).
- Whether Interface optimization is enabled or disabled. (See The Interface Optimizer for information about the SET OPTIMIZATION command.)
- The type of JOIN: non-unique or unique.

The SET ALL Command

You can include short paths in a report with the FOCUS SET ALL command. The syntax is

```
SET ALL = { OFF }
          { ON }
          { PASS }
```

where:

- OFF Is the default. In a JOIN, omits host rows from the report if they lack a corresponding cross-referenced row.
- ON Includes all host rows in the report. **Note:** See [SET ALL=ON and Join Optimization](#) for a list of conditions under which this setting passes a left outer join to the RDBMS.
- PASS Includes all host rows in the report, even if their corresponding cross-referenced rows are eliminated by a WHERE or IF record selection test. **Note:** This setting is supported only for the Sybase SQL Server and Sybase System 10 interfaces.

SET ALL=ON and Join Optimization

Under the following conditions, SET ALL=ON causes the interface to pass a left outer join to the RDBMS:

- The target RDBMS is Oracle.
- The target RDBMS is Sybase SQL Server or Sybase System 10, and your request does not include screening conditions on cross-referenced segments.

In all other cases, SET ALL=ON temporarily disables optimization to allow FOCUS to handle the join.

For the Microsoft, Sybase SQL Server, and Sybase System 10 interfaces, SET ALL=PASS passes a left outer join to the RDBMS.

See Missing Rows in Unique Descendants, and Missing Rows in Non-Unique Descendants, for a complete discussion of how the optimization setting and the SET ALL command affect join processing. For your convenience, Summary Chart provides a summary of this discussion.

Missing Rows in Unique Descendants

In a unique JOIN, the engine that handles the JOIN controls the output. Report results depend on whether OPTIMIZATION is enabled or disabled because:

- If OPTIMIZATION is disabled, FOCUS handles the JOIN. FOCUS treats a unique cross-referenced table as a logical extension of the host or parent table. Since FOCUS never considers a unique cross-referenced row to be missing, it displays short paths regardless of whether SET ALL is ON or OFF.

Note: With optimization disabled, if you describe a JOIN as unique when the cross-referenced table actually contains more than one matching record for a row in the host table, FOCUS displays only the first matching cross-referenced row on the report. Do not specify a unique JOIN to FOCUS when the data structure is non-unique.

- If optimization is enabled, the RDBMS handles the JOIN:
 - If SET ALL = OFF, the RDBMS performs an inner join; it omits short paths from the report and includes multiple matching cross-referenced rows.
 - If SET ALL = ON, the RDBMS performs a left outer join; it includes all host rows in the report and includes multiple matching cross-referenced rows. **Note:** If your request does not meet the conditions listed in SET ALL=ON and Join Optimization, issuing the SET ALL=ON command temporarily disables optimization to allow FOCUS to handle the join.

Since the RDBMS has no concept of a unique join, its behavior is identical whether the join is unique or non-unique. See Missing Rows in Non-Unique Descendants for a complete discussion.

OPTIMIZATION Disabled With a Unique JOIN

With optimization disabled, FOCUS handles the join and substitutes default values for any missing cross-referenced data (because it does not consider them to be missing in a unique join). FOCUS recognizes that you have defined the join to be unique and, regardless of the ALL setting:

- It displays short paths with blanks or zeros in missing columns.
- It includes only the first instance found of any multiple matching cross-referenced rows.

In the following example (using the DB2/2 Interface), a unique JOIN connects the EMPINFO table, containing employee information, to the FUNDTRAN table, containing direct deposit account information for the employees. OPTIMIZATION is OFF.

```
JOIN EMP_ID IN EMPINFO TAG FILE1 TO EMP_ID IN FUNDTRAN TAG FILE2 AS J1
```

```
SQL SQLDBM SET OPTIMIZATION OFF
```

```
TABLE FILE EMPINFO  
PRINT BANK_NAME BANK_ACCT  
BY DEPARTMENT_CODE  
BY EMP_ID  
END
```

Since the JOIN is unique and optimization is disabled, FOCUS displays one row per employee on the report. If there is no data for the cross-referenced table (FUNDTRAN), FOCUS appropriately substitutes zeros or blanks for its fields.

Two new employees, John Royce (333121200, no department) and Donna Lee (455670000, MIS) have no bank accounts. Six other employees also lack bank accounts:

DEPARTMENT_CODE	EMP_ID	BANK_NAME	BANK_ACCT
.	333121200		0
MIS	112847612		0
	117593129	STATE	40950036
	219984371		0
	326179357	ASSOCIATED	122850108
	455670000		0
	543729165		0
	818692173	BANK ASSOCIATION	163800144
PRODUCTION	071382660		0
	119265415		0
	119329144	BEST BANK	160633
	123764317	ASSOCIATED	819000702
	126724188		0
	451123478	ASSOCIATED	136500120

Note: Employee 333121200 has not yet been assigned a department; its null value is indicated by the NODATA symbol. However, BANK_NAME and BANK_ACCT are not considered missing because the JOIN is unique.

OPTIMIZATION Enabled With a Unique JOIN

With optimization enabled, the RDBMS has responsibility for the join. Since the RDBMS does not recognize the concept of a unique join, it performs an inner join if SET ALL=OFF and a left outer join if SET ALL=ON, just as it does if the join is non-unique. See Missing Rows in Non-Unique Descendants for a complete discussion. **Note:** If your request does not meet the conditions listed in SET ALL=ON and Join Optimization, issuing the SET ALL=ON command temporarily disables optimization to allow FOCUS to handle the join. In this case, see OPTIMIZATION Disabled With a Unique JOIN.

The Interface Optimizer discusses factors that determine whether optimization is enabled.

Missing Rows in Non-Unique Descendants

In a non-unique join (JOIN...TO ALL) with missing cross-referenced rows, report results depend entirely on the SET ALL command because:

- If SET ALL=OFF, the RDBMS and FOCUS both perform an inner join. Therefore, the OPTIMIZATION setting has no effect on the report results. Short paths are omitted from the report; multiple matching rows are included.
- If SET ALL=ON, the RDBMS and FOCUS both perform a left outer join. Therefore, the OPTIMIZATION setting has no effect on the report results. Short paths display in the report with missing columns represented by the NODATA symbol (.); multiple matching rows are included. **Note:** If your request does not meet the conditions listed in SET ALL=ON and Join Optimization, issuing the SET ALL=ON command temporarily disables optimization to allow FOCUS to handle the join; the result, in either case, is a left outer join.

SET ALL=OFF With a Non-Unique JOIN

With SET ALL=OFF, optimization may be enabled or disabled. In both cases, however, the report results are the same: an inner join. Host rows that lack corresponding cross-referenced rows are not included in the report; multiple matching rows are included (even if there is only one).

For each of the following examples, the same non-unique JOIN is in effect. It connects the EMPINFO table, containing employee information, to the DEDUCT table, containing salary deduction information.

The examples also execute the same report request. OPTIMIZATION is set ON and only the SET ALL command changes.

Two employees, John Royce (333121200, no department) and Donna Lee (455670000, MIS) have not been paid yet; therefore, they have no deductions.

When SET ALL is OFF, a host row that lacks a corresponding cross-referenced row is rejected (regardless of whether FOCUS or the RDBMS processes the JOIN):

```

join clear j1
>
join emp_id in empinfo tag file1 to all dedeid in deduct tag file2 as j2
>
set all=off
>
table file empinfo
print ded_code ded_amt
by department by emp_id by deddate
end
NUMBER OF RECORDS IN TABLE=          448 LINES=          448

```

The report displays multiple deduction records in the cross-referenced table for each of eight long-time employees in the host table. Employees who do not have corresponding cross-referenced rows in the DEDUCT table are not listed in the report. For example, John Royce (333121200, no department) and Donna Lee (455670000, MIS department).

PAGE 1				
DEPARTMENT_CODE	EMP_ID	PD_DATE	DED_CODE	DED_AMT
-----	-----	-----	-----	-----
MIS	112847612	07/30/82	HLTH	22.75
			LIFE	13.65
	117593129	07/30/82	HLTH	30.80
			LIFE	18.48
	326179357	07/30/82	HLTH	45.37
			LIFE	27.22
818692173	07/30/82	HLTH	33.82	
		LIFE	20.29	
PRODUCTION	071382660	07/30/82	HLTH	8.68
			LIFE	5.21
	123764317	07/30/82	HLTH	55.96
			LIFE	33.58
	126724188	07/30/82	HLTH	26.40
			LIFE	15.84
	451123478	07/30/82	HLTH	16.50
			LIFE	9.90

SET ALL=ON With a Non-Unique JOIN

With a non-unique join, SET ALL=ON produces a left outer join regardless of whether FOCUS or the RDBMS handles the join:

- If optimization is enabled, the Interface passes a left outer join to the RDBMS in the cases of Sybase SQL Server, Sybase System 10 and System 11, and Oracle.
- If optimization is disabled, FOCUS handles the join. Honoring the SET ALL command, FOCUS includes all host rows; since the join is non-unique, it also includes multiple matching rows. **Note:** If your request does not meet the conditions listed in SET ALL=ON and Join Optimization, issuing the SET ALL=ON command temporarily disables optimization to allow FOCUS to handle the join; the result, in either case, is a left outer join.

If there are no cross-referenced rows for a host row, the cross-referenced columns display the NODATA value.

Note:

- SET ALL=PASS is supported only for the Microsoft/Sybase SQL Server, and Sybase System 10 and System 11 interfaces. Like SET ALL=ON, it passes a left outer join to the RDBMS. For a discussion of the differences between the two settings, see SET ALL=ON With Screening Conditions.
- The report that results from passing a request to the RDBMS with SET ALL=ON may be sorted in a slightly different order from the report produced if FOCUS processes the join; however, both results are equally correct.

The following example executes the report request illustrated in SET ALL=OFF With a Non-Unique Join, except that SET ALL is ON:

```
JOIN EMP_ID IN EMPINFO TAG FILE1 TO ALL EMP_ID IN DEDUCT TAG FILE2 AS J1

SET ALL=ON

TABLE FILE EMPINFO
PRINT DED_CODE DED_AMT
BY DEPARTMENT_CODE BY EMP_ID BY PD_DATE
END
```

John Royce (333121200) and Donna Lee (455670000) are now included in the report. The department column for John Royce (333121200) displays the NODATA value, since the field has MISSING=ON:

DEPARTMENT_CODE	EMP_ID	PD_DATE	DED_CODE	DED_AMT
.	333121200	.	.	.
MIS	112847612	07/30/82	HLTH	22.75
			LIFE	13.65
	117593129	07/30/82	HLTH	30.80
			LIFE	18.48
	219984371	.	.	.
	326179357	07/30/82	HLTH	45.37
			LIFE	27.22
	455670000	.	.	.
	543729165	.	.	.
	818692173	07/30/82	HLTH	33.82
			LIFE	20.29
PRODUCTION	071382660	07/30/82	HLTH	8.68
			LIFE	5.21
	119265415	.	.	.
	119329144	.	.	.
	123764317	07/30/82	HLTH	55.96
			LIFE	33.58
	126724188	07/30/82	HLTH	26.40
			LIFE	15.84
	451123478	07/30/82	HLTH	16.50
			LIFE	9.90

SET ALL=ON With Screening Conditions

When SET ALL=ON, screening conditions affect report results. Except for the Sybase SQL Server, and Sybase System 10 and System 11 interfaces, if a screening test specifies a field from the cross-referenced structure, host rows whose cross-referenced rows were screened out are not represented, regardless of the SET ALL command.

Note: When they process a join, Sybase SQL Server, and Sybase System 10 and System 11 ignore selection criteria on cross-referenced rows. They always return a row for missing cross-referenced columns. This behavior is consistent with the FOCUS command SET ALL=PASS. Therefore, in the Microsoft/Sybase SQL Server, and Sybase System 10 and System 11 interfaces, when there are selection tests on the cross-referenced table, SET ALL=ON does not pass a left outer join to the RDBMS; instead, FOCUS processes the left outer join and eliminates screened-out records from the report. You must specify SET ALL=PASS if you want to pass the join to the RDBMS.

The following example includes a WHERE test to screen for health deduction records:

```
JOIN EMP_ID IN EMPINFO TAG FILE1 TO ALL EMP_ID IN DEDUCT TAG FILE2 AS J1

SET ALL=ON

TABLE FILE EMPINFO
PRINT DED_CODE DED_AMT
BY DEPARTMENT_CODE BY LAST_NAME BY FIRST_NAME
WHERE DED_CODE EQ 'HLTH'
END
```

Employees that have no deduction records are omitted from the report; for example, Royce and Lee:

PAGE	1			
DEPARTMENT_CODE	LAST_NAME	FIRST_NAME	DED_CODE	DED_AMT
-----	-----	-----	-----	-----
MIS	BLACKWOOD	ROSEMARIE	HLTH	45.37
	CROSS	BARBARA	HLTH	33.82
	JONES	DIANE	HLTH	30.80
	SMITH	MARY	HLTH	22.75
PRODUCTION	IRVING	JOAN	HLTH	55.96
	MCKNIGHT	ROGER	HLTH	16.50
	ROMANS	ANTHONY	HLTH	26.40
	STEVENS	ALFRED	HLTH	8.68

Selective ALL. Prefix

Even if SET ALL=OFF, you can apply the effect of the ON setting to specific tables. To do this, add the ALL. prefix to one of the *host* fields in the request. The ALL. prefix causes FOCUS to process host rows even if they have missing cross-referenced rows. Like SET ALL=ON, the report results depend on whether screening tests exist for the cross-referenced fields.

Note: If the request includes screening tests on cross-referenced fields, the Sybase SQL Server and Sybase System 10 interfaces will not pass the join to the RDBMS. Instead, FOCUS will perform the left outer join.

For instance, when the ALL. prefix is applied to the field DEPARTMENT, the report results are the same as for SET ALL=ON. (The report is displayed in SET ALL=ON With a Non-Unique JOIN).

```
JOIN EMP_ID IN EMPINFO TAG FILE1 TO ALL EMP_ID IN DEDUCT TAG FILE2 AS J1

SET ALL=OFF

TABLE FILE EMPINFO
PRINT DED_CODE DED_AMT
BY ALL.DEPARTMENT_CODE BY EMP_ID BY PD_DATE
END
```

Note: The ALL. prefix is only effective when the SET ALL value is OFF; SET ALL=ON overrides the ALL. prefix.

Dynamic JOIN Summary Chart

The following summary chart lists possible report results for dynamic JOINS and multi-table file descriptions (embedded JOINS).

JOIN Type	Optimization Enabled (RDBMS Behavior)		Optimization Disabled (FOCUS Behavior)	
	SET ALL=ON	SET ALL=OFF	SET ALL=ON	SET ALL=OFF
NON-UNIQUE Dynamic (JOIN...TO ALL) or Embedded (SEGTYPE=S0 or KL) Short paths More than one matching row	Outer join Appear with NODATA (.) value for missing values Short path rows appear in addition to all matching rows	Inner join Do not appear All matching rows appear	Outer join Appear with NODATA (.) value for missing values Short path rows appear in addition to all matching rows	Inner join Do not appear All matching rows appear
UNIQUE Dynamic (JOIN...TO) or Embedded (SEGTYPE=U or KLU) Short paths More than one matching row	Outer join Appear with NODATA (.) value for missing values Short path rows appear in addition to all matching rows	Inner join Do not appear All matching rows appear	Appear with blank or zero for missing values Only first matching row appears	Appear with blank or zero for missing values Only first matching row appears

Note:

- For non-unique dynamic and embedded JOINS, use SET ALL=ON to display the short paths.
- For unique dynamic and embedded JOINS, use either OPTIMIZATION OFF or SET ALL=ON to display the short paths.
- For unique dynamic and embedded JOINS, when OPTIMIZATION is OFF, FOCUS produces an outer join unless there is more than one matching row. If there is more than one matching row, only the first matching row appears.

Search Limits

The FOCUS READLIMIT and RECORDLIMIT features set a maximum for the number of rows the Interface fetches from the answer set returned by the RDBMS. They do not restrict the RDBMS in its construction of the answer set. However, if the RDBMS does not have to sort the answer set, using these features may result in a significant reduction in response time.

The READLIMIT and RECORDLIMIT features are effective in reducing RDBMS-to-FOCUS communication traffic, the volume of formatted report data, and terminal and disk I/Os.

Search limit tests are helpful when:

- Testing a new file description. Reporting requires only a few rows to indicate if the description is accurate.
- Testing the format of a new report.

In all three cases, a few rows are sufficient to verify results.

To restrict the maximum number of SQL FETCH commands performed for RDBMS-returned data, add the following phrase to the TABLE request

```
IF READLIMIT IS n
```

where:

`n` Is the number of records to be included in the TABLE request.

For DB2/2, READLIMIT appends an OPTIMIZE FOR n ROWS clause to the SQL generated by the Interface. This can improve DB2/2 performance if you know the size of the desired answer set in advance. For more information, consult your IBM DB/2 documentation.

For Oracle, READLIMIT appends a WHERE ROWNUM <=1 clause to the SQL generated by the Interface. This can improve Oracle performance if you know the size of the desired answer set. For more information, consult your Oracle documentation.

To restrict the maximum number of SQL FETCH commands performed for RDBMS-returned data without appending the OPTIMIZE FOR n ROWS clause or WHERE ROWNUM <=1 clause, use the following phrase in the TABLE request instead

```
IF RECORDLIMIT IS n
```

where:

`n` Is the number of records to be included in the TABLE request.

The FOCUS database administrator may place the READLIMIT or RECORDLIMIT test in a file description to limit the fetching of rows for all users. If the file description and the report request both contain such a test, the Interface uses the lower of the two values.

Note: In some instances, the RDBMS performs a substantial volume of work before returning any data to FOCUS. To limit the amount of work performed by the RDBMS, add another IF or WHERE test to one or more of the tables in the request.

Using READLIMIT With JOIN

The Interface knows how many RDBMS tables will be searched and the number of reads permitted. It calculates how many rows will be in the report and only requests sufficient data for the report.

Multiple Retrieval Paths

This topic discusses retrieval performance for multi-table file descriptions and dynamic JOINS of three or more tables. It also explains the role of unique cross-referenced tables.

File descriptions and JOIN commands define data retrieval paths. The Interface queries only those RDBMS tables necessary for the report. These include tables containing fields specified in the request plus any connecting tables needed to construct a retrieval plan (subtree). The retrieval sequence of a subtree is top to bottom, left to right.

Note: A FOCUS TABLE request that references a dynamic JOIN generates SQL join predicates for all segments in the subtree *above* the highest segment referenced in the request. These predicates are not generated for multi-table file descriptions; in a multi-table structure the subtree effectively begins with the highest referenced segment. This effect may cause identical TABLE requests to produce different reports when run against a dynamic JOIN structure and a multi-table file description that represent the same tree structure.

FOCUS treats unique tables as extensions of their hosts. In cases where the host has both unique and non-unique cross-referenced tables, FOCUS always retrieves the unique cross-referenced tables first.

To display retrieval paths, use the CHECK FILE command with the RETRIEVE option. (See *Debugging Techniques* in the online *Language Reference*.)

Multiple Retrieval Paths With Sort Phrases and Screening Tests

Fields specified for sort phrases (BY and ACROSS) and screening tests (IF and WHERE) must lie on the same retrieval path as the other requested fields. That is, the table containing the BY field or column being screened must precede or follow other tables referenced in the request. A field that is not on the same path generates FOCUS error message FOC029.

Direct SQL Passthru

Topics:

- Invoking Direct SQL Passthru.
- Issuing commands and requests.
- Controlling SQL Passthru transactions.
- Parameterized SQL Passthru.

With the Direct SQL Passthru facility, you can issue any native SQL command directly to the RDBMS from FOCUS. This facility, included with the Interface, provides Interface support for both native SQL and Interface environmental commands. You must know native SQL to use this feature for issuing SQL statements, but not for the Interface SET commands.

Issuing requests through the Direct SQL Passthru facility eliminates the need for FOCUS file descriptions and access files, but retains all FOCUS reporting capabilities.

Note: Direct SQL Passthru is not available within MODIFY.

Direct SQL Passthru provides the following advantages:

- Support for native SQL commands, including SQL SELECT statements.

With Direct SQL Passthru, FOCUS provides a storage area for the RDBMS-supplied data (answer sets) that result from SELECT statements. Therefore, you can issue SELECT statements.

- Support for all SQL SELECT options.

You can issue any SELECT syntax supported by the RDBMS, regardless of whether an equivalent FOCUS operation exists. Neither the Interface nor the Direct SQL Passthru facility translates SQL to FOCUS in order to process a request.

For example, no FOCUS syntax exists that would cause the Interface to generate one SELECT with a UNION or with a subquery. The Direct SQL Passthru facility supports both operations.

- Support for parameter markers in Direct SQL Passthru commands, so you can execute them repeatedly with varying input values.
- An alternative to the FOCUS SQL Translator.

Some applications use the FOCUS SQL Translator to access the RDBMS via a FOCUS Interface. The Translator translates SQL to FOCUS, and then the Interface uses this FOCUS code to generate SQL that it passes to the RDBMS. The Direct SQL Passthru facility bypasses internal translation and offers a more direct means of accessing the RDBMS.

The Translator supports one SQL dialect, ANSI standard Level 2 SQL. You can use the Translator to produce reports from any data source that is described to FOCUS (for instance, a FOCUS database, or the DB2/2 RDBMS). For more information about the FOCUS SQL Translator and access to FOCUS databases, see online help and the online *Language Reference*.

Invoking Direct SQL Passthru

To invoke the Direct SQL Passthru facility, you must establish the target RDBMS in either of the following two ways:

- Issue the Interface SET SQLENGINE command to establish the target RDBMS for the duration of the FOCUS session or until you issue another SET SQLENGINE command. The Interface automatically passes subsequent SQL commands to the specified RDBMS.
- Specify the target RDBMS in your request. (See Issuing Commands and Requests for information about issuing commands and requests.)

The syntax for the SET SQLENGINE command is

```
SET SQLENGINE = { OFF  
                | SQLSYB |  
                | SQLORA |  
                | SQLDBM }  
                | SQLDBC |  
                | SQLODBC |  
                | SQLINF |  
                | EDA   }
```

where:

OFF	Indicates that the FOCUS SQL Translator will process the request. OFF is the default setting.
SQLSYB	Sybase SQL Server.
SQLORA	Oracle.
SQLDBM	IBM DB2/2.
SQLDBC	Teradata DBC/1012.
SQLODBC	Microsoft ODBC.
SQLINF	Informix.
EDA	Enterprise Data Access/SQL.

You can change the SET SQLENGINE setting at any point in your FOCUS session.

Issuing Commands and Requests

If you do not issue the SET SQLENGINE command, you must specify the target RDBMS in any SQL command you want to pass directly to the RDBMS.

The following is a request syntax summary for native SQL commands, including SELECT statements, and for Interface environmental commands; subsequent topics provide examples

```
SQL [target\_db]  
command [;]  
[TABLE FILE SQLOUT]  
[options]  
END
```

where:

[target_db](#) Is the target RDBMS. Valid values are:

SQLSYB	Sybase SQL Server.
SQLORA	Oracle.
SQLDBM	IBM DB2/2.
SQLDBC	Teradata DBC/1012.
SQLODBC	Microsoft ODBC.
SQLINF	Informix.
EDA	Enterprise Data Access/SQL.

Note: Omit if you previously issued the SET SQLENGINE command.

[command](#) Is one SQL command, or one or more Interface SET commands.

[;](#) For SQL SELECT requests only, the semicolon is required if you specify additional FOCUS report options.

[TABLE FILE](#)

<code>SQLOUT</code>	For SQL SELECT requests only, allows you to specify additional report options or subcommands. To create a file description you can use throughout the FOCUS session, see <i>Creating FOCUS Views With Direct SQL Passthru</i> .
<code>options</code>	For SQL SELECT requests only, are report formatting options or operations.
<code>END</code>	Terminates the request. Is optional for Interface SET commands, the SQL commands COMMIT WORK and ROLLBACK WORK, and the Passthru commands BEGIN SESSION, END SESSION, and PURGE (see <i>Parameterized SQL Command Summary</i>). Required for all other commands.

Including a target RDBMS in your command overrides the SET SQLENGINE command. For example, you can specify your RDBMS and override an existing OFF setting. If you do not specify a target RDBMS (either in your command or with the SET SQLENGINE command), the SQL keyword invokes the Translator.

Note:

- The OFF setting is valid only for the SET SQLENGINE command.
- You cannot issue an SQL command together with Interface SET commands in one request.

Example: Issuing Interface Environmental Commands

You can issue one or more Interface SET commands in a request using Direct SQL Passthru. Interface SET commands are not passed to the RDBMS; the Interface maintains them in memory.

The following procedure issues four Interface SET commands, and issues the SQL ? query command to display the updated parameters:

```
SQL SQLDBM SET DATABASE SAMPLE
SQL SQLDBM SET USER USERID
SQL SQLDBM SET PASSWORD PASSWORD
SQL SQLDBM SET OPTIMIZATION OFF
SQL SQLDBM ?
```

The resulting output would be similar to:

```
(FOC1765) DBM SERVER IS           - :
(FOC1766) DBM DATABASE IS        - : SAMPLE
(FOC1767) DBM USERID IS         - : USERID
(FOC1520) AUTOLOGON OPTION IS    - : ON
(FOC1445) OPTIMIZATION OPTION IS - : OFF
```

See *The Interface Environment for Interface SET commands*.

Example: Issuing Native SQL Commands (Non-SELECT)

When the Interface identifies SQL commands, it passes them to the RDBMS for immediate execution.

With Direct SQL Passthru, one SQL command can span several lines without any prefix or continuation characters. You must complete the command or request with the END keyword.

This procedure creates the DPBRANCH table, and inserts rows:

```
SQL SQLDBM CREATE TABLE DPBRANCH
  (BRANCH_NUMBER    SMALLINT    NOT NULL,
   BRANCH_NAME      CHAR(5)     NOT NULL,
   BRANCH_MANAGER   CHAR(5)     NOT NULL,
   BRANCH_CITY      CHAR(5)     NOT NULL);
END

SQL SQLDBM INSERT INTO DPBRANCH VALUES( 1, 'WEST', 'PIAF', 'NY');
END

SQL SQLDBM INSERT INTO DPBRANCH VALUES( 2, 'EAST', 'SMITH', 'NY');
END
```

See your RDBMS documentation for information on native SQL commands.

Displaying the Effects of UPDATE and DELETE Commands

You can use the SET PASSRECS command to display the number of rows affected by a successfully executed Direct SQL Passthru UPDATE or DELETE command. The syntax is

```
SQL [target_db] SET PASSRECS {OFF}
                               {ON }
                               [ ]
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLINF</code>	Informix.
<code>EDA</code>	Enterprise Data Access/SQL.

Note: Omit if you previously issued the SET SQLENGINE command.

`OFF` Is the default. As in previous releases, the Interface provides no information as to the number of records affected by a successfully executed Direct SQL Passthru UPDATE or DELETE command.

`ON` Provides the following FOCUS message after the successful execution of a Direct SQL Passthru UPDATE or DELETE command:

```
(FOC1364) ROWS AFFECTED BY PASSTHRU COMMAND: #/operation
```

For example, a DELETE command that executes successfully and affects 20 rows generates the following message:

```
(FOC1364) ROWS AFFECTED BY PASSTHRU COMMAND: 20/DELETE
```

In addition to this message, the Interface updates the &RECORDS FOCUS system variable with the number of rows affected. You can access this variable via Dialogue Manager, and display it with the ? STAT query.

Note:

- Although you can use native SQL syntax to issue the SET PASSRECS command, if you use it to issue subsequent UPDATE or DELETE commands, the Interface will not issue the FOC1364 message or update the &RECORDS system variable. You must use Direct SQL Passthru syntax to issue UPDATE or DELETE commands in order to invoke the SET PASSRECS command.
- Since, by definition, the successful execution of an INSERT command always affects one record, INSERT does not generate the FOC1364 message.
- The FOC1364 message is for informational purposes only and does not affect the &FOCERRNUM setting.

Example: Issuing SQL SELECT Commands

When you issue an SQL SELECT request using the Direct SQL Passthru facility, the Interface passes the request directly to the RDBMS. FOCUS does not examine it. The RDBMS evaluates the request and sends storage requirements to FOCUS for future request results (answer sets).

Based on the storage requirements, FOCUS creates an internal file description named SQLOUT. The SQLOUT file description functions as a template for reading and formatting the request results.

After FOCUS prepares the storage area and the SQLOUT file description, the RDBMS executes the SQL SELECT request, retrieves the rows, and returns the answer set to FOCUS. The answer set is, in effect, a default report. FOCUS performs minimal additional formatting. When the report is complete, the internal SQLOUT file description is discarded.

The following example illustrates a SELECT statement and its default report. The SELECT statement retrieves, from the inventory table, the total number of individual units of each vendor's products for all branches located in New York (as shown in the resulting report). The subquery retrieves New York branch numbers. The request does not specify additional report formatting:

```
SQL SQLDBM
SELECT VENDOR_NUMBER, PRODUCT, SUM(NUMBER_OF_UNITS)
FROM DPINVENT
WHERE BRANCH_NUMBER IN
  (SELECT BRANCH_NUMBER
   FROM DPBRANCH
   WHERE BRANCH_CITY = 'NY')
GROUP BY VENDOR_NUMBER, PRODUCT
ORDER BY VENDOR_NUMBER, PRODUCT;
END
```

This will produce a report similar to:

PAGE	1		
VENDOR_NUMBER	PRODUCT		3
-----	-----		-
1	RADIO		15
2	MICRO		9

Internally, the process produces an SQLOUT file description that you can use for subsequent FOCUS report formatting commands.

The SQLOUT file description resides in memory; you cannot edit or print it. It exists only for the duration of the request, so subsequent requests cannot reference it. To create an internal file description that exists for the entire FOCUS session, see [Creating FOCUS Views With Direct SQL Passthru](#).

To produce the preceding default report, the Direct SQL Passthru facility implicitly issues the following TABLE request against the SQLOUT file description:

```
TABLE FILE SQLOUT
PRINT *
END
```

The RDBMS reads data once for SELECT requests via Direct SQL Passthru. FOCUS does not hold or re-read data locally, except for some types of TABLE subcommands (for instance, to re-sort or summarize rows). For performance reasons, you should incorporate as many operations as possible (particularly, sorting and aggregation operations) in your SELECT statement and rely on FOCUS for formatting and operations not available through the RDBMS.

For an example of a SELECT request that includes FOCUS report formatting commands, see [The SQLOUT File Description](#).

The SQLOUT File Description

To give you access to FOCUS report formatting facilities, the Interface generates the SQLOUT file description for each SQL SELECT query. The SQLOUT file description supports read-only access. The Interface creates this internal file description in memory based on information from the RDBMS. You cannot edit or print the SQLOUT file description; it exists only for the immediate request, so subsequent requests cannot reference it.

Note: The Interface does not generate an associated access file, since the SQL statement is stored in memory.

The SQLOUT file description describes the answer set. Each field represents one data element in the outermost SELECT list, reflecting the flat row that the RDBMS returns. The following is the SQLOUT file description created for the example in the previous topics:

```
FILENAME=SQLOUT, SUFFIX=SQLDBM, $
  SEGNAME=SQLOUT, SEGTYPE=S0, $
    FIELD='VENDOR_NUMBER' , E01, USAGE=I6 ,ACTUAL=I2 ,MISSING=OFF, $
    FIELD='PRODUCT'      , E02, USAGE=A5 ,ACTUAL=A5 ,MISSING=OFF, $
    FIELD='3'             , E03, USAGE=I9 ,ACTUAL=I4 ,MISSING=OFF, $
```

The FILENAME and the SEGNAME values are SQLOUT. The SUFFIX reflects the RDBMS being accessed. The SEGTYPE is S0. These values are constant.

The Interface uses the RDBMS DESCRIBE function to obtain column information:

- The FIELDNAME value is the column name. For expressions or functions, the FIELDNAME value depends on whether the RDBMS returns a column name:
 - If the RDBMS returns a column name then FOCUS uses it as a fieldname.
 - If the RDBMS does not return a column name, the fieldname and the alias are assigned the same default values.
- The ALIAS value is set to E0n (n starts at 1 and is incremented by 1 for each data element in the outermost SELECT list).
- The MISSING value is ON if the column allows NULLs; otherwise, it is set to OFF.

The Interface calculates USAGE and ACTUAL formats based on the column datatype and length. The datatype conversions needed are listed in the topics specific to each interface.

You can use the SET CONVERSION command to alter the length and scale of numeric columns returned by a SELECT request. The syntax is

```
SQL [target_db] SET CONVERSION
      {RESET } [RESET
      {INTEGER} | PRECISION {nn [mm] } |
      {DECIMAL} | {MAX } |
      {REAL } | [ { } ] |
      {FLOAT } ] ]
```

where:

`target_db` Is the target RDBMS. Valid values are:

- `SQLSYB` Sybase SQL Server.
- `SQLORA` Oracle.
- `SQLDBM` IBM DB2/2.
- `SQLDBC` Teradata DBC/1012.
- `SQLODBC` Microsoft ODBC.
- `SQLINF` Informix.
- `EDA` Enterprise Data Access/SQL.

Note: Omit if you previously issued the SET SQLENGINE command.

- `RESET` Returns precision and scale values that you previously altered back to the Interface defaults. If you specify RESET immediately following the SET CONVERSION command, all datatypes return to the defaults. If you specify RESET following a particular datatype, only columns of that datatype are reset.
- `INTEGER` Applies the command only to INTEGER and SMALLINT columns.
- `DECIMAL` Applies the command only to DECIMAL columns.
- `REAL` Applies the command only to single precision floating point columns.
- `FLOAT` Applies the command only to double precision floating point columns.

nn Is the precision. Must be greater than 1 and less than the maximum allowable value for the datatype. (See description of MAX.)

mm Is the scale. Valid with DECIMAL, REAL, and FLOAT datatypes. If you do not specify a value for scale, the current scale setting remains in effect.

MAX Sets the precision to the maximum allowable value for the indicated datatype:

DATATYPE	MAX Precision
INTEGER	11
REAL	9
FLOAT	20
DECIMAL	33

Note: You must include space for the decimal point and for a negative sign (if applicable) in your precision setting.

For example, to set the precision for all INTEGER and SMALLINT fields to 7:

```
SQL SQLDBM SET CONVERSION INTEGER PRECISION 7
```

To set the precision for all DOUBLE PRECISION fields to 14 and the scale to 3:

```
SQL SQLDBM SET CONVERSION FLOAT PRECISION 14 3
```

To set the precision for all INTEGER and SMALLINT fields to the default:

```
SQL SQLDBM SET CONVERSION INTEGER RESET
```

To set the precision and scale for all fields to the defaults:

```
SQL SQLDBM SET CONVERSION RESET
```

Example: Customizing a Default Report

The following example illustrates how to customize the default report output from the example in Example: Issuing SQL SELECT Commands. The example adds the TABLE FILE SQLOUT statement to the SELECT statement, and follows it by a report heading and AS phrases that rename column headings. The primary purpose of the TABLE FILE extension is for report formatting.

The following

```
SQL DB2
.
.
.
ORDER BY VENDOR_NUMBER, PRODUCT;
TABLE FILE SQLOUT
"Number of Units of Each Vendor's Products"
"          in New York Branches          "
" "
PRINT E01 AS 'Vendor,Number'
      E02 AS 'Product,Name'
      E03 AS 'Total,Units'
END
```

produces a report similar to:

```
PAGE      1

Number of Units of Each Vendor's Products
          in New York Branches

Vendor  Product      Total
Number  Name          Units
-----  -
      1  RADIO          15
      3  MICRO           9
```

When customizing a report, standard FOCUS report request syntax applies, subject to the following rules:

- You must specify fieldnames or aliases from the SLOUT file description.
- You may include any FOCUS TABLE formatting options or subcommands that you can code using the SLOUT file description.
- Most reporting operations are available. For instance, you can use FOCUS direct operators, calculate COMPUTE fields, re-sort the answer set, reorder or suppress the printing of fields, or create extract files with various formats, including HOLD FORMAT SQL.
- DEFINE fields are not permitted. They are permitted for FOCUS views created with Direct SQL Passthru.
- The ?F query command is not available for the SLOUT file description; it is available for FOCUS views created with Direct SQL Passthru.

Creating FOCUS Views With Direct SQL Passthru

You can create a named, internal file description (FOCUS view) for a particular SELECT statement with the Interface SQL PREPARE command. Unlike the SLOUT file description, you can generate reports with this FOCUS view for the entire FOCUS session.

In any FOCUS session, you can describe an unlimited number of FOCUS views. The syntax is

```
SQL [target_db] PREPARE view_name FOR
SELECT...[;]
END
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLINF</code>	Informix.
<code>EDA</code>	Enterprise Data Access/SQL.

Note: Omit if you previously issued the SET SQLENGINE command.

`view_name` Names the file description (FOCUS view). The name can be eight characters long and must conform to FOCUS naming conventions for file descriptions (see Describing Relational Tables).

`SELECT...` Is any SELECT statement.

Note:

- You cannot include FOCUS formatting options or subcommands. Issue a TABLE request against the view name to create a formatted report.
- FOCUS views created with SQL PREPARE provide read-only access to data; write operations are not permitted.
- Do not confuse creating a FOCUS view with creating an RDBMS view.

The keywords and default aliases in the generated file description are the same as those for the SLOUT file description. Only its file and the FILENAME value reflect the view name you specify in the PREPARE command.

With Direct SQL Passthru, the PREPARE command only creates an internal file description. The RDBMS does not return data until you execute a TABLE request referencing the FOCUS view. PREPARE does not generate an access file; you supply the tablename in SELECT statement FROM clauses.

File descriptions created with PREPARE reside in memory, so you cannot edit or print them. They function like any other FOCUS file description; for instance, you can specify them in report requests. You can assign DEFINE fields to them or use them in MATCH FILE commands. You can also issue the ?F query in TABLE requests to list the fieldnames of the FOCUS view. However, you cannot use any interactive tools such as Report Painter, Report Assist, or Graph Assist with FOCUS views since no physical file description exists.

In this example, DB2/2 is the target RDBMS. The SQL PREPARE command creates a view named TOTPROD:

```
SQL PREPARE TOTPROD FOR
  SELECT VENDOR_NUMBER, PRODUCT, SUM(NUMBER_OF_UNITS)
  FROM DPINVENT
  WHERE BRANCH_NUMBER IN
    (SELECT BRANCH_NUMBER
     FROM DPBRANCH
     WHERE BRANCH_CITY = 'NY')
  GROUP BY VENDOR_NUMBER, PRODUCT
  ORDER BY VENDOR_NUMBER, PRODUCT;
END
```

After the TOTPROD view is created, you can assign a temporary HI_STOCK field to the TOTPROD file description and specify the temporary field in a report request:

```
>
  DEFINE FILE TOTPROD
    HI_STOCK/A2 = IF (E03 GT 10) THEN '**' ELSE ' ' ;
  END
>
  TABLE FILE TOTPROD
  "Number of Units of Each Vendor's Products"
  "      in New York Branches          "
  " "
  PRINT E01 AS 'Vendor,Number'
        E02 AS 'Product,Name'
        E03 AS 'Total,Units'
        HI_STOCK AS ' '
  FOOTING
  "** = Too many units in stock,"
  "   reevaluate purchasing. "
  END
>
```

A report similar to the following is produced:

PAGE 1		
Number of Units of Each Vendor's Products in New York Branches		
Vendor Number	Product Name	Total Units
-----	-----	-----
1	RADIO	15 **
3	MICRO	9
** = Too many units in stock, reevaluate purchasing.		

The one restriction on FOCUS views created with Direct SQL Passthru involves using them with the FOCUS JOIN command. You cannot JOIN two FOCUS views or a view with another FOCUS-readable source. You can, however, create a HOLD file of data extracted from the FOCUS view and use the HOLD file in the JOIN.

Controlling SQL Passthru Transactions

It is possible to group together a sequence of SQL Passthru commands using the BEGIN SESSION and END SESSION commands, and to control when and if the transactions are committed using the COMMIT and ROLLBACK commands.

BEGIN/END SESSION

The BEGIN SESSION command begins a sequence of Direct SQL Passthru commands; the END SESSION command terminates the sequence. The syntax is

```
SQL [target_db] { BEGIN }  
                  { END   } SESSION
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLINF</code>	Informix.

Note: Omit if you previously issued the SET SQLENGINE command.

<code>BEGIN</code>	Indicates that a sequence of SQL commands is to be passed to the RDBMS. While the BEGIN option is in effect, the END syntax that terminates each Direct SQL Passthru statement does not automatically release resources.
<code>END</code>	Indicates the end of a sequence of SQL commands. Closes all cursors and releases all resources. Executes actions specified in SET action ON COMMAND (see Transaction Control Commands). Purges all statements prepared inside the BEGIN SESSION/END SESSION pair.

After the END SESSION command is executed, cursors and statements prepared in the BEGIN SESSION/END SESSION pair are unavailable. The sequence of statements in the BEGIN SESSION/END SESSION pair can include:

- SQL Passthru commands PREPARE, BIND, and EXECUTE.
- FOCUS commands that refer to Direct SQL Passthru-created views.
- Other FOCUS commands such as TABLE and MODIFY against SQL or other files.

Note:

- FIN issues END COMMAND and then purges all statements prepared outside the BEGIN SESSION/END SESSION pair.
- FOCUS commands must use the same RDBMS server.
- SQL commands. The SQL commands COMMIT WORK and ROLLBACK WORK are particularly useful in Parameterized SQL Passthru requests.
- Environmental commands such as SET SERVER.

If you omit the BEGIN SESSION/END SESSION pair, the Interface automatically brackets each individual Direct SQL Passthru command with BEGIN SESSION and END SESSION. The execution of the END SESSION (either implicitly or explicitly) in a Direct SQL Passthru statement invokes actions requested in SET action ON COMMAND (see Transaction Control Commands). You can use statements prepared without an explicit BEGIN SESSION/END SESSION pair in TABLE requests, but you cannot use them in the EXECUTE statement.

COMMIT WORK (Oracle, DB2/2 only)

COMMIT WORK terminates a unit of work and makes all database changes permanent. The syntax is

```
SQL [target_db] COMMIT WORK
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLINF</code>	Informix.

Note: Omit if you previously issued the SET SQLENGINE command.

After execution of the COMMIT WORK command, the RDBMS drops the prepared status of SQL statements; it also releases locks. If you need a prepared version of the statement, you must issue the SQL PREPARE statement again.

ROLLBACK WORK (Oracle, DB2/2 only)

ROLLBACK WORK terminates a unit of work and restores all data changed by SQL statements to their state at the last commit point. The syntax is

```
SQL [target_db] ROLLBACK WORK
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server
<code>SQLORA</code>	Oracle
<code>SQLDBM</code>	IBM DB2/2
<code>SQLDBC</code>	Teradata DBC/1012
<code>SQLODBC</code>	Microsoft ODBC
<code>SQLINF</code>	Informix.

Note: Omit if you previously issued the SET SQLENGINE command.

After execution of the ROLLBACK WORK command, the RDBMS drops the prepared status of SQL statements. If you need a prepared version of the statement, you must issue the SQL PREPARE statement again.

Parameterized SQL Passthru

The Direct SQL Passthru facility supports parameterized SQL statements. These statements incorporate parameter markers to indicate where a value should be substituted, so you can execute the SQL statements multiple times with varying input values.

The following is an example of a parameterized SQL statement for Oracle:

```
INSERT INTO INVENTORY (PARTNO) VALUES (:0001)
```

The INSERT statement is executed once for each value you provide for the parameter marker and a new row with that value is placed in the PARTNO column. The parameter marker varies for each Interface.

Parameterized SQL Passthru provides the following advantages:

- You can execute an SQL statement using varying values without keying in the entire SQL statement for each value.
- Internally, it utilizes the optimal SQL for the native RDBMS.
- The execution of a FOCUS session becomes equivalent to that of a 3GL program, with a framework consisting of such elements as compiled statements and parameter markers (input values).

Note:

- All errors are posted to Dialogue Manager variables &RETCODE and &FOCERRNUM. &RETCODE will contain the last RDBMS error code; &FOCERRNUM will contain the last FOCUS error code.
- Direct SQL Passthru and Parameterized SQL Passthru are not available in MODIFY.

Parameterized SQL Command Summary

With parameterized SQL you can compile, bind, and repeatedly execute a series of SQL commands.

To incorporate parameter markers in SQL statements, first compile the statements with the PREPARE command, then bind them with the BIND command, and subsequently execute them with the EXECUTE command. Place this group of actions in a BEGIN SESSION/END SESSION pair. You can also include other FOCUS, SQL, and Interface environmental commands in the BEGIN SESSION/END SESSION pair.

Subsequent topics explain the individual commands involved in Parameterized Passthru design. Sample Session: Oracle and Sample Session: Teradata contain sample sessions. The general syntax is

```
SQL [target_db] {PREPARE }
                  {BIND   }
                  {EXECUTE } [;]
                  {PURGE  }
[TABLE FILE statement-name]
[options]
END
```

where:

target_db Is the target RDBMS. Valid values are:

- SQLSYB** Sybase SQL Server.
- SQLORA** Oracle.
- SQLDBM** IBM DB2/2.
- SQLDBC** Teradata DBC/1012.
- SQLODBC** Microsoft ODBC.
- SQLINF** Informix.
- EDA** Enterprise Data Access/SQL.

Note: Omit if you previously issued the SET SQLENGINE command.

; For SQL SELECT requests only, the semicolon is required if you intend to specify additional FOCUS report options.

TABLE FILE Is permitted only with PREPARE and EXECUTE commands that invoke SQL SELECT requests; invokes FOCUS report formatting options or operations. By including a TABLE FILE request, you can produce different customized reports with one SQL query. The answer set is returned at EXECUTE time. If you include a TABLE FILE request in both a PREPARE and EXECUTE command for the same SQL statement, the EXECUTE request takes precedence.

statement-name Is the name of a prepared SELECT statement.

options Are FOCUS report formatting options.

END Terminates the request. Is optional for Interface SET commands, the SQL commands COMMIT WORK and ROLLBACK WORK, and the Passthru commands BEGIN SESSION, END SESSION, and PURGE (discussed in subsequent topics). Required for all other commands.

Note: Do not confuse the SQL PREPARE and BIND statements with the RDBMS prepare and bind; the RDBMS versions are not available through the Interface.

PREPARE

The PREPARE command prepares (checks syntax, then compiles) an SQL statement and stores the compiled version for later use. The SQL statement can contain parameter markers. The syntax is

```
SQL [target_db] PREPARE statement-name FOR sql-statement [;]
```

where:

target_db Is the target RDBMS. Valid values are:

<i>SQLSYB</i>	Sybase SQL Server.
<i>SQLORA</i>	Oracle.
<i>SQLDBM</i>	IBM DB2/2.
<i>SQLDBC</i>	Teradata DBC/1012.
<i>SQLODBC</i>	Microsoft ODBC.
<i>SQLINF</i>	Informix.
<i>EDA</i>	Enterprise Data Access/SQL.

Note: Omit if you previously issued the SET SQLENGINE command.

statement-name Is the 1-to 8-character name of an SQL variable that will contain the prepared (compiled) version of an SQL statement.

sql-statement Is a character-string expression that can include parameter markers; it represents the SQL statement to PREPARE. The statement must be one of the following:

- UPDATE (WHERE CURRENT OF CURSOR is not supported)
- DELETE (WHERE CURRENT OF CURSOR is not supported)
- INSERT
- SELECT (File description is created to represent the returned answer set)
- CREATE
- DROP
- ALTER
- COMMENT
- LABEL
- GRANT
- REVOKE
- COMMIT
- ROLLBACK
- LOCK

- EXPLAIN

;

Is required if sql-statement is a SELECT statement followed by TABLE FILE report options.

Note: For Oracle, use :n for parameter markers; for Teradata, use ?.

For example, consider the following SQL PREPARE command:

```
SQL SQLORA PREPARE D FOR DELETE FROM EMPLOYEE
WHERE EMP_ID = :0001
```

Variable D will contain the prepared version of the SQL string: DELETE FROM EMPLOYEE WHERE EMP_ID = :0001.

You supply values for the parameter marker in the statement by issuing an EXECUTE statement for variable D. The parameter marker allows you to execute the same DELETE statement many times with different values of EMP_ID. You can use a parameter marker anywhere a literal value appears in an SQL statement.

Note:

- Answer sets for FOCUS TABLE requests in prepared SELECT statements are returned at EXECUTE time even if they were specified in the PREPARE statement.

If both the PREPARE and EXECUTE commands specify a TABLE FILE request for the same statement, the EXECUTE request takes precedence.

- PREPARE for an already prepared statement unprepares the statement by means of PURGE. As a side effect, BIND for this statement (if any) is cleared.

EXECUTE

This statement executes a previously prepared SQL statement. If the SQL statement includes parameter markers, you must supply their values in the USING clause. If the SQL statement is a SELECT statement, you can use the TABLE FILE extension to produce a formatted report.

The syntax is

```
SQL [target_db] EXECUTE statement-name [USING data_list] [;]
```

where:

target_db	Is the target RDBMS. Valid values are:
SQLSYB	Sybase SQL Server.
SQLORA	Oracle.
SQLDBM	IBM DB2/2.
SQLDBC	Teradata DBC/1012.
SQLODBC	Microsoft ODBC.
SQLINF	Informix.
EDA	Enterprise Data Access/SQL.

Note: Omit if you previously issued the SET SQLENGINE command.

statement-name Is the 1-to 8-character name of an SQL variable that contains the prepared (compiled) version of the SQL statement to execute. Use this name in a TABLE FILE extension if you want a formatted report.

data_list Is a list of arguments to substitute for the parameter markers in the prepared SQL statement. Separate arguments in the list with commas.

;

Is required if the prepared SQL statement is a SELECT statement followed by TABLE FILE report options.

For example, to execute the PREPARE statement in the previous example, issue:

```
SQL SQLORA EXECUTE D USING 'AAA' ;
```

The DELETE statement from the previous topic is sent to the Oracle RDBMS; the RDBMS deletes all rows with an EMP_ID value of 'AAA' from the database.

You can execute this statement many times in the same unit of recovery by supplying different values for the EMP_ID to delete.

The SQL commands COMMIT WORK and ROLLBACK WORK destroy all statements prepared in a unit of recovery. Thus, after a COMMIT or ROLLBACK, you must again PREPARE any statement you want to EXECUTE.

Note:

- Use of EXECUTE outside a BEGIN SESSION/END SESSION pair produces fatal error FOC1477, "Invalid use of BIND or EXECUTE".
- Use of EXECUTE for a statement prepared outside a BEGIN SESSION/END SESSION pair produces fatal error FOC1477, "Invalid use of BIND or EXECUTE".
- Answer sets for FOCUS TABLE requests in compiled SELECT statements are returned at EXECUTE time even if they were specified in the PREPARE statement.

If both the PREPARE and EXECUTE commands specify a TABLE FILE request for the same statement, the EXECUTE request takes precedence.

- To specify missing values in an EXECUTE statement, you can use the word NULL or denote the missing column values by a comma. For example

```
EXECUTE xyz USING 8,9,NULL,, 'abcd'
```

PURGE

The PURGE command clears results from a previously issued PREPARE or BIND command. It is optional. The syntax is

```
SQL [target_db] PURGE statement-name [;]
```

where:

target_db Is the target RDBMS. Valid values are:

- SQLSYB** Sybase SQL Server.
- SQLORA** Oracle.
- SQLDBM** IBM DB2/2.
- SQLDBC** Teradata DBC/1012.
- SQLODBC** Microsoft ODBC.
- SQLINF** Informix.
- EDA** Enterprise Data Access/SQL.

Note: Omit if you previously issued the SET SQLENGINE command.

statement-name Is the 1-to 8-character name of an SQL variable that contains a prepared (compiled) version of an SQL statement.

For example:

```
SQL SQLORA PREPARE D FOR DELETE FROM EMPLOYEE
WHERE EMP_ID = :0001
```

```
SQL SQLORA PURGE D;
```

BIND

You can use the BIND command to define the format of each parameter specified in a PREPARE command. The list of formats is comma delimited; each element is a datatype supported by the RDBMS. The syntax is

```
SQL [target_db] BIND statement-name USING format_list;
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLINF</code>	Informix.

`EDA` Enterprise Data Access/SQL.

Note: Omit if you previously issued the SET SQLENGINE command.

`statement-name` Is the 1-to 8-character name of an SQL variable that contains a prepared (compiled) version of an SQL statement.

`format_list` Is a comma-delimited list of datatypes used in the request. The following datatypes are supported by the RDBMS:

- SMALLINT
- INTEGER
- DECIMAL(m,n)
- FLOAT
- REAL
- DOUBLE
- VARCHAR(n)
- LONGVARCHAR(n)
- CHARACTER(n)
- DATE

- If a statement is prepared and executed without a corresponding BIND, the Interface uses default formats based on the RDBMS storage formats it detects for the columns referenced by parameter markers in the statement.
- Use of BIND outside a BEGIN SESSION/END SESSION pair produces fatal error FOC1477, "Invalid use of BIND or EXECUTE".
- BIND without parameters clears a previous BIND for the statement.

BIND is also cleared when you issue PREPARE for an already prepared statement.

Sample Session: Oracle

The following sample session illustrates the design of a Parameterized SQL application:

```
SQL SQLORA BEGIN SESSION

SQL SQLORA PREPARE ADDRECS FOR
  INSERT INTO DPVENDOR (VENDOR_NUMBER, VENDOR_NAME, VENDOR_CITY)
  VALUES (:1,:2,:3);
END

SQL SQLORA PREPARE REP FOR
  SELECT * FROM DPVENDOR WHERE VENDOR_CITY = :1;
END

-* (Optionally) Bind variables

SQL SQLORA BIND ADDRECS USING INTEGER, CHAR(40), CHAR(2);
END

SQL SQLORA BIND REP USING CHAR(2);
END

-* Add records to table

SQL SQLORA EXECUTE ADDRECS USING 1, 'All Electronics Supply Inc.', 'NY' ;
END

SQL SQLORA END SESSION
```

Notice that the BIND commands provide formats and the EXECUTE commands provide values for the parameter markers. Since DEF represents a SELECT statement, the EXECUTE command for DEF can include a TABLE FILE DEF request. Alternatively, the TABLE FILE DEF request could have been included in the PREPARE DEF command instead of in the EXECUTE DEF command.

Sample Session: Teradata

The following sample session illustrates the design of a Parameterized SQL application:

```
SQL SQLDBC BEGIN SESSION

SQL SQLDBC PREPARE ADDRECS FOR
  INSERT INTO DPVENDOR (VENDOR_NUMBER, VENDOR_NAME, VENDOR_CITY)
  VALUES (?, ?, ?);
END

SQL SQLDBC PREPARE REP FOR
  SELECT * FROM DPVENDOR WHERE VENDOR_CITY = ?;
END

-* (Optionally) Bind variables

SQL SQLDBC BIND ADDRECS USING INTEGER, CHAR(40), CHAR(2);
END

SQL SQLDBC BIND REP USING CHAR(2);
END

-* Add records to table

SQL SQLDBC EXECUTE ADDRECS USING 1, 'All Electronics Supply Inc.', 'NY' ;
END

SQL SQLDBC EXECUTE ADDRECS USING 2, 'ABC Radio Co.', 'NJ';
END

SQL SQLDBC EXECUTE ADDRECS USING 3, 'Tri-State Electronics', 'NY';
END

SQL SQLDBC EXECUTE REP USING 'NY';
TABLE FILE REP
PRINT *
END

SQL SQLDBC END SESSION
```

Transaction Control Commands

Topics:

- [SET AUTOaction ON event command.](#)
- [Action and event combinations.](#)
- [Combinations of SET AUTOaction commands.](#)

The Interface includes a variety of COMMIT and connection control capabilities. The SET AUTOaction ON event command implements these facilities.

The SET AUTOaction ON Event Command

Actions are RDBMS commands, such as COMMIT WORK, that the Interface issues in response to events in a FOCUS session. Events include the end of a MODIFY and MAINTAIN or TABLE request, interaction with the terminal, and the end of a FOCUS session.

For DB2/2, you can make a MODIFY transaction pseudo-conversational by issuing CLOSE and DISCONNECT automatically at all COMMIT points in the MODIFY procedure. Or, by delaying COMMITs until a group of commands is issued, you can combine FOCUS commands and native SQL commands in one Logical Unit of Work.

The SET AUTOaction command allows you to control when the Interface issues actions. Subsequent topics discuss each action. The syntax is

```
SQL [target_db] SET {AUTOCOMMIT } ON {CRTFORM}
                   {AUTODISCONNECT} {COMMAND}
                   {FIN }
                   {COMMIT }
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLINF</code>	Informix.

Note: Omit if you previously issued the SET SQLENGINE command.

`AUTOCOMMIT` Is the SQL command COMMIT WORK.

`AUTODISCONNECT` Severs the connection between the RDBMS and the user.

`CRTFORM` Is valid only in MODIFY procedures with the AUTOCOMMIT action. Issues a COMMIT before each interaction with the user's terminal. At the end of the MODIFY, the event setting reverts to its value prior to the AUTOCOMMIT ON CRTFORM. Omit `target_db` from the command. Refer to Maintaining Relational Tables for more information on MODIFY.

Note: For MAINTAIN, COMMITs are only performed explicitly using the MAINTAIN COMMIT command.

`COMMAND` Executes the specified action at the end of a MAINTAIN or MODIFY procedure, a TABLE request, or a Direct SQL Passthru request.

Note: The Interface does not generate an end-of-MODIFY COMMIT if there is no open Logical Unit of Work.

FIN Executes the specified action automatically only after the FOCUS session has been terminated by the FOCUS FIN command. (You can explicitly issue the COMMIT command using SQL Direct Passthru, MAINTAIN, or MODIFY.)

COMMIT Executes the specified action whenever a COMMIT or ROLLBACK is issued either as a native SQL command or because of a current AUTOCOMMIT setting. This setting is only valid when used with AUTODISCONNECT.

Note:

- Depending on how often the event occurs (and the corresponding command is issued), the AUTOAction setting may result in considerable overhead. Almost all of this overhead is not FOCUS related; it is RDBMS or network related.
- All settings are session level and can be issued from either a procedure or the FOCUS session command line, except AUTOCOMMIT ON CRTFORM, which can only be issued in a MODIFY procedure. In a MODIFY procedure, the only valid command is AUTOCOMMIT ON CRTFORM.
- If AUTOCOMMIT is set on COMMAND, the Interface issues a COMMIT WORK at the end of a MODIFY procedure provided there is an open Logical Unit of Work (LUW).

Actions

Actions control lock retention and the user's connection to the RDBMS.

AUTOCOMMIT

SET AUTOCOMMIT issues an SQL COMMIT WORK each time the specified event occurs. Until a COMMIT WORK, changes to the target database are conditional and locks are held on the affected data. Other users may have their work delayed waiting for locks to be released.

COMMIT WORK completes the changes to the database and releases locks, improving concurrency. On the other hand, delaying the COMMIT preserves the integrity of processed data through several FOCUS commands or an entire FOCUS session.

AUTODISCONNECT

DISCONNECT completely detaches the user from the RDBMS. After a DISCONNECT, FOCUS must reestablish its connection to the RDBMS before doing any database work. FOCUS procedures that frequently issue the DISCONNECT command are connected to the RDBMS for shorter periods of time, allowing other tasks to connect and acquire threads as needed. However, there is significant overhead associated with frequently connecting and disconnecting, and the possibility exists that no thread will be immediately available when the procedure attempts to reconnect.

Action and Event Combinations

The following table summarizes possible combinations of actions and events:

- D indicates a default combination.
- X indicates a combination that either is not supported or does not apply.
- S indicates a supported combination.

Actions	Events			
	COMMIT	COMMAND	CRTFORM	FIN
COMMIT	X	D	S*	S
DISCONNECT	S	S	X	D

Note: * Supported in a MODIFY procedure only.

Effects of Action and Event Combinations

The following topics discuss the advantages and disadvantages of certain combinations of actions and events:

[SET AUTOCOMMIT ON CRTFORM](#)

[SET AUTOCOMMIT ON FIN](#)

[SET AUTODISCONNECT ON COMMIT](#)

[SET AUTODISCONNECT ON FIN](#)

SET AUTOCOMMIT ON CRTFORM

This command works only in a MODIFY procedure and requires a slightly different syntax:

```
SQL SET AUTOCOMMIT ON CRTFORM
```

Note the absence of the target database qualifier after the SQL keyword. Including a qualifier generates a syntax error.

This is the "COMMIT as often as possible" strategy. FOCUS issues a COMMIT prior to displaying each CRTFORM, thus releasing all locks before presenting data to the user. AUTOCOMMIT ON CRTFORM invokes Change Verify Protocol (CVP); you must check the FOCCURENT variable to determine whether CVP detected a conflict with another user. Maintaining Relational Tables discusses this setting and some limits on its use. This setting also requires KEYS> 0 in the access file (see Describing Relational Tables).

With this strategy, more concurrent users can access the same RDBMS data. However, a COMMIT carries overhead and may be unnecessary for the application.

At the end of the MODIFY procedure, the event setting reverts to its value before the AUTOCOMMIT ON CRTFORM was issued.

SET AUTOCOMMIT ON FIN

During the FOCUS session, only those COMMIT and ROLLBACK commands you issue explicitly are executed. The Interface automatically issues a COMMIT at the end of the FOCUS session. The syntax is

```
SQL [target\_db] SET AUTOCOMMIT ON FIN
```

where:

[target_db](#) Is the target RDBMS. Valid values are:

SQLSYB	Sybase SQL Server.
SQLORA	Oracle.
SQLDBM	IBM DB2/2.
SQLDBC	Teradata DBC/1012.
SQLODBC	Microsoft ODBC.
SQLINF	Informix.

Note: Omit if you previously issued the SET SQLENGINE command.

With this command, you do not incur COMMIT overhead until the end of the FOCUS session. You can also use it to suspend the COMMIT action for a period of time. See Example: Explicit Control of a Logical Unit of Work (LUW) for an example.

Holding on to unneeded locks can cause contention in the RDBMS system and make other users wait for data. Delaying the COMMIT also puts your data changes at risk in case of system or machine failure.

SET AUTOCOMMIT ON FIN gives you full control of Logical Units of Work (LUWs) in your FOCUS session. FOCUS relies on your explicit COMMIT and ROLLBACK commands. No implicit COMMIT or ROLLBACK is ever forced until the end of the FOCUS session (FIN command). Use this option cautiously to avoid possible locking problems and unpredictable consequences in cases of conflict with other AUTOaction settings.

Note: With AUTOCOMMIT ON FIN, SQL errors do not trigger an automatic ROLLBACK by the Interface. Examine each return code and take appropriate action to prevent unwanted changes from being committed at FIN.

SET AUTODISCONNECT ON COMMIT

When a COMMIT is issued, the FOCUS session is disconnected from the RDBMS. The syntax is

```
SQL [target_db] SET AUTODISCONNECT ON COMMIT
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLINF</code>	Informix.

Note: Omit if you previously issued the SET SQLENGINE command.

This setting frees the RDBMS connection for use by other users. The disadvantage is the cost of repeatedly connecting to the RDBMS. Connections, once released, may not be available when needed, so you may be unable to reconnect.

SET AUTODISCONNECT ON FIN

This command disconnects FOCUS from the RDBMS at the end of the FOCUS session, duplicating Interface default behavior. The syntax is

```
SQL [target_db] SET AUTODISCONNECT ON FIN
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLINF</code>	Informix.

Note: Omit if you previously issued the SET SQLENGINE command.

Combinations of SET AUTOaction Commands

Think of actions as a nested sequence; the COMMIT action is the innermost level, then DISCONNECT. Events are also organized hierarchically; CRTFORM is the innermost level, then COMMIT, then COMMAND, and, finally, FIN. In general, you should not set the outer of two actions to occur more frequently than the inner action.

Recommended combinations follow:

When AUTOCOMMIT is set to FIN,
AUTODISCONNECT must be FIN*.

When AUTOCOMMIT is set to COMMAND or CRTFORM,
AUTODISCONNECT may be FIN, COMMAND or COMMIT.

When AUTODISCONNECT is set to COMMIT,
AUTOCOMMIT may be COMMAND or CRTFORM.

* Violations of this rule can cause unpredictable behavior.

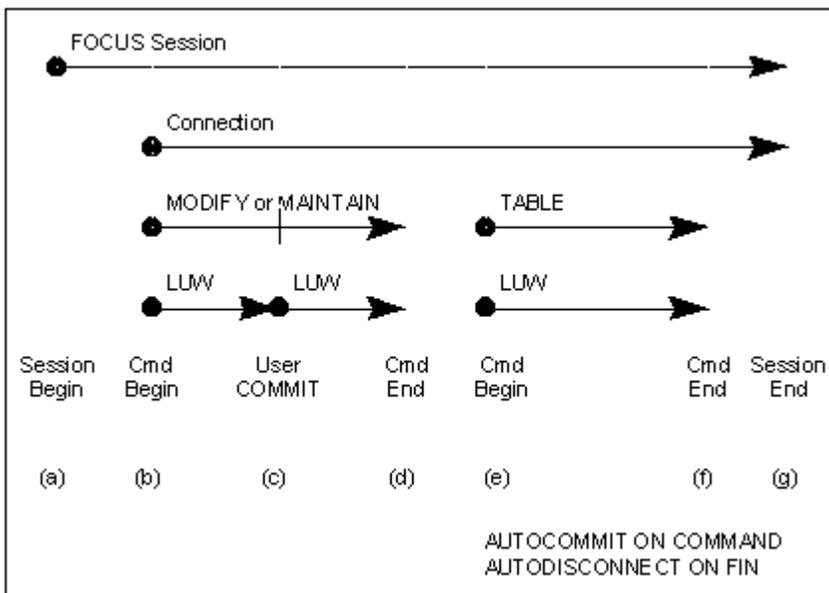
Types of FOCUS Sessions

You can establish the following three types of FOCUS sessions by varying the combinations of SET AUTOaction ON event commands.

- The default Interface session corresponds to the settings AUTOCOMMIT ON COMMAND and AUTODISCONNECT ON FIN.
- The user-controlled session corresponds to the settings AUTOCOMMIT ON FIN and AUTODISCONNECT ON FIN.
- The pseudo-conversational session corresponds to the settings AUTOCOMMIT ON CRTFORM and AUTODISCONNECT ON COMMIT.

The Default Interface Session

The following illustration shows the duration of connections and Logical Units of Work (LUWs) using the Interface default settings:



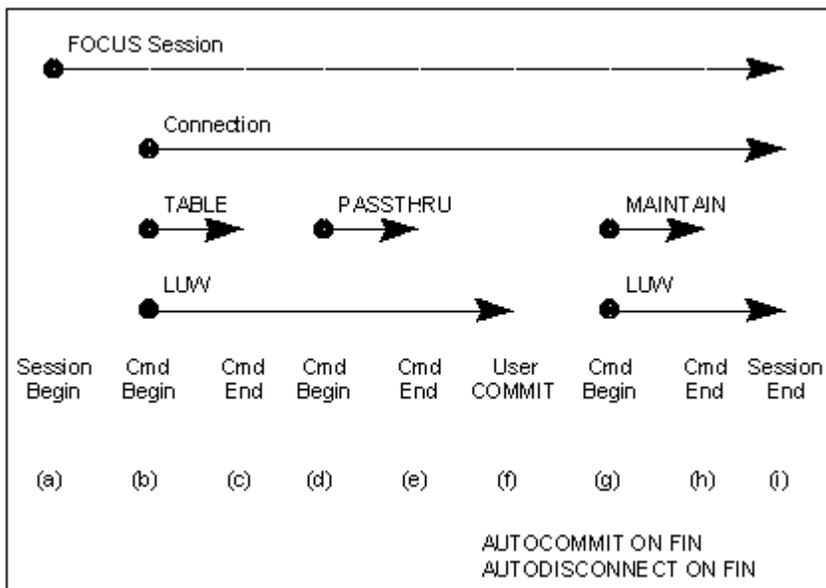
The FOCUS session begins at point (a) and ends at point (g), with the FIN command. The connection is established at point (b), the first MODIFY or MAINTAIN call to the RDBMS, and is retained for the entire FOCUS session.

The LUW initiated at point (b) by the MODIFY or MAINTAIN is terminated by the explicit COMMIT issued at point (c) in the MODIFY or MAINTAIN. Point (c) also marks the start of the next LUW which is retained until the end of the MODIFY or MAINTAIN command at point (d), where the Interface automatically generates a COMMIT.

A third LUW begins for the TABLE request. It is terminated by the COMMIT automatically generated at the end of the TABLE command. At FIN, the Interface terminates the connection to the RDBMS. No COMMIT is generated at FIN.

The User-Controlled Session

The following illustration shows a session in which the user completely controls the duration of each Logical Unit of Work (LUW). The Interface does not automatically issue any COMMIT, CLOSE, or DISCONNECT command until after the FIN at the end of the FOCUS session:



The FOCUS session begins at point (a) and ends at point (i) with the FIN command.

The connection is established by the TABLE command at point (b) and retained until FIN. The LUW also starts at point (b). It is completed at point (f) when the user issues COMMIT WORK as an SQL Passthru request.

The final LUW, triggered by the MAINTAIN, is not terminated until FIN since there is no other user-issued COMMIT.

Example: Explicit Control of a Logical Unit of Work (LUW)

Prior releases of the Interface automatically followed each native SQL request by a COMMIT; therefore, you could not have more than one native SQL command in a single LUW.

The following annotated DB2/2 example demonstrates how to explicitly control the scope of an LUW using the expanded AUTO command settings. The numbers on the left refer to the explanatory notes that follow the example:

```
1.  -TOP
    SQL SQLDBM SET AUTODISCONNECT ON FIN
2.  SQL SQLDBM SET AUTOCOMMIT ON FIN

3.  SQL SQLDBM LOCK TABLE XYZ IN EXCLUSIVE MODE
    END
4.  -IF &RETCODE NE 0 GOTO ROLLBACK ;

5.  SQL SQLDBM INSERT INTO XYZ VALUES ('A','B','C','D') ;
    END
    -IF &RETCODE NE 0 GOTO ROLLBACK ;

    SQL SQLDBM INSERT INTO XYZ VALUES ('E','F','G','H') ;
    END
    -IF &RETCODE NE 0 GOTO ROLLBACK ;

    SQL SQLDBM INSERT INTO XYZ VALUES ('I','J','K','L') ;
    END
    -IF &RETCODE NE 0 GOTO ROLLBACK ;

6.  SQL SQLDBM COMMIT WORK;
    END
    -IF &RETCODE NE 0 GOTO ROLLBACK ;

    -GOTO OUT

    -ROLLBACK
    SQL SQLDBM ROLLBACK WORK;
    END

    -OUT
7.  SQL SQLDBM SET AUTOCOMMIT ON COMMAND
```

Note:

1. Dialogue Manager control statements govern COMMIT or ROLLBACK processing based on the &RETCODE value. The example treats any value other than 0 as a failure.
2. AUTOCOMMIT is set to FIN. No COMMIT is issued unless specifically coded. As is required for this AUTOCOMMIT setting, AUTODISCONNECT is also set to FIN.
3. Table XYZ is locked for this program's exclusive use. This lock will be terminated at a COMMIT or ROLLBACK point.
4. The procedure checks &RETCODE. If it is not zero, the SQLCODE is displayed to show that the lock was not completed; the procedure issues a ROLLBACK and terminates. This &RETCODE test is executed after every SQL statement passed to DB2.
5. Three rows are inserted in XYZ.
6. The program issues an explicit COMMIT making the inserts to XYZ permanent and releasing the exclusive lock.
7. SET AUTOCOMMIT is reissued to restore automatic command-level COMMITs.

The Interface Environment

Topics:

- [Interface environment SET commands.](#)
- [Interacting with the Interfaces.](#)

Interface environmental commands can change certain parameters that govern your FOCUS session.

Interface Environment SET Commands

Interface SET commands enable you to control how the interface behaves. The following topics describe the available settings. In general, the syntax is of the form

```
SQL [target_db] SET command value
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLINF</code>	Informix.
<code>EDA</code>	Enterprise Data Access/SQL.

Note: Omit if you previously issued the SET SQLENGINE command.

`command` Is an environmental command.

`value` Is an acceptable value for the environmental command.

? Query (All Interfaces)

The Interface SQL query command displays Interface defaults and current settings for the RDBMS.

To check your defaults, issue the query from the FOCUS command level or use the ? Query tool. The syntax is

```
SQL [target_db] ?
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLINF</code>	Informix.
<code>EDA</code>	Enterprise Data Access/SQL.

Note: Omit if you previously issued the SET SQLENGINE command.

For example, the following query

```
sql sqlsyb ?
```

produces a message box similar to the following:

```
(FOC1751) SYBASE SERVER NAME IS      -  
(FOC1752) SYBASE USER NAME IS       -  
(FOC1753) SYBASE TIME OUT FACTOR IS - : 60  
(FOC1754) SYBASE LOGIN TIME IS      - : 60  
(FOC1441) WRITE FUNCTIONALITY IS    - : ON  
(FOC1445) OPTIMIZATION OPTION IS    - : ON
```

FETCHSIZE and INSERTSIZE (Oracle Only)

The Oracle Interface supports array blocking of SELECT and INSERT requests. Array blocking is a method that Oracle utilizes to buffer repeated executions of the same SQL command (in this case, FETCH or INSERT), and then execute them in bulk when the buffer is full. This feature can substantially increase efficiency for certain requests.

The default array block size for both SELECT and INSERT requests is 20. The block size for SELECT requests applies to TABLE FILE requests, MATCH file requests, MODIFY MATCH requests, and Direct SQL Passthru SELECT statements. The block size for INSERT requests applies to MODIFY INCLUDE requests and parameterized Direct SQL Passthru INSERT statements.

To enable array blocking for SELECT requests, enter the following command

```
SQL SQLORA SET FETCHSIZE n
```

where:

n Is the number of rows to be buffered and retrieved using blocked FETCH commands. Acceptable values are 1 to 5000. The default is 20.

To enable array blocking for INSERT requests, enter the following command

```
SQL SQLORA SET INSERTSIZE n
```

where:

n Is the number of INSERT rows to be buffered before the block of rows is actually transmitted to the RDBMS. Acceptable values are 1 to 5000. The default is 20.

The sole purpose of the FETCHSIZE and INSERTSIZE settings is to improve performance. High values increase the efficiency of requests that involve many rows, at the cost of higher memory usage. The best way to find optimum values is to experiment with different values. However, once values exceed 100, the increased efficiency provided is generally negligible.

The INSERTSIZE parameter is only functional for consecutive executions of INSERT statements that are identical to each other (except for the values to be inserted). No other intervening SQL statements are allowed, including COMMIT WORK. If a statement is issued that in any way (other than the inserted values) differs from the current blocked INSERT statement in effect, the block is immediately transmitted to the RDBMS, even if it is only partially full.

OPTIMIZATION (All Interfaces)

Depending on the OPTIMIZATION setting, the Interface may generate SQL SELECT statements that allow the RDBMS to perform functions (such as join and aggregation) and return the data to the Interface for FOCUS report generation (see The Interface Optimizer).

To invoke Interface Optimization, enter the following at the FOCUS command level

```
SQL [target_db] SET { OPTIMIZATION } { ON }  
                   { SQLJOIN   } { OFF }  
                   {           } { FOCUS }  
                   {           } { SQL   }
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLINF</code>	Informix.
<code>EDA</code>	Enterprise Data Access/SQL.

Note: Omit if you previously issued the SET SQLENGINE command.

`SQLJOIN` Is an alias for OPTIMIZATION.

`OFF` Instructs the Interface to create SQL statements for simple data retrieval from each table. FOCUS processes the returned sets of data in your address space or virtual machine to produce the report.

`ON` Instructs the Interface to create SQL statements that take advantage of the RDBMS join, sort, and aggregation capabilities. This parameter is compatible with previous releases in regard to the multiplicative effect: misjoined unique segments and multiplied lines in PRINT and LIST based report requests do not disable optimization. ON is the default.

`FOCUS` Passes join logic to the RDBMS only when the results will be the same as from a FOCUS-managed request. Misjoined unique segments, the multiplicative effect, and multiplied lines in PRINT and LIST based report requests temporarily disable optimization.

`SQL` Passes join logic to the RDBMS in all possible cases. The multiplicative effect does not disable optimization, even in cases involving aggregation (SUM, COUNT). Join logic is not passed to the RDBMS for an outer join (SET ALL = ON), for tables residing on multiple subsystems, and for tables residing on multiple DBMS platforms.

PASSRECS

You can use the SET PASSRECS command to display the number of rows affected by a successfully executed Direct SQL Passthru UPDATE or DELETE command. The syntax is

```
SQL [target_db] SET PASSRECS {OFF}
                               {ON }
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLINF</code>	Informix.
<code>EDA</code>	Enterprise Data Access/SQL.

Note: Omit if you previously issued the SET SQLENGINE command.

`OFF` Is the default. As in previous releases, the Interface provides no information as to the number of records affected by a successfully executed Direct SQL Passthru UPDATE or DELETE command.

`ON` Provides the following FOCUS message after the successful execution of a Direct SQL Passthru UPDATE or DELETE command:

```
(FOC1364) ROWS AFFECTED BY PASSTHRU COMMAND: #/operation
```

For example, a DELETE command that executes successfully and affects 20 rows generates the following message:

```
(FOC1364) ROWS AFFECTED BY PASSTHRU COMMAND: 20/DELETE
```

In addition to this message, the Interface updates the &RECORDS FOCUS system variable with the number of rows affected. You can access this variable via Dialogue Manager, and display it with the ? STAT query.

Note:

- Since, by definition, the successful execution of an INSERT command always affects one record, INSERT does not generate the FOC1364 message.
- The FOC1364 message is for informational purposes only and does not affect the &FOCERRNUM setting.

ERRORTYPE (All Interfaces)

With SET ERRORTYPE, you can instruct the Interface to return native RDBMS error messages, as well as FOCUS error messages, for those error conditions that are reported by the RDBMS.

The syntax is

```
SQL [target_db] SET ERRORTYPE { FOCUS }  
                               { DBMS }
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLINF</code>	Informix.
<code>EDA</code>	Enterprise Data Access/SQL.

Note: Omit if you previously issued the SET SQLENGINE command.

`FOCUS` Generates only FOCUS error messages.

`DBMS` Produces native DBMS error messages as well as FOCUS error messages.

ANSITOOEM (All Interfaces)

When using FOCUS with a relational interface, FOCUS allows ANSI to OEM conversion to be handled either by the database vendors client software or automatically by FOCUS.

By default, FOCUS expects to interact with each supported relational database system (RDBMS) using characters from the OEM (or IBM PC) character set. However, many relational database management systems provide a configuration option that allows the RDBMS Windows client software to work directly with the ANSI character set commonly used in the Windows environment. In this latter case, FOCUS can be configured to interact with the RDBMS client software using the ANSI character set.

This configuration option is particularly important when using FOCUS in the international community where the OEM and ANSI characters sets use different numeric values to represent international characters.

To specify how FOCUS should interact with a specific RDBMS, issue an SQL `sqlsuffix SET` command. You can issue this command from the FOCUS/DLL command prompt, or from a FOCEXEC. The syntax is

```
SQL [target_db] SET ANSITOOEM ON/OFF
```

where:

`target_db` Indicates the relational interface for which the ANSITOOEM parameter is being set. Allowable values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLINF</code>	Informix.

[ON/OFF](#)

OFF is the default, and indicates that FOCUS will interact with the RDBMS using the OEM (or IBM PC) character set.

ON instructs FOCUS to interact with the RDBMS using the ANSI character set.

With this latter setting, FOCUS will automatically convert characters to and from the OEM format (which FOCUS uses internally) to the ANSI format used by RDBMS client software.

Note: If the RDBMS client software is configured to use the ANSI character set, then this parameter should be set to ON. If the RDBMS client software is configured to use the OEM character set, the default setting of OFF should be used.

To determine the current setting, issue the following command from the FOCUS/DLL prompt or from a FOCEXEC:

```
SQL target_db ?
```

This command will show all the existing settings for the specified relational interface, including the ANSI to OEM conversion setting.

When the SQL SET command is issued, the setting is automatically saved in the WFOCUS.INI file and will be the active setting for all subsequent FOCUS sessions. It is not necessary to issue the SQL SET command each time FOCUS is used.

AUTOLOGON (All Interfaces)

FOCUS automatically prompts for logon information when you first access a relational database using the Server Settings tool. (Note that the automatic logon only occurs if the SQL engines server, userid, and password have not been set.)

You can disable this feature using the following SET command

```
SQL target_db SET AUTOLOGON OFF
```

where:

`target_db` Is the target RDBMS. Valid values are:

<code>SQLSYB</code>	Sybase SQL Server.
<code>SQLORA</code>	Oracle.
<code>SQLDBM</code>	IBM DB2/2.
<code>SQLDBC</code>	Teradata DBC/1012.
<code>SQLODBC</code>	Microsoft ODBC.
<code>SQLINF</code>	Informix.
<code>EDA</code>	Enterprise Data Access/SQL.

Interacting With the Interfaces

The following topics explain how FOCUS interacts with the various interfaces, and how you can acquire information about the interaction.

Default DATE Considerations

With this release of FOCUS, the default DATE value the Interface uses for the RDBMS DATE data type is different from that used by earlier versions of FOCUS on other platforms. If you use the FOCUS MAINTAIN or MODIFY facility on other FOCUS platforms to maintain RDBMS tables containing DATE columns, this change may have some impact on your applications if you move them to this release of FOCUS.

The Default DATE Value

The Interface uses the default DATE value in conjunction with RDBMS DATE columns described in a Master File Description as ACTUAL=DATE. Under certain circumstances, if your MAINTAIN or MODIFY procedure does not provide a value for a DATE column, the Interface substitutes a default value.

In prior releases of FOCUS on other platforms, the Interface default value for RDBMS tables was '1901-01-01' (for the sake of convenience, all DATE values are in DB2 ISO format unless otherwise indicated). The FOCUS DBMS default value has always been '1900-12-31'. Also, with the FOCUS DBMS, default DATE values are suppressed in report output; they print as blanks. With the Interface, the old default DATE value was not suppressed; it appeared on reports.

With this release of FOCUS, the default DATE value used by the Interface is identical to that used by the FOCUS DBMS: '1900-12-31'. This value, like the FOCUS DBMS default DATE, does not display on reports; it prints as blanks.

For more information on default dates on other FOCUS platforms, refer to the interface documentation for that environment.

Status Return Variable: &RETCODE

The Dialogue Manager status return variable, &RETCODE, indicates the status of FOCUS query commands. You can use it to test RDBMS return codes. The &RETCODE variable contains the last return code resulting from an executed report request, MODIFY request, or native SQL command (issued with or without Direct SQL Passthru).

In a Dialogue Manager request, you can use an -IF statement to test the &RETCODE value against a specified SQL return code. Then, you can take corrective actions based on the result of the -IF test. An SQL return code of zero (0) indicates a successful execution.

Note: Another useful Dialogue Manager variable, &FOCERRNUM, stores the last FOCUS or Interface (not RDBMS) error number generated by the execution of a procedure (FOCEXEC). See the *Language Reference* manual for information about &FOCERRNUM and other statistical variables.

Standard FOCUS and Interface Differences

In the design of the Interface, every effort has been made to retain compatibility with the FOCUS product. In some situations, however, these goals conflict with the need for pragmatic and efficient use of the relational model as implemented within the RDBMS.

In the following areas, results may be different from those you expect, or some features may not be available:

- When describing an embedded join in a multi-table Master File Description, the FOCUS FIELDTYPE keyword (FIELDTYPE=I) is not required for the cross-referencing field of the SQL table. The Interface ignores the keyword.
- You may not use the logical relations INCLUDES and EXCLUDES with non-FOCUS files in a FOCUS IF or WHERE test.
- FOCUS GROUP fields are not supported.
- The FOCUS CREATE FILE command cannot write over an existing table as it can with a FOCUS database. Instead, you must use the RDBMS DROP TABLE command to drop the table before you can recreate it.
- HOLD FORMAT SQL tables created with the default name HOLD are not dropped at the end of a FOCUS session. To drop the table, issue the RDBMS DROP TABLE command.
- JOIN results (embedded or dynamic) are a function of file descriptions or a JOIN specification and an OPTIMIZATION setting. All variations are discussed in *Advanced Reporting Techniques*.
- Since MODIFY or MAINTAIN does not create a table if it does not exist, you must create the table prior to executing the MODIFY or MAINTAIN.
- With the FOCUS Direct SQL Passthru facility, you can invoke all native SQL statements, including SELECT statements, from within FOCUS. Without Direct SQL Passthru, you can invoke most native SQL statements from within FOCUS, but not those that return data in addition to return codes. SELECT is an example of an SQL statement that returns data and that, therefore, must be executed with Direct SQL Passthru.
- The maximum number of bytes per native SQL request is 4096 for requests issued without Direct SQL Passthru. With Direct SQL Passthru, the maximum is 32764 bytes.

- A COMBINE statement in a MODIFY procedure may link several external (non-FOCUS DBMS) files. However, you cannot use the COMBINE command to link a FOCUS database with an RDBMS table. However, MAINTAIN can be used to link FOCUS and non-FOCUS data.
- The Interface allows you to change the value of a table's primary key with the UPDATE command in a MAINTAIN or MODIFY statement. When modifying a FOCUS database, you cannot change key field values. Also, with the Interface, you can match on any field or combination of fields in a row; you need not include the full primary key.
- MODIFY does not support FOCUS alternate file views and remote segment descriptions.
- When modifying non-FOCUS files, including RDBMS tables, you must code the keywords TRACE or ECHO on the line following the MODIFY FILE statement in order for tracing or echoing to occur.
- FOCUS evaluates a report request using SUM on an alphanumeric field differently for a FOCUS database and a relational table. For a FOCUS database, SUM on an alphanumeric field displays the value for the last record retrieved. For relational tables, SUM on an alphanumeric field is translated to the SQL MAX(field) operator, and displays the field with the highest value.

Maintaining Relational Tables

Topics:

- Types of relational transaction processing.
- The role of the primary key.
- Index considerations.
- Modifying data.
- Referential integrity.
- The COMBINE facility.
- The LOOKUP function.
- The FIND function.
- Issuing SQL commands in MODIFY.
- LOADONLY.
- AUTOCOMMIT ON CRTFORM.

These topics describe how to use the Interface to maintain RDBMS tables. In particular, they identify aspects of the FOCUS file maintenance facilities, MODIFY and MAINTAIN, that are unique for the RDBMS environment. MODIFY and MAINTAIN requests read, add, update, and delete rows in tables. You can modify single tables, sets of tables defined in a multi-table file description, or unrelated sets of tables.

Note: MAINTAIN is not supported in the current release of FOCUS. MODIFY is only supported in batch mode, using FREEFORM and FIXFORM. The Interactive MODIFY CTRFORM facility is not yet available.

MAINTAIN provides a graphical user interface and event-driven processing. In a MAINTAIN procedure, temporary storage areas called stacks collect data, transaction values, and temporary field values. You can use the MAINTAIN Window Painter facility to design Winforms—windows that display stack values, collect transaction values, and invoke triggers. A trigger implements event-driven processing by associating an action (such as performing a specific case in the MAINTAIN procedure) with an event (such as pressing a particular button). MAINTAIN also provides set-based processing through enhanced NEXT, UPDATE, DELETE, and INCLUDE commands.

These topics describe differences between the MODIFY and MAINTAIN facilities when these differences affect Interface processing.

The online *Language Reference* contains a detailed discussion of file maintenance with the FOCUS MODIFY facility. Read the MODIFY chapter carefully before developing MODIFY procedures to use with RDBMS tables.

Note: You can maintain up to 64 tables in a single MODIFY or MAINTAIN procedure. The limit for a MODIFY COMBINE or a MAINTAIN procedure is 16 file descriptions; however, each file description can describe more than one table, for a total of 64 segments per procedure minus one for the artificial root segment created by the COMBINE command. In addition, a MAINTAIN procedure can call other MAINTAIN procedures that reference additional tables.

The following are prerequisites for running MODIFY and MAINTAIN requests:

- You must be using a version of FOCUS that supports these commands.
- Proper RDBMS authorization to perform maintenance operations.

You can update some RDBMS views in accordance with database rules. In general, the RDBMS permits updates to views that are subsets of columns, rows, or both; it does not permit updates to views that perform joins or involve aggregation. For rules regarding the maintenance of RDBMS views, see the *SQL Reference Manual* for the appropriate RDBMS.

- A WRITE keyword value of YES in the table's access file (for INCLUDE, UPDATE, and DELETE operations).

- Existing tables to modify. If you intend to load new tables with MODIFY or MAINTAIN, you must generate them first. Do so with the FOCUS CREATE FILE command, discussed in Describing Multi-Table Relational Structures, or with the SQL CREATE TABLE command through the Direct SQL Passthru facility, discussed in Direct SQL Passthru.

The examples in these topics refer to the EMPINFO, COURSE, PAYINFO, ECOURSE, and EMPPAY file descriptions and access files that are included with your product media.

Types of Relational Transaction Processing

You can process incoming transactions by comparing (or matching on):

- The primary key.
- A non-key field or a subset of primary key fields (for example, two columns of a three-column primary key).
- A superset of the primary key (all key fields plus non-key fields).

In MODIFY, a MATCH on a partial key or on a non-key may retrieve more than one row. MATCH returns only the first row of this answer set.

In MAINTAIN, MATCH always matches on the full primary key and retrieves at most one row. To match on a partial key or non-key in MAINTAIN, use the NEXT command without a prior MATCH. The MAINTAIN implementation of the NEXT command fetches the entire answer set returned by the RDBMS directly into a stack. It also includes three optional phrases:

- The FOR phrase determines how many rows to retrieve.
- The WHERE phrase defines retrieval criteria.
- The INTO phrase names a stack to receive the returned rows.

The Role of the Primary Key

In a table, the primary key is the column or combination of columns whose values uniquely identify a row in a table. Such columns may not contain null data.

A table's file descriptions and access files identify its primary key. See KEYS for information about how to describe primary key columns.

You can modify tables without primary keys as well as those with primary keys. Tables that lack primary keys have KEYS=0 in their access files. Modifying Tables Without Primary Keys explains how to maintain tables without primary keys.

You can implement RDBMS referential integrity by defining primary and foreign keys in SQL CREATE TABLE statements. Defining the primary key to the RDBMS is optional. Most tables have primary keys (and unique indexes created to support them) whether or not the CREATE TABLE statement explicitly identifies them. In these topics, the term *primary key* or *key* refers to those columns that compose each row's unique identifier.

Index Considerations

Indexes enhance the performance of data maintenance routines, especially indexes created on the table's primary key. Without an index, the RDBMS must read the entire table to locate particular rows; with an index, the RDBMS can access rows directly when given search values for the indexed columns. A table can have several associated indexes. Indexes created for performance reasons can be unique or non-unique. To use either RDBMS or FOCUS referential integrity, create indexes on foreign keys.

You can define a unique index on one or more columns in a table. When an index is unique, the concatenated values of the indexed columns in one row cannot be duplicated in any other row. A unique index is generally defined on a primary key. Once you create it, the RDBMS automatically prevents the insertion of duplicate index values. Any attempt to insert a duplicate row generates an error message.

An example of a unique index on a primary key is the employee ID (EMP_ID) in the sample EMPINFO table. Since no two employees can have the same employee number, the value in the EMP_ID column makes each row unique:

EMP_ID	LAST_NAME	FIRST_NAME
111111111	SMITH	JON
123456789	JONES	ROBERT
222222222	GARFIELD	THOMAS
234567890	SMITH	PETER

You cannot add another row with EMP_ID 111111111 to this table.

Modifying Data

With the FOCUS MODIFY and MAINTAIN facilities, you can add new rows to a table, update column values for specific rows, or delete specific rows.

The Interface processes a MODIFY or MAINTAIN transaction with the following steps:

1. FOCUS reads the transaction for incoming data values.
2. The Interface generates the appropriate SQL SELECT statement.
3. The RDBMS either returns an answer set consisting of one or more rows that satisfy the SELECT request, or determines that the row does not exist.
4. After the RDBMS returns the answer set and/or return code, the Interface does one of the following:
 - * Performs the update operation (UPDATE or DELETE) on the returned answer set. With MODIFY, the Interface processes one row at a time; with MAINTAIN, it can either process one row at a time or a set of rows.
 - * Creates the new row (INCLUDE). In MAINTAIN, it may create multiple rows.
5. The RDBMS changes the database appropriately.

In MODIFY, you must use the NEXT statement to process a multi-row answer set one row at a time. Each NEXT statement puts you physically at the next logical row in the answer set. In MAINTAIN, one NEXT command can process a multi-row answer set without a prior MATCH.

The MATCH Statement

In response to a MATCH statement, the RDBMS selects the first row in the table that meets the MATCH criteria. The MATCH statement compares incoming data with one or more field values and then performs actions that depend on whether or not a row with matching field values exists in the table. MATCH processing for multi-table file descriptions is the same as for a multi-segment FOCUS database.

Acceptable actions for MATCH statements fall into eight groups. They are operations that:

- Include, change, or delete rows.
- Control MATCH processing, such as rejecting the current transaction.
- Read incoming data fields.
- Perform computations and validations, or type messages to the terminal.
- Control Case Logic.
- Control multiple-record processing.
- Activate and deactivate fields in MODIFY.
- Permanently store data in the RDBMS.

See the *Language Reference* manual for MATCH syntax in MODIFY.

Interface MATCH Behavior

In MODIFY requests, there are two major differences in the way MATCH statements function for the Interface and for native FOCUS:

- With the Interface, you can change the value of a table's primary key (subject to RDBMS limitations) using the UPDATE statement. When modifying a FOCUS database, you cannot change key field values.
- You can MATCH on *any* field or combination of fields in the row. However, if the full primary key is not included in the MATCH criteria, the Interface may retrieve more than one row as a result of the MATCH.
For example, if the primary key is EMP_ID and the incoming value for MATCH LAST_NAME is SMITH, the answer set contains all rows with last name SMITH.

In MAINTAIN, MATCH functions identically for the Interface and for native FOCUS.

NEXT

In MODIFY and MAINTAIN, the KEYORDER parameter in the access file controls the sort order (by primary key) for NEXT. It determines whether to retrieve primary key values in low (ascending order) or high (descending order) sequence. *Describing Relational Tables* explains how to specify the KEYORDER parameter. The default is to sort by primary key in ascending order.

You can also use NEXT statements with multi-table structures (FOCUS views) to modify or display data. If your MATCH or NEXT specifies a row from a parent table in a multi-table structure, that row becomes the current position in the parent table. A subsequent NEXT on a field in a descendant of that table retrieves the first descendant row in the related table.

In MODIFY, NEXT statements provide a flexible means of processing multi-row answer sets by moving the current position in the answer set from one row to the next. Your choice of MATCH and NEXT statement combinations determines the contents of the answer set.

- NEXT statement without a MATCH statement. The Interface requests the retrieval of all rows in the table sorted by primary key.
- MATCH with the primary key or a superset of primary key columns. The MATCH returns the single row that is the starting point for any subsequent NEXT statements.
- MATCH on a non-key field or a subset of primary key columns. The RDBMS returns a multi-row answer set in which each row satisfies the MATCH criteria.

In MAINTAIN:

- The syntax of the NEXT command includes optional FOR and WHERE phrases that control the number of rows retrieved into a stack.
- NEXT always starts its retrieval at the current database position; it will not retrieve a row it has already passed in its retrieval path unless you use the REPOSITION command to reset the current position.
- The UPDATE, DELETE, and INCLUDE commands also incorporate the optional FOR phrase to process multiple rows from a stack. The system variables FOCERROR and FOCERRORROW inform you whether the entire set of rows processed successfully and, if not, which row caused the problem.

The following topics illustrate different combinations of MATCH and NEXT statements with annotated examples. The requests have been kept simple for purposes of illustration; you can create more sophisticated procedures. Assume KEYORDER=LOW for all of the examples.

NEXT Processing Without MATCH

If you use a NEXT statement without a previous MATCH statement in a MODIFY request, the RDBMS returns an answer set consisting of all rows in the table sorted by the primary key in KEYORDER sequence. Use the ON NEXT statement to view each row in ascending primary key order. In a MAINTAIN request, the FOR and WHERE phrases in the NEXT statement determine the number of rows retrieved, sorted by the primary key in KEYORDER sequence.

In this MODIFY example, the NEXT command retrieves each row in ascending order of employee ID number (EMP_ID):

```
MODIFY FILE EMPINFO
NEXT EMP_ID
  ON NEXT TYPE "EMPLOYEE ID: <D.EMP_ID LAST NAME <D.LAST_NAME "
  ON NONEXT GOTO EXIT
DATA
END
```

The TYPE statements display the following on the screen:

```
EMPLOYEE ID: 071382660 LAST NAME STEVENS
.
.
.
.
EMPLOYEE ID: 222222222 LAST NAME GARFIELD
EMPLOYEE ID: 234567890 LAST NAME SMITH
EMPLOYEE ID: 326179357 LAST NAME BLACKWOOD
EMPLOYEE ID: 451123478 LAST NAME MCKNIGHT
EMPLOYEE ID: 543729165 LAST NAME GREENSPAN
EMPLOYEE ID: 818692173 LAST NAME CROSS
```

The following MAINTAIN procedure retrieves the same answer set into a stack named INSTACK and displays the retrieved values on a Winform named WIN1:

```
MAINTAIN FILE EMPINFO
FOR ALL NEXT EMP_ID INTO INSTACK
WINFORM SHOW WIN1
END
```

NEXT Processing After MATCH on a Full Key or on a Superset

In MODIFY, NEXT processing is identical for either a MATCH on a full primary key or a MATCH on a superset (full key plus a non-key field).

When the initial MATCH is successful, the RDBMS retrieves one row. This establishes the logical "position" in the table. The subsequent NEXT statement causes the RDBMS to retrieve all rows following the matched row in key sequence.

The following is an example of NEXT processing after a MATCH on a full primary key, the EMP_ID field:

```
MODIFY FILE EMPINFO
CRTFORM LINE 1
" PLEASE ENTER VALID EMPLOYEE ID </1"
1. " EMP: <EMP_ID "
2. MATCH EMP_ID
   ON NOMATCH REJECT
3. ON MATCH GOTO GETREST
CASE GETREST
4. NEXT EMP_ID
   ON NEXT CRTFORM LINE 10
   " EMP_ID: <D.EMP_ID LAST_NAME: <D.LAST_NAME "
   ON NEXT GOTO GETREST
5. ON NONEXT GOTO EXIT
ENDCASE
DATA
END
```

The MODIFY procedure processes as follows:

1. The user enters the employee ID for the search, 222222222.
2. The MATCH statement causes the RDBMS to search the table for the entered value. If no such row exists, the transaction is rejected.

3. If the specified value matches a value in the EMP_ID column of the table, the procedure branches to the GETREST case; it contains the NEXT statement.
4. The NEXT statement retrieves the next logical row based on the sequence of EMP_ID. If such a row exists, the procedure displays the values of the EMP_ID and LAST_NAME fields. It continues to display each row in order of the key field, EMP_ID.
5. If there are no more rows, the procedure ends.

The output after executing this MODIFY procedure is:

```
PLEASE ENTER VALID EMPLOYEE ID                (line 1)
EMP: 222222222                                (line 3)
EMP_ID: 234567890   LAST_NAME: SMITH           (line 10)
EMP_ID: 326179357   LAST_NAME: BLACKWOOD       (line 10)
EMP_ID: 451123478   LAST_NAME: MCKNIGHT        (line 10)
EMP_ID: 543729165   LAST_NAME: GREENSPAN       (line 10)
EMP_ID: 818692173   LAST_NAME: CROSS           (line 10)
```

Because of the NEXT statement, all employees after 222222222 display one at a time on the screen. Notice that the rows are retrieved in key sequence.

The following MAINTAIN procedure retrieves the same answer set into a stack named EMPSTACK. Assume that when MAINTAIN displays the Winform called WIN1, the user enters the transaction value, 222222222, into a stack named TRANS and presses a button to invoke the NEXTRECS case:

```
MAINTAIN FILE EMPINFO
WINFORM SHOW WIN1
CASE NEXTRECS
  FOR ALL NEXT EMP_ID INTO EMPSTACK WHERE EMP_ID GT TRANS.EMP_ID
ENDCASE
END
```

NEXT Processing After MATCH on a Non-Key Field or Partial Key

In a MODIFY request processed by the Interface, you do not have to MATCH on the full set of key fields. You can match on a non-key field or partial key. (MAINTAIN always matches on the full primary key, regardless of which fields you specify in the MATCH command.)

When you MATCH on a non-key column or subset of key columns, multiple rows may satisfy the MATCH condition. The MATCH operation retrieves the first row of the answer set, and the NEXT command makes the remaining rows in the answer set available to the program in primary key sequence.

This annotated procedure is the previous MODIFY procedure altered to MATCH on the non-key field LAST_NAME. The NEXT operation retrieves the subsequent rows from the answer set:

```
MODIFY FILE EMPINFO
CRTFORM LINE 1
" PLEASE ENTER A LAST NAME </1 "
1. " LAST_NAME: <LAST_NAME </1"
2. MATCH LAST_NAME
   ON NOMATCH REJECT
3.   ON MATCH CRTFORM LINE 5
   " EMP_ID: <D.EMP_ID LAST_NAME: <D.LAST_NAME "
4.   ON MATCH GOTO GETSAME
   CASE GETSAME
5. NEXT LAST_NAME
   ON NEXT CRTFORM LINE 10
   " EMP_ID: <D.EMP_ID LAST_NAME: <D.LAST_NAME "
   ON NEXT GOTO GETSAME
6.   ON NONEXT GOTO EXIT
ENDCASE
DATA
END
```

The MODIFY procedure processes as follows:

1. The user enters the last name (LAST_NAME) for the search, SMITH.
2. The MATCH statement causes the RDBMS to search the table for all rows with the value SMITH and return them in EMP_ID order. If the value SMITH does not exist, the transaction is rejected.
3. If the incoming value matches a value in the table, the procedure displays the employee ID and last name. (This is the first row of the answer set.)
4. After displaying the row, the procedure goes to the GETSAME case; it uses NEXT to loop through the remaining rows in the answer set.
5. Instead of retrieving the next logical row with a higher key value as in the previous example, the procedure retrieves the next row in the answer set (all rows in the answer set have the last name SMITH). If any exist, they display on the screen in order of the key, EMP_ID.
6. When no more rows exist with the value SMITH, the procedure ends.

The output from this MODIFY procedure follows:

```
PLEASE ENTER A LAST NAME

LAST_NAME  smith

EMP_ID:  111111111  LAST_NAME:  SMITH

EMP_ID:  112847612  LAST_NAME:  SMITH
```

A line displays on the screen for each employee with the last name SMITH. Employee ID 111111111 is the result of the MATCH operation; employee ID 112847612 is the result of the NEXT operation.

The following MAINTAIN procedure retrieves the entire answer set into a stack named EMPSTACK. Assume that when MAINTAIN displays the Winform named WINA, the user enters the transaction value (SMITH) in the first row of a stack named TRANS and presses a button to invoke the NEXTRECS case:

```
MAINTAIN FILE EMPINFO
WINFORM SHOW WINA
CASE NEXTRECS
  FOR ALL NEXT EMP_ID INTO EMPSTACK WHERE LAST_NAME EQ TRANS.LAST_NAME
ENDCASE
END
```

INCLUDE, UPDATE, and DELETE Processing

While MATCH and NEXT operations in MODIFY can operate on primary key or non-key columns and return single or multi-row answer sets, the MODIFY statements INCLUDE, UPDATE, and DELETE must always identify the target rows by their primary key. Therefore, in MODIFY, each update operation affects at most one row. (In MAINTAIN, the FOR phrase in the update statement determines the number of rows affected.)

For example, suppose you want to display all the employees in a department and to increase certain salaries:

```
MODIFY FILE EMPINFO
CRTFORM LINE 1
" PLEASE ENTER A VALID DEPARTMENT </1"
1. " DEPARTMENT: <DEPARTMENT "
2. MATCH DEPARTMENT
   ON NOMATCH REJECT
   ON MATCH CRTFORM LINE 10
3.   "EMP_ID: <D.EMP_ID   SALARY: <T.CURRENT_SALARY> "
4.   ON MATCH UPDATE CURRENT_SALARY
   ON MATCH GOTO GETREST
CASE GETREST
5.   NEXT EMP_ID
   ON NEXT CRTFORM LINE 10
   " EMP_ID: <D.EMP_ID   SALARY: <T.CURRENT_SALARY> "
   ON NEXT UPDATE CURRENT_SALARY
   ON NEXT GOTO GETREST
6.   ON NONEXT GOTO EXIT
ENDCASE
DATA
END
```

The MODIFY procedure processes as follows:

1. The user enters the department (DEPARTMENT) for the search, PRODUCTION.
2. The MATCH statement causes the RDBMS to search the table for the first row with the value PRODUCTION and return them in key sequence (EMP_ID). If none exists, the transaction is rejected.
3. If the supplied value matches a database value, the procedure displays it.
4. The procedure updates the salary field for the first retrieved row using the turnaround value from the CRTFORM. EMP_ID establishes the target row for the update.
5. Each time it executes the NEXT, the procedure retrieves the next row with the same department, PRODUCTION. It displays each one in EMP_ID order. It updates the salary field for each retrieved row with the turnaround value.
6. When no more rows exist for department PRODUCTION, the procedure ends.

In MAINTAIN, you can use stack columns as turnaround values to update a table. The following annotated MAINTAIN request updates the same rows as the preceding MODIFY request:

```
MAINTAIN FILE EMPINFO
1. WINFORM SHOW WIN1
2. CASE MATCHREC
   FOR ALL NEXT EMP_ID INTO EMPSTACK WHERE DEPARTMENT EQ VALSTACK.DEPARTMENT
ENDCASE
3. CASE UPDSAL
   FOR ALL UPDATE CURRENT_SALARY FROM EMPSTACK
ENDCASE
END
```

The MAINTAIN processes as follows:

1. A Winform named WIN1 appears. Assume that it displays an entry field labeled DEPARTMENT (whose source and destination stack is called VALSTACK) and a grid (scrollable table) with columns EMP_ID and CURRENT_SALARY.

2. The user enters a DEPARTMENT value for the search and presses a button to invoke case MATCHREC. Case MATCHREC retrieves the rows that satisfy the NEXT criteria and stores them in a stack named EMPSTACK. The Winform displays the retrieved rows on the grid.
3. The user edits all the necessary salaries directly on the Winform grid and then presses a button to invoke case UPDSAL which updates all salaries.

RDBMS Transaction Control

The Interface supports the Logical Unit of Work (LUW) concept defined by the RDBMS. An LUW consists of one or more FOCUS maintenance actions (UPDATE, INCLUDE, or DELETE) that process as a single unit. The maintenance operations in the LUW can operate on the same or separate tables.

The RDBMS defines a transaction as all actions taken since the application first accessed the RDBMS, last issued a COMMIT WORK, or last issued a ROLLBACK WORK.

In a Logical Unit of Work, the RDBMS either executes all statements completely, or else it executes none of them. If the RDBMS detects no errors in any of the statements in the LUW:

- FOCUS issues a COMMIT WORK statement. The changes indicated by the updates in the transaction are recorded in the table.
- The RDBMS releases locks on the target data.
- Database changes become available for other tasks.

In response to unsuccessful execution of any statement in the transaction, the Interface:

- Issues a ROLLBACK WORK command. Target data returns to its state prior to the unsuccessful transaction. All changes attempted by the statements in the transaction are backed out.
- Does not execute the remaining statements in the transaction.
- Releases locks on the target data.
- Discards partially-accumulated results.

The RDBMS and the Interface provide a level of automatic transaction management but, in many cases, this level of management alone is not sufficient. FOCUS supports explicit control of RDBMS transactions with the commands SQL COMMIT WORK and SQL ROLLBACK WORK in MODIFY, and with the commands COMMIT and ROLLBACK in MAINTAIN.

Note: For MODIFY, SQL COMMIT WORK and SQL ROLLBACK WORK are native SQL commands—commands that the Interface passes directly to the RDBMS for immediate execution. (You can issue SQL commands in MODIFY, but not in MAINTAIN.)—Do not confuse these commands with the FOCUS COMMIT WORK and ROLLBACK WORK commands that apply to FOCUS databases only. The Interface ignores COMMIT WORK and ROLLBACK WORK without the SQL qualifier.

With the default AUTOCOMMIT setting (see Transaction Control Commands), unless you specify SQL COMMIT WORK and/or SQL ROLLBACK WORK in your MODIFY procedure (or COMMIT and/or ROLLBACK in your MAINTAIN procedure), all FOCUS maintenance actions until the END statement constitute a single LUW. If the procedure completes successfully, the Interface automatically transmits a COMMIT WORK command to the RDBMS, and the changes become permanent. If the procedure terminates abnormally, the Interface issues a ROLLBACK WORK to the RDBMS, and the database remains untouched. Since locks are not released until the end of the program, a long MODIFY or MAINTAIN procedure that relies on the default, end-of-program COMMIT WORK can interfere with concurrent access to data. In addition, you may lose all updates in the event of a system failure.

Note:

- MAINTAIN does not respect the SET AUTOCOMMIT ON FIN command that postpones automatic COMMIT processing until the end of the FOCUS session (see Transaction Control Commands).
- You cannot use the FOCUS CHECK facility when updating a table. The Interface ignores the CHECK statement. You must use the SQL COMMIT WORK statement in MODIFY or the COMMIT statement in MAINTAIN.

You can COMMIT after each transaction, or you can use a counter in the procedure to COMMIT after a set number of transactions. This technique can reduce some of the overhead associated with frequent COMMIT processing.

- You can also issue the AUTOCOMMIT ON CRTFORM command in MODIFY CRTFORM procedures.
- The actions described in the following topics do not apply to non-fatal transaction level RDBMS errors, such as an attempt to include a row that violates an RDBMS uniqueness constraint. In such cases, processing continues normally to the next transaction record. Check the FOCERROR variable to determine whether or not to ROLLBACK in the MODIFY or MAINTAIN procedure.

RDBMS Transaction Termination (COMMIT WORK)

The native SQL COMMIT WORK statement signals the successful completion of a transaction at the request of the procedure. Execution of a COMMIT statement makes changes to the tables permanent. The syntax in a MODIFY request is:

```
SQL COMMIT WORK
```

You can issue a COMMIT WORK as an ON MATCH, ON NOMATCH, ON NEXT, or ON NONEXT condition, after an update operation (INCLUDE, UPDATE, DELETE), or in cases of a Case Logic request.

A COMMIT WORK example using Case Logic follows:

```
CASE PROCESS
  CRTFORM
  MATCH field1 ...
    ON MATCH insert, update, delete, ...
  GOTO EXACT
ENDCASE
CASE EXACT
  SQL COMMIT WORK
  GOTO TOP
ENDCASE
```

The PROCESS case handles the MATCH, ON MATCH, ON NOMATCH processing. Then it transfers to CASE EXACT which commits the data, instructing the RDBMS to write the entire Logical Unit of Work to the database.

Note: In MAINTAIN, you must use the MAINTAIN facility's COMMIT command to transmit an SQL COMMIT WORK to the RDBMS.

RDBMS Transaction Termination (ROLLBACK WORK) (Oracle, ODBC, DB2/2 Only)

The native SQL ROLLBACK WORK statement signals the unsuccessful completion of a transaction at the request of the procedure. Execution of a ROLLBACK statement backs out all changes made to the tables since the last COMMIT statement. The syntax in a MODIFY request is:

```
SQL ROLLBACK WORK
```

You can design a MODIFY procedure to issue a ROLLBACK WORK statement if you detect an error. For example, if a FOCUS VALIDATE test finds an inaccurate input value, you may choose to exit the transaction, backing out all changes since the last COMMIT. You can issue ROLLBACK WORK as an ON MATCH, ON NOMATCH, ON NEXT, or ON NONEXT condition, or in cases of a Case Logic request.

The following is an example of the ROLLBACK WORK statement using Case Logic:

```
ON NOMATCH CRTFORM ...
ON NOMATCH VALIDATE ...
  ON INVALID GOTO ROLLCASE
.
.
.
CASE ROLLCASE
  SQL ROLLBACK WORK
  GOTO TOP
ENDCASE
```

Code the ROLLBACK WORK statement before a REJECT statement. FOCUS ignores any action following the rejection of a transaction, except for GOTO or PERFORM.

For example:

```
ON MATCH SQL ROLLBACK WORK
ON MATCH REJECT
```

Note: In MAINTAIN, you must use the MAINTAIN facility's ROLLBACK command to transmit an SQL ROLLBACK WORK to the RDBMS.

The Interface automatically executes an SQL ROLLBACK WORK statement when you exit from a transaction early. For example, if you exit a CRTFORM without specifying some action, the Interface automatically issues a ROLLBACK WORK statement on your behalf.

The RDBMS automatically issues a ROLLBACK WORK statement in case of system failure or when it detects a fatal data error, such as a reference to a column or table that does not exist.

RDBMS Transaction Control Example

Each time an employee's salary changes, the following example updates the salary in the EMPINFO table and posts a historical pay record for the new salary in the related PAYINFO table. To ensure that both updates complete or neither one does, the MODIFY procedure places both actions prior to a COMMIT WORK statement. If the descendant table is not processed, ROLLBACK WORK discards the whole logical transaction.

```
MODIFY FILE EMPPAY
COMPUTE DAT_INC=&YMD;
CRTFORM LINE 1
"/2 <25 MODIFY FOR SALARY CHANGE </2 "
"<20 ENTER THE EMPLOYEE ID <EMP_ID "
MATCH EMP_ID
  ON MATCH CRTFORM LINE 7
  "<D.FIRST_NAME <D.LAST_NAME CURRENT SALARY <T.CURRENT_SALARY> "
  "PLEASE CHANGE THE SALARY "
  ON MATCH UPDATE CURRENT_SALARY
  ON NOMATCH REJECT
MATCH DAT_INC
  ON NOMATCH COMPUTE
    SALARY=CURRENT_SALARY;
  ON NOMATCH INCLUDE
  ON NOMATCH SQL COMMIT WORK
  ON MATCH SQL ROLLBACK WORK
  ON MATCH REJECT
DATA
END
```

DBC/1012 Transaction Control Within MODIFY (Teradata only)

FOCUS supports the concept of a transaction defined by Teradata. A transaction may be defined as one or more FOCUS maintenance actions (UPDATE, INCLUDE, or DELETE, for example) processed as a single unit.

The Teradata DBC/1012 recognizes explicit and implicit transactions.

- An **explicit** transaction consists of one or more actions enclosed by DBC/SQL BEGIN and END TRANSACTION statements. An explicit transaction is treated as a single unit of work (LUW). In an explicit transaction, the actions are mutually-dependent upon each other; if one fails during execution, the transaction or unit fails.
- An **implicit** transaction consists of one action which is not enclosed by DBC/SQL BEGIN and END TRANSACTION statements. An implicit transaction is treated as a single unit of work. It is processed independently of other actions; if it fails, it does not impact the processing of other actions.

For example, an explicit transaction is one which ensures that a withdrawal of \$100 from a bank checking account is deposited into a savings account. To ensure that both actions are performed or aborted, BEGIN and END TRANSACTION statements are specified in a MODIFY procedure which references either a multi-table Master File Description or a COMBINE structure. If the BEGIN and END TRANSACTION statements are omitted and both actions are treated as implicit transactions, any failure may interrupt processing and cause incorrect balances or a loss of credit.

The action(s) within a transaction are either executed successfully or unsuccessfully. A successful execution results in the following:

- Permanent changes are recorded in the table as indicated by the updates within the transaction.
- Locks placed on the target data by the DBC/1012 are released.
- Database changes become available for other users.

Unsuccessful transactions (those which fail to meet specifications) result in the following:

- All changes attempted within the statements of the transaction are backed out and the target data returns to its original state.
- Locks on the target data are released.
- Partially-accumulated results are discarded.

Note: You cannot use the FOCUS CHECK facility when you are updating a table. The CHECK statement is ignored by the Interface.

Transaction Control Syntax

To indicate an explicit transaction or logical unit of work, specify the following syntax as ON MATCH, ON NOMATCH, ON NEXT, or ON NONEXT conditions, or include them in cases of Case Logic requests (The semicolon is optional.):

```
SQLDBC BEGIN TRANSACTION[ ; ]
```

and

```
SQLDBC END TRANSACTION[ ; ]
```

The BEGIN TRANSACTION statement indicates the logical starting point of the LUW. The END TRANSACTION statement indicates the end of the LUW and that processing is completed. The DBC/1012 releases data locks when it encounters the END TRANSACTION statement or the DBC/SQL ROLLBACK WORK statement. See Teradata DBC/1012 Transaction Termination (ROLLBACK WORK) for an explanation of the ROLLBACK WORK statement.

Examples of Transaction Control (Sample Code)

The following examples are partial FOCEXECs which illustrate explicit transactions.

This Case Logic request illustrates one explicit transaction representing one LUW. It treats the operations in CASE PROCESS (MATCH, UPDATE, or INCLUDE) as a single LUW. The DBC/1012 locks the rows until the update is complete and the END TRANSACTION statement is processed. The FOCUS TRACE facility is also specified on a separate line to trace the Case Logic.

```
MODIFY FILE dbcfile
TRACE
.
.
.

CASE STARTIT
  SQLDBC BEGIN TRANSACTION;
  GOTO PROCESS
ENDCASE
CASE PROCESS
  crtform, prompt,...
  MATCH keyfields
    ON MATCH update, delete,...
    ON MATCH GOTO ENDIT
    ON NOMATCH include,...
    ON NOMATCH GOTO ENDIT
ENDCASE
CASE ENDIT
  SQLDBC END TRANSACTION;
  GOTO STARTIT
ENDCASE
```

This next example illustrates multiple update transaction control. It also illustrates the use of the DBC/SQL ROLLBACK WORK command.

```
MATCH tab1_keyfields
  ON MATCH SQLDBC BEGIN TRANSACTION;
  ON MATCH DELETE
  ON MATCH CONTINUE
  ON NOMATCH SQLDBC ROLLBACK WORK;
  ON NOMATCH REJECT
  ON NOMATCH GOTO TOP
MATCH tab2_keyfields
  ON MATCH UPDATE anyfield
  ON MATCH SQLDBC END TRANSACTION;
  ON NOMATCH SQLDBC ROLLBACK WORK;
  ON NOMATCH REJECT
```

Teradata DBC/1012 Transaction Termination (ROLLBACK WORK)

The Teradata DBC/1012 and the Interface provide for a level of automatic transaction management; but, in many cases, this level of management alone is not sufficient. FOCUS supports explicit control of Teradata DBC/1012 transactions with the DBC/SQL ROLLBACK WORK statement.

The native DBC/SQL ROLLBACK WORK statement signals the unsuccessful completion of an explicit transaction at the request of the procedure. Execution of a ROLLBACK statement backs out all changes made to the table(s) since the last BEGIN TRANSACTION statement. Specify the syntax as an ON MATCH, ON NOMATCH, ON NEXT, or ON NONEXT condition, or within cases of a Case Logic request:

```
SQLDBC ROLLBACK WORK [;]
```

You can include a ROLLBACK WORK statement from within a MODIFY procedure in the event that you detect an error. For instance, when a FOCUS VALIDATE test detects an inaccurate input value, you may choose to exit the transaction, backing out all changes since the last BEGIN TRANSACTION statement.

The END TRANSACTION statement is not required when the ROLLBACK WORK statement is specified. The END TRANSACTION statement is implied and, if it is encountered, you may ignore the message:

```
(FOC1500) Too many END TRANSACTION statements
```

Note:

- ROLLBACK WORK is a native DBC/SQL command which is passed directly to the Teradata DBC/1012 for immediate execution. It should not be confused with the FOCUS ROLLBACK command that applies only to FOCUS databases. ROLLBACK WORK without the SQLDBC qualifier is ignored by the Interface.
- A synonym for ROLLBACK WORK is ABORT.
- The ROLLBACK WORK statement is not used with implicit transactions.

This is an example using the ROLLBACK WORK statement in Case Logic:

```
ON NOMATCH CRTFORM ...
ON NOMATCH VALIDATE ...
  ON INVALID GOTO ROLLCASE
  .
  .
  .

CASE ROLLCASE
  SQLDBC ROLLBACK WORK
  GOTO TOP
ENDCASE
```

Code the ROLLBACK WORK statement *before* a REJECT statement. This follows the FOCUS coding convention that ignores any action following the rejection of a transaction, except for GOTO or PERFORM. For example,

```
ON MATCH SQLDBC ROLLBACK WORK
ON MATCH REJECT
```

The Interface automatically executes a DBC/SQL ROLLBACK WORK statement when you exit early from a transaction. For example, if you exit a CRTFORM without specifying some action, the Interface automatically issues a ROLLBACK WORK statement on your behalf.

The Teradata DBC/1012 automatically issues a ROLLBACK WORK statement in cases of system failure or data error. For instance,

- A column or table is referenced which does not exist.
- An attempt to include a row with a key value that is already present in the DBC/1012 unique index.
- An attempt to include or update a column with a value larger than the acceptable datatype defined for the column.

Using the Return Code Variable: FOCERROR

The RDBMS produces a return code that reflects the success or failure of SQL statements; FOCUS stores this return code in the variable FOCERROR. You can test FOCERROR in a MODIFY or MAINTAIN procedure and take the appropriate action if you encounter a non-fatal error. A return code of 0 indicates successful completion of the last SQL command issued (either a "native" SQL command, such as SQL DELETE or SQL COMMIT WORK, or an SQL command generated by MODIFY or MAINTAIN).

Fatal error conditions, such as an inactive RDBMS server, automatically terminate the FOCUS procedure. Non-fatal errors, such as an attempt to include a duplicate value for a unique index, allow the procedure to continue.

You can test the FOCERROR variable in FOCUS VALIDATE or IF statements to determine whether to continue or terminate processing. For example, in DB2/2, if FOCERROR is -803, the INCLUDE or UPDATE operation failed in an attempt to include a value for a unique index. This condition might indicate the need to ROLLBACK the transaction or re-prompt the user for new input values.

For a complete list of error codes, refer to the documentation for your RDBMS.

Using the Interface SET ERRORRUN Command

With SET ERRORRUN ON, MODIFY processing continues even when a serious error occurs, allowing applications to handle their own errors in the event that an RDBMS error is part of the normal application flow. Code this command explicitly in the MODIFY program, preferably in CASE AT START where it executes once.

Note: MAINTAIN does not support the SET ERRORRUN command.

The syntax is

```
CASE AT START
    SQL SET ERRORRUN {OFF}
                   {ON }
ENDCASE
```

where:

OFF Stops MODIFY processing when the RDBMS detects a fatal error (for example, when it cannot find the table name). OFF is the default.

ON Enables MODIFY processing to continue despite fatal errors. Test the value of FOCERROR to determine desired action after an RDBMS call.

When SET ERRORRUN is ON, the MODIFY procedure reports the error but continues execution. The MODIFY code can then test the value of FOCERROR to determine the cause of the error and take appropriate action. Be careful in evaluating the contents of FOCERROR, as failure to respond to a severe error can cause unpredictable errors in subsequent RDBMS or MODIFY processing.

SET ERRORRUN returns to its default setting of OFF at the end of the MODIFY procedure.

Modifying Tables Without Primary Keys

If the RDBMS permits tables with duplicate rows, such tables cannot possibly have a primary key, since no combination of column values can make a given row unique.

The DB2/2 Interface provides a way of maintaining tables with duplicate rows. You describe these tables to FOCUS with KEYS = 0 in the access file, and the Interface invokes special SQL syntax to maintain them. It adds the "WHERE CURRENT OF cursor_name" clause to the SQL UPDATE and DELETE statements it generates. (cursor_name is an Interface-generated value.) This syntax causes the RDBMS to keep track of the rows being updated.

Note:

- MAINTAIN does not support modifying unkeyed tables.
- When the access file describes a table with a primary key to FOCUS, the NEXT statement retrieves rows in a known, logical sequence, by primary key in ascending or descending order.

When KEYS = 0, FOCUS does not request the rows in a particular order. As a result, rows retrieved with the NEXT command arrive unordered. Moreover, the RDBMS determines the retrieval sequence, and it may vary each time the MODIFY procedure executes.
- The FOCURRENT variable is not supported; its value is always 0 for tables without primary keys. The AUTOCOMMIT ON CRTFORM option is not available for unkeyed tables.
- To guarantee the integrity of the row on which the select cursor is positioned, the Interface adds a "FOR UPDATE OF" clause to the SELECT statement. This places an update lock on the selected rows.

Referential Integrity

Since primary and foreign key values establish relationships between separate tables, it is important to maintain these values in a consistent manner throughout the database. The term *referential integrity* defines the type of consistency that should exist between foreign keys and primary keys.

Performance considerations usually make it preferable to have RDBMS indexes on both primary and foreign keys.

The following definitions help explain referential integrity:

- **Primary key** is the column or combination of columns whose value uniquely identifies a row in the table. None of the key columns can contain null values. A table can only have one designated primary key.

The employee ID (EMP_ID) is the primary key in the sample EMPINFO table. The values for the EMP_ID field make each row unique, since no two employees can have the same identification number.
- **Foreign key** is a column or combination of columns in one table whose values are the same as the primary key of another table. The foreign key may be unique or non-unique, but its value must match a primary key value in the other table or be null.

The employee ID in the sample COURSE table is a foreign key. This field is similar to the primary key in the EMPINFO table in that it contains the employee ID of every employee who has taken a course. It contains multiple rows for those employees who have taken more than one course.
- **Referential integrity** describes the synchronization of these key field values:
 - A value must exist as a primary key before it can be entered as a foreign key. For example, a specific employee ID must exist in the EMPINFO table before a course can be added for that employee in the COURSE table (INCLUDE referential integrity).
 - If a primary key is deleted, all references to its value as a foreign key must be deleted, set to null, or changed to reflect an existing primary key value (DELETE referential integrity).

The RDBMS can define and enforce referential integrity rules (constraints).

FOCUS can also provide referential integrity for those tables described in a multi-table file description.

RDBMS Referential Integrity

The RDBMS provides the ability to define relationships between tables by embedding referential integrity constraints in the table definitions. The RDBMS prohibits data changes that violate the rules, and applications using the Interface respect these defined constraints.

Note: Tables you create with the FOCUS CREATE FILE command do not contain primary or foreign key definitions and, therefore, do not participate in RDBMS referential integrity. Of course, you can use the ALTER TABLE command to add primary and foreign key definitions to such tables.

Violation of RDBMS referential integrity rules results in an SQL error code. The Interface posts this return code to the FOCERROR variable; your MODIFY or MAINTAIN procedure can test it.

Referential integrity violations do not terminate MODIFY or MAINTAIN procedures, so you need not use the FOCUS SET ERRORRUN ON command to continue processing.

With RDBMS referential integrity in place, you do not need FOCUS referential integrity to invoke some level of automatic referential integrity support. You may wish to maintain your tables in separate file descriptions and let the RDBMS take care of all referential integrity enforcement.

You may also choose to describe the tables as related (using multi-table file descriptions and access files) and take advantage of FOCUS referential integrity.

If you use both FOCUS referential integrity and RDBMS referential integrity, the RDBMS referential integrity takes precedence in cases of conflict. Make sure you are familiar with the RDBMS referential integrity constraints on the tables involved as well as FOCUS referential integrity behavior. Check with your RDBMS database administrator for specific referential integrity constraints.

FOCUS Referential Integrity

The Interface provides some level of automatic referential integrity for tables described in a multi-table file description.

The following topics describe the rules and techniques for ensuring or inhibiting FOCUS INCLUDE and DELETE Referential Integrity. The examples use the ECOURSE file description, a multi-table description that relates the EMPINFO and COURSE tables.

FOCUS INCLUDE Referential Integrity

FOCUS MODIFY facility syntax provides automatic referential integrity for inserting new rows in a related set of tables. The following rules apply:

- You must describe the related set of tables in one multi-table file description. The multi-table description establishes the relationship (an embedded join) based on the primary and foreign keys in the tables.
- The primary key rows belong to the parent table in the description. The foreign key rows belong to the related table in the description.

With a multi-table file description, you cannot add a related row (foreign key) using the FOCUS MODIFY facility unless the primary key value already exists. Therefore, a MODIFY procedure that inserts rows must MATCH on the parent table before adding a row in a related table.

The following examples demonstrate referential integrity when adding new rows. The scenarios are:

1. Add a course for an employee *only* if data for the employee ID already exists.
2. The employee ID does not exist. Add both a new employee ID and a course.

A simple, annotated FOCUS MODIFY procedure for each scenario follows.

The first example adds course information only if a row already exists for the employee:

```
MODIFY FILE ECOURSE
CRTFORM LINE 2
"ADD COURSE INFORMATION FOR EMPLOYEE </1"
1. "EMPLOYEE ID: <EMP_ID </1 "
   "COURSE NAME: <CNAME          GRADE: <GRADE "
   " YEAR TAKEN: <YR_TAKEN      QUARTER: <QTR  "
2. MATCH EMP_ID
3.   ON MATCH CONTINUE
4.   ON NOMATCH REJECT
5. MATCH CNAME
   ON NOMATCH INCLUDE
   ON MATCH REJECT
DATA
END
```

The MODIFY procedure processes as follows:

1. The user enters the employee ID and information about the course taken. This constitutes the incoming transaction record.
2. The MATCH statement causes the RDBMS to search the table for an existing row with the specified employee ID.
3. If the employee row exists, the MODIFY continues to the next MATCH statement.
4. If no row in the EMPINFO table exists with the specified employee ID, MODIFY rejects this transaction and routes control to the top of the FOCEXEC.
5. MATCH CNAME causes the RDBMS to search the COURSE table for an existing row with the specified course for the employee ID located in Step 2. If no such row exists, the MODIFY adds a row in the COURSE table. If the course row already exists, the MODIFY rejects the transaction as a duplicate.

The second example adds a row to the EMPINFO table for the new employee and adds a course for that employee to the COURSE table. If the employee ID already exists, the procedure adds only the course information to the COURSE table:

```
MODIFY FILE ECOURSE
CRTFORM LINE 1
1.  "ID: <EMP_ID "
2.  MATCH EMP_ID
3.  ON NOMATCH CRTFORM LINE 2
   "      LAST: <LAST_NAME      FIRST: <FIRST_NAME </1 "
   "  HIRE DATE: <HIRE_DATE      DEPT: <DEPARTMENT "
   "      JOB: <CURR_JOBCODE      SALARY: <CURRENT_SALARY </1"
   "  BONUS PLAN: <BONUS_PLAN      ED HRS: <ED_HRS </1"
   "COURSE NAME: <CNAME          YEAR: <YR_TAKEN  QTR: <QTR "
   "      GRADE: <GRADE "
   ON NOMATCH INCLUDE
4.  ON MATCH CRTFORM LINE 9
   "COURSE NAME: <CNAME          YEAR: <YR_TAKEN  QTR: <QTR "
   "      GRADE: <GRADE "
5.  MATCH CNAME
   ON NOMATCH INCLUDE
   ON MATCH REJECT
DATA
END
```

The MODIFY procedure processes as follows:

1. The user enters EMP_ID.
2. The MATCH statement causes the RDBMS to search the EMPINFO table for an existing row for the specified employee ID.
3. If the employee row does not exist, the user enters the data for both the employee and the specified course. The procedure adds a row to each table.
4. If the employee already exists, the user enters only the course data.
5. The MATCH CNAME statement causes the RDBMS to search the COURSE table for the specified course. If this course does not exist for this employee, the procedure adds it. If it does exist, the procedure rejects the transaction.

Notice that the MATCH statement only identifies CNAME. FOCUS automatically equates the value of EMP_ID with the EMP_ID part of the key to COURSE.

FOCUS DELETE Referential Integrity

FOCUS provides automatic referential integrity for deleting rows in a related set of tables. Just as with INCLUDE referential integrity, only tables described in a multi-table file description invoke FOCUS DELETE Referential Integrity.

Note: An attempt to use FOCUS DELETE Referential Integrity in conjunction with tables that have an RDBMS ON DELETE RESTRICT constraint on the child segments produces an error condition; when FOCUS attempts to delete a parent segment, the RDBMS restriction takes precedence and prevents the deletion.

When you delete a parent row (primary key) in a MODIFY or MAINTAIN procedure, FOCUS automatically deletes all related rows (foreign keys) at the same time.

For example, when you delete an employee from the EMPINFO table in the ECOURSE file description, FOCUS also deletes all rows from the COURSE table that represent courses the employee has taken:

```

MODIFY FILE ECOURSE
CRTFORM LINE 2
"DELETE EMPLOYEE AND ALL COURSES </1"
1. "EMPLOYEE ID: <EMP_ID  "
2. MATCH EMP_ID
    ON MATCH COMPUTE DOIT/A1 = 'N' ;
    ON MATCH CRTFORM LINE 6
3. "EMPLOYEE TO BE DELETED: <D.EMP_ID </1"
   "      LAST NAME: <D.LAST_NAME </1"
   "      FIRST NAME: <D.FIRST_NAME </1"
   "      HIRE DATE: <D.HIRE_DATE </1"
   "      DEPARTMENT: <D.DEPARTMENT </1"
   "      JOB CODE: <D.CURR_JOBCODE </2 "
   "IS THIS THE EMPLOYEE YOU WISH TO DELETE? (Y,N): <DOIT "
   ON MATCH IF DOIT EQ 'N' THEN GOTO TOP;
4. ON MATCH DELETE
   ON NOMATCH REJECT
DATA
END

```

The MODIFY procedure processes as follows:

1. The user enters the employee ID.
2. The MATCH statement causes the RDBMS to search the EMPINFO table for an existing row with the specified employee ID.
3. If the row exists, the MODIFY displays information for verification purposes.
4. Once verified, FOCUS deletes the employee and all associated rows in both the EMPINFO and the COURSE tables. When FOCUS deletes a parent, it automatically deletes all associated related instances.

Inhibiting FOCUS Referential Integrity

You may not always want to enforce FOCUS referential integrity. Consider a relationship in which COURSE is the parent table that contains the primary key and EMPINFO is the related table that contains the foreign key. If you delete a course offering, you do not want to delete all employees who have taken the course.

To handle this problem, specify the parameter WRITE=NO in the access file for the related (foreign key) table. This gives you the ability to modify the COURSE table without affecting the data in the EMPINFO table. You can still use the data in the EMPINFO table for browsing or lookup tasks. This technique bypasses FOCUS referential integrity.

Another technique is to COMBINE single tables rather than using a multi-table file description. COMBINE of single tables does not invoke FOCUS referential integrity.

The COMBINE Facility

Some applications require that you use a single input transaction to update several tables in the same MODIFY procedure. If the tables are not defined in the same file description, you can use the COMBINE facility to modify them as if they are one.

In MAINTAIN, you do not issue a COMBINE command to modify unrelated tables. Instead, you reference multiple tables in the MAINTAIN FILE statement. For example:

```
MAINTAIN FILES EMPINFO AND COURSE
```

You can maintain up to 63 tables in a single MODIFY procedure that operates on a COMBINE structure. The COMBINE limit is 16 file descriptions; however, each file description can describe more than one table, for a total of 64 per procedure, minus one for the artificial root segment created by the COMBINE command.

The COMBINE facility links multiple tables and assigns a new name to them so FOCUS can treat the tables as a single structure. Tables in a COMBINE structure can have different SUFFIX attributes, but you cannot combine a FOCUS database with anything except other FOCUS databases.

In MAINTAIN, you can modify FOCUS databases and RDBMS tables in the same procedure.

When you issue a COMBINE command, the COMBINE structure remains in effect for the duration of the FOCUS session or until you enter another COMBINE command. Only one COMBINE structure can exist at a time, so each subsequent COMBINE command replaces the existing structure.

Do not confuse COMBINE with the dynamic JOIN command. You use JOIN to *report* from multiple tables that share at least one common field or for LOOKUP functions. With the COMBINE facility, you can *MODIFY* multiple tables. COMBINE is part of the MODIFY command; only the MODIFY and CHECK FILE commands process COMBINE structures. The FIND function also works in conjunction with COMBINE.

Note that COMBINE considers the component structures to be unrelated. Although RDBMS referential integrity is enforced, FOCUS referential integrity does not apply to a COMBINE of single-table file descriptions. Your procedure should check for and enforce referential integrity, if necessary.

The basic syntax for the COMBINE command is

```

COMBINE FILES  file1  [PREFIX pref1]
                |TAG tag1  | [AND]
                |
                |
                |
                filen  [PREFIX prefn]
                |TAG tagn  | AS asname
    
```

where:

- `file1 - filen` Are the file description names of the tables you want to modify. You can specify up to 16 file descriptions.
- `pref1 - prefn` Are prefix strings for each file; up to four characters. They provide uniqueness for fieldnames. You cannot mix TAG and PREFIX in a COMBINE structure. Refer to the online *Language Reference* for additional information.
- `tag1 - tagn` Are aliases for the table names; up to eight characters. FOCUS uses the tag name as the table name qualifier for fields that refer to that table in the combined structure. You cannot mix TAG and PREFIX in a COMBINE, and you can only use TAG if FIELDNAME is set to NEW or NOTRUNC.
- `AND` Is an optional word to enhance readability.
- `asname` Is the required name of the combined structure to use in MODIFY procedures and CHECK FILE commands.

Once you enter the COMBINE command, you can modify the combined structure.

For example, the EMPINFO table contains employee number, last name, first name, hire date, department code, current jobcode, current salary, number of education hours, and bonus plan information. A second table, PAYINFO, is a historical record of the employee's pay history. It contains the employee number, date of increase, percent of increase, new salary, and jobcode.

Each time a salary changes, both the EMPINFO and PAYINFO tables must reflect the change. Since both tables need to share data entered for employee number, salary and jobcode, this application is appropriate for the COMBINE facility. You can update both tables at the same time without having to define multi-table file descriptions and access files.

The following figures represent the tables as separate entities.

EMPINFO table		PAYINFO table	
01	EMPINFO	01	PAYINFO
	S0		S0
*****		*****	
*EMP_ID	**	*PAYEID	**
*LAST_NAME	**	*DAT_INC	**
*FIRST_NAME	**	*PCT_INC	**
*HIRE_DATE	**	*SALARY	**
*	**	*	**
*****		*****	
*****		*****	
	EMPINFO		PAYINFO

To modify the tables simultaneously, issue the following sequence of commands at the FOCUS command level or in a procedure:

```
COMBINE FILES EMPINFO PAYINFO AS EMPSPAY
MODIFY FILE EMPSPAY
.
.
.
```

In the following picture, generated by the CHECK FILE command, FOCUS defines a new segment, identified as SYSTEM99, to be the root segment of the combined structure. SYSTEM99 acts as the traffic controller for this structure; it is a virtual (artificial) segment. It counts as one segment towards the total of 64 segments allowed in the COMBINE structure.

```
check file empipay pict
NUMBER OF ERRORS=      0
NUMBER OF SEGMENTS=   3 ( REAL=   3 VIRTUAL=   0 )
NUMBER OF FIELDS=    15 INDEXES=   0 FILES=   3
TOTAL LENGTH OF ALL FIELDS=  95
SECTION 01
                STRUCTURE OF SQLDS      FILE EMPINFO ON 07/22/93 AT 09.54.27

                SYSTEM99
01      S0
*****
*          **
*          **
*          **
*          **
*          **
*****
*****
                I
                +-----+
                I          I
                I EMPINFO  I PAYINFO
02      I S0      03      I S0
*****          *****
*EMP_ID      **  *PAYEID      **
*LAST_NAME   **  *DAT_INC     **
*FIRST_NAME  **  *PCT_INC     **
*HIRE_DATE   **  *SALARY      **
*            **  *            **
*****          *****
*****          *****
                EMPINFO      PAYINFO
```

The COMBINE facility makes it easy to modify many files with the same transaction. For additional information regarding the COMBINE facility of MODIFY, refer to the online *Language Reference*.

The LOOKUP Function

The LOOKUP function, used in FOCUS MODIFY procedures, retrieves data values from cross-referenced tables joined dynamically with the JOIN command. The function is valid in both MODIFY COMPUTE and VALIDATE statements.

The syntax for the LOOKUP function is

```
rfield/I1 = LOOKUP(field);
```

where:

rfield Contains the return code (1 or 0) after the LOOKUP function executes.

field Is the name of any field in a cross-referenced table. After the LOOKUP, this fieldname contains the field's value for you to use as needed.

To use this feature most efficiently with an RDBMS, specify a cross-referenced field for which an RDBMS index has been established.

Note:

- The LOOKUP function is not supported between RDBMS tables and FOCUS databases in either direction.
- The extended syntax of the LOOKUP function (parameters GE and LE) is not valid for RDBMS tables. LOOKUP can only retrieve values that match exactly. Refer to the online *Language Reference* for more information.

The FIND Function

The FIND function, used with COMBINE structures in FOCUS MODIFY procedures or with any file in a MAINTAIN procedure, verifies the existence of a value in another file structure. The FIND function sets a temporary field to 1 if the value exists in the other file and to 0 if it does not. FIND does not return any actual data values.

Use FIND only with a table referenced in a COMBINE statement or a MAINTAIN FILE statement. With COMBINE, if the FIND is for a field in a Btrieve file, this field must be the index or alternate index field. For MAINTAIN, the field must be indexed only if it is in a FOCUS database.

The syntax for the FIND function is

```
rfield/I1 = FIND(fieldname AS dbfield IN file);
```

where:

rfield Contains the return code (1 or 0) after the FIND function executes.

fieldname Is the comparison field from one COMBINE table or one table referenced in a MAINTAIN FILE statement.

dbfield For MODIFY, is the field in another COMBINE file structure or an indexed field in a Btrieve file, to use for the value comparison. The AS dbfield clause is optional if rfield and dbfield have the same name.

 For MAINTAIN, is a fieldname from one of the files listed in the MAINTAIN FILE statement, qualified with its filename.

 To use this feature most efficiently with an RDBMS, specify a field for which an RDBMS index has been established.

file In MODIFY, names the table or Btrieve file in which dbfield resides. In MAINTAIN, is ignored.

The FIND function is only supported in MODIFY or MAINTAIN procedures. For more information, consult the online *Language Reference*.

Issuing SQL Commands in MODIFY

With the Interface, you can execute a wide range of native SQL commands in FOCUS. You can issue them in a MODIFY procedure or at the FOCUS command level.

In fact, you can execute all SQL commands that return only SQL return codes—not data—in MODIFY. Since SQL SELECT returns data (rows), you cannot execute it from a MODIFY FOCEXEC. (You can, however, execute SQL SELECT requests at the FOCUS command level using the Direct SQL Passthru facility discussed in Direct SQL Passthru.)

Note: MAINTAIN does not support SQL commands.

To place native SQL commands in a FOCUS MODIFY procedure, prefix them with the environmental qualifier SQL. Do not include a target RDBMS qualifier. If the command exceeds one line, end the first line with a hyphen and continue the command on the next line, prefixing this continued line with the SQL qualifier. Semicolons are optional.

For example, the following section of a MODIFY procedure contains the SQL DELETE, CREATE TABLE, COMMIT WORK, and DROP TABLE statements:

```
CASE AT START
  SQL DELETE FROM PERSONNEL.TEMP1 WHERE ACCT_ID > 1000;
  SQL CREATE TABLE PERSONNEL.TEMP2 -
  SQL      (ACCT_ID INTEGER, AMOUNT DECIMAL (13,2)) -
  SQL      IN PUBLIC.SPACE0;
ENDCASE
.
.
.
NEXT keyfield
  ON NEXT UPDATE anyfield
  ON NEXT SQL COMMIT WORK;
  ON NONEXT SQL DROP TABLE PERSONNEL.TEMP2;
.
.
.
```

The SQL reference manual for the applicable RDBMS contains a comprehensive list of SQL commands.

LOADONLY

The Fastload facility increases the speed of loading data into tables. Use the LOADONLY command exclusively in those MODIFY procedures that insert rows. The syntax is:

```
SQL SET LOADONLY
```

You can use the Fastload feature only with ON NOMATCH INCLUDE operations; other MODIFY operations generate an error message if SQL SET LOADONLY is invoked.

In MODIFY processing without Fastload, the Interface uses an SQL SELECT statement to test the existence of a single row. By examining the SQL return code, the Interface determines whether the row exists and directs MODIFY processing to the appropriate ON MATCH or ON NOMATCH logic.

The Fastload option eliminates this SELECT operation. It loads rows into the table without first evaluating their existence. The RDBMS ensures the uniqueness of stored rows with unique indexes.

AUTOCOMMIT ON CRTFORM

The Interface AUTOCOMMIT ON CRTFORM facility provides application developers with an automated Change Verify Protocol to use as an alternative to the standard RDBMS method of concurrency and integrity in MODIFY CRTFORM procedures.

AUTOCOMMIT ON CRTFORM is not available in MAINTAIN.

Without Change Verify Protocol, the Interface relies solely on the RDBMS to manage the concurrent update and retrieval activities of its applications. Using the SQL COMMIT WORK statement, application developers can define transactions, or Logical Units of Work (LUWs), consisting of one or more database actions. In the LUW, the RDBMS guarantees database integrity; database objects manipulated by the LUW will not change in an unpredictable manner before the termination of the LUW. Furthermore, if any failure occurs in the boundaries of the LUW, the RDBMS will undo, or ROLLBACK, any outstanding updates in the LUW.

The RDBMS uses a locking mechanism to prevent other concurrent transactions from interfering with the LUW. The locking mechanism allocates and isolates database resources for the LUW. However, this approach suffers from at least one basic drawback: terminal I/O locks data for an indeterminate amount of time. The lock remains allocated to the LUW until the user chooses to react to the screen display. Database transaction throughput, in many cases, becomes more a function of RDBMS lock management than of RDBMS transaction processing performance.

AUTOCOMMIT ON CRTFORM automatically invokes the Change Verify Protocol through the following series of steps:

1. Before displaying a CRTFORM, the Interface issues an SQL COMMIT WORK statement to release all locks held on the underlying table. The COMMIT releases all locks for the record displayed on the terminal.
2. If the application requests an update to the displayed record (UPDATE, DELETE, or INCLUDE), the Interface retrieves the row from the table again. The Interface compares the held and newly-retrieved images of the transaction record to determine whether a conflict exists with a transaction from another user. If no conflict exists, FOCUS processes the update as expected. If another user changed the record in the interim, FOCUS rejects the update. The application should test the value of the FOCURRENT variable to redirect the logic flow in the FOCEXEC.
3. Many applications retrieve a record and perform VALIDATE tests on that record. If the record satisfies those tests, the MODIFY branches to a case that matches the record again and (potentially) updates it. In this situation, AUTOCOMMIT ON CRTFORM retrieves and compares the displayed and current versions of the record. The second MATCH subcommand and the update request each sponsor the Change Verify Protocol action. If no conflict exists, FOCUS submits the update as expected.

Note: In this scenario, the MODIFY application itself must be coded to check FOCURRENT; the second MATCH does not automatically perform the check. Any maintenance action issued subsequent to the second MATCH subcommand still performs an automatic check.

The Change Verify Protocol does not have the unit-of-work capabilities of the RDBMS protocol, but it never holds locks on CRTFORM-displayed records. By eliminating locks from an application's run-time path, it can substantially increase rates of transaction throughput and decrease terminal response times for interactive applications.

You can only issue the AUTOCOMMIT ON CRTFORM command in a MODIFY CRTFORM procedure. Place it in CASE AT START, not in the TOP case. You cannot switch AUTOCOMMIT modes in the MODIFY procedure.

The AUTOCOMMIT facility only works on single-record transactions. (For example, MODIFY MATCH and NEXT commands retrieve single records.) It is not designed for set processing with FOCUS multiple-record operations (REPEAT and HOLD, for example). For multiple-record processing, the Change Verify Protocol applies only to the last record.

The syntax is

```
SQL SET AUTOCOMMIT { OFF }  
                   { ON CRTFORM }
```

where:

OFF Is the default; it retains the native RDBMS locking protocol.

ON CRTFORM Invokes the Change Verify Protocol.

To ensure data integrity in conjunction with AUTOCOMMIT ON CRTFORM, you must use an RDBMS Isolation Level of Repeatable Read (RR).

The FOCURRENT Variable

A MODIFY procedure that invokes the Change Verify Protocol can test the value of the FOCURRENT variable to determine whether there is a conflict with another transaction. FOCUS stores a zero in FOCURRENT if there is no conflict, and the transaction is accepted. A non-zero value indicates a conflict; the transaction is rejected, an error message is displayed, and you can redirect the MODIFY activity.

The possible values for FOCURRENT are:

- 0 Transaction accepted
- 1 Invalid, record input will create duplicate
- 2 Invalid, record has been deleted
- 3 Invalid, record has been changed

Your MODIFY application should test FOCURRENT and branch according to its value. For example, a typical procedure using AUTOCOMMIT ON CRTFORM submits a transaction, tests FOCURRENT, and, if FOCURRENT is non-zero, resubmits the transaction.

Note: The FOCURRENT variable is not supported for tables that do not have primary keys; its value is always 0. The AUTOCOMMIT ON CRTFORM option is not available for unkeyed tables.

Rejected Transactions and T. Fields

FOCUS treats transactions rejected because of a conflict as if they failed a VALIDATE test. Transactions that use CRTFORM turnaround fields (T. fields) may require special handling in this case. If, in the logic of your application, you wish to re-retrieve a VALIDATE-rejected record from the database to display its current image, you must issue the MODIFY command DEACTIVATE INVALID. If you do not, the turnaround fields display the rejected values you attempted to enter. Decisions based on these values may be logically incorrect.

Note: MAINTAIN does not support field activation or deactivation.

For example, the following MODIFY request updates the CURRENT_SALARY field and contains a FOCURRENT test:

```
MODIFY FILE EMPINFO
-*
CRTFORM LINE 1
" EMP_ID <EMP_ID "
GOTO EMPLOYEE
-*
CASE EMPLOYEE
  MATCH EMP_ID
  ON NOMATCH REJECT
  ON MATCH CRTFORM LINE 3
    " EMP_ID <D.EMP_ID "
    " SALARY <T.CURRENT_SALARY> "
  ON MATCH UPDATE CURRENT_SALARY
  ON MATCH IF FOCURRENT NE 0 THEN GOTO UNDO; <== Check FOCURRENT and direct
ENDCASE                                     process for appropriate
-*                                         action.
CASE UNDO
  SQL ROLLBACK WORK
  DEACTIVATE INVALID
  GOTO EMPLOYEE
ENDCASE
-*
CASE AT START
  SQL SET AUTOCOMMIT ON CRTFORM <==Releases record after display
ENDCASE
-*
DATA
END
```

In the following example, the CHANGE case (or second MATCH subcommand) applies the Change Verify Protocol action; the example also contains two VALIDATE tests:

```

MODIFY FILE EMPINFO
CRTFORM LINE 1
  " EMP_ID <EMP_ID "
GOTO VALIDATE
_*
CASE VALIDATE
  MATCH EMP_ID
  ON NOMATCH REJECT
  ON MATCH CRTFORM LINE 3
  " EMP_ID <D.EMP_ID "
  " SALARY <T.CURRENT_SALARY> "
  " BONUS <T.BONUS_PLAN> "
  ON MATCH VALIDATE
    SALTEST= (CURRENT_SALARY GE 0) AND (CURRENT_SALARY LE 100000);
    BONTEST = (BONUS_PLAN GE 0) AND (BONUS_PLAN LE 100);
    ON INVALID TYPE "VALUES OUT OF RANGE "
    ON INVALID GOTO UNDO
  ON MATCH GOTO CHANGE
ENDCASE
_*
CASE CHANGE
  MATCH EMP_ID
  ON NOMATCH REJECT
  ON MATCH IF FOCURRENT NE 0 THEN GOTO UNDO; <== Check FOCURRENT and
  ON MATCH UPDATE CURRENT_SALARY BONUS_PLAN direct process for
  ON MATCH IF FOCURRENT NE 0 THEN GOTO UNDO; <== appropriate action.
  ON MATCH SQL COMMIT WORK
ENDCASE
_*
CASE UNDO
  SQL ROLLBACK WORK
  DEACTIVATE INVALID <==FOCUS does not return to old
  COMPUTE EMP_ID =EMP_ID; screen with invalid data,
  GOTO VALIDATE but verifies data, then shows
  refreshed data on the screen.
ENDCASE
_*
CASE AT START
  SQL SET AUTOCOMMIT ON CRTFORM <==Releases record after display
ENDCASE
_*
DATA
END

```

Note: If the application (and not the Interface) sponsors the second MATCH, the MODIFY application itself must check FOCURRENT; the second MATCH does not automatically perform the check. Any maintenance action issued subsequent to the second MATCH subcommand still initiates an automatic check.

Index

&

&FOCERRNUM.....	86
&RECORDS.....	83
&RETCODE	79, 86

?

? query	80
? STAT	58
? STAT query	83

A

ACCEPT.....	14
Access file	9, 10, 14, 15, 17, 18, 24, 25, 26, 27, 28, 38, 39, 75
ACROSS	36
Action	73, 74, 75
Aggregation.....	33, 34, 82
ALIAS	19
ALL.....	46, 47, 49, 50, 51
ANSITOOEM.....	84
Arithmetic expression	34
Array blocking.....	81
AUTOaction.....	73, 74, 76
AUTOCOMMIT	73, 74, 75, 76
AUTODISCONNECT.....	74, 76
AUTOLOGON	85
AVE	33, 34, 36

B

BEGIN SESSION	64
BIND.....	70
Block size	81
BY	36

C

Case Logic	90, 97
Change Verify Protocol	75, 109, 110, 111, 112
Character string expression.....	35
CNT.....	36
Combinations	74, 75, 76, 77

COMBINE.....	105, 106, 107, 108
Commands.....	73
COMMIT.....	37, 73, 74, 76
COMMIT WORK.....	65, 96, 97
Concurrency.....	74
Connection.....	73
CONVERSION.....	60, 61
COUNT.....	33, 34
Creating a multi-table structure.....	21
CRFILE.....	27, 28
Cross-referenced table.....	42, 46, 47
CRTFORM.....	73, 74, 109, 110

D

DATE.....	85, 86
DB2/2.....	75, 76, 80
Declaration.....	14, 15, 17
Default interface session.....	77
Default report.....	61
DEFINE.....	14
DEFINE field.....	34, 35
DELETE.....	58, 83, 95, 104, 105
DESCRIPTION.....	14
Direct SQL Passthru.....	62, 63, 65, 66, 81, 83, 86
Dynamic JOIN.....	25, 42, 43, 45

E

EMPINFO.....	105, 106, 107
END SESSION.....	64
Environment.....	80
Error messages.....	27
ERRORRUN.....	101
ERRORTYPE.....	84
Event.....	73, 74, 75
EXECUTE.....	68, 69
Expression.....	35
Extract file.....	37, 38, 39

F

FALLBACK.....	17
Fatal error.....	101
FETCHSIZE	81
Fields.....	25
FIELDTYPE.....	14
File description	9, 10, 11, 18, 19, 21, 22, 25, 26
File Painter	9
Filename.....	9
FIN	73, 74, 75, 76
FIND.....	108
FOC\$HOLD.....	39
FOCCURRENT.....	75
FOCERROR.....	101
FOCLIST	38, 39
FOCURRENT.....	111
FOCUS.....	103
Foreign key.....	40, 102
FORMAT	13, 37
FST	33
Full key.....	92

G

GROUP	14
-------------	----

H

HOLD FORMAT	37, 39
-------------------	--------

I

IBM DB2/2	6
IF	31, 36
INCLUDE.....	95, 103, 104
Index	89, 90
Inner JOIN.....	48
INSERTSIZE	81
Interface	9, 73, 77, 78, 80, 91
Interface Optimizer	29
IXFLD	24

J

JOIN	21, 24, 25, 40, 41, 42, 43, 44, 45, 46, 53
------------	--

K

KEYFLD	24
KEYORDER	91
KEYS.....	101, 102
KLU segment.....	27

L

LIKE	31
Link.....	25
LOADONLY.....	109
LOCATION	15
Lock.....	74
Logical expression.....	35
Logical unit of work.....	73, 74, 79
Logon	85
Long fieldname.....	17, 18
LOOKUP	108
LST.....	33
LUW	77, 78, 79

M

MAINTAIN	6, 73, 74, 88, 89, 90, 91, 95, 96, 97, 101, 102, 103
MATCH	36, 89, 90, 91, 95
MAX	33, 34, 36
Microsoft ODBC	73, 75, 76, 80
MIN.....	33, 34, 36
MODIFY	6, 73, 74, 88, 89, 90, 91, 95, 96, 97, 101, 105, 106, 107, 108, 109, 110
Multi-field.....	25
Multiple JOIN.....	41
Multi-table structure.....	21, 40

N

Native SQL.....	57, 109
NEXT.....	89, 91, 92, 93, 94
NODATA	52
Non-key.....	93, 94
Non-unique JOIN.....	42, 48, 49

O

OCCURS.....	18, 19, 20
ODBC.....	6

OEM character set	85
Optimization	29, 31, 32, 33, 52
OPTIMIZATION command	29
Optimization logic	31
Oracle.....	6, 71, 73, 75, 76, 80
ORDER	19, 20
Outer JOIN	40, 49, 50, 52

P

Parameterized SQL Passthu.....	64, 65, 66
Partial key.....	93
PASSRECS.....	58, 83
PAYINFO.....	106, 107
PREPARE	62, 63, 67, 68
Primary key	12, 42, 89, 92, 101, 102
PRINT	36
Projection	31
PURGE	69

R

RDBMS	6, 7, 8, 29, 96, 97, 98, 101, 102, 103
READLIMIT	52, 53
Record selection.....	31
RECORDLIMIT.....	52, 53
Referential integrity	102, 103, 104, 105, 106
Relational database.....	6
Relational transaction.....	89
Remote segment description.....	11, 22, 23, 27
Repeating columns.....	18
Reporting.....	36, 53
REPOSITION	91
Retention.....	74
Retrieval.....	46
Return code.....	86, 101
RETYPE	36
ROLLBACK	74
ROLLBACK WORK.....	65, 96, 97, 98
Root segment.....	27

S

Search limit.....	53
Security	8
Segment declaration	10
SEGNAME	22
Sequence	53
Server.....	85
Short path.....	52
Single-field dynamic JOIN	41
Sorting.....	33
SQL	9, 35, 37
SQL Passthru	55, 64
SQL reserved words.....	10
SQL SELECT	58
SQL Server.....	6
SQL Translator	55
SQLENGINE	55, 56, 73, 80
SQLJOIN.....	29
SQLOUT.....	58, 59, 60
Subtree.....	41, 53
SUFFIX	106
Suffix value.....	9
SUM	33, 34, 36
Superset.....	92
Sybase	6, 73, 75, 76, 80
SYSTEM99.....	107

T

TABLEF.....	33, 36, 37
TABlename.....	15, 16
Tables	9, 18, 25, 88, 89
Tables or views	26, 27
Teradata.....	6, 17, 72, 73, 75, 76, 80
TITLE	14
Transaction control.....	73, 96, 98, 99, 100
Transaction processors	6

U

Unique index	12
Unique JOIN.....	41, 42, 44, 47, 48
Unit of work	96
UPDATE.....	58, 83, 95
USAGE.....	13
User-controlled session.....	78

V

VALIDATE	101, 108
Valued expressions	34
View	41, 62, 63
Virtual construct.....	18

W

WHERE.....	31, 36
WRITE.....	33, 34