TIBCO FOCUS®

Relational Data Adapter User's Manual

Release 8207.27.0
March 2021
DN1001155.0321

Contents

Chapter 1

Introduction to Adapters for Relational Data Sources

With the relational adapters, you can use FOCUS to access DB2, Teradata, Oracle, and CA-IDMS/DB (with the SQL option) tables and views. FOCUS is well adapted to relational environments and fully supports the relational data model.

Beginners as well as advanced data processing professionals can take advantage of adapter retrieval and analysis facilities, including easy-to-use, menu-driven query tools and a powerful reporting language that can satisfy virtually any requirement. FOCUS Master and Access Files integrate all facilities and provide transparent access to the underlying relational data source.

In this chapter:

Adapter Capabilities

The relational adapters provide sophisticated data retrieval facilities as well as transaction processing and application development tools for updating and maintaining tables resident on the Relational Database Management System (RDBMS). The FOCUS transaction processors Maintain and MODIFY support interactive maintenance procedures as well as batch maintenance using external input files.

The adapters can manage multi-table, multi-record processing supported by all the constructs of a complete programming language (for example, PERFORM groups, computations, data value validations, error handling, GOTO commands, and IF-THEN-ELSE logic). They permit you to update multiple tables in a single procedure. They respect all RDBMS referential integrity rules while, optionally, maintaining their own referential integrity restrictions that you can specify within the update procedures themselves or in the Master Files.

Additionally, you can embed SQL statements (the language required by the RDBMS) directly into FOCUS application code to take advantage of SQL commands that control updates and locks (COMMIT WORK, ROLLBACK WORK), create RDBMS objects (CREATE INDEX, CREATE TABLE), and define security privileges (GRANT, REVOKE). With the Direct SQL Passthru facility, you can even include SQL SELECT statements in report requests and retrieve answer sets from the RDBMS. A Dialogue Manager variable indicates the success or failure of each SQL command. You can create new tables with SQL commands or with the FOCUS CREATE FILE command.

FOCUS complements RDBMS security features by permitting controlled data access at the user, table, field, or field-value levels. You can use FOCUS security to enhance existing RDBMS security.

The adapters translate FOCUS retrieval and update requests into an equivalent set of SQL statements. They also initiate and monitor communication between themselves and the RDBMS, and provide descriptive error message and recovery support when necessary.

The two distinct components of the adapters are Read and Write:

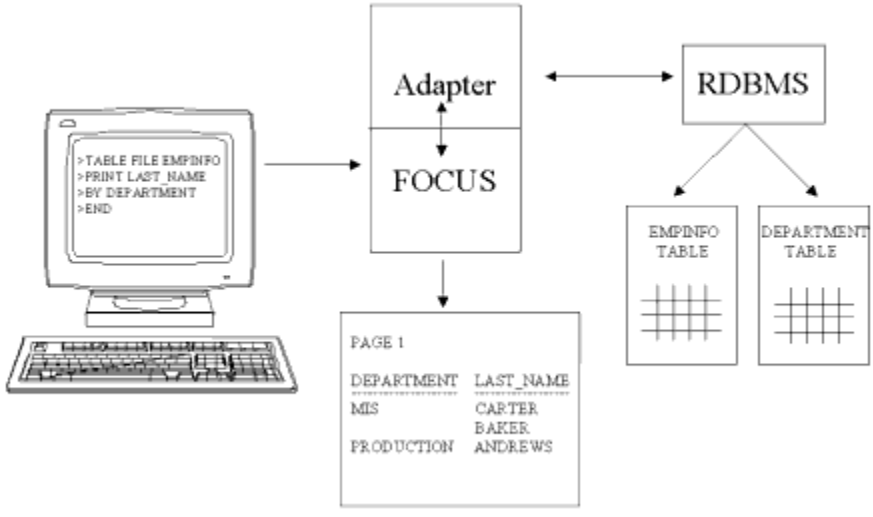
- The Read component translates FOCUS retrieval requests (such as TABLE and GRAPH) into SQL statements that define the request to the RDBMS. When the RDBMS returns an answer set in response to these statements, the adapter passes the answer set to the FOCUS report writer. The report writer can process data from any FOCUS-readable file.
- The Write component generates SQL statements from standard Maintain or MODIFY requests that retrieve and maintain data stored in RDBMS tables.

FOCUS and RDBMS Interaction

FOCUS and the RDBMS interact as follows:

1. Given a TABLE, Maintain, or MODIFY request, the adapter builds SQL statements that define the request in terms the RDBMS can understand.
2. Having received these SQL statements from the adapter, 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 adapter, which in turn passes it to FOCUS for further processing (for example, CRTFORM display and value changes).

The following diagram illustrates FOCUS interacting with a DB2 data source:



The Adapter as an RDBMS Application

Each relational adapter is an RDBMS application. The particular RDBMS determines how its adapter is implemented.

Implementing the Adapter for DB2 as an RDBMS Application

The adapter functions as an RDBMS application that normally executes dynamic SQL. As such, it must be registered with the RDBMS in the same way as any other application.

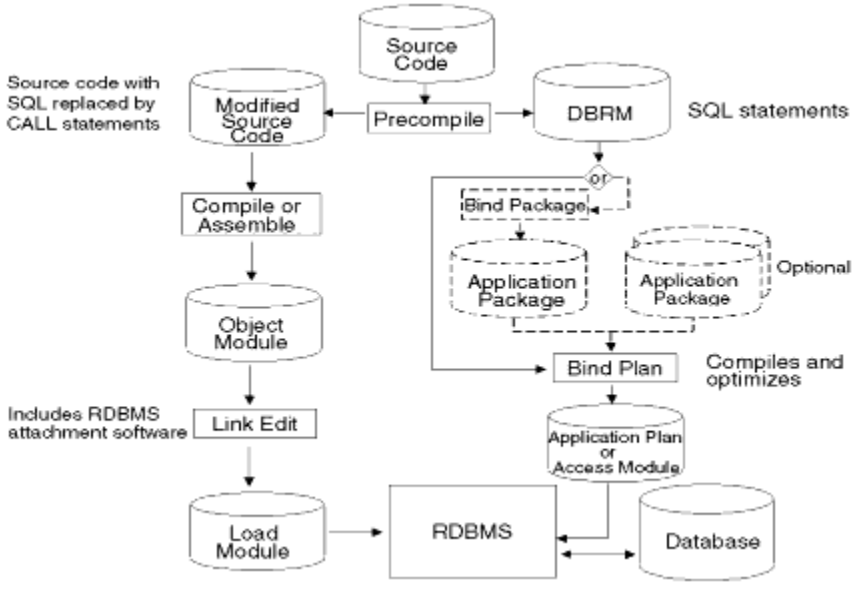
All applications that access DB2 tables on z/OS go through a precompile procedure prior to normal compilation. The precompiler copies all of the SQL statements from the application into a separate module called a Database Request Module (DBRM). It also modifies the original application program by transforming its SQL statements into comments and replacing them with calls to the DBRM. The modified application program can then go through the usual process for creating an executable load module.

The SQL statements from the original application program must also be made executable. The DBRM undergoes its own compilation and optimization process called a *bind*. The bind optimizes the SQL code, performs security checking, and determines the most efficient *access path* to the required data (the access path identifies available resources, such as specific indexes and scan methodologies for traversing the data source). The result of the bind is called an *application plan* or an *application package*.

The bind procedure used with the adapter binds separate DBRMs into small application packages, which you then bind into a special type of application plan that consists of pointers to each application package. The advantage of this is that you can re-bind individual packages without having to re-bind the entire application plan. In a distributed database environment, application packages are essential.

During execution, the program load module works in conjunction with the application plan or *access module* under control of the RDBMS.

The following diagram illustrates the process of preparing an application for execution:



You can use either dynamic or static SQL in MODIFY requests. Dynamic SQL is the default for the adapter. However, even to use static SQL, you must have the adapter installed. The choice of dynamic or static SQL has the following effects on compilation and bind processing:

- With dynamic SQL, the adapter installation includes the precompile, compile, link-edit, and bind (adapter installation produces an application plan or application package for the adapter). The source program for the precompiler is an Assembler program that the adapter calls when it needs to execute SQL statements.
- With static SQL, you issue a FOCUS command to convert your FOCEXEC into a source program for the precompiler. The source program contains the SQL that would be generated by the FOCEXEC.

  At run time, the adapter executes static MODIFY procedures without further processing.

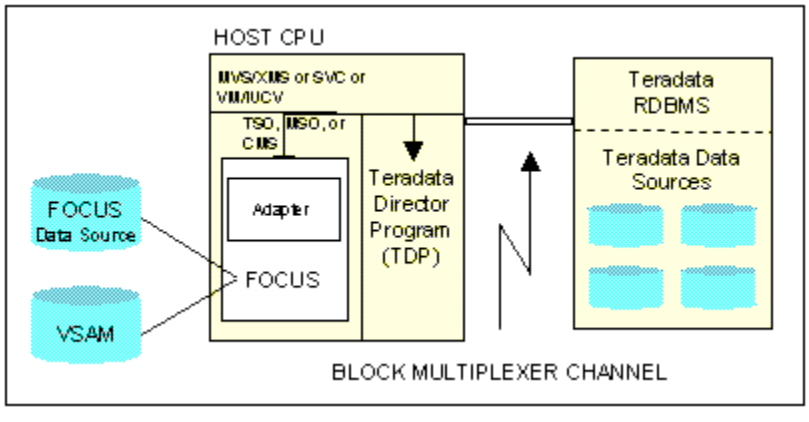
Static SQL (DB2) on page 401 discusses the advantages of static SQL and describes static module creation.

Implementing the Adapter for Teradata as an RDBMS Application

The Adapter for Teradata is an application program that contains dynamic SQL only. No preprocessing is required when installing the adapter. The Adapter for Teradata is delivered as a load library. On z/OS, you must link-edit it with the Teradata libraries.

The Teradata RDBMS and data sources reside on a UNIX box. The adapter communicates with the Teradata RDBMS on UNIX using the Teradata Director Program (TDP) communicating across a Block Multiplexor Channel. The communications link between your address space and the Teradata Director Program (TDP) is implemented with either z/OS XMS (Cross Memory Services) or Supervisor Call (SVC). Your systems programming group determines the type of link during Teradata installation.

The following diagram illustrates this configuration:



Implementing the Adapter for IDMS/SQL as an RDBMS Application

The Adapter for IDMS/SQL is an application program that contains dynamic SQL only. No preprocessing is required when installing the adapter. The Adapter for IDMS/SQL is delivered as a fully executable load module.

Implementing the Adapter for Oracle as an RDBMS Application

The Adapter for Oracle is an application program that contains dynamic SQL only. No preprocessing is required when installing the adapter.

Environment

The relational adapters operate in conjunction with FOCUS to access:

- DB2 under z/OS in TSO and batch.
- Teradata under z/OS in TSO and batch.
- CA-IDMS/DB with the SQL option under z/OS in TSO and batch.
- Oracle under z/OS in TSO and batch.

The adapters are compatible with all currently supported RDBMS releases except where noted. In addition to DB2, Teradata, Oracle, and CA-IDMS/DB tables, FOCUS accesses FOCUS data sources, sequential and delimited data sources, VSAM files, and, with the appropriate adapters installed, many other data sources such as IMS□, CA-IDMS®/DB, CA-Datacom®/DB, ADABAS®, and Model 204®.

Ease of Use

With any of the relational adapters installed, you can use the FOCUS language to request access to RDBMS tables and views. There is no need for specialized subroutines or embedded SQL commands, although with the Direct SQL Passthru facility you can include all SQL commands in report requests.

To make a table intelligible to FOCUS, describe each table or view once in FOCUS terminology. FOCUS stores this description as a Master File and an associated Access File. Once you create Master and Access Files for an RDBMS table, you can refer to the individual columns of the table using the Master File field name or the RDBMS column name. The AUTODB2 and AUTODBC facilities can create Master and Access Files for you automatically. (Issuing requests through the Direct SQL Passthru facility eliminates the need for Master and Access Files but retains all FOCUS reporting capabilities.)

Once you have a Master and Access File for a table, you can use all FOCUS facilities available at your site, such as the Report Writer, the database maintenance language, graphics, and statistics. You do not need to know SQL.

Efficiency

The adapters retrieve from the RDBMS only those rows or columns referenced in the FOCUS report request. Additionally, the adapters may pass to the RDBMS all of the work required to join, sort, and aggregate data. This reduces the volume of RDBMS-to-FOCUS communication, resulting in faster response times for adapter users.

The Adapter for DB2 fully supports the DB2 Distributed Data Facility. It appends a FOR FETCH ONLY clause to SELECT statements passed to DB2. This clause assists the DB2 optimizer in access path selection, and offers significant performance improvement in the distributed database environment. AUTODB2 also supports three-part table names. In addition, the adapter supports the IBM Distributed Relational Database Architecture (DRDA®). For a description of adapter DDF and DRDA support, see *Additional Topics* on page 417.

Security

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

FOCUS also provides its own security facilities. You can use them 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 the specified field's values.

Refer to the *Describing Data* manual for information on DBA security.

Chapter 2

Invoking Relational Adapters

This chapter explains how to invoke the relational adapters. It contains:

- Sample CLISTs, batch JCL, and required DDNAMEs. (See *Getting Started Under z/OS* on page 25.)
- The adapter SQL ? command for displaying current session defaults. (See *Issuing Commands* on page 43.) *Adapter Commands* on page 309 describes environmental commands for changing adapter defaults.

Note: The samples provided in this chapter do not contain site-specific information. Please check with your database administrator for MVS high-level qualifiers, passwords, and proper authorization.

In this chapter:

Getting Started Under z/OS

This section contains information about accessing each adapter under z/OS. Before you can invoke an adapter, it must be installed and operational.

In the z/OS operating environment, TSO controls the adapter and the FOCUS session. The adapter can access RDBMS tables interactively from TSO, with TSO batch processing, or as a z/OS batch job.

You can allocate the FOCUS and adapter load libraries (both from the same version and release of FOCUS) directly in your CLIST, or your site may choose to allocate them to DDNAME STEPLIB in your TSO logon procedure. (The FOCUS installation guide discusses the FOCUS CLIST in greater detail.)

To display the current FOCUS version and release, issue the ? RELEASE query command at the FOCUS prompt.

To run FOCUS interactively, you invoke a CLIST from TSO using standard allocations for DDNAMEs FOCEXEC, ERRORS, and MASTER. Allocate FOCSQL to the library containing Access Files. Allocate FOCLIB to the FOCUS product load library. You must create a new member in the ERRORS concatenation called EDASERVE which will contain the access parameters for the relational adapter you will be accessing.

The allocations in batch are similar to those in the CLIST, with a few changes. The STEPLIB allocation replaces the FOCLIB allocation from the CLIST. You must allocate the file containing executable FOCUS commands to DDNAME SYSIN. Output is written to the file or SYSOUT class allocated to DDNAME SYSPRINT.

The following topics provide CLIST and JCL examples for accessing each adapter. The FOCUS commands are coded in-stream in the samples. However they could have been stored in a data set. The FIN command is required to terminate FOCUS.

For proper JOB card specifications and data set names for your site, consult with your system support staff. For additional information on FOCUS and batch processing, refer to your FOCUS documentation.

Note: The discussions in this section assume that all of your FOCUS and adapter libraries are catalogued under the same z/OS high-level qualifier. The examples throughout this section use the identifier *hlq* to refer to this high-level qualifier.

Accessing DB2 Under z/OS

You must know whether the adapter was installed with the Call Attachment Facility (CAF) or Call Level Interface (CLI). The CLIST requirements are different in each case.

You must also know the four-character subsystem identifier (SSID) of the DB2 subsystem you will access and the plan name assigned to the Adapter for DB2 when it was installed. The defaults for the SSID and plan values are DSN and DSQL respectively, unless your site changed these defaults at installation time.

If the adapter was installed with the Call Attachment Facility (CAF), you can use the SET SSID and SET PLAN environmental commands to specify the SSID and plan name from FOCUS (see *Adapter Commands* on page 309). For CLI installations, you will connect to DB2 using a CONNECT statement.

After you prepare your CLIST or JCL, ask your database administrator whether you require SELECT, INSERT, and/or UPDATE privileges for the tables or views you wish to access.

Reference: **Accessing the Adapter for DB2 Interactively Under CAF Using an EDASERVE Configuration File**

You can use this sample CLIST as a template. Edit it to conform to the standards of your site.

```
ALLOC F(FOCLIB)    DA('hlq.FOCLIB.LOAD') SHR REUSE
ALLOC F(ERRORS)    DA('user.DB2CAF.CFG' -
                      'hlq.ERRORS.DATA') SHR REUSE
ALLOC F(MASTER)    DA('user.MASTER.DATA' -
                      'hlq.MASTER.DATA') SHR REUSE
ALLOC F(FOCSQL)    DA('user.FOCSQL.DATA' -
                      'hlq.FOCSQL.DATA') SHR REUSE
ALLOC F(FOCEXEC)   DA('user.FOCEXEC.DATA' -
                      'hlq.FOCEXEC.DATA') SHR REUSE
CALL 'hlq.FOCLIB.LOAD(FOCUS)'
```

where:

hlq

Is the high-level qualifier for your FOCUS production data sets.

user

Is the high-level qualifier for the private version of a data set.

Note:

- Before executing your CLIST, you must be sure that your DB2 load libraries are available either by allocating them to DDNAME STEPLIB or by having them in the LINKLIST.
- The data set *user*.DB2CAF.CFG in the allocation for DDNAME ERRORS contains the EDASERVE member, which will contain the following attributes:

  ```
  db2_caf = y
  db2_rel = 10
  db2_access = y
  ```

  where db2_rel is the version of DB2 that is being accessed.
- You will need to set your SSID and PLAN either in the procedure or in a profile (FOCPROF) using the following commands:

  ```
  SQL DB2 SET SSID ssid

  SQL DB2 SET PLAN planname
  ```

Execute the CLIST to invoke FOCUS and issue the following commands at the FOCUS prompt, pressing *Enter* after each line.

```
SQL DB2 ?
SQL DB2
SELECT * FROM SYSIBM.SYSDUMMY1;
END
```

If the adapter settings and the request output displayed, the installation and connection were successful. Issue the following command to exit FOCUS.

```
FIN
```

Reference: Accessing the Adapter for DB2 Interactively Under CAF Using the RRSET Module

You can use this sample CLIST as a template. Edit it to conform to the standards of your site.

```
ALLOC F(FOCLIB)    DA('hlq.USE.FOCSQL.LOAD' -
                      'hlq.FOCLIB.LOAD') SHR REUSE
ALLOC F(ERRORS)    DA('hlq.ERRORS.DATA') SHR REUSE
ALLOC F(MASTER)    DA('user.MASTER.DATA' -
                      'hlq.MASTER.DATA') SHR REUSE
ALLOC F(FOCSQL)    DA('user.FOCSQL.DATA' -
                      'hlq.FOCSQL.DATA') SHR REUSE
ALLOC F(FOCEXEC)   DA('user.FOCEXEC.DATA'
                      'hlq.FOCEXEC.DATA') SHR REUSE
CALL 'hlq.FOCLIB.LOAD(FOCUS)'
```

where:

hlq

Is the high-level qualifier for your FOCUS production data sets.

user

Is the high-level qualifier for the private version of a data set.

Note:

- Before executing your CLIST, you must be sure that your DB2 load libraries are available either by allocating them to DDNAME STEPLIB or by having them in the LINKLIST.
- You will need to set your SSID and PLAN either in the procedure or in a profile (FOCPROF) using the following commands:

  ```
  SQL DB2 SET SSID ssid

  SQL DB2 SET PLAN planname
  ```

- The data set *hlq*.USE.FOCSQL.LOAD in the FOCLIB allocation contains the RRSET module.

Execute the CLIST to invoke FOCUS and issue the following commands at the FOCUS prompt, pressing *Enter* after each line.

```
SQL DB2 ?
SQL DB2
SELECT * FROM SYSIBM.SYSDUMMY1;
END
```

If the adapter settings and the request output displayed, the installation and connection were successful. Issue the following command to exit FOCUS.

```
FIN
```

Reference: Accessing the Adapter for DB2 in Batch Under CAF Using an EDASERVE Configuration File

You can use this sample JCL as a template. Edit it to conform to the standards of your site

```
//job card goes here
//*
//FOCDB2  EXEC PGM=FOCUS
//STEPLIB  DD  DSN=hlq.FOCSQL.LOAD,DISP=SHR
//         DD  DSN=hlq.FOCLIB.LOAD,DISP=SHR
//         DD  DSN=DSNn10.SDSNLOAD,DISP=SHR
//ERRORS   DD  DSN=hlq.ERRORS.DATA,DISP=SHR
//         DD  DSN=user.DB2CAF.CFG,DISP=SHR
//MASTER   DD  DSN=user.MASTER.DATA,DISP=SHR
//         DD  DSN=hlq.MASTER.DATA,DISP=SHR
//FOCEXEC  DD  DSN=user.FOCEXEC.DATA,DISP=SHR
//         DD  DSN=hlq.FOCEXEC.DATA,DISP=SHR
//FOCSQL   DD  DSN=user.FOCSQL.DATA,DISP=SHR
//         DD  DSN=hlq.FOCSQL.DATA,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
SQL DB2 ?
SQL DB2
SELECT * FROM SYSIBM.SYSDUMMY1;
END
FIN
/*
```

where:

hlq

Is the high-level qualifier for your FOCUS production data sets.

n

Is the version of DB2 you will use with FOCUS.

user

Is the high-level qualifier for the private version of a data set.

Note:

If the adapter settings and the request output displayed, the installation and connection were successful.

- The data set *user*.DB2CAF.CFG contains the EDASERVE member and must be allocated to DDNAME ERRORS. The EDASERVE member contains the following attributes

```
db2_caf = y
db2_rel = 10
db2_access = y
```

where the release is the version of DB2 that is being accessed.

- You will need to set your SSID and PLAN either in the procedure or in a profile (FOCPROF) using the following commands:

```
SQL DB2 SET SSID ssid

SQL DB2 SET PLAN planname
```

Reference: Accessing the Adapter for DB2 in Batch Under CAF Using the RRSET Module

You can use this sample JCL as a template. Edit it to conform to the standards of your site

```
//job card goes here
//*
//FOCDB2  EXEC PGM=FOCUS
//STEPLIB  DD  DSN=hlq.USE.FOCSQL.LOAD,DISP=SHR
//         DD  DSN=hlq.FOCSQL.LOAD,DISP=SHR
//         DD  DSN=hlq.FOCLIB.LOAD,DISP=SHR
//         DD  DSN=DSNm10.SDSNLOAD,DISP=SH
//ERRORS   DD  DSN=hlq.ERRORS.DATA,DISP=SHR
//MASTER   DD  DSN=user.MASTER.DATA,DISP=SHR
//         DD  DSN=hlq.MASTER.DATA,DISP=SHR
//FOCEXEC  DD  DSN=user.FOCEXEC.DATA,DISP=SHR
//         DD  DSN=hlq.FOCEXEC.DATA,DISP=SHR
//FOCSQL   DD  DSN=user.FOCSQL.DATA,DISP=SHR
//         DD  DSN=hlq.FOCSQL.DATA,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
SQL DB2 ?
SQL DB2
SELECT * FROM SYSIBM.SYSDUMMY1;
END
FIN
/*
```

where:

hlq

Is the high-level qualifier for your FOCUS production data sets.

n

Is the version of DB2 you will use with FOCUS.

user

Is the high-level qualifier for the private version of a data set.

Note:

- The data set *hlq*.USE.FOCSQL.LOAD contains the RRSET module and must be in the allocation for DDNAME STEPLIB.
- You will need to set your SSID and PLAN either in the procedure or in a profile (FOCPROF) using the following commands:

```
SQL DB2 SET SSID ssid

SQL DB2 SET PLAN planname
```

If the adapter settings and the request output displayed, the installation and connection were successful.

Reference: Accessing the Adapter for DB2 Interactively Using CLI

Edit the following CLIST to conform to the standards at your site. This CLIST assumes that the adapter was installed with the Call Level Interface (CLI)

```
ALLOC F(FOCLIB)    DA('hlq.FOCLIB.LOAD') SHR REUSE
ALLOC F(ERRORS)    DA('user.DB2CLI.CFG' -
                      'hlq.ERRORS.DATA') SHR REUSE
ALLOC F(MASTER)    DA('user.MASTER.DATA' -
                      'hlq.MASTER.DATA') SHR REUSE
ALLOC F(FOCSQL)    DA('user.FOCSQL.DATA' -
                      'hlq.FOCSQL.DATA') SHR REUSE
ALLOC F(FOCEXEC)   DA('user.FOCEXEC.DATA' -
                      'hlq.FOCEXEC.DATA') SHR REUSE
CALL 'hlq.FOCLIB.LOAD(FOCUS)'
```

where:

hlq

Is the high-level qualifier under which you installed FOCUS. In this example, *user*.DB2CLI.CFG contain the members EDASERVE and FOCPROF.

user

Is the high-level qualifier for a library allocated under the user ID of a specific user.

Note: *user*.DB2CLI.CFG contains the members EDASERVE and FOCPROF.

- The EDASERVE member will contain the following attributes, where the db2_rel is the release of DB2 being accessed:

```
db2_cli = y
db2_rel = 10
db2_access = y
```

- FOCPROF can contain the connection command or the connection can be done directly in the procedure. The connection string is:

```
-SET &CONSTR='db2locname/userid,password';
ENGINE DB2 SET CONNECTION_ATTRIBUTES <local> &CONSTR
```

Execute the CLIST to invoke FOCUS and issue the following commands at the FOCUS prompt, pressing *Enter* after each line.

```
SQL DB2 ?
SQL DB2
SELECT * FROM SYSIBM.SYSDUMMY1;
END
```

If the adapter settings and the request output displayed, the installation and connection were successful. Issue the following command to exit FOCUS.

Your DB2 database administrator can supply the parameters and the attachment facility chosen at the time the adapter was installed.

Reference: Accessing the Adapter for DB2 in Batch Using CLI

If the FOCUS load libraries are not allocated in your TSO logon procedure, use the following JCL after editing it to conform to the standards at your site. This CLIST assumes that the adapter was installed with the Call Level Interface (CLI)

```
//FOCUSDB2  EXEC PGM=FOCUS,REGION=64M
//STEPLIB   DD DSN=DSNn10.SDSNEXIT,DISP=SHR
//          DD DSN=DSNn10.SDSNLOAD,DISP=SHR
//          DD DSN=DSNn10.SDSNLOD2,DISP=SHR
//          DD DSN=hlq.FOCLIB.LOAD,DISP=SHR
//DB2LOAD   DD DSN=DSNn10.SDSNLOAD,DISP=SHR
//ERRORS    DD DSN=user1.DB2CLI.CFG,DISP=SHR
//          DD DSN=hlq.ERRORS.DATA,DISP=SHR
//FOCLIB    DD DSN=hlq.FOCLIB.LOAD,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSTSPRT  DD SYSOUT=*
//FOCEXEC   DD DSN=hlq.FOCEXEC.DATA,DISP=SHR
//MASTER    DD DSN=hlq.MASTER.DATA,DISP=SHR
//FOCSQL    DD DSN=user1.FOCSQL.DATA,DISP=SHR
//SYSIN DD *
  SQL DB2 ?
  FIN
/*
```

where:

n

Is your version of DB2, for example, 10 (A).

hlq

Is the high-level qualifier under which you installed FOCUS. In this example, *hlq*.FOCUS.DB2CLI.CFG contain the members EDASERVE and FOCPROF.

user1

Is the high-level qualifier for a library allocated under the user ID of a specific user.

Note: *user1*.DB2CLI.CFG contains the members EDASERVE and FOCPROF.

- The EDASERVE member will contain the following attributes, where db2_rel is the release of DB2 being accessed:

  ```
  db2_cli = y
  db2_rel = 10
  db2_access = y
  ```

- FOCPROF can contain the connection command or the connection can be done directly in the procedure. The connection string is:

  ```
  -SET &CONSTR='db2locname/userid,password';
  ENGINE DB2 SET CONNECTION_ATTRIBUTES <local> &CONSTR
  ```

Your DB2 database administrator can supply the parameters and the attachment facility chosen at the time the Ladapter was installed.

Accessing Teradata Under z/OS

The communications link between your address space and the Teradata Director Program (TDP) is implemented with either MVS/XMS (Cross Memory Services) or Supervisor Call (SVC). (Your systems programming group determines the type of link during Teradata installation.)

After you prepare your CLIST or JCL, you must:

- Issue the adapter SET CONNECTION_ATTRIBUTES command at the beginning of your FOCUS session.
- Obtain SELECT and/or UPDATE privileges from your DBC administrator.

For DBC/SQL GRANT information and the SET CONNECTION_ATTRIBUTES command, see *Connection, Authentication, and Security* on page 45.

For information about the adapter SET AUTOCLOSE command, which enables you to control logon interaction with Teradata, see *Controlling Connection Scope* on page 295.

Reference: Interactive Access to Teradata Under z/OS

Create a CLIST by editing the following sample CLIST to conform to the standards at your site.

```
ALLOC F(ERRORS)  DA('hlq.DBCCLI.CFG' +
                    'hlq.ERRORS.DATA') SHR REUSE
ALLOC F(FOCLIB)  DA('hlq.FOCLIB.LOAD') SHR REUSE
ALLOC F(FOCEXEC) DA('hlq.FOCEXEC.DATA') SHR REUSE
ALLOC F(MASTER)  DA('hlq.MASTER.DATA') SHR REUSE
ALLOC F(ACCESS)  DA('user.ACCESS.DATA') SHR REUSE
CALL 'hlq.FOCLIB.LOAD(FOCUS)'
```

where:

hlq

Is the high-level qualifier for your FOCUS production data sets.

user

Is the high-level qualifier for the private version of a data set.

Note: In this example, *hlq*.DBCCLI.CFG contains the members EDASERVE and FOCPROF. The EDASERVE member contains the following attributes:

```
dbc_cli = y
dbc_access = y
```

Reference: Batch Access to Teradata Under z/OS

Create a job by adding a JOB card and editing the following sample JCL to conform to the standards at your site.

```
job card goes here
//SQLMX    EXEC PGM=FOCUS,REGION=0M
//STEPLIB  DD DSN=hlq.FOCLIB.LOAD,DISP=SHR
//ERRORS   DD DSN=hlq.DBCCLI.CFG,DISP=SHR
//         DD DSN=hlq.ERRORS.DATA,DISP=SHR
//FOCLIB   DD DSN=hlq.FOCLIB.LOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//FOCEXEC  DD DSN=hlq.FOCEXEC.DATA,DISP=SHR
//MASTER   DD DSN=hlq.MASTER.DATA,DISP=SHR
//ACCESS   DD DSN=user.ACCESS.DATA,DISP=SHR
//SYSIN DD *
   ENGINE SQLDBC ?
  FIN
/*
```

where:

hlq

Is the high-level qualifier for your FOCUS production data sets.

user

Is the high-level qualifier for the private version of a data set.

Note: In this example, *hlq*.DBCCLI.CFG contains the members EDASERVE and FOCPROF. The EDASERVE member contains the following attributes:

```
dbc_cli = y
dbc_access = y
```

Accessing IDMS SQL Under z/OS

Steps for invoking the adapter vary from site to site. You may be required to complete a series of menus or to execute a CLIST. Since the adapter can operate under IDMS Central Version or in Local Mode, you need to include allocations for the appropriate mode in your CLIST or JCL.

After you prepare your CLIST or JCL, ask your database administrator whether you require SELECT, INSERT, and/or UPDATE privileges for the tables or views you need to access.

If your database administrator (DBA) has turned on IDMS SQL security, the proper privileges must be granted to every adapter user. The DBA should grant you:

- Table privileges that authorize certain operations, such as SELECT or UPDATE, on tables and views.

- The proper CREATE DDL authorization, if you create tables with the FOCUS CREATE FILE command or the Direct SQL Passthru facility.

For more information on security and table privileges, consult *Connection, Authentication, and Security* on page 45. Additional prerequisites and file types are discussed in this chapter.

Example: Sample JCL for Central Version Access to IDMS

You can use the following sample JCL as a template for CV access. Add a valid job card and edit it to conform to the standards and naming conventions at your site:

```
//job card goes here
//IDMSCV   EXEC PGM=FOCUS
//STEPLIB  DD  DSN=highlvl.DBA.LOADLIB,DISP=SHR
//         DD  DSN=highlvl.LOADLIB,DISP=SHR
//         DD  DSN=prefix.IDMS.LOAD,DISP=SHR
//         DD  DSN=prefix.FOCLIB.LOAD,DISP=SHR
//ERRORS   DD  DSN=prefix.ERRNLS.DATA,DISP=SHR
//         DD  DSN=prefix.ERRORS.DATA,DISP=SHR
//MASTER   DD  DSN=prefix.MASTER.DATA,DISP=SHR
//         DD  DSN=userid.MASTER.DATA,DISP=SHR
//FOCIDMS  DD  DSN=prefix.ACCESS.DATA,DISP=SHR
//FOCEXEC  DD  DSN=prefix.FOCEXEC.DATA,DISP=SHR
//         DD  DSN=userid.FOCEXEC.DATA,DISP=SHR
//SYSCTL   DD  DSN=highlvl.SYSCTL,DISP=SHR
//SYSIDMS  DD  DSN=highlvl.SYSIDMS,DISP=SHR
//SYSIN    DD   *
? REL
TABLE FILE EMPFILE
PRINT EMP_NAME_0415
END
FIN
```

where:

highlvl

Is the high-level qualifier for Computer Associates supplied data sets.

prefix

Is the high-level qualifier for FOCUS production data sets.

userid

Is the high-level qualifier for a private data set belonging to the user.

Example: Sample CLIST for Central Version Access to IDMS

You can use the following sample CLIST as a template for CV access. Edit it to conform to the standards and naming conventions at your site:

```
ALLOC DD(STEPLIB) DA('highlvl.DBA.LOADLIB'-
                     'highlvl.LOADLIB') SHR REUSE
ALLOC F(FOCEXEC) DA('prefix.FOCEXEC.DATA' -
                    'userid.FOCEXEC.DATA') SHR REUSE
ALLOC F(MASTER)  DA('prefix.MASTER.DATA' -
                    'userid.MASTER.DATA') SHR REUSE
ALLOC F(FOCIDMS) DA('prefix.ACCESS.DATA') SHR REUSE
ALLOC F(USERLIB) DA('prefix.IDMS.LOAD') SHR REUSE
ALLOC F(FOCLIB)  DA('prefix.FOCLIB.LOAD') SHR REUSE
ALLOC F(ERRORS)  DA('prefix.ERRNLS.DATA' -
                    'prefix.ERRORS.DATA') SHR REUSE
ALLOC F(SYSCTL)  DA('highlvl.SYSCTL') SHR REUSE
ALLOC F(SYSIDMS) DA('highlvl.SYSIDMS') SHR REUSE
CALL 'prefix.FOCLIB.LOAD(FOCUS)'
```

where:

highlvl

Is the high-level qualifier for Computer Associates supplied data sets.

prefix

Is the high-level qualifier for FOCUS production data sets.

userid

Is the high-level qualifier for a private data set belonging to the user.

Local Mode Access

The user must allocate all IDMS database files. These files must be allocated to the ddnames that are assigned in the IDMS/DB Schema.

All journal file allocations must be made available along with the default local mode journal, SYSJRNL, assigned to DD DUMMY.

In some cases, the libraries containing the subschema, DMCL, and IDMSINTB load modules may not be authorized. If STEPLIB cannot be used for unauthorized IDMS libraries, in a local mode job you can allocate these unauthorized modules to ddname CDMSLIB. With CDMSLIB allocated, IDMS will search the CDMSLIB before STEPLIB to obtain all IDMS/DB specific load modules.

Both Central Version and Local Mode

The IDMS load modules IDMS and IDMSINTB must be made available at run-time, allocated to the ddname STEPLIB.

When running a CLIST, the STEPLIB ddname is not valid. These members must be allocated to either the link list or in the TSO logon procedure. Contact your Systems Programmer to add these members.

The member names of the FOCUS Master Files and Access Files to read the subschema must be identically named and made available at run time.

SYSIDMS can be allocated to identify the DMCL for both CV and LOCAL modes.

Accessing Oracle Under z/OS

You must indicate which Oracle subsystem is to be accessed.

Note: All Oracle tools and programs that use the Oracle Pro or High Level Interfaces (such as the Adapter for Oracle) require the services of the SQL Storage Anchor Module (SQLANKOR). This module resides in a library member for which the Oracle-recommended name is:

```
'ORACLE.V10203.CMDLOAD(CMDLOAD)' (for Oracle Version 10).
```

This module is generally placed in a STEPLIB, JOBLIB or system linklist library. It must be present at run time.

Any application that accesses the Oracle RDBMS must provide a means of connecting to the Oracle subsystem (or kernel).

Procedure: How to Connect to the Oracle Subsystem

You can set the Oracle SID by using the Oracle SET SSID command. You can issue this command in the FOCPROF profile. Alternatively, if the SID is at most four characters, you can specify it in your CLIST or JCL.

To set the SID using the SET SSID command, issue the following:

```
ENGINE SQLORA SET SID sid
```

where:

sid

Is the Oracle Subsystem ID.

This should be set before setting the USER or CONNECTION parameters and can be placed in the FOCPROF member.

If the sid is at most four characters, you can specify it in either your CLIST or JCL, instead of in FOCPROF, using the following DD allocation:

```
//ORA@sid DD DUMMY
```

where:

sid

Is the Oracle Subsystem ID.

In your CLIST, you also need to add the following allocation:

```
ALLOC F(ORA$LIB)  DA('ORACLE.V10203.MESG') SHR REUSE
```

Where ORACLE.V10203.MESG is your installation Oracle MESG library.

The TSO session also needs to have the following data sets available to it in the STEPLIB allocation, either explicitly or implicitly, using the naming conventions at your site:

```
ORACLE.V10203.CMDLOAD
ORACLE.V10203.MESG
```

If you are running in batch, the STEPLIB allocation in the JCL should have the following as part of the concatenation:

```
//          DD DSN=ORACLE.V10203.CMDLOAD,DISP=SHR
//          DD DSN=ORACLE.V10203.MESG,DISP=SHR
```

The job also needs the ORA$LIB allocation:

```
//ORA$LIB  DD DSN=ORACLE.V10203.MESG,DISP=SHR
```

Reference: Creating a Batch Job to Invoke FOCUS

Create a job by adding a JOB card and editing the following sample JCL to conform to the standards at your site.

```
job card goes here
//SQLMX    EXEC PGM=FOCUS,REGION=0M
//STEPLIB  DD DSN=hlq.FOCLIB.LOAD,DISP=SHR
//ERRORS   DD DSN=hlq.ORACLI.CFG,DISP=SHR
//         DD DSN=hlq.ERRORS.DATA,DISP=SHR
//ORA$LIB  DD DSN=ORACLE.V10203.MESG,DISP=SHR
//FOCLIB   DD DSN=hlq.FOCLIB.LOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//FOCEXEC  DD DSN=hlq.FOCEXEC.DATA,DISP=SHR
//MASTER   DD DSN=hlq.MASTER.DATA,DISP=SHR
//ACCESS   DD DSN=user.ACCESS.DATA,DISP=SHR
//SYSIN    DD *
  SQL SQLORA
   SELECT TABLE_NAME, OWNER, TABLESPACE_NAME FROM ALL_TABLES
   WHERE ROWNUM <= 10
  END
 FIN
/*
```

where:

hlq

Is the high-level qualifier for your FOCUS production data sets.

user

Is the high-level qualifier for the private version of a data set.

ORACLE.V10203.MESG

Is your installation Oracle MESG library.

Note: If the Oracle sid is at most four characters, you can specify it in your JCL, instead of in FOCPROF. For information, see *How to Connect to the Oracle Subsystem* on page 38.

In this example, *hlq*.ORACLI.CFG has the member EDASERVE. EDASERVE contains the following attributes:

```
ora_oci = y
ora_rel = 10
ora_access = y
```

If the job produces a list of tables, owners, and tablespaces, the adapter installation is verified.

Note: The JESLOG may produce a message similar to the following, which can be ignored:

```
14.11.28 JOB23212  $HASP708 JOBNAME SYSIN    OPEN FAILED
   959             RC=03 DATA SET ALREADY OPENED
   959             DSNAME=USERID.JOBNAME.JOBjobid.D0000101.?
```

where:

`USERID`

Is the USERID of the running JOB.

`JOBNAME`

Is the name of the JOB from the JOB card.

`jobid`

Is the system-assigned job number of the job.

`D0000101`

Is the system-generated identifier to keep the DSNAME unique.

Reference: Creating a CLIST to Invoke FOCUS Interactively

Create a CLIST by editing the following sample CLIST to conform to the standards at your site.

```
ALLOC F(ERRORS)  DA('hlq.ORACLI.CFG' +
                    'hlq.ERRORS.DATA') SHR REUSE
ALLOC F(ORA$LIB) DA('ORACLE.V10203.MESG') SHR REUSE
ALLOC F(FOCLIB)  DA('hlq.FOCLIB.LOAD') SHR REUSE
ALLOC F(FOCEXEC) DA('hlq.FOCEXEC.DATA') SHR REUSE
ALLOC F(MASTER)  DA('hlq.MASTER.DATA') SHR REUSE
ALLOC F(ACCESS)  DA('user.ACCESS.DATA') SHR REUSE
CALL 'hlq.FOCLIB.LOAD(FOCUS)'
```

where:

hlq

Is the high-level qualifier for your FOCUS production data sets.

user

Is the high-level qualifier for the private version of a data set.

`ORACLE.V10203.MESG`

Is your installation Oracle MESG library.

Note: If the Oracle sid is at most four characters, you can specify it in your CLIST, instead of in FOCPROF. For information, see *How to Connect to the Oracle Subsystem* on page 38.

In this example, *hlq*.ORACLI.CFG has the member EDASERVE. EDASERVE contains the following attributes:

```
ora_oci = y
ora_rel = 10
ora_access = y
```

Use your CLIST to invoke FOCUS, and issue the following command at the FOCUS prompt.

```
SQL SQLORA
SELECT TABLE_NAME, OWNER, TABLESPACE_NAME FROM ALL_TABLES
WHERE ROWNUM <= 10
END
```

If the command produces a list of tables, owners, and tablespaces, the adapter installation is verified.

Issue the following command to exit FOCUS.

```
FIN
```

Additional Prerequisites: File Descriptions

The adapter requires a Master and Access File for each RDBMS table referenced by FOCUS. In z/OS, file descriptions and FOCEXECs are stored as members of partitioned data sets (PDSs). The partitioned data sets are allocated to the following DDNAMEs:

DDNAME	Contents (PDS Members)
MASTER	Master Files.
FOCSQL or ACCESS	Access Files.
FOCEXEC	Stored procedures.

Execute the AUTODB2 FOCEXEC supplied with the Adapter for **DB2** or the AUTODBC CLIST supplied with the Adapter for **Teradata**, to automatically create Master and Access Files for existing RDBMS tables. You can customize the resulting descriptions with a text editor. *Automated Procedures* on page 113, describes step-by-step instructions for the AUTO facilities.

You can create new table definitions in the RDBMS from the FOCUS environment by issuing either the FOCUS CREATE FILE command (after creating a Master File and an Access File) or the SQL CREATE TABLE command. See *Automated Procedures* on page 113, for the FOCUS CREATE FILE command.

Issuing Commands

Direct SQL Passthru is a facility for passing native SQL commands directly to the RDBMS without intervention by FOCUS, and for issuing environmental commands that display or change adapter default settings. For a detailed discussion of the Direct SQL Passthru facility, consult *Direct SQL Passthru* on page 265.

Adapter Environmental Commands

The adapter provides environmental commands that display or change adapter default settings for the duration of the FOCUS session. You can issue these commands from the FOCUS command line or include them in a FOCEXEC. To display current adapter settings during a FOCUS session, issue the following command:

```
{ENGINE|SQL} sqlengine ?
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2, SQLDS, SQLDBC, SQLIDMS, or SQLORA.

Example: **Displaying Adapter for DB2 Settings**

To display the current adapter settings for a DB2 session within FOCUS, issue the following adapter query command:

```
> engine db2 ?
(FOC1440) CURRENT SQL INTERFACE SETTINGS ARE :
(FOC1442) CALL ATTACH FACILITY IS              -  : ON
(FOC1447) SSID FOR CALL ATTACH IS              -  : DBAA
(FOC1448) ACTIVE PLAN FOR CALL ATTACH IS       -  : USERCAF
(FOC1459) USER SET PLAN FOR CALL ATTACH IS     -  : USERCAF
(FOC1460) INSTALLATION DEFAULT PLAN     IS     -  : M727703B
(FOC1503) SQL STATIC OPTION IS                 -  : OFF
(FOC1444) AUTOCLOSE OPTION IS                  -  : ON FIN
(FOC1496) AUTODISCONNECT OPTION IS             -  : ON FIN
(FOC1499) AUTOCOMMIT OPTION IS                 -  : ON COMMAND
(FOC1449) CURRENT SQLID IS                     -  : SYSTEM DEFAULT
(FOC1424) ISOLATION LEVEL FOR DB2 TABLE INTERFACE IS  :
(FOC1491) FETCH BUFFERING FACTOR               -  : 100
(FOC1441) WRITE FUNCTIONALITY IS               -  : ON
(FOC1445) OPTIMIZATION OPTION IS               -  : ON
(FOC1763) IF-THEN-ELSE OPTIMIZATION IS         - : ON
(FOC1484) SQL ERROR MESSAGE TYPE IS            -  : DBMS
(FOC1497) SQL EXPLAIN OPTION IS                -  : OFF
(FOC1552) INTERFACE DEFAULT DATE TYPE          -  : NEW
(FOC1446) DEFAULT DBSPACE IS                   -  : DBUSER01.FOCUS
```

See *Adapter Commands* on page 309 for an explanation of the adapter environmental commands.

Chapter 3

Connection, Authentication, and Security

This chapter describes attributes and settings needed for connection to and authentication by the RDBMS.

In any computer system, it is important to secure data from unauthorized access. Both the RDBMS and FOCUS provide security mechanisms to ensure that users access only those objects for which they have authorization.

In this chapter:

SQL GRANT and REVOKE

The SQL GRANT and REVOKE commands control access to tables and views within the RDBMS environment. Without proper authorization, users are denied access to RDBMS tables. Only the RDBMS database administrator and the creator of the table or view implicitly hold these privileges.

The SQL GRANT command distributes privileges to others. The SELECT privilege authorizes the user to generate reports from a table. The INSERT, UPDATE, and DELETE privileges each provide specific capabilities for changing the values within a table. Contact your RDBMS database administrator for more details on RDBMS security.

FOCUS honors the security established by the SQL GRANT mechanism. For example, without SELECT access for the specified table, FOCUS cannot generate a requested report.

DB2 Security

This section examines the DB2 SET CURRENT SQLID command.

DB2 CURRENT SQLID (z/OS)

The DB2 RDBMS on z/OS accepts two types of ID, the primary authorization ID and one or more optional secondary authorization IDs. It also recognizes the CURRENT SQLID setting.

Any interactive user or batch program that accesses a DB2 subsystem is identified by a primary authorization ID. A security system such as RACF® normally provides the ID to DB2. During the process of connecting to DB2, the primary authorization ID may be associated with one or more secondary authorization IDs (usually RACF groups). Each site controls whether it uses secondary authorization IDs.

The DB2 database administrator (DBA) may grant privileges to a secondary authorization ID that are not granted to the primary ID. Thus, secondary authorization IDs provide the means for granting the same privileges to a group of users. (The DBA associates individual primary IDs with a secondary ID and grants the privileges to the secondary ID.)

The DB2 CURRENT SQLID may be the primary authorization ID or any associated secondary authorization ID. At the beginning of the FOCUS session, the CURRENT SQLID is the primary authorization ID.

You can reset the CURRENT SQLID using the following adapter command

```
ENGINE [DB2] SET CURRENT SQLID  = 'sqlid'
```

where:

`DB2`

Is required if you did not previously issue the SET SQLENGINE command for DB2 (see *Direct SQL Passthru* on page 265).

`sqlid`

Is the desired primary or secondary authorization ID, enclosed in single quotation marks. All DB2 security rules are respected.

Unless you issue the SET OWNERID command described in *Adapter Commands* on page 309, the CURRENT SQLID is the implicit owner for unqualified table names and the default owner ID for DB2 objects, such as tables or indices, created with dynamic SQL statements. (For example, the FOCUS CREATE FILE command issues dynamic SQL statements.) The CURRENT SQLID is also the sole authorization ID for GRANT and REVOKE statements. It must have all the privileges needed to create objects and must have GRANT and REVOKE privileges.

Other types of requests, such as FOCUS TABLE (SQL SELECT) and MODIFY (SQL SELECT, INSERT, UPDATE, or DELETE) requests, automatically search for the necessary authorization using the combined privileges of the primary authorization ID and all of its associated secondary authorization IDs, regardless of the DB2 CURRENT SQLID setting.

The CURRENT SQLID setting remains in effect until the communication thread to DB2 is disconnected, when it reverts to the primary authorization ID.

Teradata Login Security

The adapter SET CONNECTION_ATTRIBUTES command enables users to identify themselves to the Teradata RDBMS. When adapter users issue the command at the beginning of a FOCUS session, Teradata verifies their authorization to access tables and views.

Note: DBCLOGON is a synonym for CONNECTION_ATTRIBUTES supported for compatibility with earlier releases of the adapter.

The SET CONNECTION_ATTRIBUTES command must be issued before any DBC/SQL commands or FOCUS requests that require Teradata services. Internally, the adapter stores the Teradata logon ID and password in the virtual storage space for the user. The DBC authorization process is deferred until a subsequent command for Teradata services is executed. The adapter initiates the logon immediately prior to executing a request for Teradata services by the user.

The SET CONNECTION_ATTRIBUTES command may be issued from the FOCUS command level or be included in any FOCUS-supported profile. The syntax is

```
ENGINE [SQLDBC] SET CONNECTION_ATTRIBUTES connection_name/userid,passwd
[;]
```

where:

`SQLDBC`

Is required if you did not previously issue the SET SQLENGINE command for Teradata (see *Direct SQL Passthru* on page 265).

`connection_name`

Is the Teradata Director Program ID (TDP ID). Valid values are site-specific and release-dependent.

Note: Specifying a TDP ID may not be necessary at your site. Contact your database administrator for the requirements at your site.

`userid`

Is your Teradata user ID, up to 30 characters long.

`passwd`

Is the associated Teradata password, up to 30 characters long.

Note: If there is no password for the connection, the userid must still be followed by a comma.

If the SET CONNECTION_ATTRIBUTES command is rejected, a DBC error message is generated, and the user should resubmit the command with correct information.

For additional security, a Dialogue Manager procedure can prompt the user for the Teradata password using -CRTFORM or -PROMPT commands, and store the password in a variable. The -CRTFORM command also provides a non-display option. For information about Dialogue Manager, see the *Developing Applications* manual.

Oracle Connection Attributes

The SET CONNECTION_ATTRIBUTES command allows you to declare a connection to one Oracle database server and to supply authentication attributes necessary to connect to the server.

You can declare connections to more than one Oracle database server by issuing multiple SET CONNECTION_ATTRIBUTES commands. The actual connection takes place when the first request referencing that connection is issued. You can issue SET CONNECTION_ATTRIBUTES commands in a FOCEXEC, at the FOCUS command prompt, or in a FOCUS-supported profile. The profile can be encrypted.

If you issue multiple SET CONNECTION_ATTRIBUTES commands:

- The first SET CONNECTION_ATTRIBUTES command sets the default Oracle database server to be used.
- If more than one SET CONNECTION_ATTRIBUTES command declares the same Oracle database server, the authentication information is taken from the last SET CONNECTION_ATTRIBUTES command.

Connecting to an Oracle Database Server

The adapter supports connections to:

- Local Oracle database servers.
- Remote Oracle database servers. To connect to a remote Oracle database server, the Oracle tnsnames file on the source machine must contain an entry pointing to the target machine, and the listening process must be running on the target machine.

Once you are connected to an Oracle database server, that server may define Oracle DATABASE LINKs that can be used to access Oracle tables on other Oracle database servers.

Authenticating a User on an Oracle Database Server

Users can issue multiple SET CONNECTION_ATTRIBUTES commands to supply valid user IDs and passwords.

If needed, the DBA or some other authorized person at your site will supply you with a valid Oracle user ID and password.

It may be desirable to prompt users for their Oracle password instead of coding it in a procedure. In this case, use a Dialogue Manager variable in its place, and retrieve the value using -CRTFORM or -PROMPT commands. If you use -CRTFORM, you can make the password field non-displayable for additional security.

A valid Oracle user ID and password must be supplied before issuing commands that access the Oracle RDBMS. If a valid Oracle login ID and password have not been supplied, an error message is returned. You should respond by correcting and re-issuing the SET CONNECTION_ATTRIBUTES command.

The SET CONNECTION_ATTRIBUTES command stores both the Oracle login ID and password in user virtual storage. It does not immediately initiate a login to Oracle. The actual login is deferred until a subsequent command is issued that requires the services of the Oracle RDBMS.

Syntax: How to Declare Connection Attributes for Oracle

```
ENGINE [SQLORA] SET CONNECTION_ATTRIBUTES [connection_name]/
userid,password
```

where:

`SQLORA`

Indicates the Adapter for Oracle. You can omit this value if you previously issued the SET SQLENGINE command.

`connection_name`

Specifies a remote instance using an Oracle TNSNAME (the net service name used as a connect descriptor to an Oracle database server across the network). If omitted, the local database server will be set as the default.

`userid`

Is the primary authorization ID by which you are known to Oracle, up to 30 characters in length.

`password`

Is the password associated with the user ID, up to 30 characters in length.

Note: SET USER is a synonym for SET CONNECTION_ATTRIBUTES, supported for compatibility with earlier releases of the adapter. However, note that the symbol used for separating the connection attribute from the authentication information and the symbol used for separating the user ID from the password changed as of FOCUS 7.2.

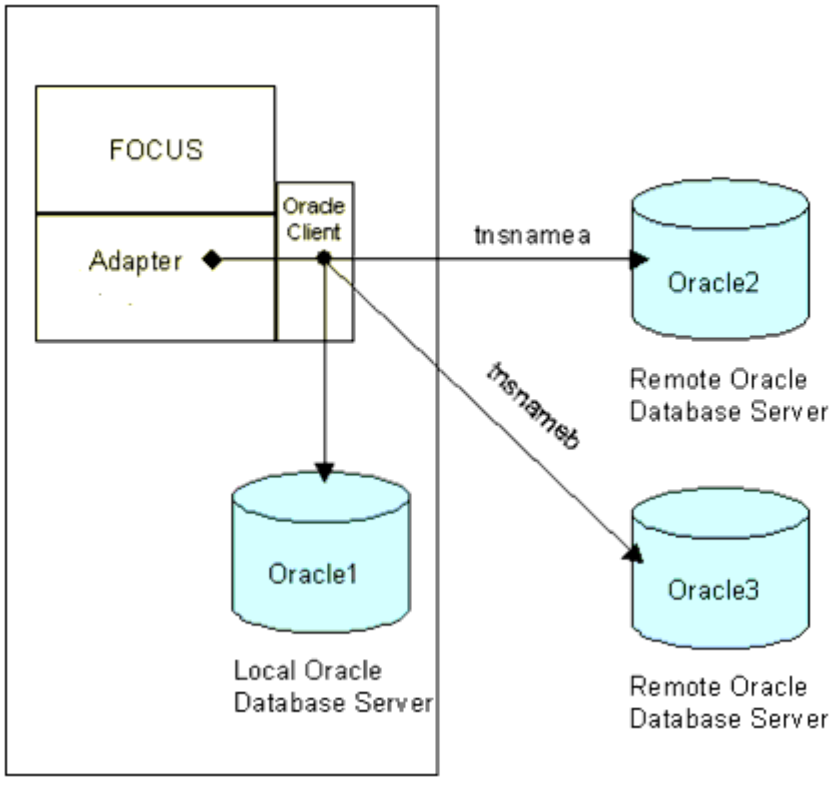
Syntax: How to Query the Declared Oracle Connections

Issue the following query command to list status information for all declared connections:

```
ENGINE SQLORA ? SERVERS
```

Example: Declaring Connection Attributes for Oracle

Refer to the following diagram in conjunction with these examples.



The following SET CONNECTION_ATTRIBUTES command connects to the Oracle database server named TNSNAMEA with an explicit user ID and password.

```
ENGINE SQLORA SET CONNECTION_ATTRIBUTES TNSNAMEA/USERA,PWDA
```

The following SET CONNECTION_ATTRIBUTES command connects to a local Oracle database server with an explicit user ID and password:

```
ENGINE SQLORA SET CONNECTION_ATTRIBUTES /USERA,PWDA
```

Selecting an Oracle Connection to Access

Once all Oracle database servers to be accessed have been declared using the SET CONNECTION_ATTRIBUTES command, there are two ways to select a specific Oracle connection from the list of declared connections:

- You can select a connection using the SET DEFAULT_CONNECTION command. If you do not issue this command, the connection name specified in the *first* SET CONNECTION_ATTRIBUTES command is used.
- You can include the CONNECTION= attribute in the Access File of the table specified in the current query. This attribute supersedes the default connection.

Syntax: How to Select an Oracle Connection to Access

```
ENGINE [SQLORA] SET DEFAULT_CONNECTION [connection_name]
```

where:

`SQLORA`

Indicates the Adapter for Oracle. You can omit this value if you previously issued the SET SQLENGINE command.

`connection_name`

Is the connection name specified in a previously issued SET CONNECTION_ATTRIBUTES command. If omitted, the local database server is set as the default. If this connection name has not been previously declared, a FOC1671 message is issued.

Note:

- If you issue the ENGINE SQLORA SET DEFAULT_CONNECTION command more than once, the connection name specified in the last command will be the active connection name.
- The SET DEFAULT_CONNECTION command cannot be issued while an uncommitted transaction (LUW) is pending. In that case, a FOC1671 message is issued.
- SET SERVER is a synonym for SET DEFAULT_CONNECTION, supported for compatibility with earlier releases of the adapter.

Example: Selecting an Oracle Connection to Access

The following SET DEFAULT_CONNECTION command selects the Oracle database server named TNSNAMEB as the default Oracle database server.

```
ENGINE SQLORA SET DEFAULT_CONNECTION TNSNAMEB
```

Note: You must have previously issued a SET CONNECTION_ATTRIBUTES command for TNSNAMEB.

Oracle Support for DATABASE LINKs

You can be connected to one Oracle database server and access a table on another Oracle database server (without actually connecting to the second server) if the first server has a DATABASE LINK defined for the table.

To access a remote Oracle table using DATABASE LINKs, the following conditions must exist.

- The Oracle database server to which you are connected must have a valid DATABASE LINK defined.
- The TABLENAME= attribute in the Access File for the Oracle table to be queried must have the following format:

  TABLENAME=[*owner.*]*tablename@databaselink*

  where:

  owner

  Is the user ID by default. It can consist of a maximum of 30 characters. Oracle prefers that the value be uppercase.

  tablename

  Is the name of the table or view. It can consist of a maximum of 30 characters.

  databaselink

  Is the valid DATABASE LINK name defined in the currently connected Oracle database server.

Once you have met these conditions, all requests for the table will be processed on the remote Oracle database server specified using the DATABASE LINK name. Using this method is another way to access multiple remote servers in one SQL request.

FOCUS DBA Security

You can implement FOCUS security as a complement to existing SQL GRANT security. FOCUS DBA security provides data control at a number of different levels. It can give users limited access to a specific table by restricting their access to particular columns and/or rows. It can further restrict their access to rows whose columns contain certain values. With this mechanism, a site can securely define a few global tables and eliminate the need to create a complex series of views for every combination of access rights.

To implement DBA security for a table, specify the security rules in the Master File, following the description of the table. The FOCUS DBA (as defined in the Master File) can prevent unauthorized viewing of the restriction rules by encrypting the Master. For information regarding DBA security or encryption, see the *Describing Data* manual.

Chapter 4

Describing Tables to FOCUS

In order to access a table or view using FOCUS, you must first describe it in two files, a Master File, and an associated Access File.

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

The Master File describes the columns of the RDBMS table using keyword-value pairs in comma-delimited format (see *Master Files* on page 56). The Access File includes additional attributes that complete the FOCUS definition of the RDBMS table (see *Access Files* on page 86). The adapters require both descriptions to generate SQL queries.

In this chapter:

Creating Master and Access Files

There are two methods for creating Master and Access Files.

- Execute an automated procedure that creates Master and Access Files for existing RDBMS tables—AUTODB2 for DB2 or AUTODBC for Teradata. *Automated Procedures* on page 113 describes these facilities.
- Use 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, data types, 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 of Master and Access Files can describe several tables. This chapter presents single-table Master and Access Files. For multi-table file descriptions, see *Multi-Table Structures* on page 99.

You can represent tables with repeating columns as FOCUS OCCURS segments. See *The OCCURS Segment* on page 94 for an explanation of these virtual constructs.

FOCUS processes a request with the following steps.

1. It locates the Master File for the table. The file name specified in the report request identifies the Master File by its member name.
2. It detects the SUFFIX value, DB2, SQLDS, SQLDBC, SQLORA, or SQLIDMS, in the Master File. Since this value indicates that the data is in an RDBMS table, FOCUS passes control to the appropriate relational adapter.
3. The adapter locates the corresponding Access File, uses the information contained in both descriptions to generate the SQL statements required by the request, and passes the SQL statements to the RDBMS.
4. The adapter 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 Master File for the DB2 table EMPINFO.

```
FILENAME=EMPINFO        ,SUFFIX=DB2,$

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=DPT          ,USAGE=A10    ,ACTUAL=A10,
MISSING=ON,$
 FIELD=CURRENT_SALARY   ,ALIAS=CSAL         ,USAGE=P9.2   ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE     ,ALIAS=CJC          ,USAGE=A3     ,ACTUAL=A3,$
 FIELD=ED_HRS           ,ALIAS=OJT          ,USAGE=F6.2   ,ACTUAL=F4,
MISSING=ON,$
 FIELD=BONUS_PLAN       ,ALIAS=BONUS_PLAN ,USAGE=I4     ,ACTUAL=I4,$
 FIELD=HIRE_DATE_TIME ,ALIAS=HDTT         ,USAGE=HYYMDm ,ACTUAL=HYYMDm ,
MISSING=ON,$
 FIELD=HIRE_TIME       ,ALIAS=HT           ,USAGE=HHIS    ,ACTUAL=HHIS    ,
MISSING=ON,$
```

The following is an Access File for the table EMPINFO.

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1,
 WRITE = YES,  DBSPACE = PUBLIC.SPACE0,$
```

File Descriptions and Tables on page 459 contains a complete list of Master Files and Access Files for the examples cited in this manual.

Master Files

A table is an RDBMS object consisting of rows and columns. A Master File represents a table as a single segment.

A Master File contains three types of declarations.

- The file declaration indicates that the data is stored in a DB2, Teradata, Oracle, or CA-IDMS 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 attributes are required. The rest are optional (see *Optional Field Attributes* on page 84).

Do not use system or SQL reserved words as names for files, segments, fields, or aliases. Specifying a reserved word generates the SQL syntax errors -104, -105, or -106.

File Attributes in the Master File

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

where:

name

Is a one- to eight-character file name. For documentation purposes, it is recommended that the Master File name be used as the FILENAME attribute.

Note: Master File names longer than eight characters are supported on z/OS, as described in *FILENAME* on page 58.

suffix

Identifies the adapter needed for accessing the table. Valid values are:

DB2

Is the SUFFIX value for the Adapter for DB2.

Note: The suffix value SQLDS is also available for backward compatibility.

SQLDBC

Is the SUFFIX value for the Adapter for Teradata.

SQLIDMS

Is the SUFFIX value for the Adapter for IDMS/SQL.

SQLORA

Is the SUFFIX value for the Adapter for Oracle.

Reference: FILENAME

The FILENAME (or FILE) attribute names the Master File. On z/OS, the name of the Master File is its member name in the PDS allocated to DDNAME MASTER.

The Master File name consists of 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.

Master File names longer than eight characters are supported on z/OS. Because file and member names are limited to eight characters, longer Master File names are assigned eight-character names to be used when interacting with the operating system. The short name consists of three parts, a prefix consisting of the leftmost characters from the long name, followed by a left brace character ({), followed by an index number.

The length of the prefix depends on how many long names have a common set of leftmost characters.

- The first ten names that share six or more leftmost characters have a six-character prefix and a one-character index number, starting from zero.
- Starting with the eleventh long name that shares the same leftmost six characters, the prefix becomes five characters, and the index number becomes two characters, starting from 00.

This process can continue until the prefix is one character and the index number is six characters. If you delete one of these members from the PDS, the member name will be reused for the next long name.

Example: **Long and Short Master File Names**

The following table lists sample long names with the corresponding short names that will be assigned under z/OS.

Long Name	Short Name
EMPLOYEES_ACCOUNTING	EMPLOY{0
EMPLOYEES_DEVELOPMENT	EMPLOY{1
EMPLOYEES_DISTRIBUTION	EMPLOY{2
EMPLOYEES_FINANCE	EMPLOY{3
EMPLOYEES_INTERNATIONAL	EMPLOY{4
EMPLOYEES_MARKETING	EMPLOY{5
EMPLOYEES_OPERATIONS	EMPLOY{6
EMPLOYEES_PERSONNEL	EMPLOY{7
EMPLOYEES_PUBLICATIONS	EMPLOY{8
EMPLOYEES_RESEARCH	EMPLOY{9
EMPLOYEES_SALES	EMPLO{00
EMPLOYEES_SUPPORT	EMPLO{01

Syntax: **How to Implement a Long Master and Access File Name in z/OS**

To relate the short name to its corresponding long name, the first line of the Master and Access File must contain the following comment:

```
$ VIRT=complete_long_file_name
```

where:

complete_long_file_name

Is the long name, up to 64 characters.

Reference: SUFFIX

The SUFFIX attribute indicates which adapter is required for interpreting requests. Valid values are DB2, SQLDS, SQLDBC, SQLIDMS, and SQLORA.

Segment Attributes in the Master File

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

If several Master Files (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 Master Files, and use the CRFILE attribute in the other Master Files to access the existing description. For a full explanation of remote segment descriptions, see *Additional Topics* on page 417.

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 Master File that contains the full definition of the columns of the RDBMS table (see *Additional Topics* on page 417).

S0

S zero indicates to the adapter that the RDBMS handles the storage order of the data.

KL

References a remote segment description (see *Additional Topics* on page 417).

crfile

Is required only to reference a remote segment description. Indicates the name of the remote Master File that contains the full definition of the columns of the RDBMS table (see *Additional Topics* on page 417).

Reference: SEGNAME in the Master File

The SEGNAME attribute identifies or links 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 Master File that contains the full definition of the columns of the RDBMS table.

The corresponding Access File must contain a segment declaration with the same SEGNAME value as the Master File. 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.

Reference: SEGTYPE

In a single table Master File, SEGTYPE always has the value 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 Master Files are discussed in *Multi-Table Structures* on page 99.

Reference: CRFILE

Include the CRFILE attribute in a segment declaration if:

- The actual description of the columns of the table is stored in another (remote) Master File. The CRFILE value must be the name of the Master File that contains the full definition of the columns of the RDBMS table. For a complete discussion of remote segment descriptions, see *Additional Topics* on page 417.
- You want to implement a conditional join to another RDBMS table. For information, see *Multi-Table Structures* on page 99. (You can also join tables using the JOIN command described in *Advanced Reporting Techniques* on page 213).

Field Attributes in the Master File

Each table consists of one or more columns. In the Master File, you define each column as a field with the primary attributes FIELDNAME, ALIAS, USAGE, ACTUAL, and MISSING. The *Describing Data* manual explains additional attributes.

You can get values for these attributes from the DB2 system catalog table SYSCOLUMNS, from the Oracle SYSTEM.COLUMNS data dictionary view, or from the IDMS SQL schema definition or standard IDMS dictionary reports. For a sample request for DB2, refer to *Additional Topics* on page 417.

Syntax: How to Describe a Column in a Master File

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

where:

name

Is a 1- to 66-character unqualified name. In requests, you can qualify a fieldname with its Master File and/or segment name. *Additional Topics* on page 417, contains a discussion of long fieldnames. For more information, consult the *Describing Data* manual.

sqlcolumn

Is the RDBMS column name, up to:

- 30 characters long for DB2.
- 30 characters for Oracle and Teradata.
- 32 characters for IDMS SQL.

display

Is the FOCUS display format for the field.

sqlfmt

Is the FOCUS definition of the RDBMS data type and length, in bytes, for the field. (See *Data Type Support* on page 64.)

OFF|ON

Indicates whether the field can contain null values. OFF, the default, does not permit null values.

FIELDTYPE=R

Indicates that the field is read-only. Any number of fields can have this attribute. A field with this attribute must represent a DB2 TIMESTAMP column.

Note: AUTODB2 does not add the FIELDTYPE=R attribute to the FIELD declaration in the generated Master File. You must edit the Master File to add this attribute.

Reference: The Primary Key

A primary key for a table 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 adapter uses information from both the Master File 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 Master File, the *first* n fields described immediately after the segment declaration constitute the primary key. Therefore, the order of field declarations in the Master File is significant.

To define the primary key in a Master 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 adapter itself does not require any index or IDMS calc key on columns comprising the primary key (although a unique index is certainly desirable for both data integrity and performance reasons).

Reference: FIELDNAME

Field names must be unique within a single-table Master File and can consist of up to 66 alphanumeric characters. Within the Master File, field names cannot include qualifiers. Column names are acceptable values if they meet the following 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 field name displays as the default column title on reports, select a name that is representative of the data. In TABLE, GRAPH, and MODIFY requests, you can specify field names, aliases, or a unique truncation of either. Maintain does not support alias names or truncated names. In all requests, you can qualify a field name with its file name and/or segment name (see your FOCUS documentation).

Reference: ALIAS

The ALIAS value for each field must be the full SQL column name (the adapter uses it to generate SQL statements). The ALIAS name must be unique within the segment. DB2, Oracle and Teradata permit 30 characters, and IDMS permits 32 characters. The ALIAS name must comply with the same naming conventions described for field names.

Reference: USAGE/FORMAT

The USAGE attribute indicates the display format of the field. An acceptable value must include the field type and length and may contain display options. The USAGE format is used for data display on reports and CRTFORMs. All standard FOCUS USAGE formats are available, except that Oracle does not support F format. For a complete list of USAGE formats appropriate for non-FOCUS data sources, see the *Describing Data* 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.
- Field type text (TX) varies for each RDBMS in terms of syntax and storage limitations. See *Data Type Support* on page 64 for a complete discussion.
- Fields with decimal and floating-point data types 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 data types exclude positions for the decimal point and negative sign.

  For example, a column defined as DECIMAL(5,2) in DB2 would have a USAGE attribute of P7.2 to allow for the decimal point and a possible negative sign.

Data Type Support

The ACTUAL attribute indicates the Master File representation of RDBMS data types.

DB2 Data Type Support

The following tables describe how the adapter maps DB2 data types.

DB2 Data Type	FOCUS Data Type		Remarks
Date-Time Data Types			
DATE	YYMD	DATE	
TIME	HHIS	HHIS	
TIMESTAMP	HYYMDm	HYYMDm	
Numeric Data Types			
SMALLINT	I6	I4	
INTEGER	I11	I4	Maximum precision is 11.
BIGINT	P20	P10	Available on UNIX and Windows only.
DECIMAL (*p,s*)	P6	P8	*p* is an integer between 1 and 31. *s* is an integer between 0 and *p*.
REAL	F9.2	F4	Maximum precision is 9.
FLOAT	D20.2	D8	Maximum precision is 20.
LOB Data Types			
BLOB	BLOB	BLOB	Up to 2 gigabytes.
CLOB	TX50	TX	Up to 2 gigabytes.
Other Data Types			
CHAR (*n*)	A*n*	A*n*	*n* is the number of bytes, and is an integer between 1 and 254.
LONG VARCHAR (*n*) in (1...32700)			Not supported

DB2 Data Type	FOCUS Data Type		Remarks
GRAPHIC (*n*)	A*m*	K*n*	*m* = (*n* * 2) + 2 *m* is the number of bytes, and *n* is the number of characters.
VARGRAPHIC (*n*)	A*m*	K*n*	*m* = (*n* * 2) + 2 VARGRAPHIC is assumed to be GRAPHIC until n = 127 for non-Unicode. The maximum length of *m* is 256.
LONG VARGRAPHIC			Not supported
DATALINK			Not supported
XML	TX50	TX	Supported with DB2 Version 9 on UNIX, Windows, and z/OS; not supported on IBM i.

The following table describes how the adapter maps non-Unicode Character data types. This mapping can be changed based on the value of LONGCHAR. The default value is ALPHA.

DB2 Data Type	Remarks	LONGCHAR ALPHA		LONGCHAR TEXT	
VARCHAR (*n*)	*n* is an integer between 1 and 256.	A*n*V	A*n*V	A*n*V	A*n*V
	n is an integer between 257and 32768.	A*n*V	A*n*V	TX50	TX

Note: The main purpose of the LONGCHAR setting is to provide compatibility with previous releases of the adapter. This SET parameter was designed to control processing of DBMS Character data types and was never intended for DBMS LOB data types.

Reference: **Data Type Support for Unicode**

The following table describes how the adapter maps Unicode Character data types. The adapter operates in character semantic when configured for Unicode. The LONGCHAR setting does not affect mapping in this case.

	FOCUS Data Type		
DB2 Data Type			**Remarks**
CHAR (*n*)	A*n*V	A*n*B	*n* is an integer between 1 and 254. The column stores *n* bytes (specified by the ACTUAL attribute) which represent up to *n* characters (specified by the USAGE attribute). On EBCDIC platforms, the ACTUAL value is multiplied by 1.5 to accommodate UTF-EBCDIC. For example, a CHAR (10) column is described by USAGE=A10V, ACTUAL=A15B.
VARCHAR (*n*)	A*n*V	A*n*VB	*n* is the number of characters, and is an integer between 1 and 32672. The column stores up to *n* bytes (specified by the ACTUAL attribute), which represent up to *n* characters (specified by the USAGE attribute). On EBCDIC platforms, the ACTUAL value is multiplied by 1.5 to accommodate UTF-EBCDIC. For example, a VARCHAR (10) column is described by USAGE=A10V, ACTUAL=A15VB.
GRAPHIC (*n*)	A*n*	A*n*	*n* is the number of characters.
VARGRAPHIC (*n*)	A*n*V	A*n*V	*n* is the number of characters.

Teradata Data Type Support

The following tables describe how the adapter maps Teradata data types.

Teradata Data Type	FOCUS Data Type		Remarks
Date-Time Data Types			
DATE	YYMD	A8	
TIME	HHISsm	HHISsm	The operation with different Teradata TIME formats depends on DateTimeFormat setting in the data source part of the $HOME/.odbc.ini file. The adapter requires DateTimeFormat=IAI. Neither the ODBC nor the CLI interface supports the integer format of TIME (I). Only the ANSI format of TIME (AT) is supported.
TIMESTAMP	HYYMdm	HYYMdm	
INTERVAL	A*n*	A*n*	INTERVAL identifies a period of time in different ranges (YEAR, MONTH, DAY, HOUR, MIN, SEC). The Teradata external (client) representation of INTERVAL is always CHAR (*n*) where $n = p + x$. Precision p from 1 to 4 and x from 1 to 11 depend on range.
Numeric Data Types			
SMALLINT	I6	I4	A 2-byte signed binary integer. Range: -2^{15} to $2^{15} - 1$.

Teradata Data Type	FOCUS Data Type		Remarks
INTEGER	I11	I4	A 4-byte binary integer Range: -2.147G to +2.147G.
BYTEINT	I6	I4	A 1-byte signed binary integer Range: -128 to +127.
BIGINT	P20	P10	Teradata representation of a signed, binary integer value from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
DECIMAL(*n*,*m*)	P31.31	P16	*n*, the precision, is an integer between 1 and 31. *m*, the scale, is an integer between 0 and n. Also referred to as NUMERIC.
FLOAT	D20.2	D8	8 bytes. Range: $-2 * 10^{307}$ to $2 * 10^{308}$. Same as REAL or DOUBLE PRECISION.
LOB Data Types			
GRAPHIC			Not supported.
VARGRAPHIC			Not supported.
LONG GRAPHIC			Not supported.
VARBYTE			Not supported.
BLOB			Not supported.

The following table lists how the adapter maps non-Unicode Character data types. This mapping can be changed based on the value of LONGCHAR. The default value is ALPHA.

		LONGCHAR ALPHA		LONGCHAR TEXT	
Teradata Data Type	**Remarks**				
CHAR	*n* is an integer between 1 and 256.	A*n*	A*n*	A*n*	A*n*
	n is an integer between 257 and 32000.	A*n*	A*n*	TX50	TX
VARCHAR (*n*)	*n* is an integer between 1 and 256.	A*n*V	A*n*V	A*n*V	A*n*V
	n is an integer between 257 and 32000.	A*n*V	A*n*V	TX50	TX
LONG VARCHAR (*n*)	*n* is an integer between 1 and 256.	A*n*V	A*n*V	A*n*V	A*n*V
	n is an integer between 257 and 64000. FOCUS supports up to 32767 bytes. All data exceeding 32K will be truncated.	A*n*V	A*n*V	TX50	TX

Note: The main purpose of the LONGCHAR setting is to provide compatibility with previous releases of the adapter. This SET parameter was designed to control processing of DBMS Character data types and was never intended for DBMS LOB.

Reference: Data Type Support for Unicode

The following table describes how the adapter maps Unicode Character data types. The adapter operates in character semantic when configured for Unicode. The LONGCHAR does not affect mapping in this case.

	FOCUS Data Type		
Teradata Data Type			**Remarks**
CHAR(*n*)	A*n*V	A*n*V	*n* is an integer between 1 and 4000.
VARCHAR (*n*)	A*n*V	A*n*V	*n* is an integer between 1 and 4000.

Oracle Data Type Support

The following tables describe how the adapter maps Oracle data types.

	FOCUS Data Type		
Oracle Data Type			**Remarks**
Date-Time Data Types			
DATE	HYYMDS	HYYMDS	Stores point-in-time values (dates and times) ranging from January 1, 4712 BCE through December 31, 9999 CE.
TIMESTAMP (*fractional_seconds_precision*) WITH LOCAL TIME ZONE	HYYMDS HYYMDs HYYMDm	HYYMDS HYYMDs HYYMDm	*fractional_seconds_precision* in (0,1,2). *fractional_seconds_precision* in (3,4,5). *fractional_seconds_precision* in (6,7,8,9). The adapter supports TIMESTAMP without the TIME ZONE portion.

Oracle Data Type	FOCUS Data Type		Remarks
INTERVAL YEAR (*year_precision*) TO MONTH	Not supported	Not supported	Stores a period of time in years and months, where *year_precision* is the number of digits in the *YEAR* datetime field. Accepted values are 0 to 9. The default is 2.
INTERVAL DAY (*day_precision*) TO SECOND (*fractional_seconds_precision*)	Not supported	Not supported	Stores a period of time in days, hours, minutes, and seconds.
Numeric Data Types			

Oracle Data Type	FOCUS Data Type		Remarks
NUMBER (*p*, s)	P*p,s*	P1,...,8	*p* is an integer between 1 and 31. s is an integer between 0 and *p*. **Note:** If the MISSING=ON attribute is present in the Master File, the ACTUAL attribute must be P8 or longer. If MISSING=OFF, the ACTUAL attribute will be generated with the length equal to the precision of a NUMERIC field in an Oracle table.
	P*p,s*	P9,...,16	*p* is an integer between 16 and 31. s is an integer between 0 and *p*.
	D20.2	D8	*p* is an integer between 32 and 37. s is an integer between 0 and *p*.
	I11	I4	*p* is 38. This value is the Oracle default. s is an integer between 0 and *p*.
INTEGER	See Number	See Number	Oracle converts and stores this type as NUMBER.
DECIMAL	See Number	See Number	Oracle converts and stores this type as NUMBER.
FLOAT	D20.2	D8	An approximate numeric type.
BINARY_DOUBLE	D20.2	D8	A 32-bit, single-precision floating-point approximate numeric type.
BINARY_FLOAT	D20.2	D8	A 64-bit, double-precision floating-point approximate numeric type.

Oracle Data Type	FOCUS Data Type		Remarks
LOB Data Types			
BLOB	BLOB	BLOB	The adapter supports BLOB through the OCI interface, with ora_oci=y set in edaserve.cfg.
CLOB	TX50	TX	Adapter supports CLOB through the OCI interface, with ora_oci=y set in edaserve.cfg.
NCLOB	TX50	TX	Adapter supports NCLOB through the OCI interface, with ora_oci=y set in edaserve.cfg.
LONG	TX50	TX	Character data of variable length up to 2 gigabytes, or $2^{31} - 1$ bytes. Provided for backward compatibility. The LONG field must be the last position in the report. This field is the subject of all limitations as documented by Oracle. As of Oracle 8i, this data type is deprecated. You can migrate to CLOB whenever possible.
LONG RAW	BLOB	BLOB	As of Oracle 8i, this data type is deprecated. You can migrate to BLOB whenever possible.
BFILE	Not supported	Not supported	Stores unstructured binary data in operating system files outside the database.
Other Data Types			

Oracle Data Type	FOCUS Data Type		Remarks
ROWID	A18	A18	The pseudo-column data types that store the physical address of a row.
UROWID	Not supported	Not supported	The pseudo-column data types that store both the physical and logical address of a row.
MLSLABEL	Not supported	Not supported	Stores variable-length, binary operating system labels.

The following table lists how the adapter maps non-Unicode Character data types. This mapping can be changed based on the value of LONGCHAR. The default value is ALPHA.

Oracle Data Type	Remarks	LONGCHAR ALPHA		LONGCHAR TEXT	
CHAR (*n*)	*n* is an integer between 1 and 256.	A*n*	A*n*	A*n*	A*n*
	n is an integer between 257 and 2000.	A*n*	A*n*	TX50	TX
NCHAR (*n*)	*n* is an integer between 1 and 256.	A*n*	A*n*	A*n*	A*n*
	n is an integer between 257 and 2000.	A*n*	A*n*	TX50	TX
VARCHAR2 (*n*) or VARCHAR (*n*)	*n* is an integer between 1 and 256.	A*n*V	A*n*V	A*n*V	A*n*V
	n is an integer between 257 and 4000.	A*n*V	A*n*V	TX50	TX

Oracle Data Type	Remarks	LONGCHAR ALPHA		LONGCHAR TEXT	
NVARCHAR2 (*n*)	*n* is an integer between 1 and 256.	A*n*	A*n*	A*n*	A*n*
	n is an integer between 257 and 4000.	A*n*	A*n*	TX50	TX
RAW (*n*)	*n* is an integer between 1 and 128, *m* = 2 * *n*	A*m*	A*m*	A*m*	A*m*
	n is an integer between 129 and 2000, *m* = 2 * *n*	A*m*	A*m*	TX50	TX

Note: The main purpose of the LONGCHAR setting is to provide compatibility with previous releases of the adapter. This SET parameter was designed to control processing of DBMS Character data types and was never intended for DBMS LOB.

Reference: Data Type Support for Unicode

The following table describes how the adapter maps Unicode Character data types. The adapter operates in character semantic when configured for Unicode. The LONGCHAR does not affect mapping in this case.

	FOCUS Data Type		
Oracle Data Type			Remarks
CHAR (*n*)	A*n*V	A*n*B	*n* is an integer between 1 and 2000. The column stores *n* bytes (specified by the ACTUAL attribute) which represent up to *n* characters (specified by the USAGE attribute).
CHAR (n CHAR)	A*n*	A*n*	*n* is the number of characters, and is an integer between 1 and 2000.

Oracle Data Type	FOCUS Data Type		Remarks
CHAR (n BYTE)	A*n*V	A*n*B	*n* is an integer between 1 and 2000. The column stores *n* bytes (specified by the ACTUAL attribute) which represent up to *n* characters (specified by the USAGE attribute).
VARCHAR2 (*n*) or VARCHAR (*n*)	A*n*V	A*n*VB	*n* is an integer between 1 and 4000. The column stores up to *n* bytes (specified by the ACTUAL attribute) which represent up to *n* characters (specified by the USAGE attribute).
VARCHAR2 (*n* CHAR)	A*n*V	A*n*V	*n* is the number of characters, and is an integer between 1 and 4000.
VARCHAR2 (*n* BYTE)	A*n*V	A*n*VB	*n* is an integer between 1 and 4000. The column stores up to *n* bytes (specified by the ACTUAL attribute) which represent up to *n* characters (specified by the USAGE attribute).

IDMS/SQL Data Type Support

The following tables describe how the adapter maps IDMS/SQL data types.

IDMS/SQL Data Type	FOCUS Data Type		Remarks
Date-Time Data Types			
Date	YYMD	DATE	
Time	A8	A8	
Timestamp	A26	A26	

IDMS/SQL Data Type	FOCUS Data Type		Remarks
Numeric Data Types			
Smallint	I6	I4	
Integer	I11	I4	
Longint/BIGINT	I11	I4	
Numeric/Decimal(*n*,*m*)	P(*n*+2,*m*)	P8	*n* is an integer between 1 and 14, m is between 0 and 31.
	P(*n*+2,*m*)	P16	*n* is an integer between 14 and 31, m is between 0 and 31.
Double Precision	D20.2	D8	
Float(*n*)	F9.2	F4	*n* is an integer between 1 and 24.
	D20.2	D8	*n* is an integer between 25 and 56.
Real	F9.2	F4	
LOB Data Types			
Vargraphic (1..16379)	TX	TX	
Other Data Types			
Binary(1..32,760)	A*n*	A*n*	*n* is less than 257.
Graphic (1..16380)(*n*)	A(2**n*)	A(2**n*)	2*n is less than or equal to 32,000.

The following table lists how the adapter maps Character data types. LONGCHAR setting is not applicable.

IDMS/SQL Data Type	FOCUS Data Type		Remarks
CHARACTER(1...32,760)	A*n*	A*n*	*n* is less than or equal to 32,000.
VARCHAR (1...32,760)	A*n*V	A*n*V	*n* is less than or equal to 32,000.

Additional Attributes

Some aspects of ACTUAL = TX or ACTUAL = DATE, TIME, or TIMESTAMP are unique for the adapters.

Reference: **ACTUAL = DATE, TIME, and TIMESTAMP**

The adapters support a comprehensive set of date and time formats.

ACTUAL = DATE, in conjunction with USAGE date formats (such as YDM, DMY, MDY), describes columns with DATE data types. FOCUS date formats containing any combination of the components year, month, and day can display dates stored with the RDBMS DATE data type. This feature makes it easier to manipulate dates, and the storage format is compatible with both FOCUS and non-FOCUS programs.

The FOCUS default date is '1900-12-31'. For information about setting the default value, see *Additional Topics* on page 417.

ACTUAL = DATE makes it possible to:

- Sort by date into 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 1999, 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, a DATE literal in a WHERE clause must conform to its USAGE format. For example, the field DATE_FLD, with a display format of YMD and an ACTUAL format of DATE, is specified as:

```
WHERE DATE_FLD EQ '990224'
```

- When using MODIFY 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 adapter assigns a default day and/or month of '01' to the incoming transaction value (for example, 1999 becomes 19990101).
- RDBMS date fields are century aware. However, if you join an RDBMS table to a nonrelational data source, use a non-aware date literal, or use a MODIFY transaction file with non-aware dates, you may need to invoke FOCUS Year 2000 solutions such as the DEFCENT and YRTHRESH parameters. See your FOCUS documentation for details.
- Teradata users who choose to use native DBC/SQL in creating tables with dates must use the Teradata default date format (YY/MM/DD).
- Oracle users who choose to use native Oracle/SQL in creating tables with dates must use the Oracle default date format (DD MON YY).

ACTUAL formats for RDBMS TIME and TIMESTAMP columns are represented by FOCUS date-time data types, which use the format code H, as described in the ACTUAL format conversion charts:

- These data types are supported for Direct SQL Passthru, TABLE, MODIFY, Maintain, and SQL Translator requests. They are not supported for Static MODIFY requests.
- AUTODB and CREATE FILE support these data types.
- Date-time manipulation handled by the FOCUS date-time functions is not converted to SQL.

Reference: ACTUAL = TX

The ACTUAL = TX field type can be used to describe CHAR, VARCHAR, and LONG VARCHAR data types (LONG for Oracle). Text fields contain variable-length text, usually lengthy descriptions or explanations. They also support text wrapping in reports. The syntax is

```
FIELDNAME=name, ALIAS=sqlcolumn, USAGE=TXnn, ACTUAL=TX,$
```

where:

nn

Defines the length of an output line for display. (Maximum line length is 254 characters.)

You can specify text fields in report and MODIFY requests. SQL restrictions do not permit them to be:

- Indexed.
- Defined as primary key fields.
- Used in joins.

The following are additional limitations on the use of text fields in MODIFY requests:

- MODIFY can reference up to 95 text fields.
- You cannot use text fields in the Scratch Pad Area (HOLD).
- You cannot use text fields in a CRTFORM or with TYPE.
- You can include at most one text field per FIXFORM subcommand. The text field must be the last field in the FIXFORM statement.

In report requests (TABLE), you cannot use text fields:

- In an IF test or WHERE clause.
- As a BY clause or an ACROSS clause.

The following HOLD formats do not support text fields:

- DIF
- IFPS
- CALC
- LOTUS
- COM, COMMA, COMT, TABT

Text fields cannot take advantage of the standard display options, nor can you use them in a DEFINE. Trailing blanks are truncated for efficient storage.

Limit: FOCUS can handle text fields up to 32K in size. (Text fields greater than 32K are truncated at the 32K boundary.)

For **Oracle**, a RAW column may contain up to 32,767 bytes of text.

For **DB2 on z/OS**, the maximum physical size of a text value is also 32,767, but the entire row must fit into the database buffer pool. A buffer pool is a main storage area reserved to hold data pages or index pages. The buffer pool size is either 4K or 32K, although 4K is generally assumed. The size of the text field is limited by the total row length and the buffer pool size. Therefore, the actual space available may be less than the size of the buffer pool. Calculate the maximum size available for TX fields using the formula

```
TXsize= (BPsize - nontext_bytes)
```

where:

BPsize

Is approximately 4000 bytes for 4K buffers or 32,000 bytes for 32K buffers.

nontext_bytes

Is the total number of bytes for field types other than text (TX).

To load text field data, use a FIXFORM or PROMPT command, or use TED in a MODIFY request. For a complete discussion of text fields, consult the *Describing Data* and *Maintaining Databases* manuals.

Reference: MISSING

The adapter 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 MODIFY or MAINTAIN procedure enters data into 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 into the FOCUS missing data display value. The default NODATA display value is the period (.).

The MISSING attribute in the Master File indicates whether the RDBMS column accepts null data. When a field declaration omits the MISSING attribute, it defaults to OFF. The syntax is

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

where:

OFF

Is the default. In the RDBMS, columns should be created with the NOT NULL attribute (or NOT NULL WITH DEFAULT—DB2 only).

ON

FOCUS displays the NODATA value for null data. The column should be created without the NOT NULL attribute.

For null data support:

- The RDBMS table definition must describe a column that allows null data (*without* the clause NOT NULL).
- The Master File 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 Master File 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.

Reference: Comparing Fields With Null Values

FOCUS and the RDBMS differ slightly in how they compare null field values:

- When two fields contain null values, FOCUS considers them equal. The RDBMS considers the result of any comparison between them to be UNDEFINED and does not return any rows.
- When one field contains a null value and another field does not, FOCUS considers them *not* equal. The RDBMS considers the result of any comparison between them to be undefined and returns the value FALSE for any such selection condition.

Consider two examples:

```
TABLE FILE X
PRINT *
WHERE (FIELD1 EQ FIELD2)
END
```

```
TABLE FILE X
PRINT *
WHERE (FIELD1 NE FIELD2)
END
```

The following table summarizes the results produced by FOCUS and the RDBMS:

Condition	Field1 Value	Field2 Value	FOCUS Result	RDBMS Result
EQ	Null	Null	**True**	**False**
	Null	Not Null	False	False
NE	Null	Null	False	False
	Null	Not Null	**True**	**False**

In most cases, the adapter translates the FOCUS WHERE clause to an SQL WHERE predicate and passes it to the RDBMS for processing. (This translation process is called adapter optimization.) If the adapter does not translate the FOCUS WHERE test to an SQL WHERE predicate, FOCUS applies the selection test and may produce a different answer set.

Because of optimization enhancements introduced in recent releases, certain requests (outer joins, for example) no longer disable optimization as they did in prior releases. Therefore, the answer set returned may differ from that produced by prior versions of the adapter when FOCUS handled the request. See *The Adapter Optimizer* on page 171, for a discussion of optimization.

Reference: Optional Field Attributes

The optional field attributes DESCRIPTION, TITLE, DEFINE, and ACCEPT are supported for use with the adapter. Refer to your FOCUS documentation for information about these attributes.

FIELDTYPE=R

A field with the FIELTYPE=R (read-only) attribute represents a DB2 TIMESTAMP column. The RDBMS automatically increments this type of field. If you supply a value, it is ignored.

Example: Using FIELDTYPE=R

SALTIME is a DB2 table with a TIMESTAMP column (field ENTRY_DATE in the Master File):

```
FILENAME=SALINFO,  SUFFIX=DB2,$

SEGNAME=SALINFO  ,SEGTYPE=S0,$
 FIELDNAME=SALEID    ,ALIAS=EID     ,USAGE=A9      ,ACTUAL=A9,$
 FIELDNAME=PAY_DATE  ,ALIAS=PD      ,USAGE=YYMD    ,ACTUAL=DATE,$
 FIELDNAME=GROSS     ,ALIAS=MO_PAY  ,USAGE=D12.2M  ,ACTUAL=D8,$
 FIELDNAME=ENTRY_DATE,ALIAS=EDATE   ,USAGE=HYYMDm  ,ACTUAL=HYYMDm,
 FIELDTYPE=R,$
```

The following MODIFY procedure asks for an employee ID and pay date. If the employee ID exists in the table, but the pay date does not, the procedure asks for a monthly salary and adds a new row to the table:

```
MODIFY FILE SALINFO
CRTFORM LINE 1
" PLEASE ENTER A VALID EMPLOYEE ID AND PAY_DATE </1"
" EMPLOYEE ID: <SALEID   PAY DATE: <PAY_DATE "
MATCH SALEID
ON NOMATCH GOTO EXIT
ON MATCH GOTO ADDCASE
CASE ADDCASE
MATCH PAY_DATE
ON MATCH GOTO EXIT
ON NOMATCH CRTFORM LINE 5
" ENTER NEW MONTHLY SALARY <T.GROSS>  "
ON NOMATCH INCLUDE
ENDCASE
DATA
END
```

The following rows exist in the table for employee ID 071382660:

```
SALEID     PAY_DATE            GROSS  ENTRY_DATE
------     --------            -----  ----------
071382660  11/11/30        $5,000.33  2011/12/25 01:05:50.595295
071382660  11/12/30        $5,000.33  2012/01/24 01:27:48.680248
071382660  12/01/27        $5,083.67  2012/02/21 03:12:38.689755
071382660  12/02/24        $5,083.67  2012/03/20 06:11:40.965379
071382660  12/03/30        $5,083.67  2012/04/24 04:54:56.465970
071382660  12/04/30        $5,083.67  2012/05/25 01:33:58.063954
071382660  12/05/28        $5,083.67  2012/06/22 04:43:38.063858
071382660  12/06/29        $5,083.67  2012/07/24 03:12:22.609716
071382660  12/07/30        $5,083.67  2012/08/24 01:43:42.648030
071382660  12/08/31        $5,083.67  2012/09/25 06:15:38.112955
```

The following example executes the MODIFY procedure to add a salary of $6,000 for the pay date 13/04/30 for this employee:

```
 PLEASE ENTER A VALID EMPLOYEE ID AND PAY_DATE

 EMPLOYEE ID:  071382660    PAY DATE:  130430

 ENTER NEW MONTHLY SALARY  6000
```

The RDBMS automatically generates a timestamp value for the ENTRY_DATE field. If the procedure had supplied a value, it would have been ignored:

```
SALEID     PAY_DATE              GROSS  ENTRY_DATE
------     --------              -----  ----------
071382660  13/04/30          $6,000.00  2013/05/12 14:43:17.852050
```

It is up to the developer to query the data source to find out what value was generated by the RDBMS for the auto-increment field.

Access Files

Each Master File has a corresponding Access File. The name of the Access File (member name in the z/OS partitioned data set allocated to DDNAME FOCSQL) must be identical to that of the Master File or, in the case of a remote segment description can be described in the cross-referenced Access File. The Access File associates a segment in the Master 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 the precision of packed fields.

For multi-table structures, the Access File also contains KEYFLD and IXFLD attributes that implement embedded equijoins. See *Multi-Table Structures* on page 99, for details.

The following is an Access File for the DB2 table EMPINFO:

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1,
 WRITE = YES,  DBSPACE = PUBLIC.SPACE0,$
```

Segment Declarations in the Access File

The segment declaration in the Access File establishes the link between the Master File and the RDBMS table or view. Attributes that constitute the segment declaration are SEGNAME, TABLENAME, DBSPACE, WRITE, KEYS, and KEYORDER. Values for SEGNAME and TABLENAME are required. The remaining attributes acquire default values if they are omitted.

Teradata supports the FALLBACK attribute described in *FALLBACK (Teradata Only)* on page 93.

Oracle supports the CONNECTION attribute described in *CONNECTION (Oracle Only)* on page 94.

Syntax: How to Describe a Table or View in an Access File

```
SEGNAME=segname, TABLENAME= tableid [,DBSPACE=storage,]
  [,WRITE= {YES|NO}] [,KEYS= {0|n}] [,KEYORDER=sequence,]  ,$
```

where:

segname

Is the one- to eight-character SEGNAME value from the Master File.

tableid

Is the RDBMS-specific table name.

For DB2, the format of the table name attribute is

```
[location.][creator.]table
```

For Teradata, the format of the table name attribute is

```
[databasename.]table
```

For Oracle, the format of the table name attribute is

```
userid.tablename
```

For IDMS/SQL, the format of the table name attribute is

```
[schema.]table
```

See *TABLENAME* on page 88 for complete information.

`DBSPACE = storage`

Is an RDBMS-specific storage area for the table used by the CREATE FILE or HOLD FORMAT DB2, SQLDBC, SQLIDMS, or SQLORA commands.

For DB2 on z/OS

```
databasename.tablespacename  or DATABASEdatabasename
```

For Teradata, N/A.

For IDMS/SQL, the format of the DBSPACE attribute is

```
segment.area
```

For Oracle, the format of the DBSPACE attribute is

```
tablespacename
```

See *DBSPACE* on page 90 for complete information.

`WRITE = {YES|NO}`

YES specifies read and write access using MODIFY and MAINTAIN. YES is the default value.

`NO` specifies read-only access using FOCUS MODIFY and MAINTAIN.

`KEYS = n`

Indicates how many columns constitute the primary key. Is a value from 0 to 64. Zero is the default.

`KEYORDER = sequence`

Indicates the primary key sort sequence. Valid values are as follows:

LOW indicates ascending primary key sort sequence. LOW is the default.

`ASC` is a synonym for LOW.

`HIGH` indicates descending primary key sort sequence.

`DESC` is a synonym for HIGH.

Reference: TABLENAME

The TABLENAME attribute specifies the RDBMS table name. This name may have multiple parts depending on the specific RDBMS.

- **DB2 RDBMS**

```
TABLENAME = [location.][creator.]table
```

where:

location

Is the DB2 subsystem location name for the Distributed Data Facility, 16 characters maximum.

creator

Defaults to the current authorization ID if not specified. Eight characters maximum.

table

Is the name of the RDBMS table or view, 18 characters maximum.

The TABLENAME attribute identifies the RDBMS table or view. It should contain both the creator ID and the table name. If not specified, the creator defaults to the current authorization ID.

The maximum length for a fully-qualified table name is 44. For a discussion of FOCUS support for the DB2 Distributed Data Facility, see *Additional Topics* on page 417. All names must conform to the rules for identifiers stated in the appropriate RDBMS manual.

- **Teradata RDBMS**

```
TABLENAME = [databasename.]table
```

where:

databasename

Is the name of the database where the table resides, 30 characters maximum. The default is the database name assigned to your Teradata logon ID, provided one exists.

table

Is the name of the table of view, 30 characters maximum.

In Teradata, the maximum length of a fully-qualified table name is 61 characters including the required period (.).

Database and table name may consist of uppercase letters (A through Z) and digits (0 through 9). Special characters, dollar sign ($), pound sign (#), and underscore (_) are also permitted.

- **IDMS/SQL RDBMS**

```
TABLENAME = [schema.]table
```

where:

schema

Is the IDMS SQL schema name for the table or view. Eight characters maximum. If not specified, IDMS searches for a temporary table definition for the named table. If that does not exist, IDMS uses the current schema in effect for the current user session.

table

Is the name of the table or view, 17 characters maximum.

The maximum IDMS length for a fully qualified table name is 25. All names must conform to the CA-IDMS rules for identifiers.

- **Oracle RDBMS**

```
TABLENAME = [creator.]tablename[@dblinkname]
```

where:

creator

Is the Oracle userid, up to 30 characters in length.

tablename

Is the name of the Oracle table being described, up to 30 characters in length.

dblinkname

Is a valid DATABASE LINK defined in the currently connected Oracle database server.

Note:

- If any part of the TABLENAME begins with a number or special character or contains special characters, enclose it in double quotation marks.
- If any part of the TABLENAME includes a dollar sign ($), enclose that part in double quotation marks, and enclose the entire TABLENAME value in single quotation marks. For example

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

Reference: DBSPACE

The DBSPACE attribute is an RDBMS-specific storage area for the table used by the CREATE FILE or HOLD FORMAT DB2, SQLDBC, SQLORA, or SQLIDMS commands.

- **DB2 RDBMS**

  DBSPACE = databasename.tablespacename

  or

  DBSPACE = DATABASE *databasename*

  The storage areas identified by the DBSPACE attribute are called tablespaces in DB2. The IBM default value is DSNDB04, a public database. (DB2 automatically generates a tablespace in DSNDB04.)

- **Teradata RDBMS** N/A

- **IDMS/SQL RDBMS**

  DBSPACE = *segment.area*

  where:

  segment

  Is the IDMS SQL segment name to be used for the CREATE TABLE DDL resulting from a CREATE FILE or HOLD FORMAT SQLIDMS command. If not specified, IDMS uses the default area associated with the schema.

area

Is the IDMS SQL segment name to be used for the CREATE TABLE DDL resulting from a CREATE FILE or HOLD FORMAT SQLIDMS command. If not specified, IDMS uses the default area associated with the schema.

- **Oracle RDBMS**

  DBSPACE = *tablespacename*

Note:

- The DBSPACE attribute is ignored for all operations except FOCUS CREATE FILE or HOLD FORMAT SQL*engine*.
- The adapter may have been installed with a default DBSPACE setting that is used if you do not either specify the DBSPACE attribute or issue a SET DBSPACE command.
- You can also declare storage areas by issuing the SET DBSPACE environmental command prior to CREATE FILE (see *Adapter Commands* on page 309).

The Access File DBSPACE attribute overrides both the SET command and the installation default.

Reference: WRITE

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

```
WRITE = {YES|NO}
```

where:

YES

Specifies read and write access using FOCUS MODIFY and MAINTAIN. YES is the default value.

NO

Specifies read-only access using FOCUS TABLE, MAINTAIN, and MODIFY. You can use MODIFY or MAINTAIN read-only functions, such as MATCH, NEXT, CRTFORM, or WINFORM, to display rows.

Note:

- The WRITE attribute has no effect on FOCUS reporting operations.

- Regardless of the WRITE value, RDBMS security must approve all operations and activities.
- The adapter must have been installed to allow Read/Write operations. Contact your system support staff for installation options installed at your site.

Reference: KEYS

The KEYS attribute indicates how many columns constitute the primary key for the table. Acceptable values range from 0 to 64. Zero, the default, indicates that the table does not have a primary key. In the corresponding Master File, primary key columns must correspond to the first *n* fields described.

The syntax is

```
KEYS = {0|n} ,
```

where:

n

Is a value from 0 to 64. Zero is the default.

The KEYS value has the following significance in reporting operations

- When you use FOCUS FST. or LST. direct operators, the adapter 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 adapter 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 adapter 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 Adapter Optimizer* on page 171, and *Advanced Reporting Techniques* on page 213, 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 operations (see *Maintaining Tables With FOCUS* on page 349, for a detailed explanation).

Reference: 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 adapter uses the KEYORDER value when you specify FST. and LST. direct operators in report requests. The syntax is

```
KEYORDER= sequence  ,
```

where:

sequence

Indicates the primary key sort sequence. Valid values are as follows:

LOW sorts the rows in ascending primary key sequence. LOW is the default value.

ASC is a synonym for LOW.

HIGH sorts the rows in descending primary key sequence.

DESC is a synonym for HIGH.

For example, 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, DBSPACE = PUBLIC.SPACE0,$
```

The adapter requests rows ordered by SALEID and PAY_DATE in descending order. FST.PAY_DATE retrieves the most recent salary data for each employee. See the Master File for SALINFO in *File Descriptions and Tables* on page 459.

KEYORDER also determines the logical sort order for MODIFY and MAINTAIN NEXT subcommands (see *Maintaining Tables With FOCUS* on page 349).

Reference: FALLBACK (Teradata Only)

The FALLBACK attribute indicates whether a secondary copy of data is maintained in addition to the primary table. The adapter incorporates this backup operation while creating a table in response to the FOCUS CREATE FILE command. You can also specify the FALLBACK parameter in native SQL/DBC CREATE TABLE statements.

The syntax is

```
FALLBACK = {YES|NO}
```

where:

YES

Establishes a backup copy. The location is determined by Teradata and becomes available if the original copy becomes unavailable.

NO

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

Reference: CONNECTION (Oracle Only)

The CONNECTION attribute selects a specific connection name from the list of Oracle database servers declared using the SET CONNECTION_ATTRIBUTES command. This setting supersedes the default connection in any request that references the Master and Access File pair. SERVER is a synonym for CONNECTION, and is supported for compatibility with earlier releases.

```
CONNECTION=connection_name
```

where:

connection_name

Is the TNSNAME used to connect to an Oracle database server. This name must have been previously referenced in a SET CONNECTION_ATTRIBUTES command. See *Connection, Authentication, and Security* on page 45, for information about the SET CONNECTION_ATTRIBUTES command.

Field Declarations (DB2 Only)

Access File field declarations define the precision type for DECIMAL fields.

The OCCURS Segment

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

The 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 within the group instead of by separate names.

To define an OCCURS segment, you need

- An additional segment declaration in the Master File 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 Master File. 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:

segname

Is the name of the OCCURS segment, up to eight characters.

parent

Is the SEGNAME value of the table that contains declarations for the repeating fields.

field

Is the name of the first repeating field in the parent table.

nnnn

Is the number of repeating fields. Acceptable values range from 1 to 4095.

Note:

- The OCCURS segment declaration in the Master File must not include the SEGTYPE attribute.
- 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 adapter optimization.

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

```
FILENAME=SALARY, SUFFIX=DB2,$
SEGNAME=SALARY,  SEGTYPE=S0,$
 FIELD=EMPID,    ALIAS=EMPID,    USAGE=A7,   ACTUAL=A7,$
 FIELD=EMPNAME,  ALIAS=EMPNAME,  USAGE=A10,  ACTUAL=A10,$
 FIELD=SALARY,   ALIAS=PAY,      USAGE=P9.2, ACTUAL=P4,$
 FIELD=DEDUCT1,  ALIAS=DEDUCT1,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT2,  ALIAS=DEDUCT2,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT3,  ALIAS=DEDUCT3,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT4,  ALIAS=DEDUCT4,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT5,  ALIAS=DEDUCT5,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT6,  ALIAS=DEDUCT6,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT7,  ALIAS=DEDUCT7,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT8,  ALIAS=DEDUCT8,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT9,  ALIAS=DEDUCT9,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT10, ALIAS=DEDUCT10, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT11, ALIAS=DEDUCT11, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT12, ALIAS=DEDUCT12, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
SEGNAME=OCC, PARENT=SALARY, POSITION=DEDUCT1, OCCURS=12,$
 FIELD=TAX,      ALIAS=TAXDEDUC, USAGE=P9.2, ACTUAL=P8, 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 = "USER1"."SALARY", KEYS = 1,
WRITE = NO,   DBSPACE = PUBLIC.SPACE0,$
```

The ORDER Field

The ORDER field is a virtual counter that assigns a sequence number to each field within 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.

I*n*

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:

```
 FILENAME=SALARY, SUFFIX=DB2,$
 SEGNAME=SALARY,  SEGTYPE=S0,$
  FIELD=EMPID,     ALIAS=EMPID,     USAGE=A7,    ACTUAL=A7,$
  FIELD=EMPNAME,   ALIAS=EMPNAME,   USAGE=A10,   ACTUAL=A10,$
  FIELD=SALARY,    ALIAS=PAY,       USAGE=P9.2,  ACTUAL=P4,$
  FIELD=DEDUCT1,   ALIAS=DEDUCT1,   USAGE=P9.2,  ACTUAL=P8,  MISSING=ON,$
  FIELD=DEDUCT2,   ALIAS=DEDUCT2,   USAGE=P9.2,  ACTUAL=P8,  MISSING=ON,$
  FIELD=DEDUCT3,   ALIAS=DEDUCT3,   USAGE=P9.2,  ACTUAL=P8,  MISSING=ON,$
  FIELD=DEDUCT4,   ALIAS=DEDUCT4,   USAGE=P9.2,  ACTUAL=P8,  MISSING=ON,$
  FIELD=DEDUCT5,   ALIAS=DEDUCT5,   USAGE=P9.2,  ACTUAL=P8,  MISSING=ON,$
  FIELD=DEDUCT6,   ALIAS=DEDUCT6,   USAGE=P9.2,  ACTUAL=P8,  MISSING=ON,$
  FIELD=DEDUCT7,   ALIAS=DEDUCT7,   USAGE=P9.2,  ACTUAL=P8,  MISSING=ON,$
  FIELD=DEDUCT8,   ALIAS=DEDUCT8,   USAGE=P9.2,  ACTUAL=P8,  MISSING=ON,$
  FIELD=DEDUCT9,   ALIAS=DEDUCT9,   USAGE=P9.2,  ACTUAL=P8,  MISSING=ON,$
  FIELD=DEDUCT10,  ALIAS=DEDUCT10,  USAGE=P9.2,  ACTUAL=P8,  MISSING=ON,$
  FIELD=DEDUCT11,  ALIAS=DEDUCT11,  USAGE=P9.2,  ACTUAL=P8,  MISSING=ON,$
  FIELD=DEDUCT12,  ALIAS=DEDUCT12,  USAGE=P9.2,  ACTUAL=P8,  MISSING=ON,$
 SEGNAME=OCC,  PARENT=SALARY,  POSITION=DEDUCT1,  OCCURS=12,$
  FIELD=TAX,       ALIAS=TAXDEDUC,  USAGE=P9.2,  ACTUAL=P8,  MISSING=ON,$
  FIELD=ORDER,     ALIAS=ORDER,     USAGE=I4,    ACTUAL=I4,$
```

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

In this example, a DEFINE command assigns the month to each counter value. You can specify a temporary field before the report request or in the Master File

```
define file salary
month/a3=decode order(1 'jan' 2 'feb' 3 'mar' 4 'apr' 5 'may'
6 'jun' 7 'jul' 8 'aug' 9 'sep' 10 'oct' 11 'nov' 12 'dec' else '   ');
end
table file salary
print last_name tot.tax if month eq 'jan'
end
```

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

```
table file salary
print empeid last_name tax
if order eq 12
end
```

Chapter 5

Multi-Table Structures

This chapter describes how to use FOCUS file descriptions to create joins between tables.

For information about creating joins using the dynamic JOIN command, see *Advanced Reporting Techniques* on page 213.

In this chapter:

Types of Embedded Joins

With the relational adapters, you can describe two or more related tables in a single Master File. This type of multi-table structure is called an *embedded join*. There are two types of embedded joins:

- The equijoin describes the related tables based on common field values in both tables. This type of embedded join is described in a multi-table Master File and an accompanying multi-table Access File.
- The conditional (or WHERE-based) join describes how to relate rows from the two tables based on any condition. In this type of embedded join, the Master File for one table contains a cross-reference to the Master File for the other table. The conditional embedded join does not require a multi-table Access File.

In an embedded equijoin, 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 Master and Access File pair can relate up to 1024 separate tables in this manner.

Multi-table Master and Access Files describe the relationships between tables. The adapter implements these relationships at run time by matching values in fields common to two or more tables for an equijoin, or by applying the specified conditions for a conditional join. A report or maintenance procedure can refer to any or all of the tables included in the multi-table description.

In this chapter, 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.

Note: This chapter describes manual methods for creating Master File and Access Files. *Automated Procedures* on page 113 describes an automated method for creating Master and Access Files.

Advantages of Multi-table Structures

The following are advantages of defining multiple tables in a single Master File:

- 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 logical view of the data tailored to contain only those columns that should be seen by a user. The DBA security feature can define additional levels of security.
- Tables described in a multi-table Master File can be maintained together using the FOCUS MODIFY and Maintain facilities and can take advantage of FOCUS referential integrity provided by the adapter. (Refer to *Maintaining Tables With FOCUS* on page 349 for a description of MODIFY, Maintain, and FOCUS referential integrity.)
- TABLE requests access only those RDBMS tables that contain columns referenced (either explicitly or implicitly) in the request.

  A TABLE request that references a dynamically joined structure generates SQL join predicates for all segments in the subtree that starts from the root segment. Multi-table Master Files do not necessarily generate these predicates. In a multi-table structure, the subtree effectively begins with the highest referenced segment. This difference may cause identical TABLE requests to produce different reports when run against a dynamic join structure and a multi-table Master File that represent the same tree structure. See *Advanced Reporting Techniques* on page 213 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 Master File. An embedded equijoin also requires that the join fields be specified in a single Access File.

Multi-table Master Files

All segment declarations in a multi-table Master File must describe RDBMS tables or views. Each segment represents one table or view, up to a total of 1024 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 Master Files (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 Master Files, and use the CRFILE attribute in the other Master Files to access the existing description. For a full explanation of remote segment descriptions, see *Additional Topics* on page 417.

Syntax: How to Define an Equijoin in the Master File

An embedded equijoin uses the PARENT segment attribute to describe the relationships between tables

```
FILENAME=mtname, SUFFIX=sqlsuffix   [,$]
 SEGNAME=table1, SEGTYPE= {S0|KL} [,CRFILE=crfile1] [,$]
  FIELD=name,...,$
  .
  .
  .
 SEGNAME=table2, SEGTYPE=relationship, PARENT=table1
[,CRFILE=crfile2][,$]
  FIELD=name,...,$
  .
  .
  .
```

where:

mtname

Is the one- to eight-character name of the multi-table Master File.

sqlsuffix

Is one of the following values: SQLDS, SQLIDMS, SQLDBC, SQLORA.

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 Master File that contains the full definition of the columns in the RDBMS table (see *Additional Topics* on page 417).

name

Is any field name.

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 Master File that contains the full definition of the columns in the RDBMS table.

relationship

Indicates the type of relationship between the table and its parent. Valid values are:

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 Master File that contains the full definition of the columns in *table1*.

crfile2

References a remote segment description. Indicates the name of the Master File that contains the full definition of the columns in *table2*.

Syntax: How to Define a Conditional Join in the Master File

Conditional joins in the Master File are supported between relational data sources only. The conditions are considered virtual fields in the Master File.

```
FILENAME=mtname, SUFFIX=sqlsuffix   [,$]
 SEGNAME=table1, SEGTYPE= {S0|KL} [,CRFILE=crfile1] [,$]
  FIELD=name,...,$
   .
   .
   .
SEGNAME=seg,  SEGTYPE=styp,  PARENT=parseg,
        [CRFILE=xmfd,] [CRSEG=xseg,]
        JOIN_WHERE=expression; ,$
```

where:

mtname

Is the one- to eight-character name of the multi-table Master File.

sqlsuffix

Is one of the following values: SQLDS, SQLIDMS, SQLDBC, SQLORA.

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 Master File that contains the full definition of the columns in the RDBMS table (see *Additional Topics* on page 417).

name

Is any field name.

seg

Is the segment name for the joined segment. Only this segment participates in the join, even if the cross-referenced Master File describes multiple segments.

styp

Is the segment type for the joined segment. Can be DKU, DKM, KU, or KM as with traditional cross-references in the Master File.

Note: If you specify a unique join when the relationship between the host and cross-referenced files is one-to-many, the results will be unpredictable.

parseg

Is the parent segment name.

xmfd

Is the cross-referenced Master File.

xseg

Is the cross-referenced segment, if *seg* is not the same name as the SEGNAME in the cross-referenced Master File.

expression

Is any expression valid in a DEFINE FILE command. All of the fields referenced the expression must lie on a single path.

Reference: SEGNAME

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

Reference: SEGTYPE

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

- SEGTYPE=S0 (S-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 is used for an embedded equijoin and 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.)
- SEGTYPE DKU or KU indicates a dynamic unique conditional join.
- SEGTYPE DKM or KM indicates a dynamic non-unique conditional join.

Reference: PARENT

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.

Reference: CRFILE

Specify the CRFILE attribute only if the actual description of the columns in the table is stored in another (remote) Master File or to create a conditional join. For a remote segment description, the CRFILE value must be the name of the Master File that contains the full definition of the columns in the RDBMS table. For a complete discussion of remote segment descriptions, see *Additional Topics* on page 417.

Reference: FIELD

Field names can consist of up to 66 alphanumeric characters. Within the Master File, field names cannot include qualifiers. Column names are acceptable values if they meet the following 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 field name displays as the default column title for reports, select a name that is representative of the data. In TABLE, GRAPH, and MODIFY requests, you can specify field names, aliases, or a unique truncation of either. MAINTAIN does not support alias names or truncated names. In all requests, you can qualify a field name with its file name and/or segment name.

Field names must be unique within a single segment. If field names are duplicated across segments, use the segment name as a qualifier when referencing them in requests.

Reference: ALIAS

The ALIAS value for each field must be the full SQL column name (the adapter uses it to generate SQL statements). The ALIAS name must be unique within the segment. DB2 permits a maximum of 18 alphanumeric characters, Teradata and Oracle permit 30, and IDMS SQL permits 32. The ALIAS name must comply with the same naming conventions described for field names.

ALIAS names may be duplicated within the Master File if they are defined for different tables.

Example: Specifying an Embedded Equijoin in a Master File

The following Master File relates the DB2 tables EMPINFO and COURSE. The EMPINFO table is described first. Therefore, its segment declaration does not include the PARENT attribute. In the COURSE segment declaration, the PARENT attribute identifies EMPINFO as the parent and notifies the adapter 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 Master File with a remote segment, refer to *Additional Topics* on page 417):

```
FILENAME=ECOURSE     ,SUFFIX=SQLDS, $
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=DPT          ,USAGE=A10   ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL         ,USAGE=P9.2  ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE  ,ALIAS=CJC          ,USAGE=A3    ,ACTUAL=A3,$
 FIELD=ED_HRS        ,ALIAS=OJT          ,USAGE=F6.2  ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN    ,ALIAS=BONUS_PLAN   ,USAGE=I4    ,ACTUAL=I4,$
SEGNAME=COURSE ,SEGTYPE=S0  ,PARENT=EMPINFO,$
 FIELD=CNAME    ,ALIAS=COURSE_NAME ,USAGE=A15, ACTUAL=A15,$
 FIELD=WHO      ,ALIAS=EMP_NO      ,USAGE=A9,  ACTUAL=A9,$
 FIELD=GRADE    ,ALIAS=GRADE       ,USAGE=A1,  ACTUAL=A1, MISSING=ON,$
 FIELD=YR_TAKEN,ALIAS=YR_TAKEN    ,USAGE=A2,  ACTUAL=A2,$
 FIELD=QTR      ,ALIAS=QUARTER     ,USAGE=A1,  ACTUAL=A1,$
```

Note:

- The join is based on matching values between a field in the parent segment and a field in the child segment. The join fields are identified in a multi-table Access File, which is described in *Multi-table Access Files* on page 108.

- Multi-table Master Files used in MODIFY and Maintain procedures invoke FOCUS referential integrity. Refer to *Maintaining Tables With FOCUS* on page 349 for a discussion of referential integrity before deciding whether to use the same file descriptions for both reporting and maintenance procedures.

Example: Specifying a Conditional Join in a Master File

The following Master File for the EMPINFO table (defined solely for the purpose of this example and not included in *File Descriptions and Tables* on page 459) contains a conditional join to the PAYINFO table. The conditions create a join that includes all employees who received a salary increase within six months of being hired:

```
FILENAME=EMPINFO ,SUFFIX=SQLDS,$
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=DPT          ,USAGE=A10     ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL          ,USAGE=P9.2    ,ACTUAL=P4      ,$
 FIELD=CURR_JOBCODE   ,ALIAS=CJC          ,USAGE=A3      ,ACTUAL=A3      ,$
 FIELD=ED_HRS         ,ALIAS=OJT          ,USAGE=F6.2    ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN     ,ALIAS=BONUS_PLAN   ,USAGE=I4      ,ACTUAL=I4      ,$
FIELD=HIRE_DATE_TIME ,ALIAS=HDTT         ,USAGE=HYYMDm ,ACTUAL=HYYMDm ,$
FIELD=HIRE_TIME      ,ALIAS=HT           ,USAGE=HHIS    ,ACTUAL=HHIS    ,$

SEGNAME=PAYINFO, SEGTYPE=KM, PARENT = EMPINFO,
CRFILE = PAYINFO,
JOIN_WHERE = DAT_INC GT HIRE_DATE AND DAT_INC LT ( HIRE_DATE + 182);
JOIN_WHERE = EMP_ID EQ PAYEID; ,$
```

Each table has a single segment Access File. No other Access File is needed.

The following report shows the employees included in the join:

```
TABLE FILE EMPINFO
PRINT SALARY HIRE_DATE DAT_INC
BY LAST_NAME BY FIRST_NAME
END
```

The output is:

```
LAST_NAME       FIRST_NAME          SALARY   HIRE_DATE  DAT_INC
---------       ----------          ------   ---------  -------
CROSS           BARBARA          $27,062.00  81/11/02   82/04/09
GREENSPAN       MARY              $9,000.00  82/04/01   82/06/11
IRVING          JOAN             $26,862.00  82/01/04   82/05/14
JONES           DIANE            $18,480.00  82/05/01   82/06/01
MCKNIGHT        ROGER            $16,100.00  82/02/02   82/05/14
SMITH           RICHARD           $9,500.00  82/01/04   82/05/14
```

Multi-table Access Files

A multi-table Master File indicates that tables are related. However, to implement an equijoin (TABLE) or FOCUS referential integrity (MODIFY and MAINTAIN), you must identify their common fields to FOCUS in the corresponding Access File.

Note:

- A conditional join does not use a multi-table Access File.
- The KEYFLD and IXFLD attributes that specify the matching fields for an equijoin should not be used when doing a conditional join with JOIN_WHERE.

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

Each segment declaration in the Access File must contain the required keywords described in *Describing Tables to FOCUS* on page 55.

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 Master File.

Syntax: How to Create a Multi-table Access File

```
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 Master File.

table2

Is the SEGNAME of the related table from the multi-table Master File.

tname1, tname2

Are the names of the parent (*tname1*) and related (*tname2*) tables:

- For DB2, `tablename` or `creator.tablename`. For the DB2 Distributed Data Facility, include a subsystem location identifier (see *Additional Topics* on page 417).
- For Teradata, `tablename` or `databasename.tablename`.
- For IDMS/SQL, `tablename` or `schema.tablename`.
- For Oracle, *`userid.tablename`*.

pkfield

Is the fieldname of the primary key column in the parent (*table1*) table.

Note: This attribute is not supported for a conditional join in the Master File (JOIN_WHERE).

fkfield

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

Note: This attribute is not supported for a conditional join in the Master File (JOIN_WHERE).

Reference: KEYFLD and IXFLD

The KEYFLD and IXFLD attributes 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 data type. 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 data type. If their length is the same, the RDBMS handles the join more efficiently.

In the ECOURSE example from *Specifying an Embedded Equijoin in a Master File*, the fields EMP_ID in EMPINFO and WHO 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 = "USER1"."EMPINFO", KEYS = 1,
   WRITE = YES,  DBSPACE = PUBLIC.SPACE0,$
SEGNAME = COURSE, TABLENAME = "USER1"."COURSE", KEYS = 2,
   WRITE = YES,     DBSPACE = PUBLIC.SPACE0,
     KEYFLD = EMP_ID, IXFLD = WHO,$
```

Multi-field Embedded Equijoins

In relational systems, a relationship or link between tables can depend on multiple columns. Embedded joins defined in multi-table Master and Access Files also provide this ability. (The dynamic JOIN command supports this feature as well. See *Advanced Reporting Techniques* on page 213).

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

Syntax: How to Implement a Multi-Field Embedded Equijoin

```
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 Master File.

name2

Is the SEGNAME of the related table from the multi-table Master File.

table1, table2

Are the names of the parent (*table1*) and related (*table2*) tables:

- For DB2, `tablename` or `creator.tablename`. For the DB2 Distributed Data Facility, also include a subsystem location identifier (see *Additional Topics* on page 417).
- For Teradata, `tablename` or `databasename.tablename`.
- For IDMS/SQL, `tablename` or `schema.tablename`.
- For Oracle, `userid.tablename`.

pkfield1/pkfield2

Are the field names that compose the primary key in the parent (or host) table.

fkfield1/fkfield2

Are the field names 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 within the table or the FOCUS Master File.

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.

Example: Creating a Multi-Field Embedded Equijoin

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

The ECOURSE1 Master File (defined solely for the purpose of this example and not included in *File Descriptions and Tables* on page 459) reflects the new fields:

```
FILENAME=ECOURSE1          ,SUFFIX=SQLDS,$
SEGNAME=EMPINFO1           ,SEGTYPE=S0,$
 FIELDNAME=LAST_NAME       ,ALIAS=LN              ,USAGE=A15  ,ACTUAL=A15,$
 FIELDNAME=FIRST_NAME      ,ALIAS=FN              ,USAGE=A10  ,ACTUAL=A10,$
 FIELDNAME=HIRE_DATE       ,ALIAS=HDT             ,USAGE=YMD
,ACTUAL=DATE,$
 FIELDNAME=DEPARTMENT_CD   ,ALIAS=DEPARTMENT_CD   ,USAGE=A10  ,ACTUAL=A10,
   MISSING=ON ,$
 FIELDNAME=CURRENT_SALARY  ,ALIAS=CURRENT_SALARY  ,USAGE=P9.2 ,ACTUAL=P4, $
 FIELDNAME=CURR_JOBCODE    ,ALIAS=CJC             ,USAGE=A3   ,ACTUAL=A3,$
 FIELDNAME=ED_HRS          ,ALIAS=OJT             ,USAGE=F6.2 ,ACTUAL=F4,
   MISSING=ON ,$
 FIELDNAME=BONUS_PLAN      ,ALIAS=BONUS_PLAN      ,USAGE=I4   ,ACTUAL=I4,$
SEGNAME=COURSE1            ,SEGTYPE=S0            ,PARENT=EMPINFO1, $
 FIELD=CNAME               ,ALIAS=COURSE_NAME     ,USAGE=A15  ,ACTUAL=A15,$
 FIELD=LNAME               ,ALIAS=LNAME           ,USAGE=A15  ,ACTUAL=A15,$
 FIELD=FNAME               ,ALIAS=FNAME           ,USAGE=A10  ,ACTUAL=A10,$
 FIELD=GRADE               ,ALIAS=GRADE           ,USAGE=A1   ,ACTUAL=A1,
   MISSING=ON ,$
 FIELD=YR_TAKEN            ,ALIAS=YR_TAKEN        ,USAGE=A2   ,ACTUAL=A2, $
 FIELD=QTR                 ,ALIAS=QUARTER         ,USAGE=A1   ,ACTUAL=A1, $
```

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

```
SEGNAME = EMPINFO1 ,TABLENAME = "USER1"."EMPINFO1" ,KEYS = 2,WRITE = YES,
   DBSPACE = PUBLIC.SPACE0,$
SEGNAME = COURSE1  ,TABLENAME = "USER1"."COURSE1"  ,KEYS = 3,WRITE = YES,
   DBSPACE = PUBLIC.SPACE0,
      KEYFLD=LAST_NAME/FIRST_NAME,IXFLD=LNAME/FNAME, $
```

Chapter 6

Automated Procedures

This chapter describes the following procedures:

- The AUTODB2 facility for DB2 creates Master and Access Files for existing DB2 tables.
- The AUTODBC facility creates Master and Access Files for existing Teradata tables.
- The FOCUS CREATE FILE command creates an RDBMS table from the description in an existing Master and Access File.
- The CREATE SYNONYM command, which can be used to generate a Master and Access File (synonym) for any relational data source.

The AUTO facilities and CREATE SYNONYM command use RDBMS catalog definitions of existing tables or views, along with user selections, to generate Master and Access Files (see *Creating File Descriptions* on page 114). The AUTODB2 and AUTODBC facilities and CREATE SYNONYM command work with any RDBMS release.

Note: Do not use a Master File created (by AUTODB2, AUTODBC, or CREATE SYNONYM) in one FOCUS release together with an Access File created in a different FOCUS release.

The FOCUS CREATE FILE command uses existing Master and Access Files to generate an RDBMS table definition. CREATE FILE also generates a unique index if the KEYS value in the Access File is greater than zero (see *Creating Tables: The CREATE FILE Command* on page 162).

In this chapter:

Creating File Descriptions

This topic discusses the AUTODB2 facility for creating Master and Access Files for DB2 tables on z/OS and the AUTODBC facility for creating Master and Access Files for Teradata tables.

AUTODB2

When you invoke the AUTODB2 facility, it presents you with a series of menus that prompt you to identify the tables you want to describe and the relationships between them. If you are unfamiliar with any of the terms used in this section, review *Multi-Table Structures* on page 99. Subsequent sections explain each menu in detail, and *AUTODB2 Sample Session* on page 146 contains a sample session.

The following is a brief overview of the AUTODB2 screens:

- The initial screen, or Main Menu, allows you to name your Master and Access Files and to identify the tables that may contain columns needed for defining them. You can use wildcard characters to generate a list of tables from which to choose. You can also change certain default values that are displayed on this screen.
- The Table Selection Screen displays the list of tables generated as a result of your entries on the Main Menu. You can select tables from the list displayed. If you choose more than one table, you must designate one to be the root and the rest to be children.

  Note: Descendant is a synonym for child. Both terms, used interchangeably in the following discussion, refer to tables related by embedded equijoins, as described in *Multi-Table Structures* on page 99.

  If you select just a root table, and no children, on the Table Selection Screen, AUTODB2 creates your single-table Master and Access File at this point. If you choose at least one child table, you proceed to the Child Selection Screen.
- The Child Selection Screen prompts you to identify pairs of related tables.
- The Common Column Selection Screen asks you to identify the primary and foreign key columns for each pair of related tables (see *Multi-Table Structures* on page 99, for an explanation of multi-table structures).

After you describe all the relationships, AUTODB2 creates the Master and Access Files. *How to Use AUTODB2* on page 116 contains sample screens and explains AUTODB2 defaults.

You can use the resulting pair of file descriptions immediately or edit them to include such options as DEFINE fields. As with any Master File, you can add additional security by including FOCUS DBA security attributes.

Note: If a table has a Master File created in a prior FOCUS release, and if you re-run AUTODB2 on the table, some field names and/or USAGE formats generated in the new Master File may differ from those in the old Master File. As a result, requests that ran against the old Master File, and DEFINE fields based on field length, may require changes.

The RDBMS system catalog table, SYSCOLUMNS, provides column information such as column names, data types, column lengths, and whether null values are allowed. System catalog tables SYSKEYS and SYSINDEXES provide unique index information. The RDBMS searches for the first unique index created and uses the columns on which this index is defined as the primary key.

In addition to the ability to create multi-table Master and Access Files in interactive mode (batch mode remains limited to one table description per execution), the AUTODB2 facility includes the following three key features:

- It provides an option on the Main Menu for assigning a USAGE attribute of either x or x.0 to table columns of data type DECIMAL(x,0).
- It provides an option on the Main Menu for changing default Main Menu entries and logging the new default values to a text file for repeated use.
- It provides an extensive on-line help facility.

AUTODB2 uses only FOCUS DYNAM dynamic allocation, freeing it from any dependence on TSO.

You can execute AUTODB2 in batch mode for single-table structures by supplying the necessary execution parameters at invocation, eliminating the need for input screens. For AUTODB2, batch mode execution is possible in the TSO environment as well as in z/OS batch. See *AUTODB2 in Batch Mode* on page 129 for an explanation of batch execution.

AUTODB2 Support for DDF

AUTODB2 supports retrieval from and Master File creation for tables from secondary locations. When a location is specified, the TABLENAME attribute in the Access File consists of three parts: *location.creator.tablename*.

During installation, a list of valid locations is included in the AUTODB2 utility. The list appears on the main menu, just below the row in which the user can enter the location. The first value in the list appears by default in the main menu. The user can enter any one of the values in the list. Entering a value not on the list generates the following error message:

```
PLEASE ENTER A VALID LOCATION.
```

Table, column, and index information is retrieved from the catalog tables for the selected location. The location is included in a three-part table name in the Access File if it is not blank.

Note: The menu option 'Use Creator Name in AFD?=N' is ignored when a non-blank location is provided. That is, the creator is included in the three-part table name even when you specify to not include it on the main menu.

The value for location that you include on the menu is logged to the parameter log file when you press PF4 to log parameters. The next session of AUTODB2 will display the logged value.

Procedure: How to Install AUTODB2 DDF Support

Create a list of valid locations by editing the following lines in the AUTODB2 FOCEXEC. Uncomment the -DEFAULT command and enter up to six location names. The start and end of the list must be enclosed in single quotation marks, and each name must be eight characters long, padded on the right with blanks if necessary:

```
-*=====================================================================
-* Change &LOC_LIST to contain possible location values, padded to 8
-* characters, separated by commas.  The entire string is enclosed in
-* single quotes.  Include a maximum of 6 locations.  The first entry is
-* the default. Example:
-* -DEFAULT &LOC_LIST='PROD    ,TEST    ,DEVELOPM'
-*                    <------>,<------>,<------>,<------>,<------>,<------>
-DEFAULT &LOC_LIST='
```

A blank location name designates the location where the plan was bound.

Example: Creating a Location List

With the following list, blank will be the default value that appears on the main menu.

```
-DEFAULT &LOC_LIST='        ,LOCDSNA ,LOCDSNC '
```

Note: Specifying blank (' ') as the first option indicates that the default is the location where the plan was bound.

How to Use AUTODB2

Before starting AUTODB2, it is helpful to know the names of the tables or views you will be using (including, if possible, their creator names and their database and location names). Having this information in advance can avert a costly search of the RDBMS catalogs using wildcard characters from the Main Menu. Get this information from your RDBMS database administrator, or query the system table SYSCOLUMNS using Master File DB2CAT. (See Appendix A, Additional Topics, for a sample request.)

The first time you execute AUTODB2, many of the entry fields on the Main Menu screen display default values. You can customize the default values for your application and use PF4 to store them in a parameter log file for future use. The PFkey functions are explained following the description of the Main Menu entry fields.

You also need enough available disk space. AUTODB2 writes the file descriptions directly to the data sets specified on the Main Menu.

To start AUTODB2, enter the FOCUS environment and issue the following command from the FOCUS command level:

```
EX AUTODB2
```

Press the *Enter* key.

Note: To allow for the display of the maximum number of characters in the names of tables and other objects, additional information about these objects is displayed on lines below the list of names. For example, on the following screen, the formats for each table display on separate lines below the list of tables. On the left half of the screen, the format for the EID column is A9, for the LN column is A15, for the FN column is A10, and for the HDT column is YYMD:

```
Master:       Master File Generation Facility for DB2
AUTOEMP             ==Common Column Selection==
Number the Primary Key Columns          | Number The Corresponding Foreign
(in sequence) In The Parent Table:      | Key Columns In The Child Table:
          USER01                        |            USER01
         EMPINFO
                          ADDRESS
 Key    Column Name                     | Key     Column Name
-------------------------------------------------------------------------------
      EID                               |       EID
      LN                                |       AT
      FN                                |       LN1
      HDT                               |       LN2
    FORMAT: A9                          |    Format: A9
    FORMAT: A15                         |    Format: A4
    FORMAT: A10                         |    Format: A20
    FORMAT: YYMD                        |    Format: A20

PF1=Help  PF2=Restart  PF3=End  PF4=Skip            PF7=Up   PF8=Down
```

The Main Menu

The following is an example of the Main Menu for AUTODB2. Complete the entry fields on the Main Menu and press *Enter* to begin processing:

```
Main Menu     Master File Generation Facility for DB2
      Master Filename ================> autoemp
 Location => locdsna     Creator => USER01
 TABLE => *
 Location values:          ,LOCDSNA ,LOCDB9A
        DATABASE NAME ================> *
      Description will be a member of:
        Master Target PDS => USER01.MASTER.DATA
        Access Target PDS => USER01.FOCSQL.DATA
        FOCDEF Target PDS => USER01.FOCDEF.DATA
        Replace Existing Description?=> N       (Y/N)
        Read/Write Functionality =====> W       (R=Read,W=Write)
        Date Display Format ==========> YYMD
        Time Stamp Display Format ====> HYYMDm
        Time Display Format ==========> HHIS
        Display Decimal when SCALE=0?=> Y       (Y/N)
        Use LABEL as Column Heading? => N       (Y/N)
        Use Remarks for FOCDEF? ======> N       (Y/N)
        Use Creator Name in AFD? =====> Y       (Y/N)
        Use Long Fieldnames? =========> Y       (Y/N)
        Parm File => USER01.FOCSQL.DATA

PF1=Help PF2=Restart PF3=Exit PF4=Log PF5=MFD PF6=AFD PF9=Picture PF10=List
```

The following list describes the Main Menu entry fields:

Location

Is one of the values on the list of location values displayed immediately below this entry field. The first location on the list is displayed initially.

Table, column, and index information is retrieved from the catalog tables for the selected location. The location is included in a three-part table name in the Access File, if it is not blank.

The menu option 'Use Creator Name in AFD?=N' is ignored when a non-blank location is provided. That is, the creator is included in the three-part table name even when you specify to not include it on the main menu.

If the first option is blank (' '), the default is the location where the plan was bound.

Master Filename

Is the 1- to 8-character name you select for referring to the data in requests. This name must be a valid member name.

Creator Name (or *)

Is the 1- to 8-character creator name of the tables that you want to describe. Specify an asterisk (*) to select tables for all creators. The default is your user ID.

You can use the asterisk (*) wildcard character in the creator, table, and database name entry fields to create a list based on the provided pattern.

Note: You cannot enter a name that contains a dollar sign ($) in the creator, table name, or database name entry fields; however, with the asterisk (*) wildcard character, you can generate a list that includes names containing dollar signs. You can then choose tables from this list.

Table Name (or *)

Is the 1- to 18-character name of an existing table that you want to describe. An asterisk (*), the default, selects all tables.

Database Name (or *)

Is the 1- to 8-character name of the database that contains any or all tables you may select. An asterisk (*), the default, selects all databases. You can omit this value if you provide creator or table.

Master Target PDS

Is the fully-qualified data set name of the Master File PDS in which to store the Master File. Do not use quotation marks (single or double) in the data set name. The default is '*userid*.MASTER.DATA'.

Access Target PDS

Is the fully-qualified data set name of the Access File PDS in which to store the Access File. Do not use quotation marks (single or double) in the data set name. The default is '*userid*.FOCSQL.DATA'.

FOCDEF Target PDS

Is required only if you specify Yes (Y) in the "Use REMARKS for FOCDEF?" field. Is the fully-qualified data set name of the FOCDEF File PDS in which to store the TableTalk® Help file. Do not use quotation marks (single or double) in the data set name. The default is '*userid*.FOCDEF.DATA'.

Replace Existing Description?

Specifies whether to overwrite existing Master and Access Files (Y/N). No (N) is the default. Yes (Y) replaces existing descriptions of the same name.

Read/Write Functionality

Indicates read-only or read/write access in the Access File. Read/Write (W), the default, allows MODIFY and MAINTAIN to update the RDBMS table. Read (R) is for reporting only.
Note: If the adapter was installed for read-only access, this field displays an R and cannot be changed.

Date Display Format

Is a valid FOCUS USAGE date format for RDBMS columns described as dates. The default is YYMD.

Time Stamp Display Format

Is HYYMDm by default. A26 is also acceptable, for compatibility with prior releases.

Time Display Format

Is HHIS by default. A8 is also acceptable, for compatibility with prior releases.

Display Decimal when SCALE=0?

Yes (Y), the default, displays the decimal point for DECIMAL values with no fractional component (for example 123.). No (N) excludes the decimal point (for example, 123).

Use LABEL as Column Heading?

Selects FOCUS report column headings. No (N), the default, uses the column name for headings. Yes (Y) uses the RDBMS LABEL.

Use Remarks for FOCDEF?

Indicates whether to include RDBMS REMARKS as HELP for TableTalk. No (N), the default, excludes the remarks. Yes (Y) includes them. You must specify a FOCDEF Target PDS in order to select Y.

Use Creator Name in AFD?

Indicates whether to include the creator name in the TABLENAME attribute of the Access File. Yes (Y), the default, includes the creator name. No (N) includes only the table name.

Main Menu PFkeys

You can implement the following functions from the Main Menu with PFkeys:

Key	Function	Description
PF1	Help	Accesses on-line help.

Key	Function	Description
PF2	Refresh the List of Tables	Clears the existing list of tables maintained by AUTODB2 for this session. When you select tables from the Main Menu, information is gathered from the system catalogs. Each time you change the selection criteria, the new tables are appended to the existing list. Pressing PF2 purges this list without exiting AUTODB2. You receive the message "Enter new table selection criteria" when the list is successfully purged. No message is displayed if you have not yet created a list of tables.
PF3	Exit	Ends the AUTODB2 session and returns to FOCUS.

Key	Function	Description
PF4	Log Default Menu Parameters	Saves the values that you want to see displayed as defaults on the Main Menu in future executions of AUTODB2. Your defaults are saved in member DB2$PARM in the parm data set indicated on the menu. This data set is either the PDS pre-allocated to DDNAME DB2$PARM, '*userid*.FOCSQL.DATA', or the first data set allocated to DDNAME FOCSQL, whichever the system finds first (see *The z/OS Parameter Log File* on page 128 for more detailed information on the search order). The following information is logged: ❏ Location ❏ Creator Name ❏ Table Name ❏ Database Name ❏ MFD Partitioned Data set Name ❏ AFD Partitioned Data set Name ❏ FOCDEF Partitioned Data set Name ❏ Replace existing description ❏ Read/Write Functionality ❏ Date Display Format ❏ Time Stamp Display Format ❏ Time Display Format ❏ Display Decimal when SCALE=0 ❏ Use LABEL as Column Heading ❏ Use REMARKS for FOCDEF ❏ Use Creator Name in AFD **Note:** The values you enter must pass all validation tests in order for the new default values to be logged.

Key	Function	Description
PF5 PF6	TED MFD TED AFD	Allows you to edit, using TED, the Master (PF5) or Access (PF6) File whose name you entered (as *mastername*) in the Master Filename entry field on the Main Menu. You access member *mastername* in the data set entered as either Master Target PDS or Access Target PDS. **Note:** You can edit these files even if they were not created with AUTODB2.
PF9	Picture of MFD	Generates a diagram of the file entered in the Master Filename entry field on the Main Menu. After the picture is displayed, type any character and press *Enter* to return to the menu. To generate the picture in z/OS, the *mastername* entered as Master Filename must be a member of a data set allocated to DDNAME MASTER (the Master Target PDS is not used).
PF10	Table List	Displays a list of all tables that meet the screening criteria provided in Creator Name, Table Name, and Database Name.

Note: The PFkey options are available only if the values you enter on the screen pass all validation tests. See *Common Errors* on page 145 for a discussion of common errors. See *Using PFkeys From non-Main Menu Screens* on page 127 for PFkey options available on other screens.

After you specify the appropriate values on the initial screen, press *Enter*. AUTODB2 informs you that it is creating a list of tables with the following message:

```
**============================================================**
**    AUTODB2 is retrieving TABLE  information from catalog.   **
**    Please wait...                                          **
**============================================================**
```

It displays this message only for the first retrieval per AUTODB2 session, or when the selection criteria have changed since the previous retrieval within the session. AUTODB2 does not access the catalog a second time for the same immediate selections in a single session.

The Table Selection Screen

If you specified a location, the table selection screen displays the location parameters for the list of tables generated by your selections on the main menu.

When table information retrieval is complete, choose the tables to include in your Master and Access Files:

```
Master:       Master File Generation Facility for DB2
AUTOEMP                ==Table Selection==

Place an 'R' next to the Table to be the root of the Master.
Place  a 'C' next to all other Tables to be described as children.
Enter 'Y' next to all selected Tables that will be updated.

                                                          Select  Write
 Location  Creator                                         (R/C)  (Y/N)
 --------  -------                                         ------  -----
           USER01
           USER01
           USER01
           USER01
DB2 TABLE: ADDRESS                                                  Y
DB2 TABLE: COURSE                                                   Y
DB2 TABLE: DATETIME                                                 Y
DB2 TABLE: DEDUCT                                                   Y

PF1=Help            PF3=End  PF4=Add Tables           PF7=Up  PF8=Down
```

The Table Selection Screen displays, in alphabetical order, the names of all creator/table combinations that pass the selection criteria you provided on the Main Menu. You can choose up to 1024 tables to include in the description.

Identify the root table by placing *r* in its Select column. Choose all other tables to be included in the Master File by placing *c* in their Select columns. Indicate those tables that can be updated in this view by placing *y* in their Write columns.

Note:

- If the adapter was installed for read-only access, the Write column displays an N and cannot be changed.
- You may have to scroll through the list using the PF7 and PF8 keys in order to view all of the available tables.

The following message displays after you complete the table selections, if you selected at least one child table in addition to the root:

```
**============================================================**
**    AUTODB2 is retrieving COLUMN information from catalog.  **
**    Please wait...                                          **
**============================================================**
```

AUTODB2 retrieves column information from the catalog only the first time you select a particular table from the Table Selection Screen. You will not see this message when you make subsequent selections using the same tables. To erase this list and create a new one, return to the Main Menu with PF3, and restart with PF2.

The Child Selection Screen

When column information retrieval is complete, specify descendants, if any, for each table.

Note: You may have to scroll through the list using the PF7 and PF8 keys in order to view all of the available tables.

To give you the opportunity to identify all parent-child relationships, every table on your list, in turn, presents its own Child Selection Screen:

```
Master:       Master File Generation Facility for DB2
AUTOEMP                ==Child Selection==

Place a 'C' next to the descendants of Parent:
           USER01
 EMPINFO
 Location  Creator                                          Select (C)
 --------  -------                                         -----------
           USER01
           USER01


DB2 TABLE: ADDRESS                                              c
DB2 TABLE: PAYINFO                                              c
DB2 TABLE:
DB2 TABLE:

PF1=Help  PF2=Restart  PF3=End  PF4=None  PF5=Picture   PF7=Up  PF8=Down
```

If the parent segment named at the top of the screen has no children, press PF4. Otherwise, place *c* in the Select column for each table that is a child of the parent table named at the top of the screen, and press *Enter*. Next, AUTODB2 displays a Common Column Selection Screen for each parent-child pair so you can identify the columns they share.

The Common Column Selection Screen

For each parent-child pair, identify the primary keys from the parent table and the foreign keys from the related table on the Common Column Selection Screen:

```
Master:       Master File Generation Facility for DB2
AUTOEMP             ==Common Column Selection==
Number the Primary Key Columns          | Number The Corresponding Foreign
(in sequence) In The Parent Table:      | Key Columns In The Child Table:
           USER01                       |            USER01
          EMPINFO
                           ADDRESS
 Key    Column Name                     | Key     Column Name
-------------------------------------------------------------------------
 1    EID                                |  1    EID
      LN                                 |       AT
      FN                                 |       LN1
      HDT                                |       LN2
    FORMAT: A9                           |    Format: A9
    FORMAT: A15                          |    Format: A4
    FORMAT: A10                          |    Format: A20
    FORMAT: YYMD                         |    Format: A20

PF1=Help  PF2=Restart  PF3=End  PF4=Skip        PF7=Up  PF8=Down
```

Note: To allow for the display of the maximum number of characters in the names of tables and other objects, additional information about these objects is displayed on lines below the list of names. For example, on the following screen, the formats for each table display on separate lines below the list of tables. On the left half of the screen, the format for the EID column is A9, for the LN column is A15, for the FN column is A10, and for the HDT column is YYMD:

Number the key fields from each table in sequence (from 1 to 16). Each primary key field must have the same format as its corresponding foreign key field. Use PF4 to skip this table if it was selected in error. This does not affect previous or subsequent selections. However, the "skipped" segments appear in subsequent lists when you return to the Child Selection Screen to define any additional paths in the hierarchy.

AUTODB2 guides you in describing the table relationships in top down, left to right order. Initially, it displays the root table and all of its selected children.

Completing the Description

After you complete the Common Column Selection Screen for the root table, AUTODB2 displays a Child Selection Screen for the first child of the root. Assign children, if any, to this table.

Child selection continues down this first path until you press PF4 to select no descendants, or you exhaust the list of descendants. Only then does AUTODB2 display a Child Selection Screen for the second child of the root. This process continues until all tables have been assigned children or all possible descendants have been exhausted.

At this point, AUTODB2 generates the Master and Access Files and returns to the Main Menu. The message "DESCRIPTION CREATED" displays at the bottom of this Status Screen with an indication, if necessary, of how many duplicate field names were generated and how many unsupported data types were found.

Note: If duplicate field names were created, either edit them to be unique, or qualify them with the segment name when referencing them in requests.

For information about the newly generated file descriptions, see *Results of the Master File Generation Facilities* on page 142.

Using PFkeys From non-Main Menu Screens

For each screen other than the Main Menu and Table Selection screens, you can erase your current selections and back up one screen with PF2. Pressing PF3 returns you to the Main Menu with all selections intact. To erase your selections from the Table Selection Screen, return to the Main Menu with PF3, and then restart with PF2.

From the Child Selection Screen, use PF5 to generate a diagram of your file structure. The description is created on disk and remains there even if you end the program with PF3. After the picture is displayed, type any character and press the *Enter* key to return to the menu. To generate the picture in z/OS, you must allocate the target master PDS to DDNAME MASTER. If a member with the selected master name exists in a data set concatenated in front of the target data set, the picture is generated from that member.

Retaining the List of Master Files Generated

AUTODB2 maintains a temporary list of the Master Files generated during any one session. This list is refreshed at the beginning of each session and erased at the end of the session. You may retain the list by executing AUTODB2 (either on-line or in the background) with the following syntax

```
EX AUTODB2 MFDLIST=Y
```

The list resides in a temporary data set allocated to DDNAME AUTODB2L, with a disposition of MOD and record length of 114. It is the responsibility of the user to free or erase this file when it is no longer required. The file layout is:

Length	Columns	Description
8	1 - 8	Master Filename
8	9 - 16	Number of duplicate fieldnames

Length	Columns	Description
5	17 - 21	Number of unsupported data types
5	22 - 26	Number of long decimal fields truncated. This entry, required in prior releases, exists for upward compatibility.
44	27 - 70	Master Target PDS name
44	71 - 114	Access Target PDS name

Changing the AUTODB2 Default Data Sets

With AUTODB2, you can save custom Main Menu defaults in a parameter log file for repeated use. You can also change the default data sets that display on the Main Menu.

To change the default data set names that appear the first time you execute AUTODB2, customize the following lines of code near the top of the AUTODB2 FOCEXEC. These are the only permanent data sets that AUTODB2 uses. All other data sets are temporary and are named by the system.

```
-SET &DSNP0=&USERID ||'.FOCSQL.DATA                    ';
-SET &DSNM0=&USERID ||'.MASTER.DATA                    ';
-SET &DSNF0=&USERID ||'.FOCSQL.DATA                    ';
-SET &DSND0=&USERID ||'.FOCDEF.DATA                     ';
```

Note: You must preserve the length of the data set name for each variable (the result of concatenating the userid with the string between the single quotation marks) at 44 characters. If necessary, pad the name with blanks to maintain the correct length. You may not change any other lines in the FOCEXEC.

The z/OS Parameter Log File

The parameter log file, if there is one, is always a data set allocated to DDNAME DB2$PARM. z/OS identifies the parameter log file with the following steps:

1. If DDNAME DB2$PARM is allocated to a PDS prior to execution of AUTODB2, AUTODB2 uses member DB2$PARM as the parameter log file. If the member does not exist, AUTODB2 creates it. This allows the user to identify the "profile" data set prior to execution of AUTODB2 and is recommended for sites that have non-standard data set naming conventions.

   If this DDNAME is allocated to a sequential data set, AUTODB2 frees it, generates a message, and disables parameter logging.

If DDNAME DB2$PARM is not allocated, AUTODB2 continues searching and attempts to allocate it in the steps that follow. AUTODB2 does not free this DDNAME upon exiting, assuming that it may be used again.

2. AUTODB2 allocates data set name '*userid*.FOCSQL.DATA', the default name provided in the code as &DSNP0, as the parameter log file. If the default has been changed, it uses the new data set name. This assumes standard Information Builders naming conventions.

   This data set must be a PDS. If it is not, AUTODB2 displays a message and disables parameter logging.

3. If the first two steps fail to identify a parameter log file, AUTODB2 allocates the first data set allocated to DDNAME FOCSQL as the parameter log file. This assumes that the user's data set is allocated first in the concatenation of data sets to DDNAME FOCSQL.

   If the data set allocated to DDNAME FOCSQL is sequential, AUTODB2 displays a message and disables parameter logging.

   Note: AUTODB2 cannot function properly if the data set is not a PDS.

Parameter logging assumes that the user has write access to the target data set. An attempt to log parameters to a data set without write access results in a security abend.

AUTODB2 Usage Notes

- AUTODB2 is a FOCEXEC that has a corresponding Master File. The FOCEXEC and Master File must be from the same release of FOCUS. If you attempt to use the old version of the Master File with the new version of the FOCEXEC, the following message is generated:

  ```
  PROGRAM AND FILE DESCRIPTION DATES DO NOT MATCH: AUTODB2 date
  UNABLE TO EXECUTE AUTODB2 - PLEASE REINSTALL
  ```

  FOCEXEC execution is terminated, and you return to the FOCUS prompt.

- AUTODB2 no longer needs the DB2CAT Master and Access Files. These Master and Access Files remain on the distribution tape. Information Builders no longer certifies the accuracy of these Master and Access Files with future RDBMS catalog table changes.

AUTODB2 in Batch Mode

You can create a single-table Master File by executing AUTODB2 in the background with an argument list that supplies the values normally entered on the Main Menu. You can invoke AUTODB2 batch mode processing in the z/OS batch or TSO environments. The syntax is

```
EX AUTODB2 BATCH=Y,MASTER=master,CREATOR=creator,TABLENAME=table
    [,option1=value1 ...]
```

where:

master

Is the 1- to 8-character name of the Master File that will be generated.

creator

Is the 1- to 8-character name of the creator of the table.

table

Is the 1- to 18-character name of the RDBMS table.

option1 ...

Is an option listed in the following chart.

value1 ...

Is an acceptable value for the corresponding option.

All options from the Main Menu are available. Specify options on the command line by entering name=value pairs separated by commas. The list can extend over several lines. The following chart presents the available options:

Option Name	Values	Description	Default
LOC	*location*	Location value	First value in the location list
REPLACE	Y=Yes,N=No	Replace existing description	N
FUNC	R=Read, W=Write	Read/Write Functionality	W
DATEDISP	*format*	Date Display Format	YYMD
TIMESTMP	*format*	Time Stamp Display Format	HYYMDm
TIME	*format*	Time Display Format	HHIS
DECIMAL	Y=Yes,N=No	Display Decimal when SCALE=0?	Y
LABELS	Y=Yes,N=No	Use Labels as Column Heading?	N

Option Name	Values	Description	Default
REMARKS	Y=Yes,N=No	Use Remarks for FOCDEF?	N
CREATAFD	Y=Yes,N=No	Use Creator Name in AFD?	Y
LONG	Y=Yes,N=No	Use Long Fieldnames?	Y
USERID	*userid*	High level qualifier of output data sets (required if target data set names not provided)	
MFDLIST	Y=Yes,N=No	Store list of Master Files created per session	N
MASTERDATA	*dsn*	Master Target PDS	&USERID.MASTER.DATA
FOCSQLDATA	*dsn*	Access Target PDS	&USERID.FOCSQL.DATA
FOCDEFDATA	*dsn*	FOCDEF Target PDS	&USERID.FOCDEF.DATA

The default for LOC is the first value in LOC_LIST (provided at installation time). If you want a value of blank, use blank as the first value in LOC_LIST or execute AUTODB2 with the following syntax:

```
EX AUTODB2 LOC=' '
```

AUTODBC

The AUTODBC facility is a TSO CLIST and is invoked from TSO. In order to generate Master and Access Files, the AUTODBC facility gathers column and table information from the ADUCOL repository. The repository is a small FOCUS database that stores information extracted from the Teradata Data Dictionary/Directory. Using AUTODBC, you can create this required repository or reuse one retained from a previous AUTODBC session.

he ADUCOL repository provides the following advantages:

- Interaction with the Teradata Data Dictionary/Directory is minimized. The Directory within the Teradata RDBMS is required for many activities and access to its services may require extended terminal response times.

- Directory information is extracted efficiently as the result of one interaction. The AUTODBC facility extracts Directory information for a potentially unlimited number of tables in a single pass. The table limit is subject to available temporary disk space.
- Users may create as many Master and Access Files as they need. An unlimited number of file descriptions may be generated based on the ADUCOL contents.
- User-supplied input is reduced. Another repository, the ADUTMP repository, is loaded as a result of creating the ADUCOL repository. The ADUTMP repository provides the default database and table names which reduces typing during the file generation process.
- Users may retain the repositories for future use. The repositories may be saved for future AUTODBC sessions during the exit process. Since these repositories are already prepared, future interaction with the Directory is unnecessary.

The file generation process is straightforward. After you invoke the AUTODBC facility and you supply your Teradata logon ID and password, the Primary Option Menu is displayed. The Primary Option Menu lists all the tasks or processes that may be performed from AUTODBC. If the repositories already exist, you can select the option that generates file descriptions and complete a series of screens. A status screen indicates successful completion of the process.

Note: You can create file descriptions that describe one or more tables or views. (See *Multi-Table Structures* on page 99.)

The resulting pair of file descriptions may be used immediately or edited to include options such as DEFINE fields. As with any Master File, additional security may be added by including FOCUS DBA security functions.

- *How to Use AUTODBC* on page 132 explains how to invoke AUTODBC and describes sample screens for most processes.
- *Results of the Master File Generation Facilities* on page 142 describes the resulting Master and Access Files.
- *Common Errors* on page 145 explains common errors.
- *AUTODBC Sample Session* on page 153 provides an AUTODBC sample session and the generated file descriptions.

How to Use AUTODBC

Before you begin your AUTODBC session, you should be aware of these requirements:

- AUTODBC must be installed.

- Proper authorization, a valid Teradata logon user ID and password. Also, the Teradata Director Program ID is unique to your site and is release-dependent.
- Database names and table names.
- The FOCUS FIDEL option. It must be available for AUTODBC.

If you have any questions about these requirements, contact your Teradata database administrator or consult the appropriate installation guide.

AUTODBC is a CLIST. Therefore, to execute the AUTODBC facility, invoke it from the TSO operating system (not from FOCUS):

```
EX AUTODBC
```

Press the *Enter* key to display the security logon screen.

Note: For TSO users, the execution syntax may be site-specific. Please ask your Teradata database administrator for the correct syntax if the command shown above does not invoke AUTODBC.

Security Logon Screen

The security logon screen, Screen A, opens after you invoke the facility or if you specify Option 4 from the Primary Option Menu, Screen B:

```
A           ------------------------------------------------------
           |                A U T O D B C                         |
           | RELATIONAL TABLE DESCRIPTOR FACILITY FOR FOCUS/DBC  |
           |       INFORMATION BUILDERS, INCORPORATED            |
            ------------------------------------------------------


          TERADATA USERID   =====>
          TERADATA PASSWORD  ====>

          TERADATA DIRECCTOR PGM  =>    0
          TERADATA PARTITION ID   =>    DBC/SQL


           SUPPLY THE REQUIRED TERADATA LOGON INFORMATION

  ENTER= PROCESS  PF3= EXIT (INITIAL ENTRY ONLY)
```

The entry fields and their acceptable values are:

USERID

Is your Teradata user ID.

PASSWORD

Is your Teradata password. (Its display is suppressed.)

DIRECTORPGMID

Is the single character value of the TDP ID. Specify a blank space instead of the default (0) if your site does not require a Teradata Director Program ID (TDPID) specification during logon.

PARTITIONID

Defaults to DBC/SQL.

These values constitute your security profile, which provides access to the Teradata Data Dictionary/Directory. Your security profile is stored in a work file that is erased when you exit the facility to prevent unauthorized use of your user ID.

Press the *Enter* key after you complete the logon screen or press PF3 to exit the facility.

Note: When you first invoke the AUTODBC facility, the PF3 key is valid for exiting from the logon screen. Otherwise, you must specify Option 5 on the Primary Option Menu to exit the facility and to delete the current session's work files. On subsequent screens, the PF3 key functions as a return key that displays a previous screen.

Primary Option Menu

The Primary Option Menu, Screen B, appears after the initial logon screen and after each completed task or process:

```
B           ------------------------------------------------
           |            A U T O D B C                       |
           |          PRIMARY OPTION MENU                   |
            ------------------------------------------------


           AVAILABLE OPTIONS:

           1. DISPLAY ADUCOL CONTENTS
           2. ADUCOL MAINTENANCE
           3. GENERATE FOCUS MASTER AND ACCESS FILE DESCRIPTION
           4. REDEFNE TERADATA SECURITY PROFILE
           5. EXIT THIS FACILITY


           OPTION ===>


  ENTER= PROCESS
```

Each task or process is assigned an option number. The options are:

1. DISPLAY ADUCOL CONTENTS

   Displays the contents (database and table names) of the ADUCOL repository.

2. ADUCOL MAINTENANCE

   Enables you to create the ADUCOL repository or add entries to it.

3. GENERATE FILE DESCRIPTIONS

   Prompts you with a series of screens to generate single- or multi-table Master and Access Files.

4. REDEFINE TERADATA SECURITY PROFILE

   Enables you to correct or change your security profile.

5. EXIT THIS FACILITY

   Enables you to save repositories and to exit the facility.

At this point, you may select tasks or processes in any order. Specify the option number and press the *Enter* key.

Subsequent sections explaining processes and sample screens are discussed by option number.

Option 1: Displaying ADUCOL Contents

In order to display the ADUCOL contents, select Option 1 from the Primary Option Menu and press the *Enter* key. The PAUSE message prompts you to press the *Enter* key again.

The AUTODBC facility produces a report of tables and views sorted by database names. FOCUS Hot Screen PF keys and commands are available. For example, you may scroll through the report or print it offline.

When you are finished, press either the *Enter* key or the PF3 key to return to the Primary Option Menu.

Option 2: Maintaining the ADUCOL

To create the ADUCOL repository or add entries to it, choose Option 2 and press the *Enter* key. The ADUCOL Maintenance screen, Screen D, appears as:

```
D           ------------------------------------------
           |            A U T O D B C                 |
           |          ADUCOL MAINTENANCE              |
            ------------------------------------------

      IDENTIFY RELATION DESCRIPTIONS TO BE REFRESHED WITHIN ADUCOL:

       DATABASENAMES                  TABLENAMES
       --------------------------     ------------------------------
=>
=>
=>
=>
=>
=>
=>
=>
=>
=>
   PF3= RETURN  PF7= TOP/REVIEW  PF8= DOWN  PF12= PROCESS
```

For each table or view, type the database name and table name in the appropriate columns. To move the cursor to a different column, use the Tab keys. To expand the input area for additional entries, scroll down using the PF8 key.

Press the PF7 key to review the entries. If the entries are correct, press the PF12 key to begin processing. When processing is successful, the Primary Option Menu is displayed again.

AUTODBC processing involves:

- Extracting Teradata table definitions from the Directory (using the Teradata BTEQ facility) and loading them into the ADUCOL repository using FOCUS. Columns that possess data types unsupported by AUTODBC are not retrieved (for example, BYTEINT, VARBYTE, and BYTE). (These data types may be added manually to generated Master Files. See *Results of the Master File Generation Facilities* on page 142.) If some of your table entries were processed during a previous AUTODBC session, the existing table definitions are replaced with new ones.
- Loading the database and table names into the ADUTMP repository.

Note:

- AUTODBC supports Teradata views larger than 50 columns and extracts column information using the DBC/SQL HELP COLUMN statement.
- The amount of writable temporary disk space determines the maximum number of table definitions loaded into the ADUCOL repository.

- Table entries cannot be deleted at this point. You may erase the ADUCOL repository during the exit process (Option 5) or selectively "blank out" tables during the file generation process (Option 3).

Option 3: Generating Master and Access Files

The AUTODBC file generation process varies, depending on the type of file description you intend to generate. If you intend to describe one table or view as a single-table Master File, you need to complete three screens. If you intend to generate a multi-table Master File (an embedded JOIN), you need to complete three screens, plus additional screens that indicate primary and foreign keys.

If the ADUCOL repository exists, select Option 3 from the Primary Option Menu and press the *Enter* key. If the repository does not exist, you must create it. (See *Option 2: Maintaining the ADUCOL* on page 135.)

The Master and Access File Generation screen, Screen E1, displays as:

```
E1          ------------------------------------------------
           |                A U T O D B C                   |
           |         MASTER AND ACCESS FILE GENERATION      |
            ------------------------------------------------


          MASTER/ACCESS FILENAME ==>

          PROCESSING OPTION =======>  1   OPTIONS:
                                              1= NEW DESCRIPTION ONLY
                                              2= NEW OR REPLACE

          USE LONG FIELD NAMES ===>   N

                  SUPPLY THE REQUIRED VALUES


 PF3= RETURN  ENTER= PROCESS
```

The entry fields and their acceptable values are:

`MASTER/ACCESS FILENAME`

Is the member name or file name for the new Master File and Access Files, up to eight characters in length.

PROCESSING OPTION

Is 1 by default. Option 2 replaces current Master and Access Files with new ones. Option 1 does not. (An error message appears for Option 1 if file descriptions exist and you may specify another name or select Option 2.)

LONG FIELDNAMES

Is no (N) by default. Field names are truncated at 12 characters, alias names are blank in the Master File, and partial field declarations appear in the Access File. Yes (Y) includes aliases up to 30 characters in the Master File. (See *Additional Topics* on page 417, for long field name limitations.)

Press the *Enter* key to continue or the PF3 key to return to the Primary Option Menu.

Another screen, Screen E2, displays and lists the contents of the ADUCOL repository.

```
E2          ------------------------------------------------
           |                A U T O D B C                   |
           |         MASTER AND ACCESS FILE GENERATION      |
            ------------------------------------------------
  IDENTIFY THOSE RELATIONS PARTICIPATING IN THE FOCUS VIEW:
   DATABASENAMES                TABLENAMES                WRITE=
   --------------------------   -------------------------  ------
=>                                                         N
=>                                                         N
=>                                                         N
=>
=>
=>
=>
=>
=>
=>
   PF3= RETURN  PF7= TOP/REVIEW  PF8= DOWN  PF12= PROCESS
```

On this screen, choose the tables or views for processing and exclude any unwanted tables. For each table, indicate WRITE functionality by specifying Y or N in the WRITE= column. The default No (N) permits read access for both reporting and MODIFY or Maintain browsing. Yes (Y) permits read-write access for MODIFY and Maintain operations.

To exclude a table and remove it from the screen, use the space bar to erase (blank out) the *database name*. This does not affect the ADUCOL repository. The tables or views that remain on this screen are described in the resulting Master and Access Files.

Before you continue, press the PF7 key to review your choices and to refresh your screen. Extra or unwanted tables are omitted from the screen. Press the PF12 key to continue.

Note:

- You may describe up to 1024 tables or views in a multi-table Master File.
- Table entries cannot be added at this point. If you attempt to do so and press the PF12 key, an error message COLUMN DATA table NOT FOUND appears.

After the tables are selected, the next screen, Screen E3, displays.

```
E3          ------------------------------------------------
           |                  A U T O D B C                 |
           |          MASTER AND ACCESS FILE GENERATION     |
            ------------------------------------------------
   IDENTIFY THE RELATIONSHIP FROM AMONG SELECTED RELATIONS:
                                                          CHILD
    DATABASENAMES                TABLENAMES                 OF
    --------------------------   ----------------------   ------
=>
=>
=>
=>
=>
=>
=>
=>
=>
=>
   PF3= RETURN  PF7= TOP/REVIEW  PF8= DOWN  PF12= PROCESS
```

This screen prompts you for table relationships. Each table entry is numbered on the left side of the screen. For each table, specify the number of its parent in the "Child of" column. Leave a blank for the first parent (the root) of the Master File.

If the screen displays one table for a single-table Master File, leave the "Child of " column blank and press the PF12 key.

If you are describing a multi-path structure, specify the number of the parent as many times as necessary.

To review your choices, press the PF7 key. Press the PF12 key to continue.

At this point, either a status screen indicates the successful generation of a pair of single-table file descriptions or additional screens prompt you for primary and foreign keys.

For a multi-table Master File, you must specify the primary and foreign keys (common columns that perform the embedded join) for each parent-dependent relationship. You may also specify multi-field embedded joins. For embedded join concepts and limitations, see *Multi-Table Structures* on page 99.

The following screen, Screen E4, appears twice—once for the dependent and then for the parent table. It displays the column names, data types, and column lengths for the table marked with the caret (>) symbol.

```
E4          ------------------------------------------------
           |                A U T O D B C                   |
           |          MASTER AND ACCESS FILE GENERATION     |
            ------------------------------------------------
    IDENTIFY THE PRIMARY/FOREIGN (KEYFLD/IXFLD) RELATIONSHIP FOR:
     PARENT RELATION ==>
  >  CHILD RELATION ===>
           -POSITION-COLUMNNAME-              -DATATYPE-
=>
=>
=>
=>
=>
=>
=>
=>
=>
=>
   PF3= RETURN  PF7= TOP/REVIEW  PF8= DOWN  PF12= PROCESS
```

In the Position column on the screen, select the common field and number it 1. If another common field exists, mark it as 2, the next as 3, and so on, up to a limit of 16. Press the PF12 key to display column information for each parent-dependent relationship. An error message appears if the data type does not match.

To return to a previous screen and change a selection, press the PF3 key. Press the PF7 key to review your selection. When all screens are completed, press the PF12 key and a CREATING MASTER message appears.

For a multi-field embedded join, you may number up to 16 common fields. You must specify the fields in the same order and you must have an equal number of specified fields.

Note: The AUTODBC facility determines the order in which the table relationships are displayed. The order is not necessarily a specific top-to-bottom, left-to-right progression.

When AUTODBC completes the file generation process, a status screen, Screen E5, appears. Besides indicating success, the status screen lists the new file descriptions and where they are stored. If errors occur, AUTODBC logs the errors in a z/OS data set which it creates, and it displays the log name on the status screen.

```
E5          ------------------------------------------------
           |                 A U T O D B C                  |
           |          MASTER AND ACCESS FILE GENERATION     |
            ------------------------------------------------
   AUTODBC SMMARY STATISTICS:
   MASTER FILE DESCRIPTION ===>
   ACCESS FILE DESCRIPTION ===>
            NO ERRORS FOUND
  ENTER= CONTINUE
```

AUTODBC writes the new Master and Access Files to Partitioned data sets as members. Partitioned data sets are allocated to ddnames ADUMAST and ADUSQL (Master and Access, respectively). If the ddnames are not allocated, the file descriptions are stored in data sets named ADUMAST.DATA and ADUSQL.DATA. If neither the ddnames nor data set names are available, AUTODBC dynamically allocates them.

To use the generated Master and Access Files, copy them from the ADUMAST and ADUSQL datasets to the MASTER.DATA and FOCDBC.DATA data sets after you exit the AUTODBC facility (Option 5).

To return to the Primary Option Menu, press the *Enter* key.

Newly generated file descriptions are discussed in *Results of the Master File Generation Facilities* on page 142.

Option 4: Redefining Your Teradata Security Profile

To redefine your security profile, select Option 4 from the Primary Option Menu and press the *Enter* key. For an explanation of the sample screen, see *Security Logon Screen* on page 133.

Option 5: Exiting AUTODBC

To exit the AUTODBC facility, select Option 5 from the Primary Option Menu and press the *Enter* key. The Utility Termination Options screen, Screen F, appears before you return to TSO:

```
F           ------------------------------------------------
           |                A U T O D B C                   |
           |          UTILITY TERMINATION OPTIONS           |
            ------------------------------------------------
   DISPOSITION OPTIONS IN AUTODBC FILE SUPPORT:
       1. REATIN ADUCOL ONLY
       2. RETAIN ADUCOL AND ADUTMP ONLY
       3. SCRATCH ALL AUTODBC MAINTENANCE FILES
       OPTION ===>
 ENTER= PROCESS
```

You may retain one or both repositories for future AUTODBC sessions or erase the repositories and the current session's work files. The options are:

1. RETAIN ADUCOL ONLY

   Saves the ADUCOL repository as ADU.ADUCOL.DATA.

2. RETAIN ADUCOL AND ADUTMP ONLY

   Saves both ADUCOL and ADUTMP repositories. The ADUTMP repository is saved as ADU.ADUTMP.DATA. This option is recommended.

3. SCRATCH ALL AUTODBC MAINTENANCE FILES

   Erases the repositories and temporary work files. TSO ADUMAST and ADUSQL data sets that store the generated file descriptions are not affected.

Note: If you did not proceed beyond the initial logon screen, press the PF3 key to return to the TSO environment.

Results of the Master File Generation Facilities

The AUTODB2 and AUTODBC facilities generate Master and Access Files containing the declarations described in *Describing Tables to FOCUS* on page 55, and *Multi-Table Structures* on page 99. This section discusses those cases in which they supply a different keyword or value. Minor changes may be required in some situations. As a rule, you can use the generated file descriptions, without changes, immediately after the AUTODB2 or AUTODBC session.

In the Master File:

- The FILENAME value is the member name or file name you specified on the AUTODB2 initial screen or on AUTODBC Screen E1.
- The SEGNAME value is:
  - The DB2 table name.
  - The Teradata table name, truncated to eight characters. If the first eight characters are not unique, the table name is truncated to six characters and a two-digit integer is appended to ensure uniqueness.
- The SEGTYPE value is always S0 for the root segment. For descendant segments it is either S0 or U, depending on the relationships between the various key fields.

- Field declarations for primary key columns are listed first, followed by those for non-key fields.
  - For AUTODB2, FIELDNAME and ALIAS values are the full RDBMS column names.
  - For AUTODBC, if fields are not unique, the FIELDNAME values are the Teradata column names truncated to eight characters plus the appended value Q*nnn* (where *nnn* is an integer). The Q*nnn* value is appended to ensure uniqueness. ALIAS values are the full Teradata column names.

    When Y is specified for long field names, field names up to 12 characters and aliases up to 30 characters are included in the Master File. Otherwise, a truncated FIELD declaration of 12 characters with a blank ALIAS appears in the Master File and the complete column name appears in the Access File, as described in *Additional Topics* on page 417.

- Data types are determined as follows:

  GRAPHIC and VARGRAPHIC data types are described with an ACTUAL attribute of Kn, and with a USAGE attribute of A(2n+2). LONG VARGRAPHIC data types are not supported.

  AUTODB2 supports the TIME and TIMESTAMP data types. TIME and TIMESTAMP data types are treated as either date-time or alphanumeric strings in the Master File, depending on the data type specified on the main menu. AUTODBC supports the TIMESTAMP data type as an alphanumeric string (A26).

- Field lengths for USAGE and ACTUAL formats are calculated automatically. The generated field declarations omit the keywords USAGE and ACTUAL. Only the values appear. Columns with RDBMS DATE data types are described with an ACTUAL attribute of DATE. Columns with VARCHAR data types longer than 254 characters are described with an ACTUAL attribute of text (TX).
  - For AUTODB2, the ACTUAL field length for DECIMAL(p,s) columns with missing data is P8 if p<=15, P((p+1)/2) if p>15.
  - For AUTODBC, the ACTUAL field lengths for numeric fields with missing data are either P8 for DECIMAL columns or I4 for SMALLINT columns.
- The MISSING parameter is included and is set ON if the RDBMS table definition allows NULLs.

In the Access File:

- The SEGNAME value matches the corresponding SEGNAME value in the Master File.

- The TABLENAME value:
  - For AUTODB2, depends on whether Y was entered on the Main Menu for the field "Use Creator Name in AFD?" If it was, the creator name and the table name compose the TABLENAME value. If a location was specified, the location, creator, and table name compose the TABLENAME value.
  - For AUTODBC, the database name and the table name compose the TABLENAME value.
- KEYS and WRITE values are supplied.

  AUTODB2 and AUTODBC use existing RDBMS indexes to determine the KEYS value and the order in which to place fields in the Master File. KEYS is set to zero if a unique index does not exist or if an RDBMS view is described.

  Note: In the case of a view, the unique indexes are described on the underlying tables, not on the view; they may not be valid as a primary key for the view. If possible, edit the Master and Access Files to reflect the index structure of the view's base tables.
- For AUTODB2, the optional keyword DBSPACE is omitted. For AUTODBC, the optional keywords FALLBACK and KEYORDER are not supplied.
- For AUTODBC, KEYFLD and IXFLD values are the Teradata columns selected during Option 3 to relate tables. KEYFLD and IXFLD values match the field names found in the Master File. For AUTODB2, KEYFLD and IXFLD values were selected on the Common Column Selection screen.

You may need to edit the generated Master File or its corresponding Access File if:

- You prefer field names different from the generated names.
- There are RDBMS data types in the tables or views that the adapter does not support. A warning about unsupported data types appears on the Status Screen.

  Field declarations are not generated for columns with unsupported data types. You can add the field declarations yourself. Consult *Describing Tables to FOCUS* on page 55 for more information on describing them.

  For DB2, to identify the unsupported columns check the system catalog, SYSCOLUMNS, with a FOCUS report request. (See *Additional Topics* on page 417, for an example.)

  AUTODB2 includes all data types that FOCUS can access.
- A unique index does not exist, or you want to use a different index from the selected one. If the table or view has a primary key, specify a value greater than zero for the KEYS attribute and rearrange the fields in the Master File so that the columns comprising the primary key are described first.

- Certain fields are classified for security reasons. To add security, delete field declarations or include FOCUS DBA attributes.
- You require options such as DEFINE fields and KEYORDER.

Common Errors

AUTODB2 and AUTODBC provide an easy, straightforward approach for defining tables and views to FOCUS. Most errors occur at the data entry level and result in error messages. Many fields trigger validation tests that prompt you for a valid entry. Common errors include:

- Leaving the Master Filename entry field blank on the AUTODB2 initial screen or AUTODBC Screen E1.
- For AUTODB2, specifying an invalid user ID for the Creator Name entry field.

  For AUTODBC, specifying an invalid user ID or password on the security logon screen. If you are creating the ADUCOL repository and your security profile is incorrect, you receive the FOCUS message:

  ```
  TERADATA ID OR PASSWORD INCORRECT: ENTER 'Y' TO CONTINUE
  ```

  After you press Y, the Primary Option Menu appears. Select Option 4 and correct your specified values or select Option 5 to exit the facility.
- For AUTODB2, leaving the Table Name entry field blank or specifying a nonexistent table or view. In these cases, no records are retrieved and you return to the Main Menu.

  For AUTODBC, specifying a nonexistent table or view when you are creating the ADUCOL repository.
- Specifying an invalid character for the Read/Write Functionality entry field.
- Specifying a duplicate member name or file name for the Master Filename entry field. If the REPLACE option is specified, AUTODB2 and AUTODBC overwrite the existing members or files.

At the Status Screen level, you may see one of the following warning messages:

- *n* DUPLICATE. The tables or views contain duplicate column names. You can edit these field names to make them unique, or qualify them with the segment name when referencing them in requests.
- *n* UNSUPPORTED. The table or view contains some columns with data types not supported by FOCUS (LONG VARGRAPHIC, for example).

- For AUTODBC:

  NO KEY DETECTED. The table or view does not possess a primary key. Edit the generated file descriptions accordingly to specify a primary key column.

  ALL DATATYPES WITHIN TARGET SQL TABLE ARE UNSUPPORTED. All of the columns possess data types that AUTODBC does not support.

  The status screen also displays the name of the error log file that AUTODBC creates when errors occur.

Other circumstances can also affect processing:

- Permanent PDSs are too small to receive the generated file descriptions as members.
- You do not have proper authorization for SELECT privileges on the system catalog tables.

AUTODB2 Sample Session

This sample session executes the AUTODB2 facility to generate Master and Access Files based on the existing EMPINFO, ADDRESS, and PAYINFO tables. It assigns the name AUTOEMP to the new file descriptions.

Note: Lowercase values represent user input.

First, execute the AUTODB2 FOCEXEC from the FOCUS command line:

```
>ex autodb2
```

AUTODB2 displays the Main Menu with its pre-set defaults.

Type *autoemp* in the Master Filename entry field and * in the TABLE entry field, and press *Enter*:

```
Main Menu     Master File Generation Facility for DB2
      Master Filename ================> autoemp
 Location =>             Creator => USER01
 TABLE => *
 Location values:          ,LOCDSNA ,LOCDB9A
        DATABASE NAME ================> *
      Description will be a member of:
        Master Target PDS => USER01.MASTER.DATA
        Access Target PDS => USER01.FOCSQL.DATA
        FOCDEF Target PDS => USER01.FOCDEF.DATA
        Replace Existing Description?=> N       (Y/N)
        Read/Write Functionality =====> W       (R=Read,W=Write)
        Date Display Format ==========> YYMD
        Time Stamp Display Format ====> HYYMDm
        Time Display Format ==========> HHIS
        Display Decimal when SCALE=0?=> Y       (Y/N)
        Use LABEL as Column Heading? => N       (Y/N)
        Use Remarks for FOCDEF? ======> N       (Y/N)
        Use Creator Name in AFD? =====> Y       (Y/N)
        Use Long Fieldnames? =========> Y       (Y/N)
        Parm File => USER01.FOCSQL.DATA

PF1=Help PF2=Restart PF3=Exit PF4=Log PF5=MFD PF6=AFD PF9=Picture PF10=List
```

The default values in the creator, table, and database name fields establish a pattern that selects all tables for creator USER01. AUTODB2 indicates that it is retrieving table information from the catalog:

```
**=============================================================**
**    AUTODB2 is retrieving TABLE  information from catalog.   **
**    Please wait...                                           **
**=============================================================**
```

Next, AUTODB2 presents the Table Selection Screen.

Designate ADDRESS as a child by placing *c* in its Select column.

Scroll down using the PF8 key to view the EMPINFO table and designate it as the root by placing *r* in its Select column.

Scroll down using the PF8 key to view the PAYINFO table and designate it as a child by placing *c* in its Select column:

```
Master:       Master File Generation Facility for DB2
AUTOEMP                ==Table Selection==

Place an 'R' next to the Table to be the root of the Master.
Place  a 'C' next to all other Tables to be described as children.
Enter 'Y' next to all selected Tables that will be updated.

                                                          Select  Writ
 Location  Creator                                         (R/C)  (Y/N
 --------  -------                                        ------  ----
           USER01
           USER01
           USER01
           USER01
DB2 TABLE: ADDRESS                                          c      Y
DB2 TABLE: COURSE                                                  Y
DB2 TABLE: DATETIME                                                Y
DB2 TABLE: DEDUCT                                                  Y

PF1=Help         PF3=End  PF4=Add Tables          PF7=Up  PF8=Down
```

AUTODB2 indicates that it is gathering column information about those tables:

```
**============================================================**
**    AUTODB2 is retrieving COLUMN information from catalog.  **
**    Please wait...                                          **
**============================================================**
```

Now, AUTODB2 displays a Child Selection Screen for the root table, EMPINFO. To define ADDRESS and PAYINFO as children of EMPINFO, place *c* in their Select columns:

```
Master:       Master File Generation Facility for DB2
AUTOEMP                ==Child Selection==

Place a 'C' next to the descendants of Parent:
           USER01
 EMPINFO
 Location  Creator                                          Select (C)
 --------  -------                                         ------------
           USER01
           USER01


DB2 TABLE: ADDRESS                                              c
DB2 TABLE: PAYINFO                                              c
DB2 TABLE:
DB2 TABLE:

PF1=Help  PF2=Restart  PF3=End  PF4=None  PF5=Picture    PF7=Up  PF8=Down
```

In a more complicated structure, some of the child tables could be descendants of other child tables. For example, PAYINFO could be a child of EMPINFO, and ADDRESS could be a child of PAYINFO. When necessary, AUTODB2 displays additional Child Selection Screens. However, in this example, all tables in the structure have been accounted for, so no additional Child Selection Screens display.

Next, identify the primary key from EMPINFO and the foreign key from ADDRESS on the Common Column Selection Screen. Identify EID as the primary key for the EMPINFO table by placing *1* in its Key column. Indicate that EID is the corresponding foreign key in the ADDRESS table by placing *1* in its Key column:

```
Master:       Master File Generation Facility for DB2
AUTOEMP             ==Common Column Selection==
Number the Primary Key Columns          | Number The Corresponding Foreign
(in sequence) In The Parent Table:      | Key Columns In The Child Table:
           USER01                       |            USER01
          EMPINFO
                          ADDRESS
 Key    Column Name                     | Key     Column Name
-------------------------------------------------------------------------
 1    EID                                |  1    EID
      LN                                 |       AT
      FN                                 |       LN1
      HDT                                |       LN2
    FORMAT: A9                           |    Format: A9
    FORMAT: A15                          |    Format: A4
    FORMAT: A10                          |    Format: A20
    FORMAT: YYMD                         |    Format: A20

PF1=Help  PF2=Restart  PF3=End  PF4=Skip          PF7=Up  PF8=Down
```

Since no other fields participate in the relationship, press *Enter*.

AUTODB2 displays another Common Column Selection Screen, for the EMPINFO and PAYINFO tables. Identify the primary and foreign keys that relate those two tables:

```
Master:       Master File Generation Facility for DB2
AUTOEMP             ==Common Column Selection==
Number the Primary Key Columns          | Number The Corresponding Foreign
(in sequence) In The Parent Table:      | Key Columns In The Child Table:
           USER01                       |            USER01
          EMPINFO
                           PAYINFO
 Key    Column Name                     | Key     Column Name
-------------------------------------------------------------------------
 1    EID                               |  1    EID
      LN                                |       DI
      FN                                |       PI
      HDT                               |       SAL
    FORMAT: A9                          |    Format: A9
    FORMAT: A15                         |    Format: YYMD
    FORMAT: A10                         |    Format: F9.2
    FORMAT: YYMD                        |    Format: D12.2

PF1=Help  PF2=Restart  PF3=End  PF4=Skip         PF7=Up  PF8=Down
```

Press the *Enter* key.

At this point, AUTODB2 creates the Master and Access Files. The Main Menu includes the "DESCRIPTION CREATED" message at the bottom of the screen:

```
Main Menu     Master File Generation Facility for DB2
      Master Filename ================> AUTOEMP
 Location =>             Creator => USER01
 TABLE => *
 Location values:          ,LOCDSNA ,LOCDB9A
        DATABASE NAME ================> *
      Description will be a member of:
        Master Target PDS => USER01.MASTER.DATA
        Access Target PDS => USER01.FOCSQL.DATA
        FOCDEF Target PDS => USER01.FOCDEF.DATA
        Replace Existing Description?=> N       (Y/N)
        Read/Write Functionality =====> W       (R=Read,W=Write)
        Date Display Format ==========> YYMD
        Time Stamp Display Format ====> HYYMDm
        Time Display Format ==========> HHIS
        Display Decimal when SCALE=0?=> Y       (Y/N)
        Use LABEL as Column Heading? => N       (Y/N)
        Use Remarks for FOCDEF? ======> N       (Y/N)
        Use Creator Name in AFD? =====> Y       (Y/N)
        Use Long Fieldnames? =========> Y       (Y/N)
        Parm File => USER01.FOCSQL.DATA
 DESCRIPTION CREATED-3 DUPLICATE
PF1=Help PF2=Restart PF3=Exit PF4=Log PF5=MFD PF6=AFD PF9=Picture PF10=List
```

Three duplicate field names were created. Either edit them to be distinct, or qualify them with the segment name in requests.

To see a picture of the structure created, press PF9:

```
 NUMBER OF ERRORS=     0
 NUMBER OF SEGMENTS=   3  ( REAL=    3  VIRTUAL=   0 )
 NUMBER OF FIELDS=    22  INDEXES=   0  FILES=     1
 TOTAL LENGTH OF ALL FIELDS=  186
SECTION 01
            STRUCTURE OF DB2      FILE AUTOEMP  ON 09/26/13 AT 14.22.32

         EMPINFO
 01      S0
**************
*EID          **
*LN           **
*FN           **
*HDT          **
*             **
***************
 ***************
       I
       +-----------------+
       I                 I
       I ADDRESS         I PAYINFO
 02    I S0        03    I S0
***************    **************
 *EID          **   *EID          **
 *AT           **   *DI           **
 *LN1          **   *PI           **
 *LN2          **   *SAL          **
 *             **   *             **
 ***************   ***************
  ***************    **************

 TYPE ANY CHARACTER AND PRESS ENTER TO CONTINUE  >
```

As indicated on the screen, type any character and press *Enter* to continue.

To edit The AUTOEMP Master File (member AUTOEMP in data set USER01.MASTER.DATA), press PF5:

```
$$$ CREATED BY AUTODB2 ON 09/26/13 AT 14.21.33 BY USER01
FILENAME=AUTOEMP,SUFFIX=DB2,$

SEGNAME='EMPINFO',SEGTYPE=S0,$
FIELD=EID               ,EID                          ,A9     ,A9
    MISSING=OFF,$
FIELD=LN                ,LN                           ,A15    ,A15
    MISSING=OFF,$
FIELD=FN                ,FN                           ,A10    ,A10
    MISSING=OFF,$
FIELD=HDT               ,HDT                          ,YYMD   ,DATE
    MISSING=OFF,$
FIELD=DPT               ,DPT                          ,A10    ,A10
    MISSING=ON,$
FIELD=CSAL              ,CSAL                         ,P9.2   ,P4
    MISSING=OFF,$
FIELD=CJC               ,CJC                          ,A3     ,A3
    MISSING=OFF,$
FIELD=OJT               ,OJT                          ,F9.2   ,F4
    MISSING=ON,$
FIELD=BONUS_PLAN        ,BONUS_PLAN                   ,I9     ,I4
    MISSING=OFF,$
FIELD=HDTT              ,HDTT                         ,HYYMDm ,HYYMDm
    MISSING=ON,$
FIELD=HT                ,HT                           ,HHIS   ,HHIS
    MISSING=ON,$

SEGNAME='ADDRESS',SEGTYPE=S0,PARENT='EMPINFO',$
FIELD=EID               ,EID                          ,A9     ,A9
    MISSING=OFF,$
FIELD=AT                ,AT                           ,A4     ,A4
    MISSING=OFF,$
FIELD=LN1               ,LN1                          ,A20    ,A20
    MISSING=OFF,$
FIELD=LN2               ,LN2                          ,A20    ,A20
    MISSING=OFF,$
FIELD=LN3               ,LN3                          ,A20    ,A20
    MISSING=OFF,$
FIELD=ANO               ,ANO                          ,I9     ,I4
    MISSING=OFF,
$
```

```
SEGNAME='PAYINFO',SEGTYPE=S0,PARENT='EMPINFO',$
FIELD=EID               ,EID                            ,A9     ,A9
    MISSING=OFF,$
FIELD=DI                ,DI                             ,YYMD   ,DATE
    MISSING=OFF,$
FIELD=PI                ,PI                             ,F9.2   ,F4
    MISSING=OFF,$
FIELD=SAL               ,SAL                            ,D12.2  ,D8
    MISSING=OFF,$
FIELD=JBC               ,JBC                            ,A3     ,A3
    MISSING=OFF,$
$DUPLICATE=EID                          ,COUNT= 3,SEGNAME=EMPINFO
$DUPLICATE=EID                          ,COUNT= 3,SEGNAME=ADDRESS
$DUPLICATE=EID                          ,COUNT= 3,SEGNAME=PAYINFO
```

To exit from the FOCUS TED editor, press PF3, or type *FILE* to save any changes you made.

To edit the AUTOEMP Access File (member AUTOEMP in data set USER01.FOCSQL.DATA), press PF6. Notice the KEYFLD and IXFLD values that define the embedded JOIN:

```
$$$ CREATED BY AUTODB2 ON 09/26/13 AT 14.21.33 BY USER01
$$$ FILENAME=AUTOEMP,SUFFIX=DB2,$

SEGNAME='EMPINFO',
TABLENAME='"USER01"."EMPINFO"',
KEYS=01,WRITE=YES,KEYORDER=LOW,$

SEGNAME='ADDRESS',
TABLENAME='"USER01"."ADDRESS"',
KEYS=02,WRITE=YES,KEYORDER=LOW,
KEYFLD=EID,
IXFLD=EID,$

SEGNAME='PAYINFO',
TABLENAME='"USER01"."PAYINFO"',
KEYS=02,WRITE=YES,KEYORDER=LOW,
KEYFLD=EID,
IXFLD=EID,$
```

To exit from the FOCUS TED editor, press PF3, or type *FILE* to save any changes you made.

AUTODBC Sample Session

This section describes a sample TSO session where the AUTODBC facility generates Master File and Access Files based on existing tables DPBRANCH, DPVENDOR, and DPINVENT. (See *File Descriptions and Tables* on page 459 for DBC table definitions.) The new file descriptions, created as a result of this sample session, will be named DEMO for demonstration. Session results are provided after the status screen at the end of this section.

Note: Lowercase values represent user input. Uppercase values represent TSO or AUTODBC responses.

First, the AUTODBC CLIST is executed from the TSO command level:

```
READY
ex autodbc
```

The security logon screen displays with its defaults as shown below:

```
A          ------------------------------------------------------
          |                   A U T O D B C                      |
          |    RELATIONAL TABLE DESCRIPTOR FACILITY FOR FOCUS/DBC |
          |          INFORMATION BUILDERS, INCORPORATED          |
           ------------------------------------------------------
         TERADATA USERID  =====>   jane
         TERADATA PASSWORD ====>
         TERADATA DIRECTOR PGM  =>    0
         TERADATA PARTITION ID  =>    DBC/SQL
            SUPPLY THE REQUIRED TERADATA LOGON INFORMATION
ENTER= PROCESS  PF3= EXIT (INITIAL ENTRY ONLY)
```

The Teradata user ID and password (suppressed display) are specified. AUTODBC provides the default values, 0 and DBC/SQL, for the Teradata Director Program ID and Partition ID entry fields.

The *Enter* key is pressed to continue and the Primary Option Menu appears.

When the Primary Option Menu displays, ADUCOL MAINTENANCE, Option 2, is selected. Screen D appears blank for this session, since the ADUCOL repository does not exist.

```
D          ------------------------------------------------------
          |              A U T O D B C                           |
          |            ADUCOL MAINTENANCE                        |
           ------------------------------------------------------
      IDENTIFY RELATION DESCRIPTIONS TO BE REFRESHED WITHIN ADUCOL:
       DATABASENAMES                TABLENAMES
       --------------------------   --------------------------------
=>    1 JANE                       DPBRANCH
=>    2 JANE                       DPVENDOR
=>    3 JANE                       DPINVENT
=>
=>
=>
=>
=>
=>
=>
   PF3= RETURN  PF7= TOP/REVIEW  PF8= DOWN  PF12= PROCESS
```

Three table entries are specified on the blank screen. The database names for this session are JANE. Then the PF7 key is pressed to review the entries. Each entry is now numbered and in uppercase.

The PF12 key is pressed to construct the ADUCOL repository.

When processing is complete, the Primary Option Menu reappears. Option 3 is selected to generate a multi-table Master File and a corresponding Access File.

The Master and Access File Generation screen, Screen E1, displays with its defaults.

```
E1          ------------------------------------------------------
           |                   A U T O D B C                      |
           |           MASTER AND ACCESS FILE GENERATION          |
            ------------------------------------------------------
            MASTER/ACCESS FILENAME ==>  demo
            PROCESSING OPTION =======>  2   OPTIONS:
                                              1= NEW DESCRIPTION ONLY
                                              2= NEW OR REPLACE
            USE LONG FIELD NAMES ===>   y
                          SUPPLY THE REQUIRED VALUES
PF3= RETURN  ENTER= PROCESS
```

DEMO is specified as the name for the new file descriptions. Instead of the default, Option 2 is specified to replace existing DEMO file descriptions. To incorporate long field names into the Master File, Y is specified.

The *Enter* key is pressed.

Screen E2 appears and lists all tables available for file generation. At this point, tables are selected for the file descriptions. Extra tables are excluded by using the space bar to "blank out" the database names. However, for this session, all three tables are required for the DEMO file descriptions.

```
E2        ---------------------------------------------------
         |               A U T O D B C                       |
         |       MASTER AND ACCESS FILE GENERATION           |
          ---------------------------------------------------
  IDENTIFY THOSE RELATIONS PARTICIPATING IN THE FOCUS VIEW:
      DATABASENAMES              TABLENAMES                  WRITE=
      -------------------------  -------------------------   ------
=>  1 JANE                       DPBRANCH                      N
=>  2 JANE                       DPVENDOR                      N
=>  3 JANE                       DPINVENT                      N
=>
=>
=>
=>
=>
=>
=>
   PF3= RETURN  PF7= TOP/REVIEW  PF8= DOWN  PF12= PROCESS
```

WRITE functionality remains unchanged. The default (N) indicates read-only access. The resulting DEMO file descriptions may be used only for reporting.

After the PF12 key is pressed, a STAND BY message displays.

Table relationships are specified on the next screen, Screen E3:

```
E3        ---------------------------------------------------------
         |                A U T O D B C                            |
         |         MASTER AND ACCESS FILE MAINTENANCE              |
          ---------------------------------------------------------
  IDENTIFY THE RELATIONSHIP FROM AMONG SELECTED RELATIONS:
                                                              CHILD
      DATABASENAMES              TABLENAMES                    OF
      -------------------------  -------------------------   ------
=>  1 JANE                       DPBRANCH
=>  2 JANE                       DPVENDOR                      3
=>  3 JANE                       DPINVENT                      1
=>
=>
=>
=>
=>
=>
=>
   PF3= RETURN  PF7= TOP/REVIEW  PF8= DOWN  PF12= PROCESS
```

The DPBRANCH table acts as the root. Its "Child of" column is left blank. The DPBRANCH table has the DPINVENT table as a descendent. The value 1 in the "Child of" column for the DPINVENT table indicates that its parent is DPBRANCH. The DPINVENT table has the DPVENDOR table as a descendent. The value 3 in the "Child of" column for the DPVENDOR table indicates that its parent is DPINVENT.

This produces a single-path structure. The PF12 key is pressed to continue.

The next screen, Screen E4, prompts for the common field (either primary key column or foreign key column) that will perform the embedded JOIN. The screen appears twice for each parent-child relationship. It displays columns, data types, and column lengths for the current table, denoted by the caret (>).

```
E4          ------------------------------------------------------
           |                A U T O D B C                         |
           |        MASTER AND ACCESS FILE GENERATION             |
            ------------------------------------------------------
  IDENTIFY THE PRIMARY/FOREIGN (KEYFLD/IXFLD) RELATIONSHIP FOR:
  PARENT RELATION ==>  DPINVENT
> CHILD RELATION ===>  DPVENDOR
       -POSITION-COLUMNNAME-            -DATATYPE-
=>               VENDOR_CITY             CF          5
=>               VENDOR_NAME             CF          5
=>      1        VENDOR_NUMBER           I2          2
=>
=>
=>
=>
=>
=>
=>
   PF3= RETURN  PF7= TOP/REVIEW  PF8= DOWN  PF12= PROCESS
```

In this session, Screen E4 displays the contents of the DPVENDOR table, as denoted by the caret (>) symbol. The DPVENDOR table acts as the descendent and the VENDOR_NUMBER column is specified as the foreign key.

When the PF12 key is pressed, the same screen appears. It displays the contents of the parent table, DPINVENT. The DPINVENT table also contains a VENDOR_NUMBER column and it is specified as the primary key.

The PF12 key repeats the screen two more times. It displays the contents of the DPBRANCH and DPINVENT tables. The BRANCH_NUMBER column that exists in both tables is specified as the common column.

After the common columns are selected for each relationship, the PF12 key is pressed to generate the file descriptions.

The CREATING MASTER and CREATING ACCESS FILE messages display while AUTODBC generates the file descriptions. The status screen, Screen E5, appears when the process is complete.

```
E5          ------------------------------------------------------
           |                A U T O D B C                         |
           |        MASTER AND ACCESS FILE GENERATION             |
            ------------------------------------------------------

   AUTODBC SUMMARY STATISTICS:
   MASTER FILE DESCRIPTION ===>   DEMO      IN PO DDNAME ADUMAST
   ACCESS FILE DESCRIPTION ===>   DEMO      IN PO DDNAME ADUSQL
         NO ERRORS FOUND
ENTER= CONTINUE
```

The AUTODBC facility successfully generates the DEMO Master and Access Files. They are stored in partitioned data sets allocated to ddnames ADUMAST and ADUSQL.

The *Enter* key is pressed to return to the Primary Option Menu.

This concludes the sample session. At the Primary Option Menu, users may exit the AUTODBC facility or continue to generate file descriptions for their applications.

Note: The generated Master and Access Files are allocated to the ADUMAST and ADUSQL data sets. In order to use them, users should copy or move them to the MASTER.DATA and FOCDBC.DATA data sets after exiting the AUTODBC facility (Option5).

The resulting DEMO Master File describes a single-path structure. Notice the PARENT values and the Q002 field suffixes:

```
JANE.MASTER.DATA(DEMO)          SIZE=50    LINE=0
00000 * * * TOP OF FILE * * *
00001
00002
00003    FILENAME=DEMO   ,SUFFIX=SQLDBC,$
00004
00005
00006    SEGNAME=DPBRANCH,SEGTYPE=S0,$
00007
00009  FIELD=BRANCH_NUMBE,
00010    ALIAS=BRANCH_NUMBER,
00011    I5      ,I2      ,MISSING=OFF,$
00012  FIELD=BRANCH_NAME,
00013    ALIAS=BRANCH_NAME,
00014    A5      ,A5      ,MISSING=OFF,$
00015  FIELD=BRANCH_MANAG,
00016    ALIAS=BRANCH_MANAGER,
00017    A5      ,A5      ,MISSING=OFF,$
00018  FIELD=BRANCH_CITY,
00019    ALIAS=BRANCH_CITY,
00020    A5      ,A5      ,MISSING=OFF,$
00021  SEGNAME=DPINVENT,PARENT=DPBRANCH,SEGTYPE=S0,$
00023

00024  FIELD=BRANCH_NQ002,
00025    ALIAS=BRANCH_NUMBER,
00026    I5      ,I2      ,MISSING=OFF,$
00027  FIELD=VENDOR_NQ002,
00028    ALIAS=VENDOR_NUMBER,
00029    I5      ,I2      ,MISSING=OFF,$
00030  FIELD=PRODUCT,
00031    ALIAS=PRODUCT,
00032    A5      ,A5      ,MISSING=OFF,$
00033  FIELD=NUMBER_OF_UN,
00034    ALIAS=NUMBER_OF_UNITS,
00035    I5      ,I2      ,MISSING=OFF,$
00036  FIELD=PER_UNIT_VAL,
00037    ALIAS=PER_UNIT_VALUE,
00038    P10.2     ,P5      ,MISSING=OFF,$
00039  SEGNAME=DPVENDOR,PARENT=DPINVENT,SEGTYPE=S0,$
00041

00042  FIELD=VENDOR_NUMBE,
00043    ALIAS=VENDOR_NUMBER,
00044    I5      ,I2      ,MISSING=OFF,$
00045  FIELD=VENDOR_NAME,
00046    ALIAS=VENDOR_NAME,
00047    A5      ,A5      ,MISSING=OFF,$
00048  FIELD=VENDOR_CITY,
00049    ALIAS=VENDOR_CITY,
00050    A5      ,A5      ,MISSING=OFF,$
00051 * * * END OF FILE * * *
```

In the corresponding DEMO Access File, notice the KEYFLD and IXFLD values that perform the embedded JOINs.

```
JANE.FOCDBC.DATA(DEMO)          SIZE=34    LINE=0


00000 * * * TOP OF FILE * * *
00001
00002
00003
00004
00005    SEGNAME=DPBRANCH,
00006
00007    TABLENAME=JANE.DPBRANCH,
00008
00009    KEYS=0 ,WRITE=NO,$
00010
00011
00012


00013    SEGNAME=DPVENDOR,
00014
00015    TABLENAME=JANE.DPVENDOR,
00016
00017    KEYS=0 ,WRITE=NO,
00018
00019    KEYFLD=VENDOR_NQ002,
00020
00021    IXFLD=VENDOR_NUMBER,$
00022
00023
00024


00025    SEGNAME=DPINVENT,
00026
00027    TABLENAME=JANE.DPINVENT,
00028
00029    KEYS=0 ,WRITE=NO,
00030
00031    KEYFLD=BRANCH_NUMBER,
00032
00033    IXFLD=BRANCH_NQ002,$
00034
00035 * * * END OF FILE * * *
```

Generating a Master and Access File Using the CREATE SYNONYM Command

You can generate a synonym (Master and Access File) for a relational data source using the CREATE SYNONYM command.

Note: The CREATE SYNONYM command attributes are subject to change without notice.

Syntax: How to Generate a Master and Access File Using the CREATE SYNONYM Command

```
CREATE SYNONYM [appname/]mastername [DROP]
  FOR tablename
  DBMS adapter
  [AT connection]
  [NOCOLS]
  [STOREDPROCEDURE
    PARMS "parm1[, parm2, ... parmn"]
  ]
END
```

where:

appname
Is the application folder in which to generate the synonym.

mastername
Is the name of the resulting synonym.

DROP
Deletes an existing synonym with the same name, if one exists, and generates the new synonym.

tablename
Is the name of the table for which the synonym is being generated.

adapter
Is the relational adapter for which the synonym is being generated.

connection
Is the name of the connection for the adapter.

NOCOLS
Creates a synonym with no columns in it and, at run time, queries the systems tables to see what columns are available.

STOREDPROCEDURE
Is required for creating a synonym for a stored procedure.

PARMS "*parm1*[, *parm2*, ... *parmn*"]
Is the list of parameter values being sent to the stored procedure.

- An input parameter is a literal value, enclosed in single quotation marks (for example, 125, 3.14, 'abcde'). You can use reserved words as input. Unlike character literals, reserved words are not enclosed in quotation marks (for example, NULL). Input is required.

- An output parameter is represented as a question mark (?). You can control whether output is passed to an application by including or omitting this parameter. If omitted, this entry will be an empty string (containing 0 characters).
- An INOUT parameter consists of a question mark (?) for output and a literal for input, separated by a slash (/). (For example: ?/125, ?/3.14, ?/'abcde'.) The out value can be an empty string (containing 0 characters).

For an example of creating and using a synonym for a stored procedure, see *Adapter for DB2 Stored Procedure Support (CLI Only)* on page 420.

Creating Tables: The CREATE FILE Command

The CREATE FILE command uses existing Master and Access Files to generate new RDBMS tables and, possibly, unique indexes.

Syntax: How to Create a Table

```
CREATE FILE name [DROP]
```

where:

name

Is the name of the Master and Access Files.

DROP

Drops the table, if it already exists, and then creates it.

CREATE FILE Prerequisites and Processing

Before issuing the CREATE FILE command, make sure you have:

- RDBMS GRANT authority to create tables (as described in *Connection, Authentication, and Security* on page 45).
- A Master File. Field declarations describing the primary key columns must be listed first.
- An Access File. If KEYS is greater than zero, a unique index will be created. To create the index in descending order, set KEYORDER to HIGH.

  If the Access File does not include a DBSPACE value, you can issue the SET DBSPACE command to establish a default tablespace or dbspace for the duration of the FOCUS session. (Consult *Describing Tables to FOCUS* on page 55 for the DBSPACE attribute and *Adapter Commands* on page 309 for the SET DBSPACE command.)

If you do not issue the SET DBSPACE command, CREATE FILE uses the adapter installation default. If your site did not establish a default during installation:

- For DB2, the table is placed in the default DB2 database, DSNDB04, and DB2 dynamically creates the tablespace.
- For Teradata, the table is placed into the database associated with the user ID.
- For IDMS/SQL, the table is placed in the IDMS default area specified in the table's schema DDL definition.
- For Oracle, the table is placed in the default tablespace of the owner of the schema containing the table

When the table is successfully generated, the FOCUS command level prompt (>) appears.

The adapter generates one table and one unique index (provided the KEYS parameter is not 0) for every segment declaration in a multi-table Master File. It accomplishes this in a single logical unit of work, so if one of the tables already exists, it does not create the others unless you specify the DROP option in the CREATE FILE command. That is, the FOCUS CREATE FILE command does not, by default, overwrite an existing RDBMS table as it may do for a FOCUS database.

Note: You can control index space parameters for DB2, Oracle, and IDMS/SQL with the adapter SET IXSPACE command described in *Adapter Commands* on page 309.

You have two choices if an SQL error occurs:

- Issue the CREATE FILE command with the DROP option.
- Change the value of the TABLENAME attribute in the Access File and reissue the CREATE FILE command.
- Discard the existing table with the SQL DROP command and reissue the CREATE FILE command.

You can also create tables by:

- Issuing the native SQL CREATE TABLE command from within the FOCUS environment. Consult *Direct SQL Passthru* on page 265 for Direct SQL Passthru.
- Using the HOLD FORMAT SQL*engine* option in a report request. See *Advanced Reporting Techniques* on page 213 for information about extract files.

Example: Using CREATE FILE to Create a DB2 Table

The following DB2 session illustrates table creation. The member name for the pair of file descriptions is EMPINFO. In order to trace the process, the example uses the SQLCALL component of the trace facility. (For more information about the adapter trace facilities, see *Tracing Adapter Processing* on page 487.)

Since the EMPINFO Master and Access Files exist (see *File Descriptions and Tables* on page 459), the CREATE FILE command can create the EMPINFO table. If the table already exists, it will be dropped first:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = SQLCALL//CLIENT
SET TRACESTAMP = OFF
CREATE FILE EMPINFO DROP
```

The trace displays the SQL statements issued by the adapter:

```
RRSET10  entered. (DB2), Gfun=  3, fun=  2; CMD_OPEN.
RRSET10  Count=  0
RRSET10  exited.  (DB2), Errcode=        0; CMD_OPEN.
RRSET10  Count=  1
RRSET10  entered. (DB2), Gfun=  1, fun=  6; EXECUTE_IM.
RRSET10  SQL:  DROP TABLE USER01."EMPINFO"
RRSET10  exited.  (DB2), Errcode=        0; EXECUTE_IM.
RRSET10  entered. (DB2), Gfun=  1, fun=  5; COMMIT_WORK.
RRSET10  exited.  (DB2), Errcode=        0; COMMIT_WORK.
RRSET10  entered. (DB2), Gfun=  1, fun=  6; EXECUTE_IM.
RRSET10  SQL:  CREATE TABLE USER01."EMPINFO"( "EID" CHAR (9) NOT NULL ,
RRSET10  SQL: "LN" CHAR (15) NOT NULL ,"FN" CHAR (10) NOT NULL ,"HDT" DATE
RRSET10  SQL:  NOT NULL ,"DPT" CHAR (10),"CSAL" DECIMAL(7, 2) NOT NULL ,
RRSET10  SQL: "CJC" CHAR (3) NOT NULL ,"OJT" REAL ,"BONUS_PLAN" INTEGER
RRSET10  SQL: NOT NULL ,"HDTT" TIMESTAMP,"HT" TIME) IN DBUSER01.FOCUS
RRSET10  exited.  (DB2), Errcode=        0; EXECUTE_IM.
RRSET10  entered. (DB2), Gfun=  1, fun=  6; EXECUTE_IM.
RRSET10  SQL:  CREATE  UNIQUE INDEX USER01."EMPINFOIX" ON USER01."EMPINFO"
RRSET10  SQL: ("EID"  ASC)
RRSET10  exited.  (DB2), Errcode=        0; EXECUTE_IM.
RRSET10  entered. (DB2), Gfun=  1, fun=  5; COMMIT_WORK.
RRSET10  exited.  (DB2), Errcode=        0; COMMIT_WORK.
RRSET10  entered. (DB2), Gfun=  3, fun=  3; CMD_CLOSE.
RRSET10  Count=  1
RRSET10  exited.  (DB2), Errcode=        0; CMD_CLOSE.
RRSET10  Count=  0
```

The adapter generates one SQL CREATE TABLE command that consists of:

- Column information from the field declarations in the Master File.
- Table information from the TABLENAME value in the Access File.

The new table resides in tablespace DBUSER01.FOCUS.

Since the KEYS value in the Access File is greater than zero, the adapter issues the SQL CREATE UNIQUE INDEX command. The first *n* fields in the Master File and the KEYS value provide the required information.

The index EMPINFOIX is created in ascending order, the RDBMS default. Its name is composed of the table name and the suffix *IX*. The parentheses around the EID field from the Master File indicate that it is the column to be indexed.

If no SQL errors result from table or index creation, the adapter issues the SQL COMMIT WORK command to permanently define the table and its index. If an error occurs, the adapter issues an SQL ROLLBACK WORK command. The ROLLBACK WORK command returns the RDBMS catalog tables to their original state, and table generation stops.

In this example there are no errors, since each generated SQL statement returns an error code of 0. The adapter issues the SQL COMMIT WORK command to permanently define the table and its index.

Example: Using CREATE FILE to Create a Teradata Table

The following session illustrates the table creation process. In order to trace the process, this example uses the SQLCALL component of the trace facility. (For more information about the adapter trace facilities, see *Tracing Adapter Processing* on page 487.)

Since the EMPINFO Master and Access Files exist (see *File Descriptions and Tables* on page 459), the CREATE FILE command is issued to create the EMPINFO table.

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = SQLCALL//CLIENT
CREATE FILE EMPINFO
```

The trace displays as:

```
>>> DBTFOC  entered. (SQLDBC), Gfun=  3, fun=  2; CMD_OPEN.
>>> DBTFOC  Count=  0
<<< DBTFOC  exited.  (SQLDBC), Errcode=        0; CMD_OPEN.
<<< DBTFOC  Count=  1
>>> DBTFOC  entered. (SQLDBC), Gfun=  1, fun=  6; EXECUTE_IM.
>>> DBTFOC  SQL:  CREATE TABLE USER01.EMPINFO( EID CHAR (0009) NOT NULL ,
>>> DBTFOC  SQL: LNAME CHAR (0015) NOT NULL ,FN CHAR (0010) NOT NULL ,HDT
>>> DBTFOC  SQL: DATE  NOT NULL ,DPT CHAR (0010),CSAL DECIMAL(15, 02) NOT
>>> DBTFOC  SQL: NULL ,CJC CHAR (0003) NOT NULL ,OJT FLOAT ,BONUS_PLAN
>>> DBTFOC  SQL: INTEGER  NOT NULL )  UNIQUE PRIMARY INDEX (EID)
<<< DBTFOC  exited.  (SQLDBC), Errcode=        0; EXECUTE_IM.
>>> DBTFOC  entered. (SQLDBC), Gfun=  3, fun=  3; CMD_CLOSE.
>>> DBTFOC  Count=  1
<<< DBTFOC  exited.  (SQLDBC), Errcode=        0; CMD_CLOSE.
<<< DBTFOC  Count=  0
```

The resulting trace shows that the adapter generated one DBC/SQL CREATE TABLE statement. The statement consists of:

- Column information from the field declarations in the Master File.
- Table information from the TABLENAME value in the Access File.
- Syntax for a unique primary index.

A unique primary index is created when the KEYS value in the Access File is greater than 0 (zero). The first *n* fields in the Master File and the KEYS value provide the required information. The field EID from the EMPINFO Master File appears in parentheses as the column to be indexed.

Note: The Teradata RDBMS requires a primary index. If the KEYS value is not specified, the RDBMS creates a non-unique primary index on the first column in the table.

In the lower portion of the trace, the error code 0 indicates success. When table creation fails, a specific DBC return code is displayed and the adapter issues a DBC/SQL ROLLBACK WORK command. The ROLLBACK WORK command causes the DBC Directory to return to its original state and table generation stops.

Example: Using CREATE FILE to Create an IDMS SQL Table

The following IDMS/SQL session illustrates table creation. The member name for the pair of file descriptions is EMPINFO. In order to trace the process, the example uses the SQLCALL component of the trace facility. (For more information about the adapter trace facilities, see *Tracing Adapter Processing* on page 487.)

Since the EMPINFO Master and Access Files exist (see *File Descriptions and Tables* on page 459), the CREATE FILE command can create the EMPINFO table:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = SQLCALL//CLIENT
SET TRACESTAMP = OFF
CREATE FILE EMPINFO
```

The trace displays as follows:

```
>>> IDQFOC  entered. (SQLIDMS), Gfun=  3, fun=  4; REFRECH_DI.
<<< IDQFOC  SQLFLAGS: 00000000 00000000 0000004D 80000000 00000001 00000002
<<< IDQFOC  SQLFLAGS: 01000032 00000000 00000001 00000000 00000000 00000000
<<< IDQFOC  SQLFLAGS: 00000000 00000000 00000000 00000000 00000000 00000000
<<< IDQFOC  SQLFLAGS: 00000000 00000000 00000000 00000000 00000000 00000000
<<< IDQFOC  SQLFLAGS: 00000000 00000000 00000000 00000000 00000000 00000000
<<< IDQFOC  SQLFLAGS: 00000000 00000000 00000000 00000000 00000000 00000000
<<< IDQFOC  SQLFLAGS: 00000000 00000000 00000000
<<< IDQFOC  APTFLAGS: 000000C0 00000001 97D00001 00000001 00000000 3002D802
<<< IDQFOC  APTFLAGS: 00000002 00001A8B 00000000 00000000 0000000E 00000000
<<< IDQFOC  APTFLAGS: 00000000 00000000 00000004 00000001 00000000
<<< IDQFOC  exited.  (SQLIDMS), Errcode=        0; REFRECH_DI.
>>> IDQFOC  entered. (SQLIDMS), Gfun=  3, fun=  2; CMD_OPEN.
>>> IDQFOC  Count=  0
<<< IDQFOC  exited.  (SQLIDMS), Errcode=        0; CMD_OPEN.
<<< IDQFOC  Count=  1
>>> IDQFOC  entered. (SQLIDMS), Gfun=  1, fun=  6; EXECUTE_IM.
>>> IDQFOC  SQL:  CREATE TABLE EMPSCHEM."EMPINFO"( EID CHAR (0009) NOT NULL ,
>>> IDQFOC  SQL: LN CHAR (0015) NOT NULL ,FN CHAR (0010) NOT NULL ,HDT DATE
>>> IDQFOC  SQL: NOT NULL ,DPT CHAR (0010),CSAL DECIMAL(15, 02) NOT NULL ,CJC
>>> IDQFOC  SQL: CHAR (0003) NOT NULL ,OJT FLOAT ,BONUS_PLAN INTEGER  NOT
>>> IDQFOC  SQL: NULL )
<<< IDQFOC  exited.  (SQLIDMS), Errcode=        0; EXECUTE_IM.
>>> IDQFOC  entered. (SQLIDMS), Gfun=  1, fun=  6; EXECUTE_IM.
>>> IDQFOC  SQL:  CREATE  UNIQUE INDEX "EMPINFOIX" ON EMPSCHEM."EMPINFO"
>>> IDQFOC  SQL: (EID  ASC)

<<< IDQFOC  exited.  (SQLIDMS), Errcode=        0; EXECUTE_IM.
>>> IDQFOC  entered. (SQLIDMS), Gfun=  1, fun=  5; COMMIT_WORK.
<<< IDQFOC  exited.  (SQLIDMS), Errcode=        0; COMMIT_WORK.
>>> IDQFOC  entered. (SQLIDMS), Gfun=  3, fun=  3; CMD_CLOSE.
>>> IDQFOC  Count=  1
<<< IDQFOC  exited.  (SQLIDMS), Errcode=        0; CMD_CLOSE.
<<< IDQFOC  Count=  0
```

The adapter generates one SQL CREATE TABLE command that consists of:

- Column information from the field declarations in the Master File.
- Table information from the TABLENAME value in the Access File.

The new table resides in IDMS area EMPSEG.EMPAREA.

Since the KEYS value in the Access File is greater than zero, adapter issues the SQL CREATE UNIQUE INDEX command. The first *n* fields in the Master File and the KEYS value provide the required information.

The index EMPINFOIX is created in ascending order, the IDMS default. Its name is composed of the table name and the suffix *IX*. The parentheses around the EID field from the Master File indicate that it is the column to be indexed.

If no SQL errors result from table or index creation, the adapter issues the SQL COMMIT WORK command to permanently define the table and its index. If an error occurs, the adapter issues an SQL ROLLBACK WORK command. The ROLLBACK WORK command returns the IDMS system tables to their original state, and table generation stops.

In this example there are no errors, since each generated SQL statement returns an error code of 0. The adapter issues the SQL COMMIT WORK command to permanently define the table and its index.

Example: Using CREATE FILE to Create an Oracle Table

The following example shows how the adapter creates the Oracle table EMPINFO using the CREATE FILE command with the sample EMPINFO Master and Access File. (For information about the adapter trace facilities, see *Tracing Adapter Processing* on page 487.)

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = SQLCALL//CLIENT
SET TRACESTAMP = OFF
CREATE FILE EMPINFO
```

The trace follows:

```
>>> ORAFOC  entered. (SQLORA), Gfun=  3, fun=  2; CMD_OPEN.
>>> ORAFOC  Count=  0
<<< ORAFOC  exited.  (SQLORA), Errcode=        0; CMD_OPEN.
<<< ORAFOC  Count=  1
>>> ORAFOC  entered. (SQLORA), Gfun=  1, fun=  6; EXECUTE_IM.
>>> ORAFOC  SQL:  CREATE TABLE USER01.EMPINFO( "EID" VARCHAR2 (0009) NOT
NULL
>>> ORAFOC  SQL: ,"LN" VARCHAR2 (0015) NOT NULL ,"FN" VARCHAR2 (0010) NOT
>>> ORAFOC  SQL: NULL ,"HDT" DATE  NOT NULL ,"DPT" VARCHAR2 (0010),"CSAL"
>>> ORAFOC  SQL: DECIMAL(07, 02) NOT NULL ,"CJC" VARCHAR2 (0003) NOT NULL ,
>>> ORAFOC  SQL: "OJT" REAL ,"BONUS_PLAN" INTEGER  NOT NULL ,"HDTT" DATE NOT
>>> ORAFOC  SQL: NULL )
<<< ORAFOC  exited.  (SQLORA), Errcode=        0; EXECUTE_IM.
>>> ORAFOC  entered. (SQLORA), Gfun=  1, fun=  6; EXECUTE_IM.
>>> ORAFOC  SQL:  CREATE  UNIQUE INDEX USER01.EMPINFOIX ON USER01.EMPINFO
>>> ORAFOC  SQL: ("EID"  ASC)
<<< ORAFOC  exited.  (SQLORA), Errcode=        0; EXECUTE_IM.
>>> ORAFOC  entered. (SQLORA), Gfun=  1, fun=  5; COMMIT_WORK.
<<< ORAFOC  exited.  (SQLORA), Errcode=        0; COMMIT_WORK.
>>> ORAFOC  entered. (SQLORA), Gfun=  3, fun=  3; CMD_CLOSE.
>>> ORAFOC  Count=  1
<<< ORAFOC  exited.  (SQLORA), Errcode=        0; CMD_CLOSE.
<<< ORAFOC  Count=  0
```

Note:

- This example assumes that the Oracle login ID is USER01. Since the login ID is the creator of the table, it is not really necessary to specify the creator in the Access File in order to use CREATE FILE.
- The Oracle table name is obtained from the Access File.
- The column definitions are taken from the USAGE attributes in the Master File.
- The unique index is based on the KEYS attribute in the Access File and the order of the field names in the Master File. In this case, KEYS=1, with EID as the first field described in the Master File. A unique index was created on the column EMP_ID.
- If the CREATE TABLE and CREATE UNIQUE INDEX commands are successful, the adapter issues a COMMIT WORK to permanently store the table definition in the Oracle RDBMS.

Note: If the adapter specifies SMALLINT, INTEGER or DECIMAL for a data type, Oracle responds by creating either full size NUMERIC or NUMERIC (n,m) columns. SMALLINT, INTEGER, and DECIMAL are acceptable keywords for use in creating Oracle tables and are used by the adapter for CREATE FILE and certain other operations. There is no need for you to be familiar with them or to use them when creating your own tables.

The order of the columns will be the same as the order in which they were described in the Master File.

Chapter 7

The Adapter Optimizer

Optimization is the process in which the adapter translates the projection, selection, join, sort, and aggregation operations of a report request into their SQL equivalents and passes them to the RDBMS for processing.

In this chapter:

Optimizing Requests

Adapter optimization allows the RDBMS to perform the work for which it is best suited, reducing the volume of RDBMS-to-FOCUS communication and improving response time. It also enables the RDBMS to exploit its own internal optimization techniques.

Syntax: How to Invoke Optimization

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

```
{ENGINE|SQL} [sqlengine] SET {OPTIMIZATION|SQLJOIN} setting
```

where:

sqlengine

Is the target RDBMS. Valid values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

SQLJOIN

Is a synonym for OPTIMIZATION.

setting

Is the optimization setting. Valid values are as follows:

OFF instructs the adapter to create SQL statements for simple data retrieval from each table. FOCUS handles all aggregation, sorting, and joining in your address space to produce the report.

ON instructs the adapter to create SQL statements that take advantage of RDBMS join, sort, and aggregation capabilities. It 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 (see *RDBMS and FOCUS Join Management* on page 178). Other cases of the multiplicative effect invoke the adapter-managed native join logic described in *Optimizing Joins* on page 178. ON is the default value.

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 (see *RDBMS and FOCUS Join Management* on page 178) invoke the adapter-managed native join logic described in *Optimizing Joins* on page 178.

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 and for tables residing on multiple DBMS platforms.

NOAGGR disables optimization of calculations (DEFINE fields) without disabling optimization of join and sort operations.

AGGR enables optimization of calculations (DEFINE fields). This is the default value for optimization of calculations.

You can invoke the adapter trace facility to evaluate the SQL statements generated by the adapter. The SQLAGGR trace component verifies whether the adapter passed aggregation and join operations to the RDBMS. The STMTRACE component displays the SQL SELECT statements that the adapter generates from report requests. A single SQL SELECT statement indicates RDBMS-managed join processing. Two or more indicate FOCUS-managed join processing. For information about adapter trace facilities, see *Tracing Adapter Processing* on page 487.

When optimizing a TABLE request, the adapter tries to ensure, within reasonable limits, that the results of execution of the optimized query will be the same as the results obtained when reading raw unfiltered data from the same data hierarchy.

Example: SQL Requests Passed to the RDBMS With Optimization OFF

This example demonstrates SQL statements generated without optimization. The report request joins tables EMPINFO and FUNDTRAN with trace components SQLAGGR and STMTRACE allocated.

When optimization is disabled, the adapter 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 (?, :000*n*, or :H, depending on the RDBMS) with the value of the host field (EMP_ID). Both SELECT statements retrieve answer sets, but FOCUS performs the join, sort, and aggregation operations:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = SQLAGGR//CLIENT
SET TRACEON = STMTRACE//CLIENT

SQL DB2 SET OPTIMIZATION OFF
 JOIN EMP_ID IN EMPINFO TO ALL WHO IN FUNDTRAN AS J1
 TABLE FILE EMPINFO
  SUM AVE.CURRENT_SALARY ED_HRS BY WHO BY LAST_NAME
  IF DEPARTMENT EQ 'MIS'
 END
```

The following trace is generated

```
(FOC2510) FOCUS-MANAGED JOIN SELECTED FOR FOLLOWING REASON(S):
(FOC2511) DISABLED BY USER
(FOC2590) AGGREGATION NOT DONE FOR THE FOLLOWING REASON:
(FOC2592) RDBMS-MANAGED JOIN HAS BEEN DISABLED
  SELECT T1.EID,T1.LN,T1.DPT,T1.CSAL,T1.OJT FROM
 "USER1"."EMPINFO" T1 WHERE (T1.DPT = 'MIS') FOR FETCH ONLY;
  SELECT T2.EID FROM "USER1"."FUNDTRAN" T2 WHERE (T2.EID = ?)
 FOR FETCH ONLY;
```

Example: SQL Requests Passed to the RDBMS With Optimization ON

This example shows the SQL generated with optimization for the report request in *SQL Requests Passed to the RDBMS With Optimization OFF*.

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

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = SQLAGGR//CLIENT
SET TRACEON = STMTRACE//CLIENT


SQL DB2 SET OPTIMIZATION ON
 >  > JOIN EMP_ID IN EMPINFO TO ALL WHO IN FUNDTRAN AS J1
 >  > TABLE FILE EMPINFO
 > SUM AVE.CURRENT_SALARY ED_HRS BY WHO BY LAST_NAME
 > IF DEPARTMENT EQ 'MIS'
 > END
```

The following trace is generated:

```
AGGREGATION DONE ...
    SELECT T2.EID,T1.LN, AVG(T1.CSAL), SUM(T1.OJT) FROM
   "USER1"."EMPINFO" T1,"USER1"."FUNDTRAN" T2 WHERE (T2.EID =
   T1.EID) AND (T1.DPT = 'MIS') GROUP BY T2.EID,T1.LN ORDER BY
   T2.EID,T1.LN FOR FETCH ONLY;
```

Both OPTIMIZATION settings produce the same report.

Reference: A Note About Examples

There are minor differences in the specific SQL syntax generated for each RDBMS. However, the adapter messages are the same and the generated SQL statements are similar enough that for the remainder of this manual most examples will illustrate SQL syntax generated by one adapter.

Optimization Logic

Optimization means passing the following tasks to the RDBMS:

- Record selection and projection
- Joins
- Sorting
- Aggregation

Aside from record selection and projection, the adapter does not offload these functions to the RDBMS if you set OPTIMIZATION to OFF. When OPTIMIZATION is not OFF, the adapter 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.

Optimizing Record Selection and Projection

Regardless of the OPTIMIZATION setting, the adapter may pass record selection and projection to the RDBMS. It is always more efficient to do so

Record Selection

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

- Certain DEFINE fields (see *Optimizing DEFINE Fields* on page 184).
- EDIT to perform data type 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.
- The only optimized selection criteria on date-time columns (criteria passed to the RDBMS in the generated SQL) are relational expressions that are valid in an IF phrase. For example, comparison of a date-time column with a date-time literal value is optimized, but comparison of a date-time column with another date-time column is not optimized.

The adapter optimizes screening conditions based on DEFINE fields that derive their values from a single segment of the join structure. The adapter optimizes these screening conditions as long as you do not set OPTIMIZATION OFF, even if it has disabled join optimization in your request.

When using 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.
- Contains the % wildcard character to denote any sequence of characters.

Escape characters in the LIKE predicate are optimized. See your FOCUS documentation 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 data types, the WHERE clause is not passed to the RDBMS if both of the following conditions are true:

- The specified pattern is not equal in length to the column used in the comparison.
- The pattern contains one or more wildcard characters but does not terminate with the percent character (%).

In this case, FOCUS performs the screening condition on all rows returned from the RDBMS.

Optimizing Selection of Relational Variable Length Character Data Types

TABLE IF criteria can reference RDBMS variable character data types such as VARCHAR, LONG VARCHAR, and CLOB (described in the Master File with USAGE=TX and ACTUAL=TX).

Certain types of IF criteria that reference variable length character data types are included in the generated SQL, causing the selection operations to be performed by the RDBMS and improving performance.

The IF test to be optimized must be a CONTAINS or OMITS test against a field described with USAGE=TX and ACTUAL=TX in the Master File. The RDBMS column must be a character variable length data type.

CONTAINS translates to LIKE in the generated SQL, and OMITS translates to NOT LIKE. The generated SQL places wildcard characters around the literal string specified in the CONTAINS or OMITS test.

Reference: Usage Notes for Optimization of Selection of Variable Length Data Types

The following options are not supported with text fields:

- CRTFORM
- TYPE

- FSCAN
- MODIFY

Example: Optimizing a Selection Test Against a Variable Length Character Column

Consider the following variation of the DB2 Master File named EMPINFO. A CLOB column named JOBDESC has been added that contains a job description:

```
FILENAME=EMPINFO ,SUFFIX=DB2,$

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=DPT          ,USAGE=A10    ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL          ,USAGE=P9.2   ,ACTUAL=P4     ,$
 FIELD=CURR_JOBCODE  ,ALIAS=CJC           ,USAGE=A3     ,ACTUAL=A3     ,$
 FIELD=JOBDESC       ,ALIAS=JDSC          ,USAGE=TX50   ,ACTUAL=TX     ,$
 FIELD=ED_HRS        ,ALIAS=OJT           ,USAGE=F6.2   ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN    ,ALIAS=BONUS_PLAN    ,USAGE=I4     ,ACTUAL=I4     ,$
```

The following request specifies a CONTAINS test against the JOBDESC field:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT

TABLE FILE EMPINFO
PRINT EMP_ID LAST_NAME FIRST_NAME DEPARTMENT CURR_JOBCODE JOBDESC
IF JOBDESC CONTAINS 'PR'
END
```

The CONTAINS operator is translated to a LIKE operator in the generated SQL:

```
    SELECT T1."EID",T1."LN",T1."FN",T1."DPT",T1."CJC",T1."JDSC"
FROM USER1."EMPINFO" T1 WHERE (T1."JDSC" LIKE '%PR%') FOR
FETCH ONLY;
```

Projection

In relational terminology, projection is the act of retrieving any subset of the available columns from a table. The adapter 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.

Optimizing Joins

This discussion applies to joins invoked either by the FOCUS dynamic JOIN command (see *Advanced Reporting Techniques* on page 213) or by the embedded join facility (see *Multi-Table Structures* on page 99).

For information about using the JOIN command to construct an outer join and about controlling outer join optimization, see *Advanced Reporting Techniques* on page 213.

To explain how optimization logic is applied to joins, first there is a discussion of how the RDBMS and FOCUS differ in their management of joins.

RDBMS and FOCUS Join Management

When the RDBMS manages a join, it effectively first generates a Cartesian product of the tables, then applies any screening conditions to the resulting rows (including any join conditions), then applies the SELECT list, and, finally, calculates column function values and expressions.

In contrast, when FOCUS manages a join, it first reads segment instances from the top to the bottom of the file structure.

Because FOCUS views RDBMS tables as segments, in very rare situations the RDBMS could return more instances of a row than FOCUS would if it were processing similar data from a FOCUS or sequential data source. This difference is called the *multiplicative effect*.

FOCUS implements a dynamic or embedded join between RDBMS tables (segments) based on a set of pairs of "from/to" fields. The "to" fields are called the foreign key.

The KEYS attribute in the Access File defines the number of primary key fields (columns) in a segment. KEYS = *n* indicates that the first *n* fields in the segment comprise the primary key. If the foreign key fields for a child segment or cross-referenced file do not completely cover its primary key, multiple rows in the child segment (table) may correspond to a single row in the parent segment. Therefore, the RDBMS-generated Cartesian product may contain multiple instances of a single parent segment row. The parent segment is multiplied. By definition, all ancestors of the multiplied segment are also multiplied.

A unique segment is defined as having exactly one instance corresponding to an instance of its parent segment. If its foreign key fields do not cover its primary key, this is not necessarily the case, even though the join was specified as unique. A segment in this situation is called *misjoined*.

Even if a unique segment is misjoined, however, the adapter does not consider it as causing its parent's multiplication. The adapter processes a "unique segment misjoined" condition separately from the multiplicative effect. it issues a warning message, and, depending on the current optimization setting, may or may not disable optimization.

In most cases, the adapter can prevent the multiplicative effect, as described in *Optimizing Joins* on page 178 and *Optimizing Aggregation* on page 183.

Join Optimization Logic

When you invoke optimization, the adapter attempts to build a single SQL statement. In order for the adapter 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.

  SUM or COUNT operations at any level other than the lowest level in the join structure cause the RDBMS to duplicate data. In effect, each host row is replicated for each associated row in the cross-referenced table. This duplication of data is known as the *multiplicative effect*.

  When all segments other than the lowest level segment in the request have primary keys, the adapter 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 adapter to eliminate duplicate rows before passing the data to FOCUS. This technique is called *adapter-managed native join optimization*. When adapter-managed native join optimization cannot be used, SUM or COUNT operations at any level other than the lowest level in the join structure cause optimization to be disabled.

  In some cases, adapter-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 adapter manages join optimization, it does not optimize aggregation or sorting, as described in *Optimizing Sorts* on page 182 and *Optimizing Aggregation* on page 183. This behavior is consistent with that of prior releases when the join was not passed. Expressions based on a single segment in the structure, however, are passed (see *Optimizing Record Selection and Projection* on page 175).

- For DB2, tables must reside at the same location (subsystem).
- The Master Files for the tables must not include any OCCURS segments.
- You must not join more than the number of tables supported by the RDBMS or FOCUS, whichever is smaller.
- For conditional joins, the join expressions must be optimizable, as described in *Optimizing DEFINE Fields* on page 184.

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

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

Note:

- A TABLE request that references a dynamically joined structure generates SQL join predicates for all segments in the subtree that starts from the root segment. The adapter does not necessarily generate these predicates for multi-table Master Files (see *Multi-Table Structures* on page 99). 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 Master File that represent the same tree structure.
- When the adapter 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 either the file (for non-RDBMS files) or the returned answer set (for RDBMS tables).

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

Optimization of Joins Between Heterogeneous Data Sources

The adapter can pass a single SELECT statement for all tables referenced in a TABLE request to the RDBMS when all active segments from the RDBMS comprise a contiguous single-path subtree. The adapter can optimize the join between relational tables even when the retrieval path includes segments from different types of data sources (for example, several DB2 segments and an IMS segment).

Single path means no non-unique RDBMS segment can have an RDBMS parent and a non-unique sibling (RDBMS or non-RDBMS). Active segments are those containing fields referenced in the request or fields involved in join operations. Contiguous segments are connected RDBMS segments with a common target RDBMS. There may be segments of different file types above or below the contiguous segments. Segments occupying intermediate positions in a file hierarchy between explicitly active segments are themselves also implicitly active.

Note:

- If the request specifies a sort operation, the adapter transfers the sort operation to the RDBMS only if the root of the SQL subtree is the topmost active segment in the data source. Additionally, all BY fields must be contained within the segments of the SQL subtree. If the multiplicative effect is detected and the adapter-managed native join is invoked, the adapter passes 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 adapter to eliminate duplicate rows before passing the data to FOCUS.
- FOCUS and the RDBMS use slightly different sorting/aggregation algorithms. Therefore, when the RDBMS processes a sort request, the sequence of report rows within a given sort key value may vary slightly from the sequence that would be produced by FOCUS. While both report results are equally correct, if such differences are unacceptable, you can set OPTIMIZATION OFF. FOCUS will then handle all aggregation, sorts and join operations required to produce the report.

Example: Optimizing Joins Between Heterogeneous Data Sources

The following example illustrates the SQL request passed to DB2 as the result of a dynamic join between two DB2 tables and an IDMS record. For information about JOIN syntax, see *Advanced Reporting Techniques* on page 213:

```
JOIN EMP_ID IN EMPINFO TO WHO IN FUNDTRAN AS JOIN1
JOIN EMP_ID IN EMPINFO TO EMP_ID IN IDMSFILE AS JOIN2
```

With optimization enabled, the adapter produces one SQL SELECT statement that joins the two DB2 tables. The RDBMS processes the join:

```
>       SELECT T1.EID,T2.BN FROM "USER1"."EMPINFO" T1,
    "USER1"."FUNDTRAN" T2 WHERE (T2.EID = T1.EID) FOR FETCH ONLY;
```

For each DB2 joined row returned, the adapter uses the join field as input to an IDMS OBTAIN record command utilizing an IDMS index or a Calc key.

With optimization disabled, the adapter generates a separate SQL SELECT statement for each DB2 table:

```
SELECT T1.EID FROM "USER1"."EMPINFO" T1 FOR FETCH ONLY;
```

After a row is fetched from table1, the join field is used as input to the second select:

```
SELECT T2.BN FROM "USER1"."FUNDTRAN" T2 WHERE (T2.EID = ?) FOR
FETCH ONLY;
```

For each DB2 joined row returned, the adapter uses the join field as input to an IDMS OBTAIN record command using an IDMS index or a Calc key.

Optimizing Sorts

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

- It can generate a single SQL statement.
- Adapter-managed join optimization is not in effect.

For a discussion of these conditions, see *Optimizing Joins* on page 178.

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

1. If the request does not use the direct operators FST. and LST., the adapter translates sort phrases (BY or ACROSS) into SQL ORDER BY clauses, thus passing responsibility for the primary sorting of data to the RDBMS. Note that BY TOTAL phrases are not optimized.
2. When the adapter 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 adapter applies this technique in requests that explicitly specify the FST. or LST. operators and in 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 adapter 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.

3. If the report request contains FST. or LST. operators on a segment that may return multiple instances, the adapter directs the RDBMS to sort the data by primary key in the sequence specified by the Access File KEYORDER attribute. From this, the adapter 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 TABLEF command in conjunction with RDBMS-managed sorting to free FOCUS from having to verify the sort order. Refer to *Advanced Reporting Techniques* on page 213, for the TABLEF command.

Optimizing 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 adapter structures its retrieval request so that the RDBMS performs the aggregation. The adapter passes aggregation to the RDBMS when:

- The IF or WHERE tests in the FOCUS request translate to SQL WHERE clauses. (Use STMTRACE, described in *Tracing Adapter Processing* on page 487, to examine the SQL generated from your request.)
- FOCUS generates a single SQL statement (as described in *Optimizing Joins* on page 178).
- Adapter-managed join optimization is not in effect (see *Optimizing Joins* on page 178).
- 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 *Optimizing Sorts* on page 182), 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* (see *Valued Expressions* on page 189) or that are otherwise restricted (see *SQL Limitations on Optimization of DEFINE Expressions* on page 190).

Note: FOCUS calculates COMPUTE fields on its internal matrix. They do not affect the ability of the adapter to pass requests for aggregation to the RDBMS.

The adapter 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* on page 189).

Optimizing DEFINE Fields

The adapter can translate certain DEFINE expressions to SQL as part of aggregation or record selection operations only. DEFINE expressions that are not translated for the RDBMS are listed in *SQL Limitations on Optimization of DEFINE Expressions* on page 190.

A virtual field defined as a constant is passed directly to the RDBMS for optimized processing that takes advantage of RDBMS join, sort, and aggregation capabilities. This reduces the volume of RDBMS-to-server communication, which improves response time.

All character constants from the initial TABLE request are passed to the RDBMS unchanged, without truncating trailing spaces.

- 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.

Controlling Optimization of Calculations

Calculations can be processed differently in different RDBMSs and operating environments. If you want FOCUS to handle calculations instead of the RDBMS, you can issue the The SQL SET OPTIMIZATION NOAGGR command. This command disables optimization of calculations (DEFINE fields) without disabling optimization of join and sort operations.

Optimizing DEFINE Fields Referenced in FOCUS BY Clauses (DB2, Teradata, Oracle)

The adapter can optimize aggregation requests in which a DEFINE field is the object of a FOCUS BY clause. The display commands in these requests must be aggregation commands (SUM, COUNT, or WRITE) or prefix operators such as MIN., MAX., and AVE.

- FOCUS and the RDBMS use slightly different sorting/aggregation algorithms. Therefore, when the adapter passes expressions as objects of SQL GROUP BY or ORDER BY phrases to the RDBMS, the sequence of report rows within a given sort key value may vary slightly from the sequence that would be produced by FOCUS. While both report results are equally correct, if such differences are unacceptable you can set adapter optimization OFF. FOCUS will then handle all aggregation, sorts and join operations required to produce the report.
- BY TOTAL phrases are not optimized.

Example: Passing Aggregation on DEFINE Fields to the RDBMS for Processing

This example shows the SQL passed following aggregation on a DEFINE field that is the object of a BY clause:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT
DEFINE FILE DEDUCT
TAX = 0.085 * DED_AMT
END

TABLE FILE DEDUCT
SUM DED_AMT TAX
BY TAX NOPRINT
END
```

- The following SQL is passed to the DB2 DBMS:

```
 SELECT
(0.085 * T1."DA"),
 SUM(T1."DA"),
 SUM((0.085 * T1."DA"))
 FROM
"USER1"."DEDUCT" T1
 GROUP BY
(0.085 * T1."DA")
 ORDER BY
(0.085 * T1."DA")
 FOR FETCH ONLY;
```

- The following SQL is passed to the Teradata DBMS:

```
SELECT (.085 * T1.DA), SUM(T1.DA)(FLOAT), SUM((.085 *
T1.DA))(FLOAT) FROM USER1.DEDUCT T1 GROUP BY (.085 * T1.DA)
ORDER BY (.085 * T1.DA);
```

- The following SQL is passed to the Oracle DBMS:

```
SELECT (.085 * T1."DA"), SUM(T1."DA"), SUM((.085 * T1."DA"))
FROM USER1.DEDUCT T1 GROUP BY (.085 * T1."DA") ORDER BY (.085 *
T1."DA");
```

IF-THEN-ELSE Optimization

The adapter can optimize TABLE requests that include DEFINE fields created using IF-THEN-ELSE syntax. In certain cases, such DEFINE fields can be passed to the RDBMS as CASE expressions, enhancing performance and minimizing the size of the answer set returned to FOCUS.

Syntax: How to Control IF-THEN-ELSE Optimization

When you issue the adapter SET OPTIFTHENELSE command, the adapter attempts to deliver the construct of a FOCUS IF-THEN-ELSEE DEFINE field to the RDBMS as a CASE expression. The DEFINE field must be an object of a selection test or an aggregation request. The DEFINE definition may be specified in the TABLE request or in the Master File.

```
ENGINE sqlengine SET OPTIFTHENELSE {ON|OFF}
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

ON

Generates a CASE statement for IF-THEN-ELSE expressions and DECODE expressions whenever possible. ON is the default value. CASE and ON CASE are synonyms for ON.

OFF

Disables IF-THEN-ELSE optimization. NOCASE and OFF NOCASE are synonyms for OFF.

There is no guarantee that the SQL that is generated will improve performance for all requests. If you find that this feature does not improve performance, set OPTIFTHENELSE OFF to disable the feature.

IF-THEN-ELSE optimization applies to SELECT statements created as a result of FOCUS TABLE requests and is subject to the limitations described in *SQL Limitations on Optimization of DEFINE Expressions* on page 190.

Example: Using IF-THEN-ELSE Optimization Without Aggregation

Consider the following request that has a WHERE condition on an IF-THEN-ELSE DEFINE field named DEF1:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT
ENGINE DB2 SET OPTIFTHENELSE ON
DEFINE FILE EMPINFO
DEF1 = IF (LAST_NAME EQ ' ') AND (FIRST_NAME EQ '  ')
       AND (DEPARTMENT EQ 'MIS') THEN 1 ELSE 0;
END

TABLE FILE EMPINFO
PRINT DEPARTMENT LAST_NAME FIRST_NAME
WHERE DEF1 EQ 1
END
```

The adapter generates an SQL request that incorporates the IF-THEN-ELSE condition corresponding to the IF-THEN-ELSE DEFINE field and the WHERE DEF1 EQ 1 test as a CASE statement:

```
SELECT
T1."LN",
T1."FN",
T1."DPT"
 FROM
USER1."EMPINFO" T1
 WHERE
((CASE WHEN (((T1."LN" = ' ') AND (T1."FN" = ' ')) AND
(T1."DPT" = 'MIS')) THEN 1 ELSE 0 END) = 1)
 FOR FETCH ONLY;
```

Example: Using IF-THEN-ELSE Optimization With Aggregation

The following request displays the maximum salary when an IF-THEN-ELSE DEFINE field named DEF2 equals 1:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT
ENGINE DB2 SET OPTIFTHENELSE ON
DEFINE FILE EMPINFO
DEF2 = IF LAST_NAME EQ 'SMITH' THEN 1 ELSE IF LAST_NAME EQ 'JONES' THEN 2
       ELSE IF LAST_NAME EQ 'CARTER' THEN 3 ELSE 0;
END
TABLE FILE EMPINFO
SUM MAX.CURRENT_SALARY IF DEF2 EQ 1
END
```

The adapter generates an SQL request that incorporates the aggregation, the IF-THEN-ELSE DEFINE field, and the condition corresponding to the WHERE DEF2 EQ 2 test as a CASE statement:

```
AGGREGATION DONE ...
 SELECT
 MAX(T1."CSAL")
 FROM
USER1."EMPINFO" T1
 WHERE
((CASE (T1."LN") WHEN 'SMITH' THEN 1 WHEN 'JONES' THEN 2 WHEN
'CARTER' THEN 3 ELSE 0 END) = 1)
 FOR FETCH ONLY;
```

Example: Using IF-THEN-ELSE Optimization With a Condition That Is Always False

The following request has a condition that is always false because the IF-THEN-ELSE DEFINE field named DEF3 is defined to be either 1 or 0, never 2:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT
ENGINE DB2 SET OPTIFTHENELSE ON
DEFINE FILE EMPINFO
DEF3 = IF FIRST_NAME EQ 'RITA' THEN 1 ELSE 0;
END
TABLE FILE EMPINFO
PRINT FIRST_NAME
IF DEF3 EQ 2
END
```

Because DEF3 EQ 2 will never be true, the adapter passes the CASE statement ((CASE (T1."FN") WHEN 'RITA' THEN 1 ELSE 0 END) = 2), which is always false to the RDBMS, returning zero records from the RDBMS:

```
 SELECT
T1."FN"
 FROM
USER1."EMPINFO" T1
 WHERE
((CASE (T1."FN") WHEN 'RITA' THEN 1 ELSE 0 END) = 2)
 FOR FETCH ONLY;
```

Valued Expressions

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

Reference: 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 data type (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;
```

Reference: 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) data type.
- 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;
```

Reference: 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 data type (including DATE fields).
- Constants of a data type 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 (described in *Valued Expressions* on page 189).
- 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 on Optimization of DEFINE Expressions

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

- FOCUS function calls.
- 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 INCLUDES and EXCLUDES.
- Some expressions involving fields with ACTUAL=DATE. The subtraction of one DATE field from another and all logical expressions on DATE fields can be optimized.
- Some types of date-time manipulation handled by the FOCUS date-time functions is not converted to SQL.
- Financial Modeling Language (FML) cell calculations.

  Note: FML report requests are extended TABLE requests. The Financial Modeling Language provides special functions for detailed reporting. Consult your FOCUS documentation for more information.

In addition, IF-THEN-ELSE optimization does not support the following features:

- Any type of DECODE expression.
- STATIC SQL.
- IF/WHERE DDNAME.
- Partial date selection.

DEFINE FUNCTION Optimization

The DEFINE FUNCTION syntax can be sent directly to an SQL engine as long as all expressions used in the function can be optimized.

Note that in order for DEFINE FUNCTION syntax to be optimized, the types and lengths of the arguments used to call the DEFINE FUNCTION must exactly match the types and lengths of the DEFINE FUNCTION parameters.

For example, the DB2 data source EMPINFO has columns LAST_NAME (alias LN) and FIRST_NAME (alias FN). The following DEFINE FUNCTION takes two arguments, N1 and N2, and sets a flag, which it returns as EMPNAME. EMPNAME has the value 1 if N1 is *Smith* and N2 is *Richard*:

```
DEFINE FUNCTION EMPNAME (N1/A15, N2/A10)
DEF1/I1 = IF N1 EQ 'SMITH' THEN 1 ELSE IF N1 EQ 'JONES' THEN 2
             ELSE IF N1 EQ 'CARTER' THEN 3 ELSE 0;
DEF2/I1 = IF N2 EQ 'RICHARD' THEN 1 ELSE IF N2 EQ 'BARBARA' THEN 2
          ELSE 0;
EMPNAME/I1 = IF DEF1 EQ 1 AND DEF2 EQ 1 THEN 1 ELSE 0;
END
```

The following request uses the result of the DEFINE FUNCTION in an aggregation command. Note that the format of LAST_NAME exactly matches the format defined for N1, and the format of FIRST_NAME exactly matches the format defined for N2:

```
SET TRACEUSER = ON
SET TRACESTAMP = OFF
SET TRACEON = STMTRACE//CLIENT
ENGINE DB2 SET OPTIFTHENELSE ON
DEFINE FILE EMPINFO
DEF3/I1 = EMPNAME(LAST_NAME, FIRST_NAME);
END
TABLE FILE EMPINFO
SUM MAX.LAST_NAME IF DEF3  EQ 1
END
```

The trace output shows that the IF-THEN-ELSE expressions from the DEFINE FUNCTION are translated to an SQL expression in the WHERE predicate of the SELECT statement passed to the RDBMS:

```
 SELECT
 MAX(T1."LN")
 FROM
USER1."EMPINFO" T1
 WHERE
((CASE WHEN (((CASE (T1."LN")  WHEN 'SMITH' THEN 1 WHEN 'JONES'
THEN 2 WHEN 'CARTER' THEN 3 ELSE 0 END) = 1) AND ((CASE
(T1."FN")  WHEN 'RICHARD' THEN 1 WHEN 'BARBARA' THEN 2 ELSE 0
END) = 1)) THEN 1 ELSE 0 END) = 1)
 FOR FETCH ONLY;
```

Optimizing Function Calls

The relational adapters can convert calls to certain FOCUS functions to calls to SQL functions.

The following FOCUS functions can be optimized:

- EDIT (for extracting characters from a column)
- EDIT(integer|real|pack)
- SUBSTR
- SUBSTV
- LOCASE
- UPCASE
- LOCASV
- UPCASV

- TRIMV
- DECODE(char)
- DATEDIF
- DATEADD
- DPART(YEAR|MONTH|DAY|QUARTER)
- HDIFF(DAY|HOUR|MINUTE) (dates only)
- HADD(YEAR|MONTH|DAY|HOUR|MINUTE|SECOND)
- HPART(YEAR|MONTH|DAY|QUARTER) SET INT OPT

Calls to the following FOCUS functions cannot be optimized:

- ABS
- INT
- MAX
- MIN
- LOG
- 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.

Optimization of the HPART, DPART, HDIFF, HDATE, and DATEDIF Functions

The relational adapters can translate certain calls to the HPART, DPART, HDIFF, HDATE, and DATEDIF functions to SQL date and time functions, allowing the optimization of some TABLE requests with DEFINE fields or WHERE phrases that call these functions.

Optimization of the DATEDIF and HDIFF Functions

The DATEDIF function returns the difference between two dates in units, and the HDIFF function returns the difference between two date-time values in units. Calls to these functions can be translated to SQL when the unit is the number of days between two values.

Syntax: **How to Calculate the Number of Days Between Date or Date-Time Values**

The syntax for the DATEDIF function with the *day* unit is

```
DATEDIF(from_date, to_date, 'D')
```

The syntax for the HDIFF function with the *day* unit is

```
HDIFF(to_date_time, from_date_time, 'DAY', outfield)
```

where:

from_date

Is the starting date or date-time value from which to calculate the difference.

to_date

Is the ending date or date-time value from which to calculate the difference.

outfield

Numeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks.

Reference: Conversion of the DATEDIF and HDIFF Functions to SQL

For Oracle, calls to DATEDIF and HDIFF translate calls to the native DATE-TIME functions with Datepart Parameter DAY, Startdate, and Enddate.

For DB2, calls to DATEDIF and HDIFF translate as calls to the DAYS function:

```
SELECT DAYS(d2) - DAYS(d1) FROM tablename
```

For Teradata, calls to DATEDIF and HDIFF translate as a simple difference between two date values for the DATE datatype and as calls to the CAST function for the TIMESTAMP datatype.

Example: Optimizing the Difference Between DB2 Date or Timestamp Columns

The following Master File represents a DB2 table named DATETIME with two DATE fields named DATE1 and DATE2 and two TIMESTAMP fields named TIMEST1 and TIMEST2:

```
FILENAME=DATETIME, SUFFIX=DB2     , $
  SEGMENT=DATETIME, SEGTYPE=S0, $
    FIELDNAME=DATE1, ALIAS=DATE1, USAGE=YYMD, ACTUAL=DATE,
      MISSING=ON, $
    FIELDNAME=DATE2, ALIAS=DATE2, USAGE=YYMD, ACTUAL=DATE,
      MISSING=ON, $
    FIELDNAME=TIMEST1, ALIAS=TIMEST1, USAGE=HYYMDm, ACTUAL=HYYMDm,
      MISSING=ON, $
    FIELDNAME=TIMEST2, ALIAS=TIMEST2, USAGE=HYYMDm, ACTUAL=HYYMDm,
      MISSING=ON, $
```

The following request calculates the number of days difference between DATE1 and DATE2 using the DATEDIF function and the number of days difference between TIMEST1 and TIMEST2 using the HDIFF function:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT
TABLE FILE DATETIME
PRINT DATE1 DATE2 TIMEST1 TIMEST2
WHERE DATEDIF(DATE2, DATE1, 'D') LT 5 AND
HDIFF(TIMEST1, TIMEST1, 'DAY', 'I11') EQ 1
END
```

The SQL generated for this request replaces the calls to DATEDIF and HDIFF with calls to the SQL DAYS function in the SQL WHERE predicate:

```
SELECT T1."DATE1", T1."DATE2", T1."TIMEST1", T1."TIMEST2"
  FROM USER1."DATETIME" T1
 WHERE ((DAYS(T1."TIMEST1") - DAYS(T1."TIMEST1")) = 1) AND
       ((DAYS(T1."DATE1") - DAYS(T1."DATE2")) < 5)
       FOR FETCH ONLY;
```

Example: Optimizing the Difference Between Oracle Date or Timestamp Columns

The following Master File represents an Oracle table named DATETIME with two DATE fields named DATE1 and DATE2 and two TIMESTAMP fields named TIMEST1 and TIMEST2:

```
FILENAME=DATETIME, SUFFIX=SQLORA  , $
  SEGMENT=DATETIME, SEGTYPE=S0, $
    FIELDNAME=DATE1, ALIAS=DATE1, USAGE=YYMD, ACTUAL=DATE,
      MISSING=ON, $
    FIELDNAME=DATE2, ALIAS=DATE2, USAGE=YYMD, ACTUAL=DATE,
      MISSING=ON, $
    FIELDNAME=TIMEST1, ALIAS=TIMEST1, USAGE=HYYMDm, ACTUAL=HYYMDm,
      MISSING=ON, $
    FIELDNAME=TIMEST2, ALIAS=TIMEST2, USAGE=HYYMDm, ACTUAL=HYYMDm,
      MISSING=ON, $
```

The following request calculates the number of days difference between DATE1 and DATE2 using the DATEDIF function and the number of days difference between TIMEST1 and TIMEST2 using the HDIFF function:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT
TABLE FILE DATETIME
PRINT DATE1 DATE2 TIMEST1 TIMEST2
WHERE DATEDIF(DATE2, DATE1, 'D') LT 5 AND
HDIFF(TIMEST1, TIMEST1, 'DAY', 'I11') EQ 1
END
```

The SQL generated for this request replaces the calls to DATEDIF and HDIFF with calls to the SQL EXTRACT and TRUNC functions in the SQL WHERE predicate:

```
SELECT T1."DATE1", T1."DATE2", T1."TIMEST1", T1."TIMEST2"
   FROM OWNER1.DATETIME T1
   WHERE (EXTRACT(DAY FROM (TRUNC(T1."TIMEST1") - TRUNC(T1."TIMEST1"))
   DAY(9) TO SECOND) = 1)
   AND (EXTRACT(DAY FROM (TRUNC(T1."DATE1") - TRUNC(T1."DATE2"))
   DAY(9) TO SECOND) < 5);
```

Optimization of the HPART and DPART Functions

The HPART function extracts a specified component from a date-time value and returns it in numeric format. The DPART function extracts a specified component from a date value and returns it in numeric format. Calls to HPART and DPART are optimized when the second parameter is a case insensitive YEAR, MONTH, QUARTER, DAY-OF-MONTH, or DAY. Calls to HPART are also optimized when the second argument is a case-insensitive HOUR, MINUTE, SECOND, or MICROSECOND.

Syntax: **How to Extract a Component From a Date-Time Value**

```
HPART(dtvalue, 'component', outfield)
```

where:

dtvalue

Date-time

Is a date-time value, the name of a date-time field that contains the value, or an expression that returns the value.

component

Alphanumeric

Is the name of the component to be retrieved enclosed in single quotation marks.

outfield

Integer

Is the field that contains the result, or the integer format of the output value enclosed in single quotation marks.

Syntax: **How to Extract a Date Component and Return a Date Component in Integer Format**

```
DPART(datevalue, 'component', outfield)
```

where:

datevalue

Date

Is a full component date.

component

Alphanumeric

Is the name of the component to be retrieved enclosed in single quotation marks. Valid values are:

For year: YEAR, YY

For month: MONTH, MM

For day: DAY, For day of month: DAY-OF-MONTH.

For quarter: QUARTER, QQ

outfield

Integer

Is the field that contains the result, or the integer format of the output value enclosed in single quotation marks.

Reference: **Conversion of the HPART Function to SQL**

For DB2, calls to HPART translate as calls to the YEAR, MONTH, or DAY function.

For Oracle and Teradata (Both DATE and TIMESTAMP data types), calls to HPART and DPART translate as calls to the EXTRACT function.

Example: **Extracting a Component From a DB2 Timestamp Column**

The following request uses the HPART function to retrieve the year component from the DB2 TIMEST1 column:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT
TABLE FILE DATETIME
PRINT TIMEST1
WHERE HPART(TIMEST1, 'YEAR', 'I4') EQ 2008
END
```

The generated SQL replaces the call to the HPART function with a call to the SQL YEAR function in the SQL WHERE predicate:

```
SELECT T1."TIMEST1" FROM USER1."DATETIME" T1
   WHERE (YEAR(T1."TIMEST1") = 2008)
    FOR FETCH ONLY;
```

Example: Extracting a Component From an Oracle Timestamp Column

The following request uses the HPART function to retrieve the year component from the Oracle TIMEST1 column:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT
TABLE FILE DATETIME
PRINT TIMEST1
WHERE HPART(TIMEST1, 'YEAR', 'I4') EQ 2008
END
```

The generated SQL replaces the call to the HPART function with a call to the SQL EXTRACT function in the SQL WHERE predicate:

```
SELECT T1."TIMEST1" FROM OWNER1.DATETIME T1
    WHERE (EXTRACT(YEAR FROM T1."TIMEST1") = 2008);
```

Optimization of the HDATE Function

The HDATE function converts the date portion of a date-time value to the date format YYMD.

Syntax: How to Extract the Date Portion of a Date-Time Value

```
HDATE(dtvalue, {'YYMD'|outfield})
```

where:

dtvalue

Date-time

Is the date-time value to be converted, the name of a date-time field that contains the value, or an expression that returns the value.

YYMD

Date

Is the output format. The value must be YYMD.

outfield

YYMD

Is the field that contains the result.

Reference: **Conversion of the HDATE Function to SQL**

For DB2 and Teradata, calls to HDATE translate as calls to the CAST function.

For Oracle, calls to HDATE translate as calls to the TRUNC function.

Example: **Extracting the Date Portion of a DB2 Timestamp Column**

The following request uses the HDATE function to extract the date portion of the TIMEST1 column:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT
TABLE FILE DATETIME
PRINT DATE1 TIMEST1
WHERE HDATE(TIMEST1, 'YYMD') GT DATE1
END
```

In the generated SQL, the call to HDATE is replaced by a call to the SQL CAST function in the SQL WHERE predicate:

```
SELECT T1."DATE1", T1."TIMEST1" FROM USER1."DATETIME" T1
   WHERE (CAST(T1."TIMEST1" AS DATE) > T1."DATE1")
   FOR FETCH ONLY;
```

Example: **Extracting the Date Portion of an Oracle Timestamp Column**

The following request uses the HDATE function to extract the date portion of the TIMEST1 column:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT
TABLE FILE DATETIME
PRINT DATE1 TIMEST1
WHERE HDATE(TIMEST1, 'YYMD') GT DATE1
END
```

In the generated SQL, the call to HDATE is replaced by a call to the SQL TRUNC function in the SQL WHERE predicate:

```
SELECT T1."DATE1", T1."TIMEST1" FROM OWNER1.DATETIME T1
    WHERE (TRUNC(T1."TIMEST1") > T1."DATE1");
```

Passing the SUBSTR Character Function to SQL

The SUBSTR character function can be passed to SQL for optimized processing that takes advantage of RDBMS substring functions, thus improving performance and response time.

Passing Function Calls Directly to a Relational Engine Using SQL.Function Syntax

The SQL adapters can pass virtual fields that call certain SQL scalar functions to the relational engine for processing. This enables you to use SQL functions in a request even when they have no equivalent in the WebFOCUS language. The function must be row-based and have a parameter list that consists of a comma-delimited list of column names or constants. In order to reference the function in an expression, prefix the function name with *SQL*.

If the virtual field is in the Master File, both TABLE requests and those SQL requests that qualify for Automatic Passthru (APT) can access the field. If the virtual field is created by a DEFINE FILE command, TABLE requests can access the field. The function name and parameters are passed without translation to the relational engine. Therefore, the expression that creates the DEFINE field must be optimized, or the request will fail.

Reference: Usage Notes for Direct SQL Function Calls

- The expression containing the SQL.*function* call must be optimized or the request will fail with the following message:

  ```
  (FOC32605) NON OPTIMIZABLE EXPRESSION WITH SQL. SYNTAX
  ```

- The function must be a row-based scalar function and have a parameter list that consists of a comma-delimited list of column names or constants. If the function uses anything other than a list of comma separated values, the SQL. syntax cannot be used to pass it.
- Constant DEFINE fields must be assigned a segment location using the WITH phrase.
- Expressions should be declared as DEFINE fields, which are supported as parameters to an SQL function.
- Data types are not supported as parameters to an SQL function. Examples of data type arguments are CHAR and INT for the CONVERT function and ISO, EUR, JIS, and USA for the CHAR function.

Example: Calling the SQL CONCAT Function in a Request

This example uses the WebFOCUS Retail demo sample. You can create this sample data source for a relational adapter by right-clicking the application in which you want to place this sample, and selecting *New* and then *Samples* from the context menu. Then, select *WebFOCUS - Retail Demo* from the *Sample procedures and data for* drop-down list and click *Create*.

The following request against the WebFOCUS Retail demo data source uses the SQL CONCAT function to concatenate the product category with the product subcategory.

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT
SET TRACESTAMP=OFF
SET XRETRIEVAL = OFF

DEFINE FILE WF_RETAIL
CAT_SUBCAT/A50 = SQL.CONCAT(PRODUCT_CATEGORY, PRODUCT_SUBCATEG);
END

TABLE FILE WF_RETAIL
PRINT CAT_SUBCAT
BY PRODUCT_CATEGORY NOPRINT
END
```

The trace output shows that the SQL function call was passed to the RDBMS.

```
SELECT
CONCAT(T2."PRODUCT_CATEGORY",T2."PRODUCT_SUBCATEG"),
T2."PRODUCT_CATEGORY",
T2."PRODUCT_SUBCATEG"
FROM
wfr_product T2
ORDER BY
T2."PRODUCT_CATEGORY"
FOR FETCH ONLY;
```

Example: Calling the SQL COALESCE Function in a Request

The following example creates the table GGDB2 which has columns Biscotti, Capuccino, Croissant, Espresso, and Latte, some of which are null. It then issues a TABLE request that calls the SQL function COALESCE. COALESCE returns the first column in the parameter list that does not contain a null value, if one exists.

The Master File for GGDB2 is:

```
FILENAME=GGDB2  , SUFFIX=DB2     , IOTYPE=STREAM, $
  SEGMENT=GGDB2, SEGTYPE=S0, $
    FIELDNAME=CATEGORY, ALIAS=CAT, USAGE=A11, ACTUAL=A11,
      MISSING=ON, $
    FIELDNAME=Biscotti, ALIAS=BIS, USAGE=I08, ACTUAL=A08,
      MISSING=ON, $
    FIELDNAME=Capuccino, ALIAS=CAP, USAGE=I08, ACTUAL=A08,
      MISSING=ON, $
    FIELDNAME=Croissant, ALIAS=CROI, USAGE=I08, ACTUAL=A08,
      MISSING=ON, $
    FIELDNAME=Espresso, ALIAS=ESP, USAGE=I08, ACTUAL=A08,
      MISSING=ON, $
    FIELDNAME=Latte, ALIAS=LAT, USAGE=I08, ACTUAL=A08,
      MISSING=ON, $
    FIELDNAME=Mug, ALIAS=MUG, USAGE=I08, ACTUAL=A08,
      MISSING=ON, $
    FIELDNAME=Scone, ALIAS=SCO, USAGE=I08, ACTUAL=A08,
      MISSING=ON, $
    FIELDNAME=Thermos, ALIAS=THER, USAGE=I08, ACTUAL=A08,
      MISSING=ON, $
```

The following procedure creates the table GGORA based on the FOCUS data source GGSALES (with MISSING=ON attributes added to the DOLLARS field in the Master File):

```
SET HOLDMISS = ON
SET HOLDLIST=PRINTONLY
SET ASNAMES = ON
TABLE FILE GGSALES
SUM DOLLARS AS ''
BY CATEGORY
ACROSS PRODUCT
WHERE PRODUCT NE 'Coffee Grinder' OR 'Coffee Pot'
ON TABLE HOLD AS GGFLAT FORMAT ALPHA
END

CREATE FILE GGDB2
MODIFY FILE GGDB2
FIXFORM FROM GGFLAT
DATA ON GGFLAT
END
```

The following TABLE request calls the SQL function COALESCE:

```
SET TRACEUSER = ON
SET TRACESTAMP = OFF
SET TRACEON = STMTRACE//CLIENT

DEFINE FILE GGDB2
VALUE/I8 MISSING ON = SQL.COALESCE(Biscotti, Capuccino, Croissant,
Espresso, Latte);
END
TABLE FILE GGDB2
PRINT Biscotti Capuccino Croissant Espresso Latte VALUE
BY CATEGORY
ON TABLE SET PAGE NOPAGE
END
```

The trace output shows that the SQL function call was passed to the RDBMS:

```
 SELECT
COALESCE(T1."BIS", T1."CAP", T1."CROI", T1."ESP", T1."LAT")
T1."CAT",
T1."BIS",
T1."CAP",
T1."CROI",
T1."ESP",
T1."LAT"
 FROM
USER1."GGDB2" T1
 ORDER BY
T1."CAT"
 FOR FETCH ONLY;
```

The output is:

```
CATEGORY     Biscotti  Capuccino  Croissant  Espresso     Latte     VALUE
--------     --------  ---------  ---------  --------     -----     -----
Coffee              .    2401556          .   3906243  11000388   2401556
Food          5387773          .    7758857         .         .   5387773
Gifts               .          .          .         .         .         .
```

The FOCUS EXPLAIN Utility (DB2 and Teradata)

The FOCUS for SQL EXPLAIN utility is an interactive development tool that helps you fine-tune FOCUS query performance. It invokes the RDBMS EXPLAIN function to analyze the efficiency of data retrieval paths for TABLE requests. You cannot use the EXPLAIN utility with MODIFY, MAINTAIN, or MATCH FILE requests. The analysis result is displayed in the FOCUS Hot Screen facility. You can save or print it for further examination.

This section provides:

- An overview of internal processing in *EXPLAIN Processing Overview* on page 204.

- Instructions for invoking the FOCUS EXPLAIN utility and descriptions of its prompting windows in *Using the EXPLAIN Utility* on page 204.
- A sample report request and its EXPLAIN result in *Sample EXPLAIN Report for DB2* on page 208 or *Sample EXPLAIN Report for Teradata* on page 210.

EXPLAIN Processing Overview

Given a TABLE request, the EXPLAIN utility invokes the adapter and generates SQL statements using its normal mechanisms, the FOCUS TABLE parser and Dialogue Manager.

However, instead of processing the request, FOCUS directs the RDBMS to invoke its own native EXPLAIN function and analyze the generated statements. The analysis produces a detailed evaluation of the *access path*, the methodology for retrieving the data for that request.

The RDBMS places this access path information into a table. The FOCUS EXPLAIN utility reads these tables and generates a clear and detailed report containing valuable information about the performance characteristics of your query, information that anyone familiar with the RDBMS and its performance characteristics can understand and analyze.

Using the EXPLAIN Utility

Enter FOCUS and execute the EXPLAIN utility with the following syntax

```
EX expproc
```

where:

expproc

Can be one of the following:

EXPDB2

Is the FOCEXEC that invokes the DB2 EXPLAIN function.

EXPDBC

Is the FOCEXEC that invokes the Teradata EXPLAIN function.

Press the *Enter* key.

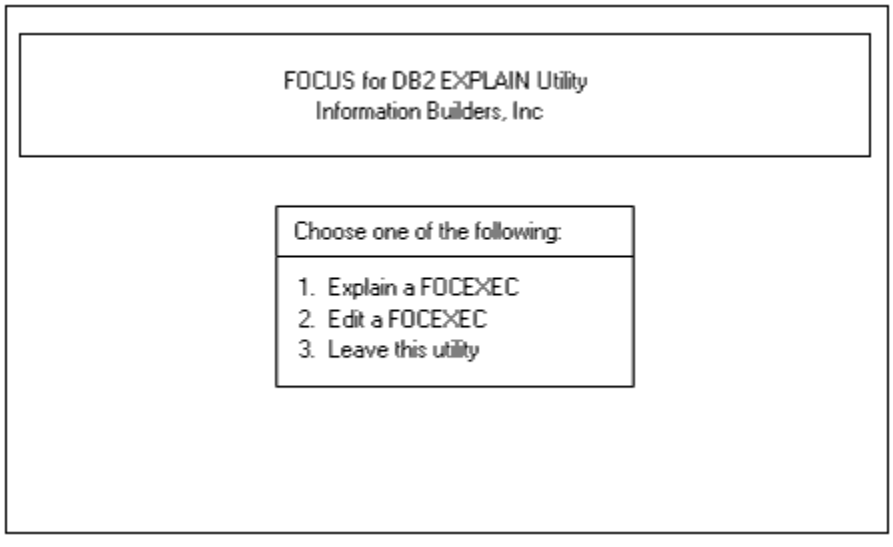
Note: If the IM parameter was set to zero when the adapter was installed, you do not have access to the EXPLAIN utility. The adapter will generate error message FOC1638 if you attempt to use it. See the adapter installation instructions for more information.

The EXPLAIN utility cannot analyze an interactive TABLE request. You must provide the name of a FOCEXEC on the main window. However, you can access the TED editor from the FOCUS EXPLAIN utility and create or change the TABLE request at will.

If you do not have the tables required for executing the RDBMS EXPLAIN function, the FOCUS EXPLAIN utility attempts to create them. In DB2, you need appropriate authorization for creating tables, otherwise an SQLCODE of -551 results. The report results from the FOCUS EXPLAIN utility are displayed in Hot Screen. You can save or print them for later examination.

Note: DB2 v10 and above require the plan table to be created in Unicode with a Unicode database.

The main window presents three choices:

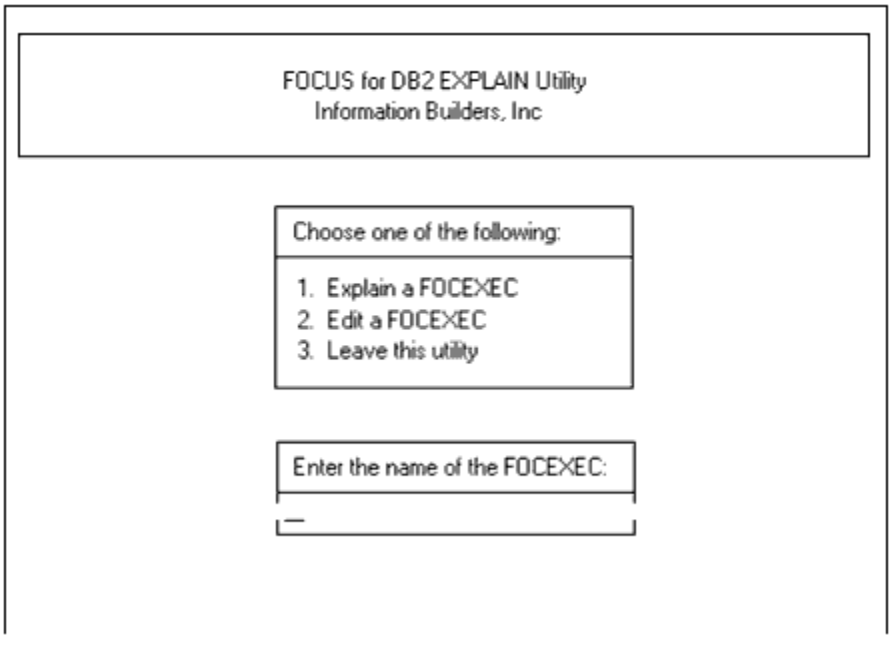


To make a selection, move your cursor under one of the numbers and press the *Enter* key. To exit the utility and return to the FOCUS command line, press the PF3 key. The choices are:

1.	Explain a FOCEXEC	Invokes the RDBMS EXPLAIN command for the TABLE request in your FOCEXEC.
2.	Edit a FOCEXEC	Invokes the TED editor from within the FOCUS EXPLAIN utility. You can create a new FOCEXEC or edit an existing one.
3.	Leave this utility	Returns you to FOCUS, deleting any entries made in the RDBMS EXPLAIN tables.

At any point, you can use the PF3 key to return to a previous window.

If you select Choice 1 or 2, you are asked for the name of your FOCEXEC:



If you are creating a new FOCEXEC, the same FOCEXEC naming conventions apply here as for the FOCUS EXECUTE (EX) command. Specify the z/OS member name. If you have already used TED (Choice 2), the name of the most recent FOCEXEC is automatically supplied.

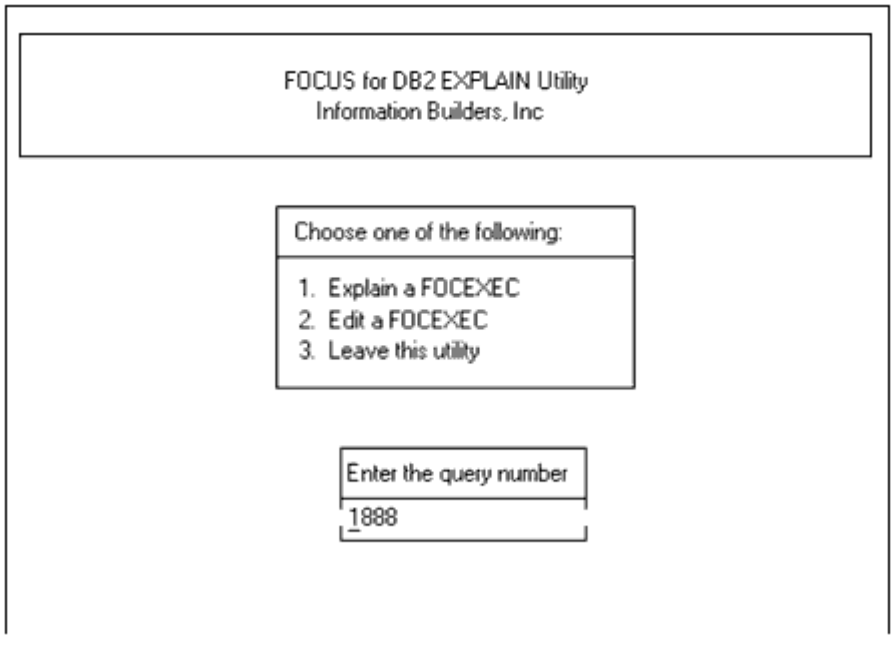
Press the *Enter* key and continue with either the EXPLAIN option or the TED editor:

- The EXPLAIN option.

  For **DB2**, the option to explain a FOCEXEC requires a query number, an internal control number used by the RDBMS when populating the EXPLAIN tables. You can choose any number between 1 and 32,767. However, if you choose a number that already exists, the EXPLAIN utility informs you of its existence and deletes all entries with that number before processing your request. Therefore, make sure the number you choose does not correspond to an existing entry that you would like to keep. If there are no existing entries in the EXPLAIN tables, any number will do.

  For **Teradata**, no query number is needed. If you are prompted for a query number, press *Enter* again.

For example, the query number 1888 has been entered on the following screen:



At this point, the message PLEASE WAIT, PROCESSING YOUR REQUEST displays, followed by your EXPLAIN report.

Your EXPLAIN report is displayed in the Hot Screen facility. All Hot Screen options are available. When you exit Hot Screen, you return to the first window and can evaluate another request.

- The TED editor. Edit a new or existing FOCEXEC. When you are finished, type FILE (to save your edits) or QUIT. Do not issue the RUN command from TED within the EXPLAIN utility. You return to the initial main window and can evaluate your FOCEXEC with Choice 1, the EXPLAIN option.

Note:

- With the EXPLAIN option (Choice 1), if you:
  - Specify a FOCEXEC that generates FOCUS or SQL errors, the EXPLAIN utility stops processing and you return to the utility main menu.
  - Misspell the name of a FOCEXEC or specify one that does not exist, the EXPLAIN utility stops processing and you return to the FOCUS command prompt.

- Do not execute large Dialogue Manager applications through the EXPLAIN utility. The FOCUS TABLE parser executes the Dialogue Manager statements and may produce unpredictable results.
- Create separate FOCEXECs for evaluating individual report requests. Experiment with alternate parameters to improve response time and RDBMS performance.
- Do not use the STMTRACE component in any FOCEXEC that you evaluate with the EXPLAIN utility.

Sample EXPLAIN Report for DB2

FOCEXEC RPT1 contains the following TABLE request:

```
TABLE FILE INVQ5
SUM PRICE BY PARTNO
WHERE DESCRIPTION EQ 'BOLT' OR 'NUT' OR 'SCREW'
WHERE PRICE GT .30
IF TOTAL PRICE GT 1
END
```

The EXPLAIN output consists of two reports. The first is a one-page report:

```
PAGE     1
    EXPLAIN REPORT 1 FOR FOCEXEC RPT1                RUN ON 06/01/99
                                                QUERY NUMBER IS 1000
                THE FOLLOWING SQL STATEMENT(S) WILL BE EXPLAINED
    SELECT T1.PARTNO, SUM(T2.PRICE) FROM "TESTID"."INVENT5" T1,
   "TESTID"."QUOT5" T2 WHERE (T2.PARTNO = T1.PARTNO) AND
   (T1.DESCRIPTION IN('BOLT', 'NUT', 'SCREW')) AND (T2.PRICE > .3)
   GROUP BY T1.PARTNO HAVING (SUM(T2.PRICE) > 1.) ORDER BY
   T1.PARTNO FOR FETCH ONLY;

MORE
```

The second is a four-page report:

```
   PAGE     1
    EXPLAIN REPORT 2 FOR FOCEXEC RPT1                    RUN ON 06/01/99
                                          QUERY NUMBER IS 1000
    1ST   ACCESS OF DATA
   TABLE NAME .......................:  TESTID.INVENT5
   TABLE NUMBER .....................:  1
   JOIN METHOD ......................:  FIRST TABLE ACCESSED
   MULTIPLE INDEX OPERATION SEQUENCE .:  0
   INDEX ACCESS TYPE ................:  DIRECT INDEX ACCESS
   # OF INDEX KEYS USED ..............:  0
   INDEX NAME .......................:  TESTID.INVENT5IX
   INDEX ONLY ACCESS? ...............:  NO
   SORT OF BASE TABLE REQUIRED FOR ...:  NOTHING
   SORT OF RESULT TABLE REQUIRED FOR .:  NOTHING
   LOCKING MODE .....................:  INTENT SHARE
   TYPE OF PREFETCH .................:  UNKNOWN/NO PREFETCH
   COLUMN FUNCTION EVALUATED AT ......:  N/A OR DETERMINED AT EXECUTION

 MORE
```

```
 PAGE     2
     EXPLAIN REPORT 2 FOR FOCEXEC RPT1                   RUN ON 06/01/99
                                          QUERY NUMBER IS 1000
   PARALLEL ACCESS DEGREE ............:  .
   PARALLEL ACCESS GROUP .............:  .
   PARALLEL JOIN DEGREE ..............:  .
   PARALLEL JOIN GROUP ...............:  .

 MORE
```

```
  PAGE     3
    EXPLAIN REPORT 2 FOR FOCEXEC RPT1                RUN ON 06/01/99
                                          QUERY NUMBER IS 1000
   2ND   ACCESS OF DATA
  TABLE NAME .....................:  TESTID.QUOT5
  TABLE NUMBER ...................:  2
  JOIN METHOD ....................:  NESTED LOOP JOIN
  MULTIPLE INDEX OPERATION SEQUENCE .:  0
  INDEX ACCESS TYPE ..............:  DIRECT INDEX ACCESS
  # OF INDEX KEYS USED ...........:  0
  INDEX NAME .....................:  TESTID.QUOT5IX
  INDEX ONLY ACCESS? .............:  NO
  SORT OF BASE TABLE REQUIRED FOR ...:  NOTHING
  SORT OF RESULT TABLE REQUIRED FOR .:  NOTHING
  LOCKING MODE ...................:  INTENT SHARE
  TYPE OF PREFETCH ...............:  UNKNOWN/NO PREFETCH
  COLUMN FUNCTION EVALUATED AT ......:  N/A OR DETERMINED AT EXECUTION

MORE
```

```
PAGE     4
    EXPLAIN REPORT 2 FOR FOCEXEC RPT1                RUN ON 06/01/99
                                          QUERY NUMBER IS 1000
  PARALLEL ACCESS DEGREE ...........:  .
  PARALLEL ACCESS GROUP ............:  .
  PARALLEL JOIN DEGREE .............:  .
  PARALLEL JOIN GROUP ..............:  .
  IF YOUR REQUEST SPONSORED A SEQUENTIAL SCAN OF THE DATA, OR
  ONE OR MORE ADDITIONAL SORTS, ESPECIALLY ON THE COMPOSITE
  RESULT TABLES, YOU MAY WISH TO REPHRASE THIS REPORT
                                END OF REPORT
```

For information about the EXPLAIN report, consult the *DB2 Administration Guide*.

Sample EXPLAIN Report for Teradata

Suppose you want to analyze the following request, stored as SAMP1 FOCEXEC:

```
JOIN EMP_ID IN EMPINFO TO ALL WHO IN FUNDTRAN
TABLE FILE EMPINFO
WRITE AVE.CURRENT_SALARY ED_HRS BY WHO BY LAST_NAME
IF DEPARTMENT_CD IS MIS
END
```

The EXPLAIN utility produces a two-page report:

```
PAGE   1

    EXPLAIN REPORT FOR SAMP1                    RUN ON 10/10/99

 -_
 SQL STATEMENT AS FOLLOWS:
 SELECT T2.EID(CHAR( 9), UPPERCASE),T1.LNAME(CHAR( 15),
 UPPERCASE), AVG(T1.CURRENT_SALARY)(DECIMAL(15, 2)),
 SUM(T1.OJT)(FLOAT) FROM EMPINFO T1,FUNDTRAN T2 WHERE (T2.EID =
 T1.EID) AND (T1.DEPARTMENT_CD = ¢MIS¢) GROUP BY T2.EID,T1.LNAME
 ORDER BY T2.EID,T1.LNAME

 1) First, we lock JANE.T2 for read, and we lock JANE.T1 for read.
 2) Next, we do an all-AMPs JOIN step from JANE.T2 by way of an
    all-rows scan with no residual conditions, which is joined to
    JANE.T2 and JANE.T1 are joined using a merge join, with a join
    condition of ("JANE.T2.EID = JANE.T1.EID"). The result goes into
    Spool 2, which is built locally on the AMPs. The size
```

```
PAGE   2

    EXPLAIN REPORT FOR SAMP1                    RUN ON 10/10/99

 _
    of Spool 2 is estimated to be 4 rows. The estimated time for this
    step is 0.10 seconds.
 3) We do a SUM step to aggregate from Spool 2 (Last Use) by way of an
    all-rows scan. Aggregate Intermediate Results are computed
    globally, then placed in Spool 3.
 4) We do an all_AMPs RETRIEVE step from Spool 3 (Last Use) by way of
    an all-rows scan into Spool 1, which is built locally on the AMPs.
    Then we do a SORT to order Spool 1 by the sort key in spool field1.
    The size of Spool 1 is estimated to be 2 rows. The estimated time
    for this step is 0.07 seconds.
 5) Finally, we send out an END TRANSACTION step to all AMPs involved
     in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of
   statement 1.
```

Chapter 8

Advanced Reporting Techniques

Using a relational adapter, you can create graphs with GRAPH requests, design interactive procedures with Dialogue Manager, or extract data from reports with the HOLD command. However, of all the facilities the adapter makes available to you for reading, retrieving, and organizing data from RDBMS tables, the most widely used is the Report Writer, invoked by the TABLE, TABLEF, and MATCH commands. Some reporting features for the adapter vary from standard FOCUS. These minor variations are discussed in this chapter. This chapter includes:

- Improving retrieval performance with the TABLEF command.
- Creating RDBMS tables from a TABLE request.
- Joined structures for reporting purposes. How multi-table Master Files compare with dynamically-joined structures, the JOIN command for unique and non-unique relationships, and the multi-field JOIN command.
- The SET ALL command and multi-table structures. How short retrieval paths affect report results.
- The join utilities CHECK FILE, ? JOIN, and JOIN CLEAR.
- Specifying search limits with the READLIMIT and RECORDLIMIT.
- For multi-table structures only, multiple retrieval paths and how sort phrases and screening conditions affect retrieval.

If you are not familiar with FOCUS syntax, you can create report requests with TableTalk, a menu-driven report generator.

In this chapter:

FOCUS and SQL Similarities

Some FOCUS and SQL query functions are similar:

- SQL SELECT is equivalent to the FOCUS command 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 file and is available as a source for further report processing. Refer to your FOCUS documentation 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* on page 265).

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 further improve efficiency, 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 the request requires fewer I/Os to data pages.

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

- If BY phrases are passed to the RDBMS as SQL ORDER BY phrases, the answer sets returned are sorted 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 Adapter Optimizer* on page 171), so you must verify that the ORDER BY is passed before relying on the sorted order of the returned answer set. Use STMTRACE (see *Tracing Adapter Processing* on page 487) to examine the SQL statements generated by the adapter.

- ACROSS phrases are not supported.
- Requests with multiple display commands (multi-verb requests) are not supported.
- The RETYPE command is not available.
- You can include the ON TABLE HOLD command 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 adapter issues a COMMIT (usually when the report display is terminated).

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. (For a detailed discussion of locks and isolation levels, see *Maintaining Tables With FOCUS* on page 349.) This consideration does not apply to regular TABLE requests, because the adapter 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 Tables Using the HOLD Command

Using 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 HOLD FORMAT DB2, SQLIDMS, SQLDBC, or SQLORA 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. Your site must also have enabled WRITE access and native SQL support when installing the adapter. Contact your site DBA for more information.

To extract data and convert it to an RDBMS table, issue the HOLD command with the FORMAT DB2, SQLDBC, SQLIDMS, or SQLORA option either in the report request or after the report has printed. The adapter generates a single-table Master and Access File, and it creates and loads one RDBMS table. If the report request uses the display command 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 adapter creates a unique index on any BY fields.

If you attempt to issue the HOLD FORMAT *sqlengine* command without first installing the adapter, the following error messages are generated:

```
(FOC1488)SQL INTERFACE IS NOT INSTALLED

(FOC1479)ERROR CONNECTING TO SQL DATABASE
```

Note: *Adapter Commands* on page 309 describes how to control index space parameters with the Adapter for DB2, IDMS/SQL, or Oracle SET IXSPACE command.

The RDBMS table name that results from the extract must not already exist. (See *Adapter Commands* on page 309 for a discussion of adapter environmental commands that allow you to control where the table is placed.)

Within the report request, the syntax is

```
ON TABLE HOLD [ AS name ] FORMAT sqlengine
```

At the command level, the syntax is

```
HOLD [ AS name ] FORMAT sqlengine
    [TABLENAME table] [CONNECTION con] [DROP]
```

where:

name

Is a name for the extract Master File. If the TABLENAME attribute is not specified, it is also the name of the resulting table. The default name is HOLD. Maximum length depends on the table name length and format for each adapter (including a period to separate the creator ID from the table name). All characters become the TABLENAME value in the Access File. Long Master File names are supported. If the HOLD command includes a long AS name, both the Master and Access File names will be long. The Master and Access Files will be named according to the procedure described in *Describing Tables to FOCUS* on page 55.

Unless the TABLENAME attribute is specified, the AS name also becomes the table name and is included in the Access File. If the AS name specified in the HOLD command is longer than the table name length supported by the RDBMS, the table cannot be created.

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 Master File. Consult *Describing Tables to FOCUS* on page 55 for information about creator IDs and TABLENAME values.

sqlengine

Identifies the type of RDBMS table created. Valid values are:

`SQL` or `DB2` indicates that the output data source is stored as a table in DB2. DB2 is a synonym for SQL.

`SQLDBC` indicates that the output data source is stored as a Teradata table.

`SQLIDMS` indicates that the output data source is stored as a CA-IDMS table.

`SQLORA` indicates that the output data source is stored as an Oracle table.

`table`

Is the name of the resulting table in the DBMS. It must conform to the name length supported by the RDBMS or the table cannot be created.

`con`

Is a connection name (CLI only).

`DROP`

Before implementing the HOLD command to create the table, drops an existing table with the same name.

Unless you give them an AS name, file descriptions are temporary and exist only for the current session. All HOLD tables are permanent and must be explicitly dropped.

To make the HOLD file descriptions permanent, specify an AS name in the HOLD command and allocate the data sets that contain the generated Master and Access files to permanent partitioned data sets. The generated Master and Access Files are created as members of the partitioned data sets allocated to DDNAMEs HOLDMAST and HOLDACC. If you do not allocate those DDNAMEs, they are allocated dynamically and deleted at the end of the session. To permanently retain the new file descriptions, allocate HOLDMAST to a permanent partitioned data set and HOLDACC to a second permanent partitioned data set. If you allocate HOLDMAST and HOLDACC, do *not* specify DCB parameters.

Example: Converting the FOCUS PROD Database to a DB2 Table

This example converts the FOCUS PROD data source to a DB2 table:

```
sql db2 set dbspace public.space0
>
table file prod
print *
on table hold as user1.prodsql format sql
end

    NUMBER OF RECORDS IN TABLE=       14  LINES=     14


    HOLDING SQLDS    FILE...
```

Master Files Generated by HOLD

Even if the original Master File describes several segments, the Master File resulting from the ON TABLE HOLD command is a single-table description. It contains the attributes described in *Describing Tables to FOCUS* on page 55. Following is a list of the generated keyword/valuepairs:

- 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. On z/OS, longer Master File names are supported (see *Describing Tables to FOCUS* on page 55). 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 Master File.
- The SUFFIX value is SQLDS for DB2, SQLDBC for Teradata, SQLIDMS for IDMS/SQL, or SQLORA for Oracle.
- The SEGNAME value is SEG01 and the SEGTYPE value is S0.
- FIELDNAME and ALIAS values from the original Master File are retained. If the original Master File does not define aliases, the original field names 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 field name becomes the new value for ALIAS. The AS phrase name is also included as a TITLE value.

  The FOCUS PRINT and LIST commands 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 the charts in *Extract File Conversion Chart for DB2 and DB2 for VM* on page 222). If the original Master File contains ACTUAL formats, they are ignored.
- MISSING parameter settings are converted to SQL NULL statements.

Example: Sample Master File Generated by HOLD

The following is the generated Master File from the ON TABLE HOLD AS PRODSQL command in *Converting the FOCUS PROD Database to a DB2 Table*:

```
FILE=PRODSQL    ,SUFFIX=SQLDS,$
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    ,$
```

Example: Creating a DB2 Table With a Long Name on OS/390

The following request creates a 15-character DB2 table named EMPLOYEEINFODB2:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID CURR_SAL
BY DEPARTMENT
BY LAST_NAME
BY FIRST_NAME
ON TABLE HOLD AS USER1.EMPLOYEEINFODB2 FORMAT DB2
END
```

This request creates the following Master File:

```
$ VIRT=EMPLOYEEINFODB2

FILE=EMPLOYEEINFODB2       ,SUFFIX=SQL
SEGNAME=SEG01   ,SEGTYPE=S0
FIELDNAME   ='DEPARTMENT'       ,'DPT'         ,A10      ,A10      ,$
FIELDNAME   ='LAST_NAME'        ,'LN'          ,A15      ,A15      ,$
FIELDNAME   ='FIRST_NAME'       ,'FN'          ,A10      ,A10      ,$
FIELDNAME   =FOCLIST            ,FOCLIST       ,I5       ,I4       ,$
FIELDNAME   ='EMP_ID'           ,'EID'         ,A9       ,A9       ,$
FIELDNAME   ='CURR_SAL'          ,'CSAL'        ,D12.2M   ,D8        ,$
```

This request also creates the following Access File. The AS name is also the table name:

```
$ VIRT=EMPLOYEEINFODB2
SEGNAME=SEG01   ,
TABLENAME=USER1.EMPLOYEEINFODB2
KEYS=04  , WRITE=YES, $
```

When the AS name is longer than the supported length for table names in DB2 on z/OS, the following messages are generated:

```
>   NUMBER OF RECORDS IN TABLE=       12  LINES=     12
 HOLDING SQL      FILE...
(FOC1400) SQLCODE IS -107 (HEX: FFFFFF95)
(FOC1414) EXECUTE IMMEDIATE ERROR.
```

The Master and Access Files are created. However, the table cannot be created because the table name specified in the Access File is too long and, therefore, invalid. Note that you will get a more descriptive message if you issue the following command:

```
SQL DB2 SET ERRORTYPE DBMS
```

For example:

```
(FOC1400) SQLCODE IS -107 (HEX: FFFFFF95)
  :  DSNT408I SQLCODE = -107, ERROR:  THE NAME name IS
TOO
  :           LONG.  MAXIMUM ALLOWABLE SIZE IS xx  :  DSNT418I SQLSTATE   =
42622 SQLSTATE RETURN CODE
  :  DSNT415I SQLERRP    = DSNHSMUD SQL PROCEDURE DETECTING ERROR
  :  DSNT416I SQLERRD    = 0  0  0  -1  15  0 SQL DIAGNOSTIC INFORMATION
  :  DSNT416I SQLERRD    = X'00000000'  X'00000000'  X'00000000'
  :           X'FFFFFFFF'  X'0000000F'  X'00000000' SQL DIAGNOSTIC
  :           INFORMATION
(FOC1414) EXECUTE IMMEDIATE ERROR.
```

Access Files Generated by HOLD

The Access File resulting from the ON TABLE HOLD command contains the declarations described in *Describing Tables to FOCUS* on page 55. 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 defaults to 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 for DB2, the schema name for IDMS, and the user ID for Oracle. Characters following the period become the table name.

  Note: For DB2, Teradata, and Oracle, if you do not specify a creator name, the ID specified by the SET OWNERID command becomes the creator. If you did not issue this command, your user ID becomes the creator, by default. For IDMS, an unqualified table name will be generated.
- The KEYS value and the fields that are indexed depend on the display command 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 command 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 display command.
- The adapter 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 adapter 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.

Example: Sample Access File Generated by HOLD

Following is the generated Access File from the ON TABLE HOLD AS PRODSQL statement in *Converting the FOCUS PROD Database to a DB2 Table*:

```
SEGNAME=SEG01   ,
TABLENAME="USER1".PRODSQL  ,
KEYS=01  , WRITE=YES, $
```

Other Files Generated by HOLD

Three work files, FOC$HOLD MASTER, FOC$HOLD FOCTEMP, and FOCSORT, are used with an internal MODIFY procedure to create and load the output table. The FOC$HOLD Master File is a fixed-format file with a corresponding sequential data file.

Usage Restrictions for HOLD

- You may not use the HOLD FORMAT *sqlengine* option in multi-verb TABLE requests. Only one display command is allowed.
- You may not use the ACROSS sort phrase.
- Names used in AS phrases to rename fields should begin with an alphabetic character. If the name begins with a digit, SQL error -105 or SQL/DBC error 3708 results and the table is not generated. For more information on AS names, refer to your FOCUS documentation.
- The CNT.DST. prefix operator is not supported.

Extract File Conversion Charts

The following charts show original USAGE formats, conditions, and resulting USAGE and ACTUAL formats for ON TABLE HOLD. The field length is represented by *n*.

Reference: **Extract File Conversion Chart for DB2 and DB2 for VM**

Non-SQL USAGE	Conditions	HOLD USAGE	HOLD ACTUAL
A*n*	none	A*n*	A*n*
D*n*	none	D*n*	D8
F*n*	none	F*n*	F4
I*n*	*n* EQ 1 or 2 *n* GT 2 MISSING=ON	I*n* I*n* I*n*	I2 I4 I4
P*n.m*	*n* LE 31 MISSING=ON	P*n.m* P*n*, *n*<15	P(trunc((*n*+2)/2)) P8
date	Date format	date	DATE
TX*nn*	TEXT field	TX*nn*	TX
HYYMDm	Timestamp field	HYYMDm	HYYMDm
HHIS	Time format	HHIS	HHIS

Note:

- USAGE field types A, D, F, HYYMDm, HHIS, and TX from the original Master File remain the same. Their generated ACTUAL formats vary slightly.
- USAGE field types I and P from the original Master File 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.
- To create SMALLINT NOT NULL columns with the CREATE FILE command or HOLD FORMAT DB2, use an ACTUAL attribute of I2, and then change the ACTUAL attribute to I4. In all other cases, use I4 for the ACTUAL attribute.

Reference: Extract File Conversion Chart for Teradata

Non-DBC/ SQLUSAGE	Conditions	HOLDUSAGE	HOLDACTUAL
A*n*	none	A*n*	A*n*
D*n*	none	D*n*	D8
F*n*	none	D*n*	D8
I*n*	*n* EQ 1 or 2 *n* GT 2 MISSING=ON	I*n* I*n* I*n*	I2 I4 I4
P*n.m*	*n* < 18 *n* = 18	P*n.m* P*n.m*	P8P10
date	date format	date	DATE
TX*nn*	TEXT field	TX*nn*	TX

Note:

- USAGE field types I and P from the original Master File are converted based on the original length and on whether null values are permitted (MISSING=ON).
- USAGE field type F is converted to HOLD USAGE and ACTUAL formats of D.
- FOCUS date formats as USAGE values remain the same. They are converted to the ACTUAL format DATE.

Reference: Extract File Conversion Chart for IDMS SQL

Non-SQL USAGE	Conditions	HOLD USAGE	HOLD ACTUAL
A*n*	none	A*n*	A*n*
D*n*	none	D*n*	D8
F*n*	none	F*n*	F4

Non-SQL USAGE	Conditions	HOLD USAGE	HOLD ACTUAL
I*n*	*n* EQ 1 or 2 *n* GT 2 MISSING=ON	I*n* I*n* I*n*	I2I4I4
P*n.m*	*n* LE 31 MISSING=ON	P*n.m* P*n.m*, *n*£15	P(trunc((*n*+2)/2))P8
date	Date format	date	DATE
TX*nn*	TEXT field	TX*nn*	TX

Note:

- USAGE field types A, D, F, and TX from the original Master File remain the same. Their generated ACTUAL formats vary slightly.
- USAGE field types I and P from the original Master File 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.
- To create SMALLINT NOT NULL columns with the CREATE FILE command or HOLD FORMAT SQLIDMS, use an ACTUAL attribute of I2, and then change the ACTUAL attribute to I4. In all other cases, use I4 for the ACTUAL attribute.

Reference: Extract File Conversion Chart for Oracle

Non-SQL USAGE	Conditions	HOLD USAGE	HOLD ACTUAL
A*n*	none	A*n*	A*n*
D*n*	none	D20.2	D8
I*n*	none	I11	I4
P*n.m*	*n* LE 31 MISSING=ON	P*n.m* P*n*, *n*£15	P(trunc((*n*+2)/2)) P8
date	Date format	HYYMDS	H08

Non-SQL USAGE	Conditions	HOLD USAGE	HOLD ACTUAL
TX*nn*	TEXT field	TX*nn*	TX

HOLD FORMAT SAME_DB

You can create a report output file, that is, a HOLD file, as a native DBMS temporary table. This increases performance by keeping the entire reporting operation on the DBMS server, instead of downloading data to your computer and then back to the DBMS server.

For example, if you temporarily store report output for immediate use by another procedure, storing it as a temporary table instead of creating a standard HOLD file avoids the overhead of transmitting the interim data to your computer.

The temporary table columns are created from the following report elements

- Display columns
- Sort (BY) columns
- COMPUTE columns

except for those for which NOPRINT is specified.

The temporary table that you create from your report will be the same data source type (that is, the same DBMS) as the data source from which you reported. If the data source from which you reported contains multiple tables, all must be of the same data source type and reside on the same instance of the DBMS server.

You can choose between several types of table persistence.

You can create extract files as native DBMS tables with the following adapters:

- DB2 (on z/OS, UNIX, and Windows)
- Oracle
- Teradata

Syntax: How to Save Report Output as a Temporary Table

The syntax to save report output as a native DBMS temporary table is

```
ON TABLE HOLD [AS filename]FORMAT SAME_DB [PERSISTENCE persistValue]
```

where:

filename

Specifies the name of the HOLD file. If you omit AS *filename*, the name of the temporary table defaults to "HOLD".

Because each subsequent HOLD command overwrites the previous HOLD file, it is recommended to specify a name in each request to direct the extracted data to a separate file, thereby preventing an earlier file from being overwritten by a later one.

PERSISTENCE

Specifies the type of table persistence and related table properties. This is optional for DBMSs that support volatile tables, and required otherwise. For information about support for volatile tables for a particular DBMS, see Temporary Table Properties for SAME_DB Persistence Values, and consult your DBMS vendor's documentation.

persistValue

Is one of the following:

VOLATILE

Specifies that the table is local to the DBMS session. A temporary synonym—a Master File and Access File—is generated automatically; it expires when the server session ends.

This is the default persistence setting for all DBMSs that support volatile tables.

For information about support for the volatile setting, and about persistence and other table properties, for a particular DBMS, see Temporary Table Properties for SAME_DB Persistence Values, and consult your DBMS vendor's documentation.

GLOBAL_TEMPORARY

Specifies that while the table exists, its definition will be visible to other database sessions and users though its data will not be. A permanent synonym—a Master File and Access File—is generated automatically.

For information about support for the global temporary setting, and about persistence and other table properties, for a particular DBMS, see Temporary Table Properties for SAME_DB Persistence Values, and consult your DBMS vendor's documentation.

PERMANENT

Specifies that a regular permanent table will be created. A permanent synonym—a Master File and Access File—is generated automatically.

Reference: **Temporary Table Properties for SAME_DB Persistence Values**

The following chart provides additional detail about persistence and other properties of temporary tables of different data source types that are supported for use with HOLD format SAME_DB.

Informix

Microsoft SQL Server

MySQL

A volatile table is created using the CREATE TEMP TABLE command with the WITH NO LOG option. The definition and the data persist, and are visible, only within the current session.

A volatile table is created as a local temporary table whose name is prefixed with a single number sign (#). Therefore, the name of a volatile table used as a HOLD file is the name specified by the HOLD phrase, prefixed with a number sign (#). The table's definition and the data persist, and are visible, only within the current session.

A volatile table is created using the CREATE TEMPORARY TABLE command. A temporary table persists and is visible only within the current session (connection). If a temporary table has the same name as a permanent table, the permanent table becomes invisible.

This type of table is not supported by Informix.

The name of a global temporary table is prefixed with two number signs (##). Therefore, the name of a global temporary table used as a HOLD file is the name specified by the HOLD phrase, prefixed with two number signs (##). The table is dropped automatically when the session that created the table ends and all other tasks have stopped referencing it. The table's definition and data are visible to other sessions.

This type of table is not supported by MySQL.

DBMS	VOLATILE	GLOBAL_TEMPORARY
DB2	DB2: on UNIX, Windows, and DB2 for z/OS: a volatile table is created using the DECLARE GLOBAL TEMPORARY TABLE command with the ON COMMIT PRESERVE ROWS option. Declared global temporary tables persist and are visible only within the current session (connection). SESSION is the schema name for all declared global temporary tables.	DB2 Release 7.1 and up for z/OS only: a global temporary table is created using the CREATE GLOBAL TEMPORARY TABLE command. The definition of a created global temporary table is visible to other sessions, but the data is not. The data is deleted at the end of each transaction (COMMIT or ROLLBACK command). The table definition persists after the session ends.
Oracle	This type of table is not supported by Oracle.	The table's definition is visible to all sessions; its data is visible only to the session that inserts data into it. The table's definition persists for the same period as the definition of a regular table.

DBMS	VOLATILE	GLOBAL_TEMPORARY
Teradata	A volatile table definition and data are visible only within the session that created the table and inserted the data. The volatile table is created with the ON COMMIT PRESERVE ROWS option.	A global temporary table persists for the same duration as a permanent table. The definition is visible to all sessions, but the data is visible only to the session that inserted the data. The global temporary table is created with the ON COMMIT PRESERVE ROWS option.

Column Names in the HOLD File

Each HOLD file column is assigned its name:

1. From the AS name specified for the column in the report request.
2. If there is no AS name specified, the name is assigned from the alias specified in the synonym. (The alias is identical to the column name in the original relational table.)
3. In all other cases, the name is assigned from the field name as it is specified in the synonym.

Primary Keys and Indexes in the HOLD File

A primary key or an index is created for the HOLD table. The key or index definition is generated from the sort (BY) keys of the TABLE command, except for undisplayed sort keys (that is, sort keys for which NOPRINT is specified). To determine whether a primary key or an index will be created:

1. If these sort keys provide uniqueness and do not allow nulls (that is, if in the synonym the column's MISSING attribute is unselected or OFF), and if the DBMS supports primary keys on the type of table being created, a primary key is created.
2. If these sort keys provide uniqueness but either
   a. some of the columns allow nulls
   b. the DBMS does not support primary keys on the type of table being created.
3. If these sort keys do not provide uniqueness, a non-unique index is created.
4. If there are no displayed sort keys (that is, no sort keys for which NOPRINT has not been specified), no primary key or index is created.

Using the Dynamic JOIN Command

With the dynamic JOIN command, you can reference two or more related tables (or non-FOCUS data sources) in a single FOCUS report request. The data structures remain physically separate, but FOCUS treats them as a single logical structure.

Two types of join are available, the equijoin and the conditional (WHERE-based join).

The terms *primary key* and *foreign key* refer to the common columns that relate host and cross-referenced tables in an equijoin.

Although the run-time effect of the dynamic join is very similar to that of the embedded join discussed in *Multi-Table Structures* on page 99, the dynamic join is easier to construct since it does not require a separate Master File and Access File. You can create a dynamic join on an as-needed basis.

Note: A TABLE request that references a dynamically joined structure generates SQL join predicates for all segments in the subtree that starts from the root segment. Multi-table Master Files do not necessarily generate these predicates. In a multi-table structure, the subtree effectively begins with the highest referenced segment. This difference may cause identical TABLE requests to produce different reports when run against a dynamic join structure and a multi-table Master File 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 Master File support both read-only and read-write operations.

FOCUS joins can be unique or non-unique. 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 *leftouter 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 Adapter Optimizer* on page 171). In order to understand join behavior, you must know whether optimization is enabled or disabled for a particular report request. The SQLAGGR trace gives you that information (see *Tracing Adapter Processing* on page 487).

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 sections describe how these factors interact. SET ALL is described in *The SET ALL Command* on page 247.

Reference: Preserving Virtual Fields During Join Parsing

By default, a JOIN command clears all DEFINE FILE commands for the host data source and the joined structure. Two methods are available for preserving virtual fields during join parsing, the KEEPDEFINES parameter and the DEFINE FILE SAVE and RETURN commands. For complete information, see your FOCUS documentation.

Constructing a Single-field Dynamic Equijoin

A single-field dynamic equijoin is a join based on relating values in one column of the host and cross-referenced tables.

Syntax: How to Construct a Single-Field Dynamic Equijoin

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

where:

fielda1

Is a field in the host file or a DEFINE field that shares values and format with fieldb1 in the cross-referenced file.

WITH *rfield*

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.

hostfile

Is the name of the host file. Use this name in subsequent TABLE requests on the joined structure.

tag1

Is the tag name for the host file, a one- to eight-character table name 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 table name in the structure.

ALL

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 discussed in *The SET ALL Command* on page 247.

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.

fieldb1

Is a field in the cross-referenced file whose values and format can match that of fielda1.

crfile

Is the name of the cross-referenced file.

tag2

Is the tag name for the cross-referenced file, a one- to eight-character table name qualifier for field and alias names in the cross-referenced file. The tag name may not be the same as any table name in the structure. In a recursive join structure, if no tag name is provided, FOCUS prefixes all field names and aliases with the first four characters of the join name.

joinname

Assigns an internal name to the join structure, up to eight characters in length. Joinname also provides a unique prefix for field names participating in a recursive join.

You can clear the join name when you no longer need this join (see *JOIN CLEAR* on page 259). Do not specify *joinname* as the file name in subsequent TABLE FILE requests; use hostfile.

`END`

Required when the JOIN command is longer than one line. Terminates the statement.

Example: **Using a Single-Field Dynamic Equijoin**

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

You can join up to 1024 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 join name using the JOIN CLEAR command (see *JOIN CLEAR* on page 259). The adapter 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 the RDBMS limit for an SQL statement. It also cannot involve more than the FOCUS limit for files in a join.

Each cross-referenced table must have at least one data field in common with its host file. Fixed sequential, ISAM, VSAM, IMS, CA-IDMS/DB, CA-Datacom/DB®, ADABAS®, Teradata, MODEL 204®, Oracle, and RDBMS 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 data source by using FOCUS database indexed fields.

If the DB2 Distributed Data Facility is installed:

- You can use the dynamic JOIN command to join tables from two different DB2 subsystems. FOCUS processes the join since DB2 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 attributes in the Access File for the tables referenced in the report request, the adapter 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 your FOCUS documentation on creating reports for additional data source combinations and examples of dynamic joins. In the chart, SQL refers to DB2, Teradata, IDMS SQL, or Oracle tables and views:

From	To	Special Rules That May Apply
SQL	SQL	Joined fields must be of the same data type. 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 data types and lengths are equal.
SQL	FOCUS	Must join to an indexed field (FIELDTYPE=I). Joined fields must have common data type and length.
SQL	IMS	Adapter for IMS must be installed. The DL/I join field must be a key field in the root segment of the data source. It can be a primary or secondary index. Join fields must have a common data type and length.
SQL	VSAM	For a unique join, the VSAM 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 data type and length.
SQL	QSAM	QSAM file must be sorted by its join field. Joined fields must have a common data type and length.
VSAM	SQL	Joined fields must have a common data type and length.
QSAM	SQL	Joined fields must have a common data type and length.

Note:

- A QSAM file is a sequential file. One common example is a HOLD file.

- When joining to a VSAM, QSAM, or IMS data structure, the field in the host data source can be shorter in length than the field in the cross-referenced data source, but performance is adversely affected. This is the only exception to the "same data type, same length" rule when joining non-relational data sources.
- 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.
- Text fields (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 join name specified with the AS phrase.
- You can construct a dynamic join based on more than one field or on conditions other than equality. See *Constructing a Multi-field Dynamic Equijoin* on page 236 for multi-field joins and *Constructing a Conditional Join* on page 240 for conditional joins.
- For information about how heterogeneous joins affect adapter optimization, see *The Adapter Optimizer* on page 171.

Example: Implementing a Single-Field Dynamic Equijoin

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

Note:

- The EMPINFO and COURSE tables are described in *File Descriptions and Tables* on page 459.
- You can reproduce the SQL requests with the STMTRACE trace. See *Tracing Adapter Processing* on page 487 for trace syntax.

The first example prints the full name and courses taken for each employee. Since the employee may have taken more than one course, the example specifies the FOCUS non-unique join.

Employee information is stored in the DB2 table EMPINFO and is represented by the fields LAST_NAME and FIRST_NAME. Course information is stored in the DB2 table COURSE and is represented by the CNAME field.

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT

JOIN EMP_ID IN EMPINFO TAG FILE1 TO ALL WHO IN COURSE TAG FILE2 AS J1
TABLE FILE EMPINFO
PRINT CNAME
BY LAST_NAME BY FIRST_NAME
END

    SELECT T1.EID,T1.LN,T1.FN,T2.COURSE_NAME FROM
   "USER1"."EMPINFO" T1,"USER1"."COURSE" T2 WHERE (T2.EMP_NO =
   T1.EID) ORDER BY T1.LN,T1.FN 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 BY phrases translate to the ORDER BY phrase. The PRINT command translates as SELECT.

The next example illustrates a screening test that lists the employees who have taken either the Advanced or Internals course.

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT

JOIN WHO IN COURSE TAG FILE1 TO ALL EMP_ID IN EMPINFO TAG FILE2 AS J2
TABLE FILE COURSE
PRINT CNAME
BY LAST_NAME BY FIRST_NAME
WHERE CNAME IS 'ADVANCED' OR 'INTERNALS'
END

    SELECT T1.COURSE_NAME,T1.EMP_NO,T2.LN,T2.FN FROM
   "USER1"."COURSE" T1,"USER1"."EMPINFO" T2 WHERE (T2.EID =
   T1.EMP_NO) AND (T1.COURSE_NAME IN('ADVANCED','INTERNALS'))
   ORDER BY T2.LN,T2.FN 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 BY phrases translate to the ORDER BY phrase.

Constructing a Multi-field Dynamic Equijoin

You can construct a dynamic join based on multiple fields from the host file and cross-referenced files. Separate the participating field names with the keyword AND.

Syntax: How to Construct a Multi-Field Dynamic Equijoin

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

where:

fielda1

Is a field in the host file that shares values and format with *fieldb1* in the cross-referenced file.

AND *fielda2*

Is a field in the host file that shares values and format with *fieldb2* in the cross-referenced file.

hostfile

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 *Constructing a Single-field Dynamic Equijoin* on page 231 for possible join combinations.

tag1

Is the tag name for the host file, a one- to eight-character table name 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 table name in the structure

ALL

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 discussed in *The SET ALL Command* on page 247.

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.

fieldb1

Is a field in the cross-referenced file whose values and format can match that of *fielda1*.

AND *fieldb2*

Is a field in the cross-referenced file whose values and format can match that of *fielda2*.

crfile

Is the name of the cross-referenced file.

tag2

Is the tag name for the cross-referenced file, a one- to eight-character table name qualifier for field and alias names in the cross-referenced file. The tag name may not be the same as any table name in the structure. In a recursive join structure, if no tag name is provided, FOCUS prefixes all field names and aliases with the first four characters of the joinname.

joinname

Assigns an internal name to the join structure, up to eight characters in length. It also provides a unique prefix for field names participating in a recursive join.

You can clear the join name when you no longer need this join (see *JOIN CLEAR* on page 259). Do not specify *joinname* as the file name in subsequent TABLE FILE requests. Use *hostfile*.

END

Required when the JOIN command is longer than one line. Terminates the command.

For a multi-field join, the joined fields from each table must be equal in number, data type, 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 1023 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.

Example: Using a Multi-field Dynamic Join

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

```
FILENAME=EMPINFO  ,SUFFIX=SQLDS,$

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=DPT         ,USAGE=A10    ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY ,ALIAS=CSAL        ,USAGE=P9.2   ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE   ,ALIAS=CJC         ,USAGE=A3     ,ACTUAL=A3,$
 FIELD=ED_HRS         ,ALIAS=OJT         ,USAGE=F6.2   ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN     ,ALIAS=BONUS_PLAN  ,USAGE=I4     ,ACTUAL=I4,$
 FIELD=HIRE_DATE_TIME ,ALIAS=HDTT        ,USAGE=HYYMDm ,ACTUAL=HYYMDm,$
 FIELD=HIRE_TIME       ,ALIAS=HT          ,USAGE=HHIS   ,ACTUAL=HHIS,$
```

In the COURSE1 Master File, LAST_NAME and FIRST_NAME have the same field formats, A15 and A10:

```
FILE=COURSE1     ,SUFFIX=SQLDS, $
SEGNAME=SEG01    ,SEGTYPE=S0, $
FIELDNAME   =FOCLIST          ,FOCLIST     ,I5       ,I4       ,$
FIELDNAME   =CNAME            ,COURSE_NAME ,A15      ,A15      ,$
FIELDNAME   =LAST_NAME        ,LN          ,A15      ,A15      ,$
FIELDNAME   =FIRST_NAME       ,FN          ,A10      ,A10      ,$
FIELDNAME   =GRADE            ,GRADE       ,A1       ,A1       ,
      MISSING=ON, $
FIELDNAME   =YR_TAKEN         ,YR_TAKEN    ,A2       ,A2       ,$
FIELDNAME   =QTR               ,QUARTER     ,A1       ,A1        ,$
```

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

```
JOIN LAST_NAME AND FIRST_NAME 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 equijoin, the adapter generates an SQL SELECT statement to satisfy the request:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT

TABLE FILE EMPINFO
PRINT LAST_NAME FIRST_NAME CNAME
END

    SELECT T1.LN,T1.FN,T2.COURSE_NAME FROM "USER1"."EMPINFO" T1,
   "USER1"."COURSE1" T2 WHERE (T2.LN = T1.LN) AND (T2.FN = T1.FN) FOR
    FETCH ONLY;
```

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

Constructing a Conditional Join

The conditional join lets you specify conditions other than equality between fields for relating two tables.

The conditional JOIN command supports FOCUS, relational, VSAM, fixed-format sequential, ADABAS, and IMS data sources. Because each data source differs in its ability to handle complex WHERE criteria, the optimization of the WHERE-based JOIN syntax differs depending on the specific data sources involved in the join and the complexity of the WHERE criteria.

Syntax: How to Construct a Conditional Join

```
JOIN FILE from_file AT from_field [TAG from_tag] [WITH fieldname]
     TO [ALL|ONE]
     FILE to_file AT to_field [TAG to_tag]
     [AS as_name]
     [WHERE expression1 ;
     WHERE expression2  ;
     ...               ;]
END
```

where:

from_file

Is the host Master File.

from_field

Is the field name in the host Master File whose segment will be joined to the cross-referenced data source. It can be any field in the segment. It must reside in the lowest level segment that will be referenced.

from_tag

Is the optional tag name that is used as a unique qualifier for fields and aliases in the host data source.

fieldname

Is a real field name used to assign a segment location for a virtual field. Required when issuing a DEFINE field-based WHERE-based JOIN.

ALL

Describes a one-to-many relationship between the *from_file* and *to_file*.

ONE

Describes a one-to-one relationship between the *from_file* and *to_file*.

Note: If you specify a unique join when the relationship between the host and cross-referenced files is one-to-many, the results will be unpredictable.

to_file

Is the cross-referenced Master File.

to_field

Is the join field name in the cross-referenced Master File. It can be any field in the segment.

to_tag

Is the optional tag name that is used as a unique qualifier for fields and aliases in the cross-referenced data source.

as_name

Is the name associated with the JOIN.

expression1, expression2

Are any expressions valid in a DEFINE FILE command. All of the fields used in all of the expressions the expressions must lie on a single path.

Note: Single line JOIN syntax is not supported. The END command is required.

To issue a DEFINE-based conditional join, the KEEPDEFINES setting must be ON. You then must create all virtual fields before issuing the join. This differs from traditional DEFINE-based joins in which the virtual field is created after the JOIN command is issued. In addition, a virtual field can be part of the JOIN syntax or WHERE criteria. For a discussion of the KEEPDEFINES setting, see your FOCUS documentation.

Example: Using a Conditional Join

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//CLIENT

JOIN   FILE EMPINFO AT EMP_ID TAG E1
TO ALL FILE PAYINFO AT PAYEID TAG P1 AS JW
WHERE EMP_ID EQ PAYEID;
WHERE (DAT_INC - HIRE_DATE) GT 0;
WHERE (DAT_INC - HIRE_DATE) LT 365;
END

TABLE FILE EMPINFO
PRINT CURRENT_SALARY
COMPUTE DIFF/I5 = DAT_INC - HIRE_DATE;
END
```

For DB2, the following SQL is generated:

```
     SELECT T1."EID",T1."HDT",T1."CSAL",T2."EID",T2."DI" FROM
 "USER1"."EMPINFO" T1,"USER1"."PAYINFO" T2 WHERE (T2."EID" =
 T1."EID") AND ((DAYS(T2."DI") - DAYS(T1."HDT")) > 0) AND
 ((DAYS(T2."DI") - DAYS(T1."HDT")) < 365) FOR FETCH ONLY;
```

The SQL request references both tables in the FROM clause and incorporates the conditions specified in the JOIN command as predicates of the WHERE clause. The PRINT command translates as SELECT.

Optimizing Non-Equality WHERE-based Left Outer Joins

A left outer join selects all records from the host table and matches them with records from the cross-referenced table. When no matching records exist, the host record is still retained, and default values (blank or zero) are assigned to the cross-referenced fields. The relational adapters may optimize any WHERE-based left outer join command in which the conditional expressions are supported by the RDBMS.

Syntax: How to Specify a Conditional Left Outer Join

```
JOIN LEFT_OUTER FILE hostfile AT hfld1 [TAG tag1]
     [WITH hfld2]
     TO {UNIQUE|MULTIPLE}
     FILE crfile AT crfld [TAG tag2] [AS joinname]
     [WHERE expression1;
     [WHERE expression2;
     ...]

END
```

where:

`LEFT OUTER`

Specifies a left outer join. If you do not specify the type of join in the JOIN command, the ALL parameter setting determines the type of join to perform.

`hostfile`

Is the host Master File.

`AT`

Links the correct parent segment or host to the correct child or cross-referenced segment. The field values used as the AT parameter are not used to cause the link. They are used as segment references.

`hfld1`

Is the field name in the host Master File whose segment will be joined to the cross-referenced data source. The field name must be at the lowest level segment in the data source that is referenced.

`tag1`

Is the optional tag name that is used as a unique qualifier for fields and aliases in the host data source.

`WITH` *`hfld2`*

Is a data source field with which to associate a DEFINE-based conditional join. For a DEFINE-based conditional join, the KEEPDEFINES setting must be ON, and you must create the virtual fields before issuing the JOIN command.

`MULTIPLE`

Specifies a one-to-many relationship between *hostfile* and *crfile*. Note that ALL is a synonym for MULTIPLE.

`UNIQUE`

Specifies a one-to-one relationship between *hostfile* and *crfile*. Note that ONE is a synonym for UNIQUE.

Note: UNIQUE returns only one instance and, if there is no matching instance in the cross-referenced file, it supplies default values (blank for alphanumeric fields and zero for numeric fields).

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.

If the RDBMS processes a join that the request specifies as unique, and if there are, in fact, multiple corresponding rows in the cross-referenced file, the RDBMS 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.

crfile

Is the cross-referenced Master File.

crfld

Is the join field name in the cross-referenced Master File. It can be any field in the segment.

tag2

Is the optional tag name that is used as a unique qualifier for fields and aliases in the cross-referenced data source.

joinname

Is the name associated with the joined structure.

expression1, expression2

Are any expressions that are acceptable in a DEFINE FILE command. All fields used in the expressions must lie on a single path.

Reference: Conditions for WHERE-based Outer Join Optimization

- In order for a WHERE-based left outer join to be optimized, the expressions must be optimizable for the RDBMS involved and at least one of the following conditions must be true:
  - The JOIN WHERE command contains at least one *field1* EQ *field2* predicate in which *field1* is in *table1* and *field2* is in *table2*.

    or
  - The right table has a key or a unique index that does not contain NULL data.

    or
  - The right table contains at least one "NOT NULL" column that does not have a long data type (such as TEXT or IMAGE).
- The adapter SQLJOIN OUTER setting must be ON (the default).

Controlling Outer Join Optimization

With the SET SQLJOIN OUTER command, you can control when the adapter optimizes outer joins without affecting the optimization of other operations. This parameter provides backward compatibility with prior releases of the adapter and enables you to fine-tune your applications.

When join optimization is in effect, the adapter generates one SQL SELECT statement that includes every table involved in the join. The RDBMS can then process the join. When join optimization is disabled, the adapter generates a separate SQL SELECT statement for each table, and FOCUS processes the join.

You can use the SQLJOIN OUTER setting to disable outer join optimization while leaving other optimization enhancements in effect.

Syntax: How to Control Outer Join Optimization

```
SQL sqlengine SET SQLJOIN OUTER {ON|OFF}
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2, SQLDBC, or SQLORA. Omit if you issued the SET SQLENGINE command.

ON

Enables outer join optimization. ON is the default value for Teradata and Oracle.

OFF

Disables outer join optimization. OFF is the default value for DB2.

Note:

- Left outer join optimization is not supported for IDMS SQL.
- The SQLJOIN OUTER setting is available only when optimization is enabled (that is, OPTIMIZATION is not set to OFF).
- The SQLJOIN OUTER setting is ignored when SET ALL = OFF.
- If SQLJOIN OUTER is set to OFF, the following message displays when you issue the SQL ? query command:

  ```
  (FOC1420) OPTIMIZATION OF ALL=ON AS LEFT JOIN - : OFF
  ```

Reference: Effects of Combinations of Settings on Outer Join Optimization

The following table describes how different combinations of OPTIMIZATION and SQLJOIN OUTER settings affect adapter behavior. It assumes that SET ALL = ON:

Settings		Results	
ON	ON	Yes	Enabled
ON	OFF	No	Enabled
OFF	N/A	No	Disabled
SQL	ON	Yes, in all possible cases	Enabled
SQL	OFF	No	Enabled
FOCUS	ON	Yes if results are equivalent to FOCUS-managed request	Enabled
FOCUS	OFF	No	Enabled

Example: Enabling Left Outer Join Optimization

The following request specifies a left outer join between the EMPINFO and FUNDTRAN tables. The SQLJOIN OUTER setting specifies that the left outer join should be optimized, and the SQLAGGR and STMTRACE trace components are activated:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = SQLAGGR//CLIENT
SET TRACEON = STMTRACE//CLIENT

SET ALL = ON
SQL DB2 SET OPTIMIZATION ON
SQL DB2 SET SQLJOIN OUTER ON
JOIN EMP_ID IN EMPINFO TO ALL WHO IN FUNDTRAN AS J1
TABLE FILE EMPINFO
SUM AVE.CURRENT_SALARY ED_HRS BY WHO BY LAST_NAME
IF DEPARTMENT EQ 'MIS'
END
```

The following trace is generated. One SELECT statement is generated. The LEFT OUTER JOIN phrase in the FROM predicate specifies the left outer join:

```
AGGREGATION DONE ...
 SELECT T2."EID",T1."LN", AVG(T1."CSAL"), SUM(T1."OJT") FROM (
USER1."EMPINFO" T1 LEFT OUTER JOIN "USER1"."FUNDTRAN" T2 ON
T2."EID" = T1."EID") WHERE (T1."DPT" = 'MIS') GROUP BY
T2."EID",T1."LN" ORDER BY T2."EID",T1."LN" FOR FETCH ONLY;
```

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 adapter optimization is enabled or disabled. (See *The Adapter Optimizer* on page 171 for information about the SET OPTIMIZATION command.)

  Note: Even if you set OPTIMIZATION ON, the adapter may disable it. Use the SQLAGGR trace component to determine whether optimization is enabled for a particular request (*Tracing Adapter Processing* on page 487 describes trace facilities).
- 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

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

where:

OFF

Is the default value. 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. (This is known as a left outer join.)

Note:

- The adapter does not support SET ALL = PASS.

- SET ALL = ON with a conditional join disables optimization in DB2, Teradata, and Oracle.

Missing Rows in Unique Descendants

In a unique join, the engine that handles the join controls the output:

- IF FOCUS handles the join, it 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. FOCUS handles the join in the following two cases:
  - OPTIMIZATION is disabled.
  - OPTIMIZATION is enabled with SET ALL = ON and SQLJOIN OUTER OFF.

    In this case the SQLJOIN OUTER OFF setting requires that FOCUS process the left outer join even though other optimization features remain in effect.

  Note: When FOCUS handles join processing, 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 the RDBMS handles the join, OPTIMIZATION must be enabled. The SET ALL command determines the output:
  - 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 SQLJOIN OUTER setting determines whether FOCUS or the RDBMS process the join. When SQLJOIN OUTER is ON, the RDBMS processes the left outer join. It includes all host rows in the report and includes multiple matching cross-referenced rows.
  - 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* on page 251 for a complete discussion.

The following topics discuss each of these settings.

Reference: How FOCUS Processes a Unique Join

FOCUS handles the join when either of the following conditions is true:

- OPTIMIZATION disabled.

- OPTIMIZATION enabled, SET ALL = ON, and SQLJOIN OUTER OFF.

FOCUS 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:

- 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.

The Adapter Optimizer on page 171 discusses factors that determine whether optimization is disabled. To find out if and why the adapter disabled OPTIMIZATION for a report request, use the SQLAGGR trace component (see *Tracing Adapter Processing* on page 487).

Example: FOCUS Unique Join Processing

In the following example, a unique join connects the EMPINFO table, containing employee information, to the FUNDTRAN table, containing direct deposit account information for the employees. (*File Descriptions and Tables* on page 459 provides the Master and Access Files.)

```
JOIN EMP_ID IN EMPINFO TAG FILE1 TO WHO IN FUNDTRAN TAG FILE2 AS J1
SQL DB2 SET OPTIMIZATION OFF
TABLE FILE EMPINFO
PRINT BANK_NAME BANK_ACCT
BY DEPARTMENT BY EMP_ID
END
NUMBER OF RECORDS IN TABLE=       14  LINES=     14
```

Since the join is unique and the SET OPTIMIZATION OFF command disables optimization, 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      EMP_ID       BANK_NAME           BANK_ACCT
----------      ------       ---------           ---------
.               333121200
MIS             112847612
                117593129    STATE                40950036
                219984371
                326179357    ASSOCIATED          122850108
                455670000
                543729165
                818692173    BANK ASSOCIATION    163800144
PRODUCTION      071382660
                119265415
                119329144    BEST BANK              160633
                123764317    ASSOCIATED          819000702
                126724188
                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. Note that the bank name and bank account values, although integers, display as blanks instead of zeros because of the *S* display option in their USAGE formats.

Reference: **How the RDBMS Processes a Unique Join**

The RDBMS processes the join under the following conditions:

- OPTIMIZATION enabled with SET ALL = OFF.

  Since the RDBMS does not recognize the concept of a unique join, it performs an inner join just as it does if the join is non-unique. See *Missing Rows in Non-unique Descendants* on page 251 for a complete discussion.

- OPTIMIZATION enabled with SET ALL = ON and SQLJOIN OUTER ON.

  If SQLJOIN OUTER is ON, the RDBMS performs a left outer join just as it does if the join is non-unique. See *Missing Rows in Non-unique Descendants* on page 251 for a complete discussion.

  (If SQLJOIN OUTER is OFF, FOCUS processes the join. It substitutes default values for short path columns and displays only one matching record as described in *How FOCUS Processes a Unique Join* on page 248.)

The Adapter Optimizer on page 171 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 (although if optimization is enabled, you can control which engine processes the outer join with the SET SQLJOIN OUTER command described in *How to Control Outer Join Optimization* on page 245). Short paths display in the report with missing columns represented by the NODATA symbol (.). Multiple matching rows are included.

Reference: 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).

Example: Non-Unique Join Processing With SET ALL = OFF

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 12 long-time employees in the host table. John Royce (333121200, no department) and Donna Lee (455670000, MIS department) are not listed in the report, because their corresponding cross-referenced rows do not exist in the DEDUCT table:

```
DEPARTMENT  EMP_ID     DEDDATE   DED_CODE          DED_AMT
----------  ------     -------   --------          -------
MIS         112847612  82/01/29  CITY                $1.43
                                 FED               $121.55
                                 FICA              $100.10
                                 HLTH               $22.75
                                 LIFE               $13.65
                                 SAVE               $54.60
                                 STAT               $20.02
                       82/02/26  CITY                $1.43
                                 FED               $121.55
                                 FICA              $100.10
                                 HLTH               $22.75
                                 LIFE               $13.65
                                 SAVE               $54.60
                                 STAT               $20.02
                          .
                          .
                          .
```

Reference: 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 and SQLJOIN OUTER is ON, the adapter passes a left outer join to the RDBMS.
- If optimization is disabled or optimization is enabled but SQLJOIN OUTER is OFF, 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.

In both cases the report results are the same.

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

Example: Non-Unique Join Processing With SET ALL = ON

The following example executes the same report request as in *Non-Unique Join Processing With SET ALL = OFF*, except that SET ALL is ON:

```
SET ALL = ON
TABLE FILE EMPINFO
PRINT DED_CODE DED_AMT
BY DEPARTMENT BY EMP_ID BY DEDDATE
END
  NUMBER OF RECORDS IN TABLE=      450  LINES=    450
```

John Royce (333121200) is now on Page 1 and Donna Lee (455670000) on Page 8. The department column for John Royce (333121200) displays the NODATA value, since the field has MISSING=ON:

```
PAGE     1

   DEPARTMENT    EMP_ID      DEDDATE      DED_CODE     DED_AMT
   ----------    ------      -------      --------     -------
   .             333121200   .            .                .
   MIS           112847612   82/01/29     CITY           $1.43
                                          FED          $121.55
                                          FICA         $100.10
                                          HLTH          $22.75
                                          LIFE          $13.65
                             .
                             .
                             .
```

Donna Lee (455670000) works for MIS:

```
PAGE     8

   DEPARTMENT    EMP_ID      DEDDATE     DED_CODE     DED_AMT
   ----------    ------      -------     --------     -------
   MIS           326179357   82/07/30    STAT          $88.93
                             82/08/31    CITY           $6.35
                                         FED          $539.96
                                         FICA         $444.67
                                         HLTH          $45.37
                                         LIFE          $27.22
                                         SAVE         $108.90
                                         STAT          $88.93
                 455670000   .           .                .
                 543729165   82/04/30    CITY           $.72
                             .
                             .
                             .
```

Note: 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.

Reference: SET ALL=ON With Screening Conditions

When SET ALL=ON, screening conditions affect report results for a non-unique join. 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.

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

```
JOIN EMP_ID IN EMPINFO TAG F1 TO ALL DEDEID IN DEDUCT TAG F2 AS JOIN1
SET ALL=ON
TABLE FILE EMPINFO
SUM DED_AMT
BY DEPARTMENT BY LAST_NAME BY FIRST_NAME BY DED_CODE
WHERE DED_CODE EQ 'HLTH'
END
```

Royce and Lee are omitted as the result of the WHERE test, because they have no deduction code records:

```
DEPARTMENT     LAST_NAME    FIRST_NAME     DED_CODE    DED_AMT
----------     ---------    ----------     --------    -------
MIS            BLACKWOOD    ROSEMARIE      HLTH        $226.87
               CROSS        BARBARA        HLTH        $330.21
               JONES        DIANE          HLTH        $121.16
               MCCOY        JOHN           HLTH         $16.50
               SMITH        MARY           HLTH        $181.99
PRODUCTION     BANNING      JOHN           HLTH         $41.25
               IRVING       JOAN           HLTH        $427.35
               MCKNIGHT     ROGER          HLTH        $122.43
               ROMANS       ANTHONY        HLTH        $105.60
               STEVENS       ALFRED         HLTH         $69.44
```

Reference: 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.

For example, 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 *Non-Unique Join Processing With SET ALL = ON*).

```
SET ALL=OFF
TABLE FILE EMPINFO
PRINT DED_CODE DED_AMT
BY ALL.DEPARTMENT BY EMP_ID BY DEDDATE
END
```

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

Summary Chart

This summary chart lists possible report results for dynamic joins and multi-table Master Files (embedded joins):

Join Type	**RDBMS Behavior (Optimization Enabled)**		**FOCUS Behavior (Optimization Disabled)**	
NON-UNIQUE Dynamic `(JOIN...TO ALL)`or Embedded `(SEGTYPE=S0 or KL)`	**Outer Join**	**Inner Join**	**Outer join** **(Also applies if Optimization Enabled and SQLJOIN OUTER OFF)**	**Inner join**
Short paths	Appear with NODATA (.) value for fields of all missing segments	Do not appear	Appear with NODATA (.) value for fields of all missing segments	Do not appear
More than one matching row	All matching rows appear	All matching rows appear	All matching rows appear	All matching rows appear
UNIQUE Dynamic `(JOIN...TO)` or Embedded `(SEGTYPE=U or KLU)`	**Outer join**	**Inner Join**	**FOCUS Unique** **(Also applies if Optimization Enabled and SQLJOIN OUTER OFF)**	**FOCUS Unique**

Join Type	RDBMS Behavior (Optimization Enabled)		FOCUS Behavior (Optimization Disabled)	
Short paths	Appear with NODATA (.) value for fields of all missing segments	Do not appear	Appear with blank or zero for missing values	Appear with blank or zero for missing values
More than one matching row	All matching rows appear	All matching rows appear	Only first matching row appears	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 or SET ALL = ON with SQLJOIN OUTER 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.

JOIN Utilities

You can check and control the status of join structures with three commands, CHECK FILE, ? JOIN query, and JOIN CLEAR.

CHECK FILE

The CHECK FILE command produces a diagram of host and cross-referenced relationships and retrieval paths. CHECK FILE also reloads the Master File and checks the Master File syntax.

To display the structure, at the FOCUS command level type

```
CHECK FILE name PICT[URE] [RETRIEVE] [HOLD] [JOIN]
```

where:

name

Is the name of the host structure (for a dynamic join) or of a multi-table Master File.

`PICT[URE]`

Displays a diagram of the structure.

`RETRIEVE`

Is optional. Modifies the diagram to show retrieval paths, especially how a unique table is treated as an extension of its host. See your FOCUS documentation for more information.

`HOLD`

Creates a temporary HOLD file containing detailed information about the fields and segments in the structure.

`JOIN`

Loads and parses the Access File, checking for syntax errors. It does not check for the existence of the tables or databases named. It checks for the existence of the fields used as the KEYFLD and IXFLD pairs that implement an equijoin, but not their data types.

Used in conjunction with the HOLD option, JOIN adds a field named INDX to the HOLD file. The value 1 in this field indicates that you can join to the field. This option is useful when joining to a data source that requires the target of a join to be a key or index field.

On the diagram, labels for non-unique (KM) or unique (KU) next to the cross-referenced structure indicate whether the dynamic JOIN is non-unique or unique. Only the first four fieldnames of each structure display.

The letter K next to a field indicates that it represents the foreign key column or the first column of a composite foreign key. If the cross-referenced structure has descendants, the descendants are labeled as either KL (keyed through linkage) or KLU (keyed through linkage unique), depending on how their relationship was declared to FOCUS.

Example: **Displaying a Structure With CHECK FILE**

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

CHECK FILE EMPINFO PICT

  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   2  ( REAL=    1  VIRTUAL=   1 )
  NUMBER OF FIELDS=    13  INDEXES=   0  FILES=     2
  TOTAL LENGTH OF ALL FIELDS=   95

 SECTION 01
 STRUCTURE OF GNTINT   FILE EMPINFO ON 06/19/90 AT 12.17.40

          EMPINFO
  01      S0
 **************
 *EMP_ID      **
 *LAST_NAME   **
 *FIRST_NAME  **
 *HIRE_DATE   **
 *            **
 ***************
  **************
        I
        I
        I
        I COURSE
  02    I KM
 ..............
 :CNAME       ::
 :WHO         ::K
 :GRADE       ::
 :YR_TAKEN    ::
 :            ::
 :............::
  ............:
  JOINED  COURSE
```

Note: "JOINED COURSE" indicates that this relationship was created by a dynamic JOIN command rather than a multi-segment Master File.

? JOIN

For all dynamic joins that are in effect, the ? JOIN query command identifies the fields representing the primary/foreign key field pair, the host structure, the cross-referenced structure, any specified join name, the type of join (non-unique or unique, as indicated by the presence or absence of the ALL keyword), and whether the join is conditional.

To list all active dynamic join structures, enter:

```
? JOIN
```

For example:

```
? join
 JOINS CURRENTLY ACTIVE

HOST                          CROSSREFERENCE
FIELD        FILE    TAG      FIELD        FILE     TAG    AS   ALL WH
-----        ----    ---      -----        ----     ---    --   --- --
PAYEID       PAYINFO  P1       EMP_ID       EMPINFO  E1           Y   Y
```

Note: The ? JOIN query command displays only the first join pair for multi-field joins.

JOIN CLEAR

A maximum of 1023 dynamic joins can be active in any FOCUS session. The JOIN CLEAR command can release either all or specific existing joins. The syntax is

```
JOIN CLEAR {*|joinname}
```

where:

`*`

Clears all existing joins.

joinname

Is a specific join to be cleared.

The effect of the JOIN CLEAR command depends on whether conditional joins exist:

- If no conditional joins exist, the JOIN CLEAR command clears all virtual fields defined for the host data source and the joined structure.
- If conditional joins exist but were issued prior to the join you want to clear, the JOIN CLEAR command clears only the specified join. Any virtual fields saved in the context of a join that is cleared will also be cleared.
- If conditional joins exist and were issued subsequent to the join you want to clear, or if the join you want to clear is a conditional join, the JOIN CLEAR command clears the specified join and all subsequent joins issued for the same host file.

The JOIN CLEAR * command clears every join that was issued along with its associated virtual fields. However, virtual fields defined in the null context (prior to any joins) remain in effect.

Implementing Search Limits

The RECORDLIMIT phrase sets a maximum for the number of rows the adapter fetches from the answer set returned by the RDBMS. It does not restrict the RDBMS in its construction of the answer set. However, if the RDBMS does not have to sort the answer set, using this feature may result in a significant reduction in response time.

The READLIMIT phrase is synonymous with RECORDLIMIT except in the following cases:

- The Adapter for Oracle passes READLIMIT screening conditions to the Oracle RDBMS as SQL WHERE ROWNUM <=*n* clauses. ROWNUM is a special pseudo column that allows Oracle to limit the number of rows retrieved and returned.
- For DB2, READLIMIT appends a FETCH FIRST *n* ROWS ONLY clause to the generated SQL, restricting the size of the answer set constructed by the RDBMS. For *n*=1, the SQL request contains the clause FETCH FIRST 1 ROW ONLY. The reduction in answer set size reduces response time and enhances performance. This can improve DB2 performance if you know the size of the desired answer set in advance. For more information, consult the IBM *DB2 SQL Reference* Manual.

The READLIMIT and RECORDLIMIT phrases 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 Master File. Reporting requires only a few rows to indicate if the description is accurate.
- Using the trace facility in problem resolution.
- Testing the format of a new report.

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

Syntax: How to Implement Search Limits

```
{WHERE|IF} {RECORDLIMIT|READLIMIT} {EQ|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 Master File to limit the fetching of rows for all users. If the Master File and the report request both contain such a test, the adapter uses the lower of the two values.

Note:

- For IDMS/SQL and Teradata, READLIMIT and RECORDLIMIT are synonymous and do not pass any special SQL syntax to the RDBMS.
- 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.

Example: Reducing DB2 Answer Set Size

The following request turns on the STMTRACE trace component and issues a retrieval request with a READLIMIT phrase. The SQL request contains the FETCH FIRST 2 ROWS clause:

```
SET TRACEUSER=ON
SET TRACEOFF=ALL
SET TRACEON=STMTRACE//CLIENT
TABLE FILE EMPINFO
PRINT LAST_NAME FIRST_NAME CURRENT_SALARY
BY EMP_ID
IF READLIMIT EQ 2
END
```

The generated SQL request contains the FETCH FIRST 2 ROWS clause:

```
 SELECT T1."EID",T1."LN",T1."FN",T1."CSAL" FROM
USER1."EMPINFO" T1 ORDER BY T1."EID" FETCH FIRST 2 ROWS ONLY FOR FETCH ONLY;
```

If the test specifies READLIMIT EQ 1, the SQL request contains the clause FETCH FIRST 1 ROW ONLY:

```
 SELECT T1."EID",T1."LN",T1."FN",T1."CSAL" FROM
 USER1.EMPINFO" T1 ORDER BY T1."EID" FETCH FIRST 1 ROW ONLY FOR FETCH ONLY;
```

Example: Using Oracle READLIMIT Optimization

Assume the EMPLOYEE Master File describes the Oracle table USER1.PAYROLL and consider the following request:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID
WHERE READLIMIT EQ 5
END
```

The Adapter for Oracle generates the following SQL for this request:

```
SELECT T1.EMP_ID FROM "USER1"."PAYROLL" T1 WHERE ROWNUM <=5;
```

Because the screening condition is passed to the RDBMS, only the first five rows retrieved by Oracle are returned. This results in less communication between the RDBMS and FOCUS and less processing by FOCUS prior to displaying the report output.

Array Blocking for SELECT Requests

The Adapters for Oracle and DB2 support array retrieval from result sets produced by executing SELECT queries or stored procedures. This technique substantially reduces network traffic and CPU utilization.

High FETCHSIZE values increase the efficiency of requests involving many rows, at the cost of higher virtual storage requirements. The default value is 20 for Oracle and 100 for DB2. A value higher than 100 is not recommended because the increased efficiency it would provide is generally negligible.

For more information, see *Adapter Commands* on page 309.

Multiple Retrieval Paths

This section discusses retrieval performance for multi-table Master Files and dynamic joins of three or more tables. It also explains the role of unique cross-referenced tables.

Master Files and JOIN commands define data retrieval paths. The adapter 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 TABLE request that references a dynamically joined structure generates SQL join predicates for all segments in the subtree that starts from the root segment. Multi-table Master Files do not necessarily generate these predicates. In a multi-table structure, the subtree effectively begins with the highest referenced segment. This difference may cause identical TABLE requests to produce different reports when run against a dynamic join structure and a multi-table Master File 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 *CHECK FILE* on page 256).

Multiple Retrieval Paths With Sort Phrases and Screening Tests

Fields specified in 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.

Chapter 9

Direct SQL Passthru

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

In this chapter:

Direct SQL Passthru Advantages

Issuing requests through the Direct SQL Passthru facility eliminates the need for Master 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, a storage area is created 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 adapter 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 adapter 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 SQL Translator.

  Some applications use the SQL Translator to access the RDBMS using an adapter. The Translator translates SQL to FOCUS, and then the adapter 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 example, IMS, a FOCUS data source, or the DB2 RDBMS). For more information about the SQL Translator and access to FOCUS data sources, see your FOCUS documentation.

  Note: If your site installed the adapter with the IM parameter set to 0, certain SQL commands will be disabled. Issuing these commands will produce error message FOC1638. You can issue SQL SELECT commands regardless of the IM setting; they are never disabled. Refer to your adapter installation guide for more information.

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 adapter SET SQLENGINE command to establish the target RDBMS for the duration of the FOCUS session or until you issue another SET SQLENGINE command. The adapter automatically passes subsequent SQL commands to the specified RDBMS.
- Specify the target RDBMS in your request. (See *Issuing Commands and Requests* on page 267 for information about issuing commands and requests.)

Invoking Automatic Passthru

If you do not establish a target RDBMS, the SQL Translator processes the request. However, the Translator will invoke Automatic Passthru (APT) when the SQL submitted contains valid syntax for the RDBMS being accessed.

As a result, the SQL code will not be processed by FOCUS, but instead will be sent directly to the RDBMS. Column names displayed in the report will be the RDBMS table column names, which correspond with the Master File ALIAS values. If this behavior is not desirable, you can disable Automatic Passthru for the request by issuing the SET APT=OFF; command in the SQL query:

```
SQL
SET APT = OFF;
 sql statement
END
```

Syntax: How to Issue the SET SQLENGINE Command

```
SET SQLENGINE = sqlengine
```

where:

sqlengine

Indicates the target RDBMS. Valid values are

DB2

Indicates the DB2 RDBMS.

SQLDBC

Indicates the Teradata RDBMS.

SQLIDMS

Indicates the CA-IDMS/SQL RDBMS.

SQLORA

Indicates the Oracle RDBMS.

OFF

Indicates that unless the request specifies a target RDBMS, the SQL Translator will process the request. OFF is the default setting.

You can change the 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.

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.

Syntax: How to Issue a Direct SQL Passthru Request

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

```
{ENGINE|SQL} [sqlengine]
command [;]
[TABLE FILE SQLOUT]
[options]
END
```

where:

sqlengine

Indicates the target RDBMS. Acceptable values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

command

Is one SQL command, or one or more adapter SET commands.

;

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

TABLE FILE SQLOUT

For SQL SELECT requests only, allows you to specify additional FOCUS report options or subcommands. To create a Master File you can use throughout the FOCUS session, see *Creating a FOCUS View With Direct SQL Passthru* on page 281.

options

For SQL SELECT requests only, are report formatting options or operations.

END

Terminates the request. Is optional for adapter SET commands, the SQL commands COMMIT WORK and ROLLBACK WORK, the DB2 CONNECT command, and the adapter parameterized Passthru commands BEGIN SESSION, END SESSION, and PURGE (see *Parameterized SQL Command Summary* on page 284). Required for all other commands.

Note:

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

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

```
{ENGINE|SQL} [sqlengine] SET PASSRECS {OFF|ON}
```

where:

sqlengine

Indicates the target RDBMS. Acceptable values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

OFF

Is the default value. The adapter 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 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 adapter updates the &RECORDS system variable with the number of rows affected. You can access this variable using Dialogue Manager, and display it with the ? STAT query.

Note:

- You must use Direct SQL Passthru syntax to issue UPDATE or DELETE commands in order to invoke the SET PASSRECS command.
- The FOC1364 message is for informational purposes only and does not affect the &FOCERRNUM setting.

The &ROWSAFFECTED variable is populated with the number of rows affected by a Direct SQL Passthru INSERT, UPDATE, or DELETE command. It is populated regardless of the PASSRECS setting. &ROWSAFFECTED is initialized to -1 and is set to -1 by any Direct SQL Passthru command that does not return the number of rows affected, such as a SELECT statement. The value of &ROWSAFFECTED is overwritten each time a Direct SQL Passthru INSERT, UPDATE, or DELETE command is executed, so if you want to retain it, you must copy it to another variable or store it in a calculated field.

Issuing Adapter Environmental Commands

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

Example: Issuing Adapter Environmental Commands

The following example specifies the DB2 RDBMS as the target RDBMS, issues four adapter SET commands, and issues the SQL ? query command to display the updated parameters. Notice that the request does not include the environmental qualifiers TSO or MVS.

```
SET SQLENGINE=DB2
SQL SET AUTOCLOSE ON FIN
SQL SET SSID DSN
SQL SET PLAN P7029910
SQL SET DBSPACE PUBLIC.SPACE0
SQL ?
```

The output is

```
(FOC1440) CURRENT SQL INTERFACE SETTINGS ARE :
(FOC1442) CALL ATTACH FACILITY IS             -  : ON
(FOC1447) SSID FOR CALL ATTACH IS             -  : DSN
(FOC1448) ACTIVE PLAN FOR CALL ATTACH IS      -  : USERCAF
(FOC1459) USER SET PLAN FOR CALL ATTACH IS    -  : P7029910
(FOC1460) INSTALLATION DEFAULT PLAN     IS    -  : M727703B
(FOC1503) SQL STATIC OPTION IS                -  : OFF
(FOC1444) AUTOCLOSE OPTION IS                 -  : ON FIN
(FOC1496) AUTODISCONNECT OPTION IS            -  : ON FIN
(FOC1499) AUTOCOMMIT OPTION IS                -  : ON COMMAND
(FOC1449) CURRENT SQLID IS                    -  : SYSTEM DEFAU
(FOC1424) ISOLATION LEVEL FOR DB2 TABLE INTERFACE IS  :
(FOC1491) FETCH BUFFERING FACTOR              -  : 100
(FOC1441) WRITE FUNCTIONALITY IS              -  : ON
(FOC1445) OPTIMIZATION OPTION IS              -  : OFF
(FOC1763) IF-THEN-ELSE OPTIMIZATION IS        - : ON
(FOC1484) SQL ERROR MESSAGE TYPE IS           -  : DBMS
(FOC1497) SQL EXPLAIN OPTION IS               -  : OFF
(FOC1552) INTERFACE DEFAULT DATE TYPE         -  : NEW
(FOC1446) DEFAULT DBSPACE IS                  -  : PUBLIC.SPACE
```

See *Adapter Commands* on page 309, for information about adapter SET commands.

Issuing Native SQL Commands (Non-SELECT)

When the adapter 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.

Example: Issuing Native SQL Commands (Non-SELECT)

This example specifies the DB2 RDBMS as the target RDBMS, creates the DPBRANCH table, and inserts rows.

```
SET SQLENGINE=DB2
SQL CREATE TABLE DPBRANCH
    (BRANCH_NUMBER    INTEGER      NOT NULL,
     BRANCH_NAME      CHAR(5)      NOT NULL,
     BRANCH_MANAGER   CHAR(5)      NOT NULL,
     BRANCH_CITY      CHAR(5)      NOT NULL)
 IN PUBLIC.SPACE0 ;
END
SQL INSERT INTO DPBRANCH VALUES (1,'WEST','PIAF','NY') ;
END
SQL INSERT INTO DPBRANCH VALUES (2,'EAST','SMITH','NY') ;
END
```

Consult *File Descriptions and Tables* on page 459 for sample tables.

Issuing SQL SELECT Commands

When you issue an SQL SELECT request using the Direct SQL Passthru facility, the adapter 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 Master File named SQLOUT (discussed in *The SQLOUT Master File* on page 273. The SQLOUT Master File functions as a template for reading and formatting the request results.

The SQLOUT Master File resides in memory. You cannot edit or print it. It exists only for the duration of the request. Subsequent requests cannot reference it. To create an internal Master File that exists for the entire FOCUS session, see *Creating a FOCUS View With Direct SQL Passthru* on page 281.

After FOCUS prepares the storage area and the SQLOUT Master File, 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 Master File is discarded.

The RDBMS reads data once for SELECT requests using Direct SQL Passthru. FOCUS does not hold or re-read data locally, except for some types of TABLE subcommands (for example, 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.

Example: Issuing SQL SELECT Commands

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 products from each vendor for all branches located in New York. The subquery retrieves New York branch numbers. The request does not specify additional report formatting:

```
ENGINE DB2
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
```

The output is:

```
VENDOR_NUMBER  PRODUCT          E03
-------------  -------          ---
            1  RADIO             10
            3  MICRO              9
```

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

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

Internally, the process produces an SQLOUT Master File (described in *The SQLOUT Master File* on page 273) that you can use for FOCUS report formatting commands.

For an example of a SELECT request that includes FOCUS report formatting commands, see *The SQLOUT Master File* on page 273.

The SQLOUT Master File

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

Note: The adapter does not generate an associated Access File, since the SQL statement is stored in memory.

The SQLOUT Master File describes the answer set. Each field represents one data element in the outermost SELECT list, reflecting the flat row that the RDBMS returns.

Example: Sample SQLOUT Master File

The following is the SQLOUT Master File created for the example in *Issuing SQL SELECT Commands* on page 271:

```
FILENAME=SQLOUT, SUFFIX=SQLDS, $
  SEGNAME=SQLOUT, SEGTYPE=S0, $
    FIELD='VENDOR_NUMBER' , E01, USAGE=I11 ,ACTUAL=I4 ,MISSING=OFF, $
    FIELD='PRODUCT'       , E02, USAGE=A5  ,ACTUAL=A5 ,MISSING=OFF, $
    FIELD='__SUM(NUMBER_OF_UNITS)',
                            E03, USAGE=I11 ,ACTUAL=I4 ,MISSING=OFF,  $
```

The FILENAME and the SEGNAME values are SQLOUT. The SUFFIX is SQLDS, SQLDBC, SQLIDMS, or SQLORA depending on the target engine for the request. The SEGTYPE is S0. These values are constant.

The Adapters for DB2 and IDMS/SQL use the SQL DESCRIBE function to obtain column information:

- The FIELDNAME value is the column name. For Teradata, the field name is the same default value assigned to the alias name. For expressions or functions, the FIELDNAME value depends on the RDBMS. For example, the DB2 RDBMS does not return a column name, so the field name 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 adapter calculates USAGE and ACTUAL formats based on the column data type and length. The following charts outline these calculations.

Reference: **SQLOUT Formats for DB2 and IDMS/SQL**

		ACTUAL Formats MISSING=	
DATE	YYMD	DATE	same
TIME	HHIS	HHIS	same
TIMESTAMP	HYYMDm	HYYMDm	same
INTEGER	I11	I4	same
SMALLINT	I6	I2	I4
DECIMAL	Pp.s	P((p+1)/2)	P((p+1)/2) if p>15, P8 if p<=15
Where p is precision and s is scale. Precision in the USAGE format includes the decimal point and sign; precision in the ACTUAL format excludes them.			
FLOAT (4 byte)	F9.2	F4	same
FLOAT (8 byte)	D12.2	D8	same
VARCHAR(n)	An	An	same
Where n <= 254 characters.			
VARCHAR(n)	TX50	TX	same
Where 254 < n <= 4094 characters. **Note:** VARCHAR strings > 4094 characters are not supported.			
CHAR(n)	An	An	same
GRAPHIC(n)	A(2n+2)	Kn	same

		ACTUAL Formats MISSING=	
Where n <= 127 characters.			
VARGRAPHIC(n)	A(2n+2)	Kn	same
Where n <= 127 characters **Note:** VARGRAPHIC(n) where n >127 characters is not supported.			

Note: Results of expressions are also one of these data types.

Reference: SQLOUT Formats for Teradata

		ACTUAL Formats MISSING=	
DATE	YYMD	DATE	same
INTEGER	I11	I4	same
SMALLINT	I6	I2	I4
DECIMAL	Pp.s	P((p+1)/2)	P((p+1)/2) if p>15, P8 if p<=15
Where p is precision and s is scale. Precision in the USAGE format includes the decimal point and sign; precision in the ACTUAL format excludes them.			
FLOAT (8 byte)	D12.2	D8	same
VARCHAR(n)	An	An	same
CHAR(n)	An	An	same
BYTE	An	An	same
BYTEINT	I6	I2	I4
VARBYTE	An	An	same

		ACTUAL Formats MISSING=	
GRAPHIC(n)	A(2n+2)	Kn	same
Where n <= 127 characters.			
VARGRAPHIC(n)	A(2n+2)	Kn	same
Where n <= 127 characters **Note:** LONG VARGRAPHIC (VARGRAPHIC(n) where n >127 characters) is not supported.			

Note: Results of expressions are also one of these data types.

Reference: SQLOUT Formats for Oracle

		ACTUAL Formats MISSING=	
DATE	HYYMDS	HYYMDS	same
NUMBER(p,s)	I11 Pp.s D12.2	I4 P((p+1)/2) D8	same, if p=38,s=0 P8 if p<=15 same, if p>15
Where p is precision and s is scale. Precision in the USAGE format includes the decimal point and sign; precision in the ACTUAL format excludes them.			
VARCHAR(n)	An	An	same
Where n £ 254 characters.			
LONG(n)	TX50	TX	same
Where 254 < n <= 4094 characters. **Note:** VARCHAR strings > 4094 characters are not supported.			
CHAR(n)	An	An	same
RAW(n)	A2n	A2n	same

ACTUAL Formats MISSING=
Where n <= 128 characters **Note:** Oracle converts RAW data to and from CHAR data. The data is represented as 1 hexadecimal character, which is equivalent to 2 alphanumeric characters.

Note: Results of expressions are always floating point.

Syntax:

How to Alter Length and Scale of Numeric Columns Returned

You can use the SET CONVERSION command to alter the length and scale of numeric columns displayed from a SELECT request. That is, you can control the USAGE attribute in the dynamically created Master File.

```
{ENGINE|SQL} [sqlengine] SET CONVERSION {RESET|dtype} [RESET|PRECISION
{value|MAX}]
```

where:

sqlengine

Indicates the target RDBMS. Acceptable values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

RESET

Returns precision and scale values that you previously altered back to the data adapter defaults. If you specify RESET immediately following the SET CONVERSION command, all data types return to the defaults. If you specify RESET following a particular data type, only columns of that data type are reset.

dtype

Applies the command only to columns of a specific data type. Valid datatypes are:

INTEGER

INTEGER (and, for Oracle, SMALLINT).

DECIMAL

DECIMAL.

REAL

Single precision floating point. Not supported for Oracle or Teradata.

FLOAT

Double precision floating point.

value

Is the precision in the following form:

nn [*mm*]

where:

nn

Must be greater than 1 and less than the maximum allowable value for the data type. (See description of MAX.)

mm

Is the scale. Valid with DECIMAL, REAL, and FLOAT data types. 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 data type:

DATA TYPE	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.

Example: **Altering the Length and Scale of Numeric Columns Returned**

To set the precision for all INTEGER fields to 7:

```
SQL DB2 SET CONVERSION INTEGER PRECISION 7
```

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

```
ENGINE DB2 SET CONVERSION FLOAT PRECISION 14 3
```

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

```
SQL DB2 SET CONVERSION INTEGER RESET
```

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

```
ENGINE DB2 SET CONVERSION RESET
```

Syntax: **How to Control the Precision of the Oracle NUMBER Data Type**

You can use the ORANUMBER setting to override the default mapping of a NUMBER data type with precision between 32 and 38.

```
SQL [SQLORA] SET ORANUMBER {COMPAT|DECIMAL}
```

where:

SQLORA

Indicates the Oracle Data Adapter. You can omit this value if you previously issued the SET SQLENGINE command.

COMPAT

Indicates that the NUMBER data type with precision between 32 and 37 will be mapped to format D20.2.This is the default.

DECIMAL

Indicates that the NUMBER data type with precision between 32 and 37 will be mapped to format P33.2.

Example: Customizing Output of a Direct SQL Passthru Report Request

The following example illustrates how to customize the default report output from the example in *Issuing SQL SELECT Commands* on page 271. The example adds the TABLE FILE SQLOUT command 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:

```
SQL DB2
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;                    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
```

The output is

```
 NUMBER OF RECORDS IN TABLE=        2  LINES=      2
PAGE     1
 Number of Units of Each Vendor's Products
          in New York Branches
 Vendor  Product      Total
 Number  Name         Units
 ------  -------      -----
      1  RADIO           10
      3  MICRO            9
```

Reference: Usage Notes for Customizing a Direct SQL Passthru Report

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

- You must specify field names or aliases from the SQLOUT Master File.
- You can include any FOCUS TABLE formatting options or subcommands that you can code using the SQLOUT Master File.
- Most reporting operations are available. For example, you can use prefix 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 DB2.

- DEFINE fields are not supported. They are supported for FOCUS views created with Direct SQL Passthru. See *Creating a FOCUS View With Direct SQL Passthru* on page 281.
- The ?F query command is not available for the SQLOUT Master File. It is available for FOCUS views created with Direct SQL Passthru.

Creating a FOCUS View With Direct SQL Passthru

You can create a named, internal Master File (FOCUS view) for a particular SELECT statement with the adapter SQL PREPARE command. Unlike the SQLOUT Master File, 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.

Syntax: How to Create a FOCUS View With Direct SQL Passthru

```
{ENGINE|SQL} [sqlengine] PREPARE view_name FOR
SELECT....[;]
END
```

where:

sqlengine

Indicates the target RDBMS. Acceptable values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

view_name

Names the Master File (FOCUS view). The name can be eight characters long and must conform to FOCUS naming conventions for Master Files (see *Describing Tables to FOCUS* on page 55).

`SELECT...`

Is any SELECT statement.

Reference: Usage Notes for Creating a FOCUS View

- You cannot include FOCUS formatting options or subcommands. Issue a TABLE request against the view name to create a 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 attributes and default aliases in the generated Master File are the same as those for the SQLOUT Master File (see *The SQLOUT Master File* on page 273). Only its file or member name 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 Master File. 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 table names in SELECT statement FROM clauses.

Master Files created with PREPARE reside in memory, so you cannot edit or print them. They function like any other FOCUS Master File. For example, you can specify them with TableTalk. You can assign DEFINE fields to them or use them in MATCH FILE commands. You can also issue the ?F query to list the field names of the FOCUS view.

Example: Creating a FOCUS View

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

```
SET SQLENGINE=DB2
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 virtual HI_STOCK field to the TOTPROD Master File and specify the virtual field in a report request:

```
DEFINE FILE TOTPROD
  HI_STOCK/A2 = IF (E03 GT 9) 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
```

The output is

```
 NUMBER OF RECORDS IN TABLE=        2  LINES=      2
PAGE     1
Number of Units of Each Vendor's Products
         in New York Branches
Vendor  Product      Total
Number  Name         Units
------  -------      -----
     1  RADIO           10  **
     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.

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:

```
INSERT INTO INVENTORY (PARTNO) VALUES(?)
```

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.

Note: You must use the proper form of parameter marker for the RDBMS you are accessing. DB2, IDMS/SQL, and Teradata use the question mark (?) as the parameter marker. Oracle uses :00n, where n is incremented by 1 for each parameter. For example:

```
SQL SQLORA PREPARE I FOR INSERT INTO EMP (EMP_ID,COURSE)
VALUES(:001,:002)
END
```

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.
- Direct SQL Passthru and Parameterized SQL Passthru are not available within MODIFY.

Parameterized SQL Command Summary

With parameterized SQL you can compile, bind, and repeatedly execute a series of SQL commands. To avoid invoking END processing between the SQL statements in the series, you place the whole sequence of SQL requests within SQL BEGIN SESSION and SQL END SESSION 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 within a BEGIN SESSION/END SESSION pair. You can also include other FOCUS, SQL, and adapter environmental commands within the BEGIN SESSION/END SESSION pair.

Subsequent sections explain the individual commands involved in Parameterized Passthru design. *Parameterized SQL Passthru Sample Session* on page 293 contains a sample session.

Syntax: How to Issue Parameterized Passthru Commands

```
{ENGINE|SQL} [sqlengine] command [;]
[TABLE FILE statement_name]
[options]
END
```

where:

sqlengine

Indicates the target RDBMS. Acceptable values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

command

Is one of the following: BEGIN SESSSION, END SESSION, PREPARE, BIND, or EXECUTE.

;

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. This 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 adapter SET commands, the SQL commands COMMIT WORK and ROLLBACK WORK, the DB2 CONNECT command, and the adapter parameterized Passthru commands BEGIN SESSION, END SESSION, and PURGE (discussed in subsequent sections). 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 adapter.

Using the SQL Passthru BEGIN/END SESSION Commands

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

Syntax: How to Begin and Terminate a Sequence of Direct SQL Passthru Commands

```
{ENGINE|SQL} [sqlengine] {BEGIN|END} SESSION
```

where:

`sqlengine`

Indicates the target RDBMS. Acceptable values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

`BEGIN`

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.

END

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 *Controlling Connection Scope* on page 295). Purges all statements PREPAREd inside the BEGIN SESSION/END SESSION pair.

After the END SESSION command is executed, cursors and statements PREPAREd within the BEGIN SESSION/END SESSION pair are unavailable. The sequence of statements within 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 DB2 plan.
- SQL commands. The SQL commands COMMIT WORK and ROLLBACK WORK are particularly useful in Parameterized SQL Passthru requests.
- Environmental commands such as SET PLAN, SET SSID, and DB2 CONNECT statements.

If you omit the BEGIN SESSION/END SESSION pair, the adapter 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 *Controlling Connection Scope* on page 295). 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.

Using the SQL Passthru COMMIT WORK Command

COMMIT WORK terminates a unit of work and makes all data source changes permanent.

Syntax: How to COMMIT Data Source Changes

```
{ENGINE|SQL} [sqlengine] COMMIT WORK
```

where:

sqlengine

Indicates the target RDBMS. Acceptable values are DB2, SQLDBC, SQLIDMS, or SQLORA. 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.

Using the SQL Passthru ROLLBACK WORK Command

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

Syntax: How to ROLLBACK Data Source Changes

```
{ENGINE|SQL} [sqlengine] ROLLBACK WORK
```

where:

sqlengine

Indicates the target RDBMS. Acceptable values are DB2, SQLDBC, SQLIDMS, or SQLORA. 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.

Using the SQL Passthru PREPARE Command

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.

Syntax: How to PREPARE an SQL Statement

```
{ENGINE|SQL} [sqlengine] PREPARE statement_name FOR sql_statement [;]
```

where:

sqlengine

Indicates the target RDBMS. Acceptable values are DB2, SQLDBC, SQLIDMS, or SQLORA. 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 (Master File 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.

Example: Preparing a Parameterized Passthru Command

Consider the following SQL PREPARE command for DB2:

```
SQL DB2 PREPARE D FOR DELETE FROM USER1.EMPLOYEE
   WHERE EMP_ID = ?
```

Variable D will contain the PREPAREd version of the following SQL string:

```
DELETE FROM USER1.EMPLOYEE WHERE EMP_ID = ?
```

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.

Reference: Usage Notes for PREPARE

- 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.

- Using the PREPARE command with the Oracle RDBMS to issue Data Definition Language statements such as CREATE or DROP causes the statements to be executed immediately. A subsequent EXECUTE statement is not necessary.

- You must use the proper form of parameter marker for the RDBMS you are accessing. DB2, IDMS SQL, and Teradata use the question mark (?) as the parameter marker. Oracle uses :00*n*, where *n* is incremented by 1 for each parameter. For example:

  ```
  SQL SQLORA PREPARE I FOR INSERV INTO EMP (EMP_ID,COURSE)
  VALUES(:001,:002)
  END
  ```

Using the SQL Passthru EXECUTE Command

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.

Syntax: How to Execute a Prepared SQL Command

```
{ENGINE|SQL} [sqlengine] EXECUTE statement_name [USING data_list] [;]
```

where:

sqlengine

Indicates the target RDBMS. Acceptable values are DB2, SQLDBC, SQLIDMS, or SQLORA. 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.

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.

Example: Executing a Prepared Passthru Command

To execute the PREPARE statement in *Preparing a Parameterized Passthru Command*, issue:

```
SQL DB2 EXECUTE D USING 'AAA';
```

The DELETE statement from *Using the SQL Passthru PREPARE Command* on page 287 is sent to the DB2 RDBMS. The RDBMS deletes all rows with an EMP_ID value of 'AAA' from the data source. The resulting SQLCODE is returned to the program.

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

Reference: Usage Notes for EXECUTE

- 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'
```

- Using the PREPARE command with the Oracle RDBMS to issue Data Definition Language statements such as CREATE or DROP, causes the statements to be executed immediately. A subsequent EXECUTE statement is not necessary.

Using the SQL Passthru PURGE Command

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

Syntax: How to Purge an SQL Command

```
{ENGINE|SQL} [sqlengine] PURGE statement_name [;]
```

where:

sqlengine

Indicates the target RDBMS. Acceptable values are DB2, SQLDBC, SQLIDMS, or SQLORA. 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.

Example: Purging an SQL Command

```
SQL DB2 PREPARE D FOR DELETE FROM USER1.EMPLOYEE
    WHERE EMP_ID = ?
SQL DB2 PURGE D;
```

Using the SQL Passthru BIND Command

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 data type supported by the RDBMS.

Syntax: How to Define Formats of SQL Parameters

```
{ENGINE|SQL} [sqlengine] BIND statement_name USING format_list;
```

where:

sqlengine

Indicates the target RDBMS. Acceptable values are DB2, SQLDBC, SQLIDMS, or SQLORA. 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 data types used in the request. The following data types are supported by the RDBMS:

DB2	Teradata	Oracle	IDMS SQL
SMALLINT	SMALLINT	NUMBER(m,n)	SMALLINT
INTEGER	INTEGER	VARCHAR(n)	INTEGER
DECIMAL(m,n)	DECIMAL(m,n)	LONG	DECIMAL(m,n)
FLOAT	FLOAT	CHARACTER(n)	FLOAT
REAL	BYTEINT	DATE	REAL
DOUBLE	BYTE(n)	RAW	DOUBLE
VARCHAR(n)	VARBYTE(n)		VARCHAR(n)
LONGVARCHAR(n)	VARCHAR(n)		LONGVARCHAR(n)
CHARACTER(n)	LONG VARCHAR(n)		CHARACTER(n)
DATE	CHARACTER(n)		
	DATE		
	TIME		

DB2	Teradata	Oracle	IDMS SQL
	TIMESTAMP		

Reference: **Usage Notes for BIND**

- The Oracle data type LONGRAW is currently not supported.
- If a statement is PREPAREd and EXECUTEd without a corresponding BIND, the adapter 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.

Parameterized SQL Passthru Sample Session

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

```
SQL DB2 BEGIN SESSION
SQL DB2 PREPARE ABC FOR UPDATE STARS SET NAME=? WHERE DISTANCE=? ;
END

SQL DB2 PREPARE DEF FOR SELECT * FROM STARS WHERE DISTANCE=? AND
DENSITY=?;
END

SQL DB2 BIND ABC USING CHAR(6),DECIMAL(5,0);
END

SQL DB2 BIND DEF USING DECIMAL(5,0),DECIMAL(6,2);
END

-* repeat with different input data...
    SQL DB2 EXECUTE ABC USING 'GAMMA',555. ;
    END
    SQL DB2 EXECUTE ABC USING 'DELTA',777. ;
    END
    SQL DB2 EXECUTE ABC USING 'ALPHA',9640. ;
    END
    SQL DB2 EXECUTE DEF USING 555.,324.27; TABLE FILE DEF PRINT *
    END
-* end repeat
```

```
SQL DB2 COMMIT WORK ;
END

SQL DB2 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.

Chapter 10

Controlling Connection Scope

The adapter includes a variety of COMMIT, connection, and thread control capabilities. The SET AUTO*action* ON event command implements these facilities.

This chapter includes:

- The SET AUTO*action* ON event command.
- The effects of various action and event combinations.
- Examples of three types of sessions, default, user controlled, and pseudo-conversational, made possible by varying the settings.

In this chapter:

Invoking Actions in Response to Events

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

For DB2, you can make a MODIFY transaction pseudo-conversational by issuing CLOSE and DISCONNECT automatically at all COMMIT points within 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 adapter issues actions. Subsequent sections discuss each action.

Syntax: How to Automatically Invoke Actions in Response to Events

```
{ENGINE|SQL} [sqlengine] SET action ON event
```

where:

sqlengine

Indicates the target RDBMS. Acceptable values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command or to set AUTOCOMMIT ON CRTFORM in a MODIFY procedure.

action

Indicates the action to be taken by the adapter. Valid values are:

AUTOCOMMIT

Issues the SQL command COMMIT WORK or the SQL/DBC command END TRANSACTION. (Teradata implicitly commits each SQL statement individually, so the COMMIT action is redundant.)

AUTOCLOSE

For **DB2**, is the DB2 Call Attachment Facility (CAF) CLOSE operation.

For **Teradata**, AUTOCLOSE sponsors a LOGOFF.

For **IDMS/SQL** and **Oracle**, AUTOCLOSE is ignored

AUTODISCONNECT

For **DB2**, severs the connection between the RDBMS and the user's address space.

For **Oracle**, sponsors a LOGOFF from the Oracle RDBMS.

For **Teradata** and **IDMS/SQL**, is ignored.

event

Is the event that triggers the action. Valid values are:

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.

For **Teradata**, this setting is not needed and is ignored because any MODIFY that does not contain BEGIN TRANSACTION and END TRANSACTION statements implicitly commits each SQL statement individually.

Omit *sqlengine* from the command. Refer to *Maintaining Tables With FOCUS* on page 349, for more information on MODIFY.

COMMAND

Executes the specified action at the end of a MODIFY procedure, a TABLE request, a Direct SQL Passthru request, or a CALLDB2 subroutine containing embedded SQL.

Note: The adapter 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. To execute the action within the session, you must issue it explicitly. Since you cannot issue CLOSE and DISCONNECT explicitly, this applies to COMMIT only.

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.

Reference: Usage Notes for SET AUTOaction ON Event

- Depending on how often the event occurs (and the corresponding command is issued), the AUTOaction setting may result in considerable overhead. Almost none of this overhead is FOCUS related. it is z/OS and RDBMS related.
- For **DB2**, you can use the CLOSE and DISCONNECT commands only if the Adapter for DB2 is installed to use CAF.

- All settings are session level and can be issued from either the FOCUS command line or a FOCEXEC, 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.
- You can issue all settings except AUTOCOMMIT ON CRTFORM in batch jobs.
- You can use the CLOSE and DISCONNECT actions with FOCUS running under TSO.
- If AUTOCOMMIT is set on COMMAND, the adapter issues a COMMIT WORK at the end of a MODIFY procedure provided there is an open Logical Unit of Work (LUW).

Understanding 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 data source 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 data source 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.

Teradata implicitly commits each SQL statement individually, so the COMMIT action is redundant.

AUTOCLOSE

SET AUTOCLOSE initiates the **DB2** Call Attachment Facility (CAF) CLOSE operation or sponsors a **Teradata** LOGOFF. It is ignored for IDMS/SQL or Oracle.

For **DB2**, it determines how long a thread (the connection between the application program in the user's address space and the DB2 application plan) is open. The thread is not the same as the address space connection to DB2. The AUTODISCONNECT setting, discussed in a subsequent section, controls that connection. In FOCUS, a **DB2** application program is one of the following:

- The dynamic FOCUS Adapter for DB2.
- A MODIFY procedure using static SQL.
- A CALLDB2 subroutine using embedded SQL.

Generally speaking, each program has a corresponding plan. (*Static SQL (DB2)* on page 401, includes a discussion of plan management.)

A site that installs a DB2 subsystem determines the maximum number of concurrent users (threads) the subsystem will support. Since each user requires enough virtual storage for his application plan, this setting controls the amount of storage the site wants to allocate to active DB2 users at any one time.

The CAF CLOSE command de-allocates the DB2 thread, releasing the virtual storage for the application plan. DB2 requires that an existing thread to a plan be closed before a thread to another plan is opened. If a thread is closed without a subsequent OPEN operation, the closed thread becomes inactive. The user is still connected to DB2, but not to a particular application plan. The user (task) still owns the thread. It is not available to other users. To release the thread, the user must disconnect completely from DB2.

For **Teradata**, the SET AUTOCLOSE command enables users to control logon and logoff interaction with the Teradata RDBMS. After the Teradata RDBMS processes a FOCUS request or native DBC/SQL command, the connection (or communications path) may be deactivated or retained depending on the AUTOCLOSE setting.

- If the AUTOCLOSE setting is ON FIN, the initial logon connection is retained and remains active for the duration of the FOCUS session. The adapter initiates a Teradata logon request at the first native DBC/SQL command or FOCUS request requiring Teradata services, provided a valid SET CONNECTION_ATTRIBUTES command has been issued. The connection is released at the FIN command.

- If the AUTOCLOSE setting is ON COMMAND, the initial logon connection is released after Teradata processing and, for each subsequent request, is re-established. The adapter logs on prior to the DBC/SQL call and generates a Teradata logoff request after completing each FOCUS command (TABLE request, MODIFY procedure, or Direct SQL Passthru request). Teradata resources are freed prior to report generation and display. Although the logoff request is issued and the connection to Teradata is disconnected, the FOCUS session remains active.

Any other setting defaults to AUTOCLOSE ON FIN. AUTOCLOSE ON FIN minimizes overhead required for repetitive interaction with the Teradata RDBMS by retaining the connection to Teradata. By contrast, AUTOCLOSE ON COMMAND frees resources that are no longer (or infrequently) required.

Note For Teradata:

- The SET CONNECTION_ATTRIBUTES command described in *Adapter Commands* on page 309 functions independently from the SET AUTOCLOSE command.

- Regardless of the AUTOCLOSE setting, the FIN command to exit the session generates an explicit logoff request. This ensures a successful user exit from the Teradata environment. Teradata frees the resources required for the session.

AUTODISCONNECT

For **DB2**, DISCONNECT completely detaches the user's address space (or task) from DB2. This differs from CLOSE because, after a CLOSE, the FOCUS task is still connected to the DB2 subsystem and can open a thread to another plan. After a DISCONNECT, the FOCUS task must reestablish its connection to DB2 before doing any database work. FOCUS tasks that frequently issue the DISCONNECT command are connected to DB2 for shorter periods of time, allowing other tasks to connect and acquire threads as needed. However, there is significant system overhead associated with frequently connecting and disconnecting, and the possibility exists that no thread will be immediately available when the task attempts to reconnect.

For **Teradata** and **IDMS/SQL**, DISCONNECT is ignored. For **Teradata**, use AUTOCLOSE instead.

For **Oracle**, DISCONNECT sponsors a LOGOFF from the Oracle RDBMS. AUTODISCONNECT ON COMMIT is not supported. It will be converted to AUTODISCONNECT ON FIN.

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	COMMIT	COMMAND	CRTFORM	FIN
	X	D	S*	S
CLOSE (**DB2 CAF** and **Teradata** only)	S***	S	X	D
DISCONNECT (**DB2**, and **Oracle** only)	S**	S	X	D

Note: * Supported within a MODIFY procedure only. Not needed for Teradata.
** Not supported for Oracle.
*** Not supported for Teradata.

The following sections discuss the advantages and disadvantages of certain combinations of actions and events.

SET AUTOCOMMIT ON CRTFORM

This command works only from within 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 FOCURRENT variable to determine whether CVP detected a conflict with another user. *Maintaining Tables With FOCUS* on page 349 discusses this setting and some limits on its use. This setting also requires the KEYS value in the Access File to be greater than zero.

With this strategy, more concurrent users can access the same DB2 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.

For **Teradata**, each SQL statement is implicitly committed. Therefore, this setting is not needed.

SET AUTOCOMMIT ON FIN

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

```
{ENGINE|SQL} [sqlengine] SET AUTOCOMMIT ON FIN
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

With this setting, 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 *Explicit Control of a Logical Unit of Work (LUW)* for an example.

Maintain does not support the AUTOCOMMIT ON FIN setting.

Holding unneeded locks can cause contention in the system and make other users wait for data. Delaying the COMMIT also puts 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) within 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 adapter. Examine each return code and take appropriate action to prevent unwanted changes from being committed at FIN.
- Teradata implicitly commits every SQL statement individually unless you enclose them in BEGIN TRANSACTION and END TRANSACTION commands.

SET AUTOCLOSE ON COMMAND

This setting closes the **DB2** thread at the end of each command or logs off from the **Teradata** RDBMS. The syntax is:

```
{ENGINE|SQL} [DB2|SQLDBC] SET AUTOCLOSE ON COMMAND
```

Omit the DB2 or SQLDBC qualifier if you previously issued the SET SQLENGINE command.

This setting releases some virtual storage, but there is a cost to repeatedly initializing a thread. Also, closing a thread does not make it available to other users. Only a disconnect can release it.

Note: AUTOCLOSE is ignored for IDMS/SQL and Oracle. See the description of AUTODISCONNECT for a comparable function.

SET AUTODISCONNECT ON COMMIT

For **DB2**, when a COMMIT is issued, the CAF facility disconnects the FOCUS session from DB2, terminating the DB2 thread. For **Oracle**, this setting logs off the Oracle RDBMS. The syntax is

```
{ENGINE|SQL} [sqlengine] SET AUTODISCONNECT ON COMMIT
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2 or SQLORA. Omit if you previously issued the SET SQLENGINE command.

This setting frees the DB2 thread for use by other users or logs off the Oracle RDBMS. The disadvantage is the cost of repeatedly connecting to DB2 and acquiring a thread. Threads, once released, may not be available when needed, so you may experience delays while your request waits for a thread.

Note:

- When you combine this setting with AUTOCOMMIT ON FIN, you can control the span of the DB2 session.
- If you attempt to issue this setting for Oracle, it will be converted to AUTODISCONNECT ON FIN.

SET AUTOCLOSE ON FIN

The adapter issues a **DB2** CAF CLOSE at the end of the FOCUS session or sponsors a LOGOFF from the **Teradata** RDBMS. The syntax is

```
{ENGINE|SQL} [DB2|SQLDBC] SET AUTOCLOSE ON FIN
```

Omit the DB2 or SQLDBC qualifier if you previously issued the SET SQLENGINE command.

This setting duplicates adapter default behavior, unless the adapter was installed with AUTOCLOSE ON COMMAND as the default.

Note: AUTOCLOSE is ignored for IDMS SQL and Oracle. See the description of AUTODISCONNECT for a comparable function.

SET AUTODISCONNECT ON FIN

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

```
{ENGINE|SQL} [sqlengine] SET AUTODISCONNECT ON FIN
```

where:

sqlengine

Indicates the target RDBMS. Valid values are **DB2** or **SQLORA**. Omit if you previously issued the SET SQLENGINE command.

Note: DISCONNECT is ignored for Teradata and IDMS/SQL.

Combinations of SET AUTO*action* Commands

Think of actions as a nested sequence. The COMMIT action is the innermost level, then CLOSE, 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,
     AUTOCLOSE      must be   FIN,
     AUTODISCONNECT must be   FIN*.

 When AUTOCLOSE      is set to FIN
     AUTODISCONNECT must be   FIN.
 When AUTOCOMMIT     is set to COMMAND or CRTFORM,
     AUTOCLOSE      may be    FIN, COMMAND or COMMIT,
     AUTODISCONNECT may be    FIN, COMMAND or COMMIT.
 When AUTOCLOSE      is set to COMMAND or COMMIT,
     AUTODISCONNECT may be    FIN, COMMAND or COMMIT.
 When AUTOCLOSE or AUTODISCONNECT is set to COMMIT,
     AUTOCOMMIT     may be    COMMAND or CRTFORM.
 * Violations of this rule can cause unpredictable behavior.
```

Note For DB2:

- Whenever you open a thread to a new application plan or connect to a new DB2 subsystem, the adapter issues a COMMIT WORK regardless of the AUTOCOMMIT setting.
- If the event that triggers AUTODISCONNECT occurs more frequently than the event that triggers AUTOCLOSE, the adapter forces a CAF CLOSE prior to any CAF DISCONNECT.
- In order to use the CLOSE and DISCONNECT commands, the Adapter for DB2 must be installed to use CAF.

Establishing Different Types of FOCUS Sessions

You can establish the following three types of FOCUS sessions by varying the combinations of SET AUTOaction ON event commands. Each is illustrated in a subsequent section:

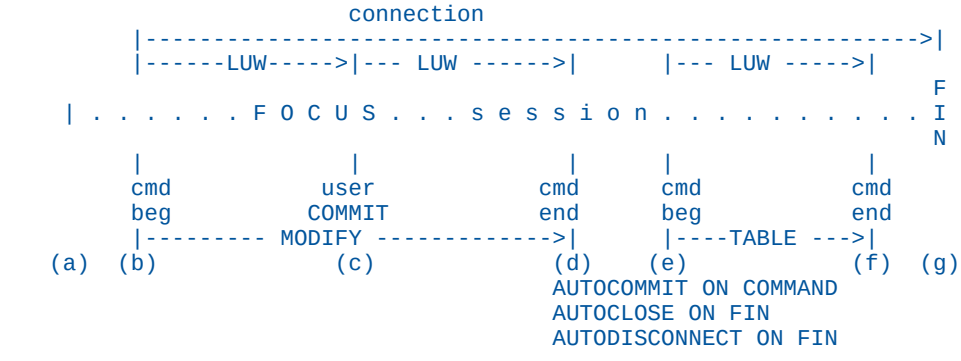
- The default adapter session uses the settings AUTOCOMMIT ON COMMAND, AUTOCLOSE ON FIN, and AUTODISCONNECT ON FIN.
- The user-controlled session uses the settings AUTOCOMMIT ON FIN, AUTOCLOSE ON FIN, and AUTODISCONNECT ON FIN.

Note: Teradata implicitly commits each SQL statement individually unless they are enclosed in BEGIN TRANSACTION and END TRANSACTION statements.

- The pseudo-conversational session uses the settings AUTOCOMMIT ON CRTFORM, AUTOCLOSE ON COMMIT, and AUTODISCONNECT ON COMMIT.

The Default Adapter Session

The following illustration shows the duration of connections, threads, and Logical Units of Work (LUWs) using the default adapter 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 call to the RDBMS, and is retained for the entire FOCUS session.

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

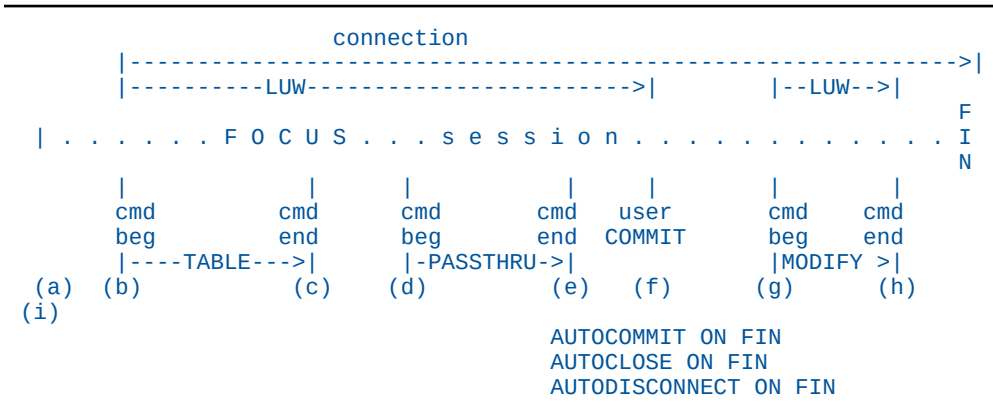
A third LUW begins for the TABLE request. It is terminated by the COMMIT automatically generated at the end of the TABLE command. At FIN, the adapter terminates the connection to the RDBMS. No COMMIT is generated at FIN.

Note: For DB2, the thread is opened when the connection is established and closed when the connection is severed.

The User-Controlled Session

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

Note: The Adapter for Teradata does not support this type of 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 MODIFY procedure, is not terminated until FIN since there is no other user-issued COMMIT.

Note: For DB2, the thread is opened when the connection is established and closed when the connection is severed.

The Pseudo-Conversational Session

The next illustration shows how connections can be dropped and re-established within a MODIFY procedure. The AUTOCOMMIT ON CRTFORM, issued within the MODIFY, automatically generates a COMMIT whenever a CRTFORM is displayed.

Note: The Adapter for Teradata does not support this type of session.

```
      |--connection--->|--connection->|-conn-->|--connection-->||
      |--LUW---------->|--LUW-------->|-LUW--->|--LUW--------->|
                                                                F
| . . . . . . F O C U S . . . s e s s i o n . . . . . . . . . . . I
                                                                N
      |              |             |       |              |
      cmd            CRTFORM      user     CRTFORM       cmd
      beg                        COMMIT                  end
      |--------------------MODIFY-------------------------->|
(a)  (b)             (c)          (d)    (e)           (f)   (g)
                                  AUTOCOMMIT ON CRTFORM
                                  AUTOCLOSE ON COMMIT
                                  AUTODISCONNECT ON COMMIT
```

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

The connection and Logical Unit of Work (LUW) are established at point (b), the beginning of the MODIFY procedure. The first CRTFORM triggers a COMMIT, terminating the LUW and connection. The adapter automatically re-establishes them at the next call to the RDBMS, and they are terminated by the user-issued COMMIT at point (d).

The process is repeated two more times. At the end of the MODIFY procedure, point (f), AUTOCOMMIT reverts to its previous (default) COMMAND setting, and the adapter generates a COMMIT.

Note: For DB2, a thread is opened when the connection is established and closed when the connection is severed.

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

The following example demonstrates how to explicitly control the scope of an LUW using the AUTO command settings. Numbers to the left refer to explanatory notes that follow the example.

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:

```
-TOP
    SQL DB2 SET AUTODISCONNECT ON FIN
    SQL DB2 SET AUTOCLOSE ON FIN
1.  SQL DB2 SET AUTOCOMMIT ON FIN


2.  SQL DB2 LOCK TABLE XYZ IN EXCLUSIVE MODE
    END
3.  -IF &RETCODE NE 0 GOTO ROLLBACK ;
```

```
4.  SQL DB2 INSERT INTO XYZ VALUES ('A','B','C','D') ;
    END
    -IF &RETCODE NE 0 GOTO ROLLBACK ;
    SQL DB2 INSERT INTO XYZ VALUES ('E','F','G','H') ;
    END
    -IF &RETCODE NE 0 GOTO ROLLBACK ;
    SQL DB2 INSERT INTO XYZ VALUES ('I','J','K','L') ;
    END
    -IF &RETCODE NE 0 GOTO ROLLBACK ;

5.  SQL DB2 COMMIT WORK;
    END
    -IF &RETCODE NE 0 GOTO ROLLBACK ;
    -GOTO OUT
    -ROLLBACK
    SQL DB2 ROLLBACK WORK;
    END
    -OUT 6.  SQL DB2 SET AUTOCOMMIT ON COMMAND
```

Note:

1. AUTOCOMMIT is set to FIN. No COMMIT is issued unless specifically coded. As is required for this AUTOCOMMIT setting, AUTODISCONNECT and AUTOCLOSE are also set to FIN.
2. Table XYZ is locked for the exclusive use of this program. This lock will be terminated at a COMMIT or ROLLBACK point.
3. 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.
4. Three rows are inserted into XYZ.
5. The program issues an explicit COMMIT making the inserts to XYZ permanent and releasing the exclusive lock.
6. SET AUTOCOMMIT is reissued to restore automatic command-level COMMITs.

Chapter 11

Adapter Commands

Adapter commands can change certain parameters that govern your FOCUS session. These parameters control or identify the Change Verify Protocol, DB2 Call Attachment Facility defaults, DBSPACE defaults, the Fastload option, the DB2 CURRENT SQLID, Oracle blocked insert and fetch sizes, adapter logon parameters, optimization, and isolation levels. Environmental commands that control connection to the RDBMS, such as SET AUTODISCONNECT, are discussed in *Controlling Connection Scope* on page 295.

You can display all the current parameter settings with the adapter SQL ? query described in *Querying Adapter Parameter Settings* on page 310.

In this chapter:

Issuing Adapter Commands

The Direct SQL Passthru facility provides support for issuing adapter environmental commands.

Note: Commands that you issue from MODIFY procedures, like LOADONLY, ERRORRUN, and AUTOCOMMIT ON CRTFORM, use a slightly different form of syntax that is described in *Parameters That Apply to MODIFY Only* on page 343.

Syntax: How to Issue Adapter Commands

```
{ENGINE|SQL} [sqlengine] SET command value
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

command

Is an environmental command.

value

Is an acceptable value for the environmental command.

Querying Adapter Parameter Settings

The SQL query command displays defaults and current settings for each adapter.

Syntax: How to Query Data Adapter Parameter Settings

```
{ENGINE|SQL} [sqlengine] ?
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

Example: Querying Adapter for DB2 Settings

The following is an example for DB2:

```
> sql db2 ?
(FOC1440) CURRENT SQL INTERFACE SETTINGS ARE :
(FOC1442) CALL ATTACH FACILITY IS             -  : ON
(FOC1447) SSID FOR CALL ATTACH IS             -  : DBAA
(FOC1448) ACTIVE PLAN FOR CALL ATTACH IS      -  : USER1CAF
(FOC1459) USER SET PLAN FOR CALL ATTACH IS    -  : USER1CAF
(FOC1460) INSTALLATION DEFAULT PLAN     IS    -  : M727703B
(FOC1503) SQL STATIC OPTION IS                -  : OFF
(FOC1444) AUTOCLOSE OPTION IS                 -  : ON FIN
(FOC1496) AUTODISCONNECT OPTION IS            -  : ON FIN
(FOC1499) AUTOCOMMIT OPTION IS                -  : ON COMMAND
(FOC1449) CURRENT SQLID IS                    -  : SYSTEM DEFAUL
(FOC1424) ISOLATION LEVEL FOR DB2 TABLE INTERFACE IS  :
(FOC1491) FETCH BUFFERING FACTOR              -  : 100
(FOC1441) WRITE FUNCTIONALITY IS              -  : ON
(FOC1445) OPTIMIZATION OPTION IS              -  : ON
(FOC1763) IF-THEN-ELSE OPTIMIZATION IS        - : ON
(FOC1484) SQL ERROR MESSAGE TYPE IS           -  : DBMS
(FOC1497) SQL EXPLAIN OPTION IS               -  : OFF
(FOC1552) INTERFACE DEFAULT DATE TYPE         -  : NEW
(FOC1446) DEFAULT DBSPACE IS                  -  : PUBLIC.SPACE0
```

Example: Querying Adapter for Teradata Settings

The following is an example for Teradata:

```
> sql sqldbc ?
(FOC1461) CURRENT DBC/SQL SETTINGS ARE:
(FOC1463) DBC/1012 CONNECTION                 -  : OFF
(FOC1467) PARTITION                           -  : DBC/SQL
(FOC1468) TERADATA DIRECTOR PROGRAM (TDP)     -  : 0
(FOC1469) DBC/1012 USER ID                    -  : DBC
(FOC1444) AUTOCLOSE OPTION IS                 -  : ON COMMAND
(FOC1465) DBC COLNAME OPTION IS               -  : TITLE
(FOC1441) WRITE FUNCTIONALITY IS              -  : ON
(FOC1445) OPTIMIZATION OPTION IS              -  : ON
(FOC1763) IF-THEN-ELSE OPTIMIZATION IS        -  : OFF
(FOC1497) SQL EXPLAIN OPTION IS               -  : OFF
(FOC1552) INTERFACE DEFAULT DATE TYPE          -  : NEW
```

Example: Querying Adapter for IDMS/SQL Settings

The following is an example for IDMS/SQL:

```
> sql sqlidms ?
(FOC1756) CURRENT IDMS DICTIONARY IS      -  : SYSTEM DEFAULT
(FOC1757) CURRENT SCHEMA IS               -  : SYSTEM DEFAULT
(FOC1758) CURRENT ISOLATION LEVEL IS       -  : READ WRITE CURSOR
STABILITY
(FOC1499) AUTOCOMMIT OPTION IS            -  : ON COMMAND
(FOC1445) OPTIMIZATION OPTION IS          -  : ON
(FOC1763) IF-THEN-ELSE OPTIMIZATION IS    -  : OFF
(FOC1552) INTERFACE DEFAULT DATE TYPE     -  : NEW
(FOC1446) DEFAULT DBSPACE IS
```

Example: **Querying Adapter for Oracle Settings**

The following is an example for Oracle:

```
> sql sqlora ?
(FOC1450) CURRENT ORACLE INTERFACE SETTINGS ARE  :
(FOC1656) DEFAULT SERVER NAME                 -  : <local>
(FOC1502) USERID AND PASSWORD ARE             -  : SCOTT
(FOC1496) AUTODISCONNECT OPTION IS            -  : ON FIN
(FOC1499) AUTOCOMMIT OPTION IS                -  : ON COMMAND
(FOC1491) FETCH BUFFERING FACTOR              -  : 20
(FOC1531) INSERT BUFFERING FACTOR             -  : 1
(FOC1379) MAXIMUM STORED PROCEDURE PARAMETERS -  : 256
(FOC1652) CHARACTER TYPE FOR INSERT OR UPDATE -  : VAR
(FOC1653) CONVERSION STYLE FOR ORACLE NUMBERS -  : COMPAT
(FOC1654) ORACLE HOME                         -  : Not set
(FOC1655) ORACLE SID                          -  : Not set
(FOC1441) WRITE FUNCTIONALITY IS              -  : ON
(FOC1445) OPTIMIZATION OPTION IS              -  : ON
(FOC1763) IF-THEN-ELSE OPTIMIZATION IS        -  : OFF
(FOC1552) INTERFACE DEFAULT DATE TYPE         -  : NEW
(FOC1446) DEFAULT DBSPACE IS
```

Parameters That Apply to Multiple Adapters

The following parameters can be used with multiple relational adapters. These include CONVERSION, DBSPACE, DEFDATE, FETCHSIZE, INSERTSIZE, OPTIFTHENELSE, OPTIMIZATION, PASSRECS, SQLJOIN OUTER, TRIM_LITERALS, and VARCHAR. This topic also describe the EXPLAIN parameter that applies to the Adapters for DB2 and Teradata. The IXSPACE parameter applies to the Adapters for DB2, Oracle, and IDMS/SQL. The OWNERID parameter applies to the Adapters for DB2, Teradata, and Oracle.

CONVERSION

You can use the SET CONVERSION command to alter the length and scale of numeric columns displayed from a SELECT request. That is, you can control the USAGE attribute in the dynamically created Master File.

Syntax: **How to Alter Length and Scale of Numeric Columns Returned**

```
{ENGINE|SQL} [sqlengine] SET CONVERSION {RESET|dtype} [RESET|PRECISION
{value|MAX}]
```

where:

sqlengine

Indicates the target RDBMS. Acceptable values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

RESET

Returns precision and scale values that you previously altered back to the adapter defaults. If you specify RESET immediately following the SET CONVERSION command, all data types return to the defaults. If you specify RESET following a particular data type, only columns of that data type are reset.

dtype

Applies the command only to columns of a specific data type. Valid data types are:

INTEGER

INTEGER (and, for Oracle, SMALLINT).

DECIMAL

DECIMAL.

REAL

Single precision floating point. Not supported for Oracle or Teradata.

FLOAT

Double precision floating point.

value

Is the precision in the following form:

nn [*mm*]

where:

nn

Must be greater than 1 and less than the maximum allowable value for the data type. (See description of MAX.)

mm

Is the scale. Valid with DECIMAL, REAL, and FLOAT data types. 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 data type:

DATA TYPE	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.

CONVERSION LONGCHAR (DB2, Oracle, Teradata)

The SET parameter CONVERSION LONGCHAR controls the mapping of supported data types listed below. By default, the adapter maps these data types as alphanumeric (A). The FOCUS data type A supports a maximum of 4096 characters for TABLE/MODIFY and 32768 characters for API applications.

The following table lists data type mappings based on the value of LONGCHAR for DB2.

		LONGCHAR ALPHA		BLOB		LONGCHAR TEXT	
DB2 Data Type	**Remarks**						
VARCHAR (*n*)	*n* is an integer between 1 and 32768	A*n*V	A*n*V	BLOB	BLOB	TX50	TX

The following table lists data type mappings based on the value of LONGCHAR for Oracle.

Oracle Data Type	Remarks	LONGCHAR ALPHA or BLOB		LONGCHAR TEXT	
CHAR (*n*)	*n* is an integer between 1 and 2000	A*n*	A*n*	TX50	TX
VARCHAR (*n*)	*n* is an integer between 1 and 4000	A*n*V	A*n*V	TX50	TX
VARCHAR2 (*n*)	*n* is an integer between 1 and 4000	A*n*V	A*n*V	TX50	TX
RAW (*n*)	*n* is an integer between 1 and 2000*m* = 2 * *n*	A*m*	A*m*	TX50	TX

The following table lists data type mappings based on the value of LONGCHAR for Teradata.

Teradata Data Type	Remarks	LONGCHAR ALPHA or BLOB		LONGCHAR TEXT	
CHAR	*n* is an integer between 1 and 4000	A*n*	A*n*	TX50	TX
VARCHAR (*n*)	*n* is an integer between 1 and 4000	A*n*V	A*n*V	TX50	TX
LONG VARCHAR (*n*)	*n* is an integer between 1 and 4000	A*n*V	A*n*V	TX50	TX

Syntax: How to Control the Mapping of Large Character Data Types

```
ENGINE sqlengine SET CONVERSION LONGCHAR {ALPHA|TEXT|BLOB}
```

where:

sqlengine

Indicates the adapter. Acceptable values are DB2, SQLDBC, or SQLORA. You can omit this value if you previously issued the SET SQLENGINE command.

ALPHA

Maps the DB2 data type VARCHAR as alphanumeric (A*n*V).

Maps the Oracle data types CHAR, NCHAR, VARCHAR, VARCHAR2, NVARCHAR2, and RAW as alphanumeric (A).

Maps the Teradata data types CHAR, VARCHAR, and LONG VARCHAR as alphanumeric (A).

ALPHA is the default value.

`TEXT`

Maps the DB2 data type VARCHAR as text (TX).

Maps the Oracle data types CHAR, NCHAR, VARCHAR, VARCHAR2, NVARCHAR2, and RAW as text (TX).

Maps the Teradata data types data types CHAR, VARCHAR, and LONG VARCHAR as text (TX).

Use this value for FOCUS applications.

`BLOB`

maps the DB2 data type VARCHAR as alphanumeric (A). This is equivalent to ALPHA.

Maps the Oracle data types CHAR, NCHAR, VARCHAR, VARCHAR2, NVARCHAR2, and RAW as binary large object (BLOB).

Maps the Teradata data types CHAR, VARCHAR, and LONG VARCHAR as binary large object (BLOB).

DBSPACE

Within a FOCUS session, you can designate a default storage space for tables you create with the FOCUS CREATE FILE or HOLD FORMAT SQL*engine* commands. For the duration of the session, the RDBMS places such tables in the IDMS/SQL segment.area, Oracle tablespace, or DB2 database you identify in the SET DBSPACE command.

Issue the SET DBSPACE command from the FOCUS command level

`{ENGINE|SQL} [`*`sqlengine`*`] SET DBSPACE `*`storage`*

where:

`sqlengine`

Indicates the target RDBMS. Valid values are DB2, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

storage

For **DB2**, is databasename.tablespacename or DATABASE databasename. In DB2, the RDBMS default value is DSNDB04, a public database. (DB2 automatically generates a tablespace in DSNDB04.)

For **IDMS SQL** is segment.area. The default is the IDMS default area for the current schema in effect for the user's SQL session.

For **Oracle**, is tablespacename. If the SET DBSPACE command is not issued, Oracle uses the default tablespace for the connected user.

The adapter may have been installed with a default, site-specific, DBSPACE specification. Use the SQL ? query command to display the setting.

DEFDATE

Use the SET DEFDATE command to change the value of the default date FOCUS uses for the RDBMS DATE data type. See in *Additional Topics* on page 417, for a complete discussion.

From the FOCUS command line, issue

```
{ENGINE|SQL} [sqlengine] SET DEFDATE {NEW|OLD}
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

NEW

Supplies '1900-12-31' as the base date for RDBMS columns with DATE data types. NEW is the default.

OLD

Supplies '1901-01-01' as the base date for RDBMS columns with DATE data types. OLD was the default in versions of the data adapter prior to FOCUS Version 6.8.

If you use the MODIFY facility to maintain RDBMS tables containing DATE columns, a change in the default date from prior releases may impact existing applications. See *Additional Topics* on page 417 for more information.

EXPLAIN (DB2, Teradata)

You can instruct the adapter to execute an RDBMS EXPLAIN command for the SQL SELECT statements issued by a FOCUS request before it actually issues the FOCUS request. At the FOCUS command level enter

```
{ENGINE|SQL} [sqlengine] SET EXPLAIN {OFF|ON [n]}
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2 or SQLDBC. Omit if you previously issued the SET SQLENGINE command.

OFF

Is the default. The adapter proceeds as usual.

ON

Instructs the adapter to issue an RDBMS EXPLAIN command for the SQL SELECT statements issued by a FOCUS request, then issue the request itself. (Do not confuse this with the adapter EXP facilities.)

n

Is the query number to use in the DB2 EXPLAIN tables only. The default value is 1.

To use the ON setting, you must satisfy all RDBMS requirements for executing an EXPLAIN, such as having EXPLAIN tables in DB2 or an XPLTRACE allocation for Teradata. The ON setting applies to TABLE and SQL Passthru SELECT requests only. It is ignored by the MODIFY facility and non-SELECT SQL Passthru requests.

Note: For **Teradata**, TABLE requests that result in multiple SQL SELECT statements and Direct SQL Passthru requests are not supported with SET EXPLAIN set to ON.

FETCHSIZE (DB2, Oracle)

The Adapters for DB2 and Oracle support array retrieval from result sets produced by executing SELECT queries or stored procedures. This technique substantially reduces network traffic and CPU utilization.

Using high values increases the efficiency of requests involving many rows, at the cost of higher virtual storage requirements. A value higher than 100 is not recommended because the increased efficiency it would provide is generally negligible.

Syntax: How to Specify Block Size for Array Retrieval

The block size for a SELECT request applies to TABLE FILE requests, MODIFY requests, MATCH requests, and DIRECT SQL SELECT statements.

```
{ENGINE|SQL} sqlengine SET FETCHSIZE n
```

where:

sqlengine

Indicates the adapter. Valid values are DB2 or SQLORA. You can omit this value if you previously issued the SET SQLENGINE command.

n

Is the number of rows to be retrieved at once using array retrieval techniques for the CLI adapter or a cursor with Rowset positioning and multi row Fetch for the CAF adapter. Accepted values are 1 to 32000 for DB2 and 1 to 5000 for Oracle. The default for DB2 is 100. The default for Oracle is 20. If the result set contains a column that has to be processed as a CLOB or a BLOB, the FETCHSIZE value used for that result set is 1.

Note: The DB2 default value of 100 is incompatible with compiled (static) MODIFY procedures. To run a compiled MODIFY procedure, you must SET FETCHSIZE to 1.

INSERTSIZE (DB2 CLI, Oracle)

In combination with LOADONLY, the block size for an INSERT applies to MODIFY INCLUDE requests. INSERTSIZE is also supported for parameterized DIRECT SQL INSERT statements.

Syntax: How to Specify Block Size for Insert Processing

```
{ENGINE|SQL} sqlengine SET INSERTSIZE n
```

where:

sqlengine

Indicates the adapter. Valid values are DB2 and SQLORA. You can omit this value if you previously issued the SET SQLENGINE command.

n

Is the number of rows to be inserted using array insert techniques. Accepted values are 1 to 5000. 1 is the default value. If the result set contains a column that has to be processed as a BLOB, the INSERTSIZE value used for that result set is 1.

IXSPACE (DB2, IDMS/SQL, Oracle)

You can override the default parameters for the DB2 or Oracle index space or for the IDMS/SQL default index segment.area and default index specifications implicitly created by FOCUS CREATE FILE and HOLD FORMAT DB2, SQLORA, or SQLIDMS. The syntax is

```
{ENGINE|SQL} [sqlengine] SET IXSPACE [index_spec]
```

where:

sqlengine

Identifies the target RDBMS. Valid values are DB2, SQLORA, or SQLIDMS. Omit if you previously issued the SET SQLENGINE command.

index_spec

Is the portion (up to 94 bytes) of the SQL CREATE INDEX statement beginning with:

- The DB2 syntax USING-BLOCK (as specified in the *IBM DB2 SQL Reference* CREATE INDEX syntax diagram).
- The IDMS/SQL syntax IN segment.area (as specified in the *CA-IDMS/DB Release 12 SQL Reference* CREATE INDEX syntax diagram).

To reset to the default index space parameters, issue the SET IXSPACE command with no operands.

Example: Providing Index Space Parameters

Use the long form of Direct SQL Passthru syntax for commands that exceed one line:

```
{ENGINE|SQL} sqlengine
SET IXSPACE index_spec
END
```

To specify the USING-BLOCK, FREE-BLOCK, and CLUSTER portions of the CREATE INDEX statement for DB2, enter the following:

```
SQL DB2
SET IXSPACE USING STOGROUP SYSDEFLT PRIQTY 100
SECQTY 20 FREEPAGE 16 PCTFREE 5 CLUSTER
END
```

To specify the segment.area and INDEX BLOCK portions of the CREATE INDEX statement for IDMS/SQL, enter the following:

```
SQL SQLIDMS
SET IXSPACE IN EMPSEG.EMPAREA INDEX BLOCK CONTAINS 5 KEYS
END
```

To specify the Oracle table space ORATS1 for an index:

```
SQL SQLORA SET IXSPACE TABLESPACE ORATS1
```

You can use the SQL ? query command to determine the current IXSPACE setting. If the current setting is the default, IXSPACE does not display in the SQL ? output.

OPTIFTHENELSE

When you issue the SET OPTIFTHENELSE command, the adapter attempts to deliver the construct of FOCUS IF-THEN-ELSE DEFINE fields to the RDBMS as an expression. The DEFINE field must be an object of a selection test or an aggregation request. The DEFINE definition may be specified in the TABLE request or in the Master File.

```
{ENGINE|SQL} [sqlengine] SET OPTIFTHENELSE {ON|OFF}
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

ON

Enables IF-THEN-ELSE optimization. ON is the default value.

OFF

Disables IF-THEN-ELSE optimization.

OPTIMIZATION

Depending on the OPTIMIZATION setting, the adapter may generate SQL SELECT statements that allow the RDBMS to perform operations (such as join and aggregation) and return the data to the adapter for FOCUS report generation (see *The Adapter Optimizer* on page 171).

To invoke adapter optimization, issue the following at the FOCUS command level

```
{ENGINE}SQL} [sqlengine] SET {OPTIMIZATION|SQLJOIN} optsetting
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

SQLJOIN

Is an alias for OPTIMIZATION.

optsetting

Indicates whether the adapter should pass sort, join, and aggregation operations to the RDBMS for processing. Valid values are:

OFF instructs the adapter 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 adapter to create SQL statements that take advantage of RDBMS join, sort, and aggregation capabilities. ON 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 adapter-managed native join logic. See *The Adapter Optimizer* on page 171, for more information. 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 requests invoke adapter-managed native join logic. See *The Adapter Optimizer* on page 171, for information.

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 tables residing on multiple subsystems and for tables residing on multiple DBMS platforms.

NOAGGR disables optimization of calculations (DEFINE fields) without disabling optimization of join and sort operations.

AGGR enables optimization of calculations (DEFINE fields). This is the default value for optimization of calculations.

OWNERID (DB2, Teradata, Oracle)

You can issue the SET OWNERID command to designate a creator name for all unqualified table names. The adapter will then use the owner ID as the creator name whenever the Access File does not include a creator value as part of the table name. Direct SQL Passthru requests are not affected by this setting. The syntax is

```
{ENGINE|SQL} [sqlengine] SET OWNERID ownerid_value
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2, SQLDBC, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

ownerid_value

Is the owner ID to assign to all unqualified table names.

The adapter uses the owner ID whenever there is ambiguity about the creator name. For example, the adapter uses the owner ID as the creator for any table you create using:

- The CREATE FILE command when the Access File does not specify a creator.
- HOLD FORMAT SQL*engine* when the name of the extract file does not include a period.

If you do not have sufficient rights to create tables using the owner ID you set, an SQL error results and the table is not created.

When this setting is in effect, the following line is added to the output of the SQL ? query:

```
(FOC1520) SQL CURRENT OWNER ID IS ownerid
```

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

```
{ENGINE|SQL} [sqlengine] SET PASSRECS {OFF|ON}
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

OFF

The adapter provides no information as to the number of records affected by a successfully executed Direct SQL Passthru UPDATE or DELETE command.

ON

Is the default. 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 adapter updates the FOCUS system variable &RECORDS with the number of rows affected. You can access this variable with Dialogue Manager and display it with the ? STAT query.

Note:

- 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.
- The &ROWSAFFECTED variable is populated with the number of rows affected by a Direct SQL Passthru INSERT, UPDATE, or DELETE command. It is populated regardless of the PASSRECS setting. &ROWSAFFECTED is initialized to -1 and is set to -1 by any Direct SQL Passthru command that does not return the number of rows affected, such as a SELECT statement. The value of &ROWSAFFECTED is overwritten each time a Direct SQL Passthru INSERT, UPDATE, or DELETE command is executed, so if you want to retain it, you must copy it to another variable or store it in a calculated field.

SQLJOIN OUTER (DB2, Teradata, Oracle)

With the SET SQLJOIN OUTER command you can control when the adapter optimizes outer joins, without affecting the optimization of other operations. (An outer join is generated when the SET ALL=ON command is in effect.) This parameter provides backward compatibility with prior releases of the adapter and enables you to fine-tune your applications.

When join optimization is in effect, the adapter generates one SQL SELECT statement that includes every table involved in the join. The RDBMS can then process the join. When join optimization is disabled, the adapter generates a separate SQL SELECT statement for each table, and FOCUS processes the join.

The syntax is

```
{ENGINE|SQL} sqlengine SET SQLJOIN OUTER {ON|OFF}
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2, SQLDBC, or SQLORA. Omit if you issued the SET SQLENGINE command.

ON

Enables outer join optimization. ON is the default for **Teradata**, and **Oracle**.

OFF

Disables outer join optimization. OFF is the default value for **DB2**.

Note:

- Left outer join optimization is not supported for IDMS/SQL.
- The SQLJOIN OUTER setting is available only when optimization is enabled (that is, OPTIMIZATION is not set to OFF).
- The SQLJOIN OUTER setting is ignored when SET ALL = OFF.

The following table describes how different combinations of OPTIMIZATION and SQLJOIN OUTER settings affect adapter behavior. It assumes that SET ALL = ON:

Settings		Results	
ON	ON	Yes	Enabled
ON	OFF	No	Enabled
OFF	N/A	No	Disabled
SQL	ON	Yes, in all possible cases	Enabled
SQL	OFF	No	Enabled
FOCUS	ON	Yes if results are equivalent to FOCUS managed request	Enabled
FOCUS	OFF	No	Enabled

If SQLJOIN OUTER is set to OFF, the following message displays when you issue the SQL ? query command:

(FOC1420) OPTIMIZATION OF ALL=ON AS LEFT JOIN - : OFF

TRIM_LITERALS (DB2, Oracle, Teradata)

By default, literal contents are preserved, including trailing blanks in string literals and the fractional part and exponential notation in numeric literals. This allows greater control over the generated SQL. In some cases, when trailing blanks are not needed, you can issue the adapter SET TRIM_LITERALS ON command to trim them.

```
ENGINE sqlengine SET TRIM_LITERALS {ON|OFF}
```

where:

sqlengine

Indicates the adapter. Acceptable values are DB2, SQLDBC, or SQLORA. You can omit this value if you previously issued the SET SQLENGINE command.

ON

Trims trailing blanks.

OFF

Preserves trailing blanks. OFF is the default value.

VARCHAR (DB2, Oracle, Teradata)

The SET parameter VARCHAR controls the mapping of the VARCHAR data type. By default, the server maps this data type as variable character (A*n*V).

Syntax: How to Control the Mapping of Variable-Length Data Types

```
ENGINE sqlengine SET VARCHAR {ON|OFF}
```

where:

sqlengine

Indicates the adapter. Acceptable values are DB2, SQLDBC, or SQLORA. You can omit this value if you previously issued the SET SQLENGINE command.

ON

Maps the VARCHAR data type as variable-length alphanumeric (A*n*V). This is required for Unicode environments. ON is the default value.

OFF

Maps the VARCHAR data type as fixed-length alphanumeric (A*n*).

Parameters That Apply to DB2 Only

The following adapter environmental commands apply only to DB2, SET BINDOPTIONS, SET CURRENT DEGREE, SET CURRENT SQLID, SET ERRORTYPE, SET ISOLATION, SET PLAN, and SET SSID. (SET AUTOCLOSE also applies to DB2 only and is discussed in *Controlling Connection Scope* on page 295.)

BINDOPTIONS

You can override the default BIND string that the adapter creates when you compile a static MODIFY procedure (with SET STATIC ON).

The syntax is

```
{ENGINE|SQL} [DB2] SET BINDOPTIONS [bind_spec]
```

where:

bind_spec

Is the portion of the BIND command that contains the BIND keywords and parameters (*not* including the BIND keyword itself). These BIND keywords and parameters must conform to rules governing the BIND PLAN and BIND PACKAGE commands described in the IBM *DB2 Command and Utility Reference*.

Note: Do not include the word BIND as part of the bind_spec.

To reset the default options, issue the command with no bind_spec:

```
SQL DB2 SET BINDOPTIONS
```

Note: Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.

Use the long form of Direct SQL Passthru syntax for commands that exceed one line:

```
SQL DB2
SET BINDOPTIONS bind_spec
END
```

The adapter default BIND string has the form:

```
BIND PLAN (focexec_name) MEM(focexec_name) ACTION(REPLACE) ISOLATION(CS)
```

You can determine the current setting for BINDOPTIONS with the adapter SQL DB2 ? query command. If the current setting is the default, BINDOPTIONS does not display in the SQL DB2 ? output.

Note: If the bind string specifies the MEM keyword, the adapter ignores it and supplies the MEM keyword with a parameter value equal to the FOCEXEC name. If the bind string omits any of the other three default keywords, PLAN/PACKAGE, ACTION, or ISOLATION, the adapter adds them automatically with their corresponding default values. To bind a package, you must specify the keyword PACKAGE, as PLAN is the default.

CURRENT DEGREE

DB2 supports parallel query I/O and parallel query CPU to improve response. You can bind static SQL requests with the DEGREE(ANY) parameter to take advantage of parallel processing. For dynamic SQL requests, the adapter supports parallel processing if you issue the SET CURRENT DEGREE command prior to the request.

The syntax is

```
SQL [DB2] SET CURRENT DEGREE {'1'|'ANY'}
```

where:

1

Is the default. Does not invoke parallel processing.

ANY

Invokes parallel processing for dynamic requests. If the thread to DB2 is closed during the session, the value resets to '1'.

Note:

- Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.
- Closing a DB2 thread restores the setting to '1', the default. Therefore, if you set AUTOCLOSE or AUTODISCONNECT to anything other than ON FIN (the adapter default), you must reissue the SET CURRENT DEGREE command prior to each request in which you want it to apply.

CURRENT SQLID

The DB2 RDBMS recognizes two types of ID, the primary authorization ID and one or more optional secondary authorization IDs. It also recognizes the CURRENT SQLID setting.

An ID known as a primary authorization ID identifies an interactive user or batch program accessing a DB2 subsystem. A security system such as RACF normally provides the ID to DB2. During the process of connecting to DB2, the primary authorization ID may be associated with one or more secondary authorization IDs (usually RACF groups). The site controls whether it uses secondary authorization IDs.

The DB2 database administrator may grant privileges to a secondary authorization ID that are not granted to the primary ID. Thus, secondary authorization IDs provide the means for granting the same privileges to a group of users. (The DBA associates individual primary IDs with the same secondary ID and then grants the privileges to the secondary ID.)

The DB2 CURRENT SQLID can be the primary authorization ID or any associated secondary authorization ID. At the beginning of the FOCUS session, the CURRENT SQLID is the primary authorization ID. You can reset it using the following adapter command

```
SQL [DB2] SET CURRENT SQLID  'sqlid'
```

where:

sqlid

Is the desired primary or secondary authorization ID, enclosed in single quotation marks. All DB2 security rules are respected.

Note: Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.

Unless you issue the SET OWNERID command, the CURRENT SQLID is the default owner ID for DB2 objects (such as tables or indexes) created with dynamic SQL statements. (For example, the FOCUS CREATE FILE command issues dynamic SQL statements.) The CURRENT SQLID is also the sole authorization ID for GRANT and REVOKE statements. It must be assigned all the privileges needed to create objects as well as GRANT and REVOKE privileges. If you do not issue the SET OWNERID command, the CURRENT SQLID is assumed to be the owner for unqualified table names.

Other types of requests, such as FOCUS TABLE (SQL SELECT) and MODIFY (SQL SELECT, INSERT, UPDATE, or DELETE) requests, automatically search for the necessary privileges using the combined privileges of the primary authorization ID and all of its associated secondary authorization IDs, regardless of the DB2 CURRENT SQLID setting. The CURRENT SQLID setting stays in effect until the communication thread to DB2 is disconnected, when it reverts to the primary authorization ID.

ERRORTYPE

With SET ERRORTYPE, you can instruct the Adapter for DB2 to return native DB2 error messages, as well as FOCUS error messages, for those error conditions that are reported by the DBMS. This feature can be enabled both as an installation option and as a run-time SET parameter.

You can override the installation default setting at run time. The syntax is

```
{ENGINE|SQL} [DB2] SET ERRORTYPE {FOCUS|DBMS}
```

where:

FOCUS

Generates only FOCUS error messages.

DBMS

Produces native DBMS error messages as well as FOCUS error messages.

Note:

- Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.
- The SET option overrides the installation default.

ISOLATION (DB2)

Each RDBMS protects data being read by one user from changes (INSERT, UPDATE, or DELETE) made by others. The isolation level setting governs the duration of the protection. That is, the isolation level determines when shared locks on rows or data pages are released, so that those rows or pages become available for updates by other users. You can dynamically set the isolation level within the FOCUS session using the SET ISOLATION command for DB2.

Note: For IDMS/SQL, see *TRANSACTION* on page 336.

You can change the isolation level by issuing this command in a MODIFY procedure or at the FOCUS command level. (For Maintain, you must issue the command at the FOCUS command level prior to invoking the Maintain procedure.) The setting remains in effect for the FOCUS session or until you reset it.

Syntax: How to Dynamically Change the Isolation Level

From the FOCUS command level, issue

```
{ENGINE|SQL} [DB2] SET ISOLATION level
```

where:

level

CS is Cursor Stability, the default. Releases shared locks as the cursor moves on in the table. Use for read-only requests.

RR is Repeatable Read. Use for MODIFY and Maintain read/write routines. Locks the retrieved data until it is released by an SQL COMMIT WORK or SQL ROLLBACK WORK.

UR is Uncommitted Read. It provides read-only access to records even if they are locked. However, these records may not yet be committed to the database.

RS is Read Stability. For more information, see the *DB2 Command and Utility Reference*.

blank resets the level to the adapter default.

Note:

- The adapter does not validate the isolation level values. If you issue the SET ISOLATION command with a level not supported for the version of the RDBMS you are using, the RDBMS will return an SQL code of -104, signifying an SQL syntax error.
- The SET ISOLATION command is enabled only for SELECT requests created as a result of FOCUS TABLE requests.
- This setting cannot override the required isolation level of RR for MODIFY and Maintain requests.

To display the isolation level setting, issue the SQL ? query command.

PLAN

In addition to the DB2 subsystem-ID, you must identify your DB2 application plan before you execute any requests. The plan is a result of the data adapter installation process or, possibly, the result of compiling a procedure that uses static SQL. See *Static SQL (DB2)* on page 401, for a discussion of static SQL procedures.

In a CAF environment, issue the SET PLAN command from the FOCUS command level

```
SQL [DB2] SET PLAN planname
```

where:

planname

Is the name of your application plan.

Note:

- Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.
- The default isolation level for the DB2 Data Adapter is CS. At installation time, your site may have chosen RR as the isolation level. Your DB2 database administrator can provide you with the isolation level for a given data adapter application plan. Refer to *ISOLATION (DB2)* on page 330 for more information.

  The isolation level called Uncommitted Read (UR) provides read-only access to records even if they are locked; however, these records may not yet be committed to the database. For more information, see the DB2 Command and Utility Reference.

- Sometimes, a site binds two different plans during the installation process; each specifies a different isolation level to control RDBMS locking. You can use the SET PLAN command to, in effect, change your Isolation Level within a FOCUS session. A level of repeatable read (RR) is required for read/write MODIFY and Maintain applications including those that invoke the Change Verify Protocol (CVP); cursor stability (CS) is recommended for TABLE and read-only MODIFY or Maintain operations. Refer to *ISOLATION (DB2)* on page 330 for information.

  This technique does not apply to non-CAF versions of the data adapter; they require a different CLIST or JCL procedure to invoke different versions of the data adapter.

You can reset the plan name at any time, regardless of the AUTOCLOSE setting. If you change the plan name, the next native SQL command or FOCUS request uses the new plan by closing and re-opening the thread.

The data adapter may have been installed with a default, site-specific, PLAN setting. Use the SQL ? query command to display this setting.

SSID

If the CAF option for the adapter is installed at your site, you must indicate which DB2 system you intend to use. The name for the DB2 system may differ from the default, or your site may have multiple copies of DB2. To specify the DB2 subsystem-ID (SSID), issue the SET SSID command before executing native SQL commands or FOCUS requests.

Issue the following from the FOCUS command level or include it in a PROFILE FOCEXEC

```
SQL [DB2] SET SSID ssid
```

where:

ssid

Is the DB2 subsystem ID. The default is DSN, unless your site changed the default at installation time.

Note:

- Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.
- Non-CAF versions of the adapter usually supply the subsystem-ID by specifying the DSN SYSTEM parameter and the RUN subcommand in the TSO CLIST that invokes FOCUS.

You can reset the SSID at any time, regardless of the AUTOCLOSE setting. If you change the SSID setting, the next native SQL command or FOCUS request accesses the new DB2 subsystem.

The adapter may have been installed with a default, site-specific, SSID setting. Use the SQL ? query command to display this setting.

NOCOLUMNTITLE

You can use the SET NOCOLUMNTITLE command to control the column names in a report when executing a stored procedure.

Syntax: How to Control Column Names

```
ENGINE DB2 SET NOCOLUMNTITLE {ON|OFF}
```

where:

`DB2`

Indicates the adapter. You can omit this parameter value if you previously issued the SET SQLENGINE command.

`ON`

Uses generated column names (for example, E01, E02, and so on) instead of the column names returned by DB2.

`OFF`

Uses the column names returned by DB2. OFF is the default value.

Parameters That Apply to Teradata Only

This section describes the Teradata SET CONNECTION_ATTRIBUTES and SET TRANSACTION commands. The SET AUTOCLOSE command is also supported and is described in *Controlling Connection Scope* on page 295.

Teradata CONNECTION_ATTRIBUTES

The adapter SET CONNECTION_ATTRIBUTES command enables users to identify themselves to the Teradata RDBMS. When users issue the command at the beginning of FOCUS sessions, the RDBMS verifies their authorization to access tables and views.

Note: DBCLOGON is a synonym for CONNECTION_ATTRIBUTES, supported for compatibility with earlier releases of the adapter.

The SET CONNECTION_ATTRIBUTES command must be issued before any DBC/SQL commands or FOCUS requests that require Teradata services. Internally, the adapter stores the Teradata logon ID and password in the user's virtual storage space and the DBC authorization process is deferred until a subsequent command for Teradata services is executed. The adapter initiates the logon immediately prior to executing the user's request for Teradata services.

The SET CONNECTION_ATTRIBUTES command may be issued from the FOCUS command level or be included in the users' PROFILE FOCEXEC. The syntax is

```
SQL [SQLDBC] SET CONNECTION_ATTRIBUTES [tdpid/]tuserid,tpassword [;]
```

where:

tdpid

Is the Teradata Director Program ID (TDP ID). Valid values are site-specific and release-dependent:

Note: Specifying a TDP ID may not be necessary at your site. Contact your database administrator for your site's requirements.

tuserid

Is your Teradata user ID, up to 30 characters long.

tpassword

Is the associated Teradata password, up to 30 characters long.

If the SET CONNECTION_ATTRIBUTES command is rejected, a Teradata error message appears and the user should resubmit the command with correct information.

For additional security, a Dialogue Manager procedure may be designed as an alternative to the PROFILE FOCEXEC method. Using -CRTFORM or -PROMPT statements, adapter users are prompted for the Teradata password and the value is stored in a variable. The -CRTFORM statement also provides a non-display option. For information about Dialogue Manager, see your documentation on developing applications.

TRANSACTION

You can control the transaction mode of a Teradata connection by issuing the SET TRANSACTION command.

Syntax: How to Set the Transaction Mode of a Teradata Connection

To set the transaction mode of a Teradata connection, issue the following command in any profile or in a procedure

```
SQL SQLDBC SET TRANSACTION [ANSI|BTET|DEFAULT]
```

where:

ANSI

Sets the Teradata connection to the ANSI transaction mode. This is the default for a CLI connection.

BTET

Sets the Teradata connection to the Teradata transaction mode (also known as the BTET transaction mode).

DEFAULT

For ODBC connections only. Sets the Teradata connection to the Teradata system default transaction mode. This is the default for an ODBC connection.

For more information about how the Teradata system default is determined, see your Teradata documentation.

Note: The connection scope is different when setting the transaction mode for ODBC and CLI connections.

- **For an ODBC Connection:** The transaction mode can be set within an active connection. There is no need to disconnect.
- **For a CLI Connection:** The transaction mode can be set only before the connection takes place. You must terminate the connection first using the SQL SQLDBC END SESSION command, if you want to change the transaction mode for a CLI connection. Failure to do so will return the following warning:

```
(FOC1722) WARNING: COULD NOT SET TRANSACTION MODE
```

Parameters That Apply to IDMS/SQL Only

This section describes the SQLIDMS SET SESSION CURRENT SCHEMA and SET TRANSACTION commands, and IDMS/SQL session control.

CURRENT SCHEMA

The Adapter for IDMS/SQL uses the user-specified schema name as the first qualifier for all SQL requests involving SQL tables or views. This command overrides the IDMS/SQL current schema in effect and precludes the specification of unqualified table names. This prevents passing unqualified table names to IDMS.

To identify the schema, issue the SET SESSION CURRENT SCHEMA command from the FOCUS command level

```
SQL [SQLIDMS] SET SESSION CURRENT SCHEMA schema
```

where:

schema

Is the name of the schema in SQL requests.

Note: Omit the SQLIDMS target RDBMS qualifier if you previously issued the SET SQLENGINE command for SQLIDMS. Use the SQL ? query command to display this setting.

TRANSACTION

IDMS protects data being read by one user from changes (INSERT, UPDATE, or DELETE) made by others. The isolation level setting governs the duration of the protection. That is, the isolation level determines when shared locks on rows are released, so that those rows or pages become available for updates by other users. IDMS/SQL allows you to dynamically set the isolation level within the FOCUS session using the IDMS SQL SET TRANSACTION command.

Note: For the equivalent command for DB2, see *ISOLATION (DB2)* on page 330.

The SET TRANSACTION CURSOR STABILITY or TRANSIENT READ command affects the duration of row or page shared locks on IDMS/SQL tables for the duration of the IDMS/SQL transaction. You can specify the command in your MODIFY procedure or from the FOCUS command level. The setting remains in effect for the FOCUS session or until you reset it.

To set the isolation level from the FOCUS command level, issue

```
SQL [SQLIDMS] SET TRANSACTION level
```

where:

level

Can be one of the following:

`CURSOR STABILITY` is Cursor Stability, the default. Provides the maximum amount of concurrency while guaranteeing the integrity of the data selected.

`TRANSIENT READ` is Transient Read. Allows the reading of records locked by other users (allows dirty reads). Recommended for TABLE only. Transient read prevents the SQL transaction from performing updates. Use this only when you do not need the data retrieved to be absolutely consistent and accurate. If you specify Transient Read, IDMS assumes Read Only.

`READ ONLY` allows data to be retrieved, but does not allow the database to be updated.

`READ WRITE` allows data to be retrieved, and allows the database to be updated.

Note: Omit the SQLIDMS target RDBMS qualifier from the command when issuing it in a MODIFY procedure or if you previously issued the SET SQLENGINE command for SQLIDMS.

IDMS SQL Session Control: The CONNECT Command

An SQL session is a connection between the FOCUS application and the IDMS database. It begins when the application connects to a dictionary. You use the CONNECT command to override the IDMS/SQL default (automatic) connection. The length of time an SQL session stays in effect depends on whether the connection began automatically or a CONNECT command was issued. If the CONNECT command was issued, the SQL session is in effect until a COMMIT RELEASE, ROLLBACK RELEASE, or RELEASE command is executed. All of these commands may be executed within the FOCUS IDMS/SQL session using Direct SQL Passthru. Refer to the appropriate CA-IDMS/DB documentation for more information regarding SQL sessions.

Issue the following from the FOCUS command level or include it in a PROFILE FOCEXEC

```
SQL [SQLIDMS] CONNECT TO dictname
```

where:

dictname

Is the database (dictionary) to start the IDMS SQL session. The default is the dictionary in effect for the user session. This default is set outside of the FOCUS session, for example, with a SYSIDMS DICTNAME parameter. Please refer to the appropriate CA-IDMS/DB documentation for a complete description.

Note: Omit the SQLIDMS target RDBMS qualifier if you previously issued the SET SQLENGINE command for SQLIDMS.

IDMS SQL Session Control: Other Session Commands

Other IDMS SQL commands that affect the IDMS/SQL session can be executed explicitly.

To issue IDMS SQL session commands such as COMMIT, COMMIT RELEASE, ROLLBACK, ROLLBACK RELEASE, and COMMIT CONTINUE, the syntax is:

```
SQL [SQLIDMS] COMMIT RELEASE
```

Parameters That Apply to Oracle Only

This section describes the Adapter for Oracle SET CONNECTION_ATTRIBUTES, SET DATETIME_PROCESS, SET DEFAULT_CONNECTION, SET ORACHAR, SET ORANUMBER, and SET SPMAZPRM commands.

Oracle CONNECTION_ATTRIBUTES

The SET CONNECTION_ATTRIBUTES command allows you to declare a connection to one Oracle database server and to supply authentication attributes necessary to connect to the server.

You can connect to more than one Oracle database server by issuing multiple SET CONNECTION_ATTRIBUTES commands. The actual connection takes place when the first request referencing that connection is issued. You can issue SET CONNECTION_ATTRIBUTES commands in a FOCEXEC, at the FOCUS command prompt or in a FOCUS-supported profile. The profile can be encrypted.

If you issue multiple SET CONNECTION_ATTRIBUTES commands:

- The first SET CONNECTION_ATTRIBUTES command sets the default Oracle database server to be used.
- If more than one SET CONNECTION_ATTRIBUTES command declares the same Oracle database server, the authentication information is taken from the last SET CONNECTION_ATTRIBUTES command.

The adapter supports connections to:

- Local Oracle database servers.
- Remote Oracle database servers. To connect to a remote Oracle database server, the Oracle tnsnames file on the source machine must contain an entry pointing to the target machine and the listening process must be running on the target machine.

Once you are connected to an Oracle database server, that server may define Oracle DATABASE LINKs that can be used to access Oracle tables on other Oracle database servers.

If needed, the DBA or some other authorized person at your site will supply you with a valid Oracle user ID and password.

It may be desirable to prompt users for their Oracle password instead of coding it in a procedure. In this case, use a Dialogue Manager variable in its place, and retrieve the value using -CRTFORM or -PROMPT facilities. If you use -CRTFORM, you can make the password field non-displayable for additional security.

A valid Oracle user ID and password must be supplied before issuing commands that access the Oracle RDBMS. If a valid Oracle login ID and password have not been supplied, an error message is returned. You should respond by correcting and re-issuing the SET CONNECTION_ATTRIBUTES command.

The SET CONNECTION_ATTRIBUTES command stores both the Oracle login ID and password in user virtual storage. It does not immediately initiate a login to Oracle. The actual login is deferred until a subsequent command is issued that requires the services of the Oracle RDBMS.

The syntax is:

```
SQL [SQLORA] SET CONNECTION_ATTRIBUTES [connection_name]/userid,password
```

where:

SQLORA

Indicates the Adapter for Oracle. You can omit this value if you previously issued the SET SQLENGINE command.

connection_name

Specifies a remote instance using an Oracle TNSNAME (the net service name used as a connect descriptor to an Oracle database server across the network). If omitted, the local database server will be set as the default.

userid

Is the primary authorization ID by which you are known to Oracle, up to 30 characters in length.

password

Is the password associated with the user ID, up to 30 characters in length.

Note: SET USER is a synonym for SET CONNECTION_ATTRIBUTES, supported for compatibility with earlier releases of the adapter. However, note that the symbol used for separating the connection attribute from the authentication information and the symbol used for separating the user ID from the password changed in FOCUS 7.2.

Issue the following query command to list status information for all declared connections:

```
SQL SQLORA ? SERVERS
```

DATETIME_PROCESS

When an SQL SELECT statement refers to CURRENT_DATE, CURRENT_TIME, or CURRENT_TIMESTAMP, you can choose a source for obtaining the system date and time. By default, the date or time value will be obtained from the system running FOCUS. Using the SET DATETIME_PROCESS command, you can override this default so that the date or time is obtained from the Oracle DBMS for reports using Automatic Passthru.

Syntax: How to Choose a Source for System Date and Time

To choose a source for system date and time, the syntax is

```
SQL SET DATETIME_PROCESS=SERVER|DBMS
END
```

where:

SERVER

Specifies that the date and time values will be taken from the system running FOCUS. This is the default.

DBMS

Specifies that the date and time values will be taken from the Oracle DBMS.

DEFAULT_CONNECTION

Once all Oracle database servers to be accessed have been declared using the SET CONNECTION_ATTRIBUTES command, you can select a default server using the SET DEFAULT_CONNECTION command. If you do not issue this command, the tnsname value specified in the *first* SET CONNECTION_ATTRIBUTES command is used.

```
SQL [SQLORA] SET DEFAULT_CONNECTION [connection_name]
```

where:

SQLORA

Indicates the Adapter for Oracle. You can omit this value if you previously issued the SET SQLENGINE command.

connection_name

Is the net service name used as a connect descriptor to an Oracle database server across the network. If omitted, then the local database server will be set as the default. If this name has not been previously declared in a SET CONNECTION_ATTRIBUTES command, message FOC1671 is issued.

Note:

- If you use the SQL SQLORA SET DEFAULT_CONNECTION command more than once, the connection name specified in the last command will be the active connection name.
- The SQL SQLORA SET DEFAULT_CONNECTION command cannot be issued while an uncommitted transaction (LUW) is pending. In that case message FOC1671 is issued.

ORACHAR

With regard to CHAR and VARCHAR2 data types, special attention must be paid to which of the two will be used. When you compare a column of CHAR data type to a column of VARCHAR2 data type, where the only difference in data is the additional space in the column of the CHAR data type, Oracle recognizes that the data types are not the same.

The ORACHAR setting lets you specify which of the two data types will be used for inserting, updating, and retrieving data.

If you create the tables outside of FOCUS, we recommend that you use either CHAR or VARCHAR2 data types, but not both. If you create a table with both data types, you might not be able to retrieve the data you inserted due to Oracle's comparison mechanism.

If you use FOCUS to generate Oracle tables and retrieve data, you will not encounter this problem, since the data type being used will be either CHAR or VARCHAR2, depending upon the ORACHAR setting.

```
SQL [SQLORA] SET ORACHAR {FIX|VAR}
```

where:

`FIX`

Uses the CHAR data type.

`VAR`

Uses the VARCHAR2 data type. This is the default.

Note: Omit the SQLORA target RDBMS qualifier if you issued the SET SQLENGINE command.

Prior to Oracle V7, the Oracle RDBMS treated all character strings transmitted by the adapter through SQL variables as variable length because all Oracle RDBMS character data types were essentially variable length. In Oracle V7, the CHAR data type was enhanced to possess fixed length characteristics similar to other relational database CHAR data types.

Important: The FIX setting may cause compatibility problems in applications developed in earlier releases of FOCUS or Oracle or under the default setting of VAR. Use of the FIX setting with Oracle V7 CHAR columns in MODIFY MATCH, INCLUDE, UPDATE and DELETE commands, as variables in parameterized Direct SQL Passthru statements, and as cross-referenced fields in non-optimized join requests should be tested extensively to verify that their behavior is as expected.

ORANUMBER

The ORANUMBER setting determines how the Oracle NUMBER data type will be described in the Master File generated for the answer set returned by a Direct SQL Passthru SELECT request.

By default a NUMBER data type whose precision is:

- Between 32 and 37 is mapped to the format D20.2.
- 38 is mapped to the format I11.

You can use the ORANUMBER setting to override the default mapping of a NUMBER data type with precision between 32 and 38.

Syntax: How to Override the Precision of the Oracle NUMBER Data Type

```
SQL [SQLORA] SET ORANUMBER {COMPAT|DECIMAL}
```

where:

`SQLORA`

Indicates the Adapter for Oracle. You can omit this value if you previously issued the SET SQLENGINE command.

`COMPAT`

Indicates that the NUMBER data type with precision between 32 and 37 will be mapped to format D20.2.This is the default.

`DECIMAL`

Indicates that the NUMBER data type with precision between 32 and 37 will be mapped to format P33.2.

SPMAXPRM

An Adapter for Oracle parameter can be used to set the maximum number of input parameters for stored procedures.

```
SQL [SQLORA] SET SPMAXPRM value
```

where:

value

Is a numeric value indicating the maximum number of input parameters that may be entered for stored procedures. The default value is 256. This value is displayed by the SQL SQLORA ? query.

Note: Omit the SQLORA target RDBMS qualifier if you issued the SET SQLENGINE command.

Parameters That Apply to MODIFY Only

The following commands are available only within FOCUS MODIFY procedures, SET LOADONLY, SET AUTOCOMMIT ON CRTFORM, and SET ERRORRUN. The SET ISOLATION command, explained in *ISOLATION (DB2)* on page 330, may also be issued from MODIFY. See *Maintaining Tables With FOCUS* on page 349, for a discussion of MODIFY.

When issuing commands from within MODIFY procedures, omit the *sqlengine* or the environmental prefix.

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 ON
```

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 adapter uses an SQL SELECT statement to test the existence of a single row. By examining the SQL return code, the adapter 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.

Note:

- For Oracle, choose the appropriate INSERTSIZE to use in combination with LOADONLY.
- The FOCUS Fastload Facility should not be confused with the Teradata Fastload feature.

Note: 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 the buffer is not full. This restriction has several ramifications:

- To use INSERTSIZE in a MODIFY request, you must use the SQL SET LOADONLY command described in *Maintaining Tables With FOCUS* on page 349. Without LOADONLY, the adapter does not insert a row without first issuing a SELECT statement to check that the row does not already exist.
- To use INSERTSIZE with Direct SQL Passthru, the INSERT statement must be parameterized and use the PREPARE, BIND, and EXECUTE command set within a BEGIN SESSION/END SESSION command pair. In this case, the BIND is not optional. See *Direct SQL Passthru* on page 265, for a discussion of parameterized Passthru.

AUTOCOMMIT ON CRTFORM

The AUTOCOMMIT ON CRTFORM facility releases locks on database objects every time a CRTFORM is displayed. The integrity of data is protected by an automated Change Verify Protocol (CVP) as an alternative to the RDBMS standard method of locking for concurrency and integrity. The Change Verify Protocol consists of a series of steps that manage the integrity and concurrent update/retrieval activity of the database by committing open transactions prior to displaying each CRTFORM.

Note: This setting is not needed for Teradata because it implicitly commits every SQL statement individually.

AUTOCOMMIT ON CRTFORM does not apply to Maintain.

CVP lacks the unit-of-work capabilities of the RDBMS protocol, but CVP never retains locks on displayed records. The elimination of locks increases rates of transaction operations and improves terminal response times for interactive applications. All open transactions are automatically committed prior to using CVP for SELECT and UPDATE.

The Change Verify Protocol initiates these steps for each CRTFORM display:

1. The adapter releases all locks held on the record. Before the record displays on the terminal, the adapter issues an SQL COMMIT WORK statement to release any existing RDBMS locks.
2. The adapter retrieves the record and displays it on the terminal. The user enters a database update that results in an UPDATE, DELETE, or INCLUDE action.

3. The adapter retrieves the record again. It compares the original and current values of the record to determine if another transaction changed the record in the interim. If the original (displayed) record has been modified in the time between the original retrieval and the update request, the update is rejected. If the original (displayed) record remains unchanged, the update is processed.

With the RDBMS standard method, SQL COMMIT WORK statements specified in applications define transactions or logical units of work (LUW). The RDBMS uses a locking mechanism to protect the LUW from interference by other concurrent transactions. The locking mechanism allocates and isolates database resources for the LUW.

The disadvantage to the RDBMS method is that terminal I/O in the LUW locks data for the indeterminate amount of time it takes the user to react to the terminal display. The locks are retained until the user completes the transactions successfully or until the RDBMS issues a ROLLBACK WORK command.

You can issue the AUTOCOMMIT ON CRTFORM command only from within a MODIFY CRTFORM procedure. Include it in CASE AT START, not in the TOP case. You may not switch AUTOCOMMIT modes within the MODIFY procedure.

The AUTOCOMMIT ON CRTFORM 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, CVP applies only to the last record.

Syntax: How to Invoke the Change Verify Protocol

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.

Note:

- A MODIFY procedure that invokes the Change Verify Protocol can test the value of the FOCURRENT variable to determine whether or not there is a conflict with another transaction. See *Maintaining Tables With FOCUS* on page 349 for more information.

- To avoid data inconsistencies with the AUTOCOMMIT feature, you must set your DB2 isolation level to Repeatable Read (RR), the required isolation level for all read/write MODIFY applications. For IDMS/SQL, you must set the isolation level to Cursor Stability. Oracle requires a Write lock.

ERRORRUN

With SET ERRORRUN ON, MODIFY processing continues even when a serious error occurs, allowing an application to handle its own errors in the event that an RDBMS error is part of the normal application flow. Code this command explicitly within the MODIFY procedure, preferably in CASE AT START, where it is executed once.

The syntax is

```
CASE AT START
SQL SET ERRORRUN {OFF|ON}
.
.
.
ENDCASE
```

where:

OFF

Causes MODIFY processing to stop when a fatal error is detected (for example, if the table name is not found). OFF is the default value.

ON

Allows MODIFY processing to continue despite fatal errors. Test the value of FOCERROR to determine appropriate 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 negative SQLCODE, non-zero return code, or RDBMS error message can cause unpredictable errors in subsequent RDBMS or MODIFY processing.

SET ERRORRUN returns to its default setting at the end of the MODIFY procedure.

Adapter Dialogue Manager Variables

All adapter environmental settings are available for display and query as Dialogue Manager variables. The following sections list the variable names for each RDBMS, and the lines displaying their settings in the SQL ? query.

Note: If the length of the current setting value is greater than 12, the Dialogue Manager variable makes only the first 12 characters available.

Dialogue Manager Variables for the Adapter for DB2

The following variables are available for DB2:

```
&SQLCAF       - (FOC1442) CALL ATTACH FACILITY IS
&SQLSSID      - (FOC1447) SSID FOR CALL ATTACH IS
&SQLPLANA     - (FOC1448) ACTIVE PLAN FOR CALL ATTACH IS
&SQLPLANU     - (FOC1459) USER SET PLAN FOR CALL ATTACH IS
&SQLPLANI     - (FOC1460) INSTALLATION DEFAULT PLAN IS
&SQLSTATIC    - (FOC1503) SQL STATIC OPTION IS
&SQLAUTOCLS   - (FOC1444) AUTOCLOSE OPTION IS
&SQLAUTODIS   - (FOC1496) AUTODISCONNECT OPTION IS
&SQLAUTOCOM   - (FOC1499) AUTOCOMMIT OPTION IS
&SQLDBSPACE   - (FOC1446) DEFAULT DBSPACE IS
&SQLID        - (FOC1449) CURRENT SQLID IS
&SQLISOLATION - (FOC1424) ISOLATION LEVEL FOR DB2 TABLE INTERFACE IS
&SQLWRITE     - (FOC1441) WRITE FUNCTIONALITY IS
&SQLOPT       - (FOC1445) OPTIMIZATION OPTION IS
&SQLITEOPT    - (FOC1763) IF-THEN-ELSE OPTIMIZATION IS
&SQLEMSG      - (FOC1484) SQL ERROR MESSAGE TYPE IS
&SQLEXPL      - (FOC1497) SQL EXPLAIN OPTION IS
&SQLQUERYNO   - (FOC1551) QUERY NUMBER TO BE USED FOR EXPLAIN
&SQLDEFDATE   - (FOC1552) INTERFACE DEFAULT DATE TYPE
&SQLOWNERID   - (FOC1520) SQL CURRENT OWNER ID IS

&SQLRELEASE   - Current release of DB2 in format Dxxx where
                xxx is release number (for example, 910 for
                DB2 Version 9 Release 1).

&SQLVVRRM     - Return from CONNECT in the form DSNvvrrM.
                vv is the Version of DB2, rr is the Release, M is the
                Modification level. For more information, consult IBM's
                DB2 SQL Reference.
```

Dialogue Manager Variables for the Adapter for Teradata

The following variables are available for Teradata:

```
&DBCCON       - (FOC1463) DBC/1012 CONNECTION
&DBCPART      - (FOC1467) PARTITION
&DBCTDP       - (FOC1468) TERADATA DIRECTOR PROGRAM (TDP)
&DBCUSER      - (FOC1469) DBC/1012 USER ID
&DBCAUTOCLS   - (FOC1444) AUTOCLOSE OPTION IS
&DBCWRITE     - (FOC1441) WRITE FUNCTIONALITY IS
&DBCOPT       - (FOC1445) OPTIMIZATION OPTION IS
&DBCITEOPT    - (FOC1763) IF-THEN-ELSE OPTIMIZATION IS
&DBCEMSG      - (FOC1484) SQL ERROR MESSAGE TYPE IS
&DBCEXPL      - (FOC1497) SQL EXPLAIN OPTION IS
&DBCDEFDATE   - (FOC1552) INTERFACE DEFAULT DATE TYPE
&DBCOWNERID   - (FOC1520) DBC CURRENT OWNER ID IS
```

Dialogue Manager Variables for the Adapter for IDMS/SQL

The following variables are available for IDMS/SQL:

```
&IDQDICTNAME  - (FOC1756) CURRENT IDMS DICTIONARY IS
&IDQCURSCHEMA - (FOC1757) CURRENT SCHEMA IS
&IDQISLEVEL   - (FOC1758) CURRENT ISOLATION LEVEL IS
&IDQAUTOCOM   - (FOC1499) AUTOCOMMIT OPTION IS
&IDQDBSPACE   - (FOC1446) DEFAULT DBSPACE IS
&IDQOPT       - (FOC1445) OPTIMIZATION OPTION IS
&IDQITEOPT    - (FOC1763) IF-THEN-ELSE OPTIMIZATION IS
&IDQDEFDATE   - (FOC1552) INTERFACE DEFAULT DATE TYPE
```

Dialogue Manager Variables for the Adapter for Oracle

The following variables are available for Oracle:

```
&ORASERVER    - (FOC1656) DEFAULT SERVER NAME
&ORAUSER      - (FOC1502) USERID AND PASSWORD ARE
&ORAAUTODIS   - (FOC1496) AUTODISCONNECT OPTION IS
&ORAAUTOCOM   - (FOC1499) AUTOCOMMIT OPTION IS
&ORADBSPACE   - (FOC1446) DEFAULT DBSPACE IS
&ORAFETCHSIZE - (FOC1491) FETCH BUFFERING FACTOR
&ORAINSERTSIZ - (FOC1531) INSERT BUFFERING FACTOR
&ORASPMAXPRM  - (FOC1379) MAXIMUM STORED PROCEDURE PARAMETERS
&ORAWRITE     - (FOC1441) WRITE FUNCTIONALITY IS
&ORAOPT       - (FOC1445) OPTIMIZATION OPTION IS
&ORAITEOPT    - (FOC1763) IF-THEN-ELSE OPTIMIZATION IS
&ORAEMSG      - (FOC1484) SQL ERROR MESSAGE TYPE IS
&ORADEFDATE   - (FOC1552) INTERFACE DEFAULT DATE TYPE
&ORAOWNERID   - (FOC1520) ORACLE CURRENT OWNER ID IS
&ORADBSPACE   - (FOC1446) DEFAULT DBSPACE IS
&ORAMSGTXT - MESSAGE RETURNED BY AN ORACLE STORED PROCEDURE
```

Chapter 12

Maintaining Tables With FOCUS

This chapter describes how to use the adapter to maintain RDBMS tables. In particular, it identifies 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 Master File, or unrelated sets of tables.

In this chapter:

Overview of Data Source Maintenance Facilities

This chapter describes differences between the MODIFY and Maintain facilities when these differences affect adapter processing.

Your FOCUS documentation contains a detailed discussion of file maintenance with the MODIFY and Maintain facilities. Read the MODIFY and Maintain information carefully before developing 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 Master Files. However, each Master File 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.

Maintaining IDMS network tables is not supported. The MODIFY facility supports true IDMS/SQL tables only.

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, which are 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 PF key). Maintain also provides set-based processing through enhanced NEXT, UPDATE, DELETE, and INCLUDE commands.

Prerequisites for running MODIFY and Maintain requests include:

- The Write Adapter. The Write component of the adapter must be installed and operational.
- Proper RDBMS authorization to perform maintenance operations.

  You can update some RDBMS views in accordance with data source 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 attribute value of YES in the Access File for the table (for INCLUDE, UPDATE, and DELETE operations). For information about this attribute, see *Describing Tables to FOCUS* on page 55.
- Existing tables to modify. If you intend to load new tables with MODIFY or Maintain, you must create them first. Do so with the FOCUS CREATE FILE command, discussed in *Automated Procedures* on page 113, or with the SQL CREATE TABLE command through the Direct SQL Passthru facility, discussed in *Direct SQL Passthru* on page 265.

  You can maintain DB2 views as long as they are updateable, as defined in the IBM *DB2 SQL Reference*.

The examples in this chapter refer to the EMPINFO, COURSE, PAYINFO, ECOURSE, and EMPPAY Master and Access Files. *File Descriptions and Tables* on page 459 contains a complete listing of Master and Access Files.

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. Subsequent sections demonstrate how to use NEXT to retrieve the remaining rows.

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, you 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.

See your FOCUS documentation on maintaining databases for complete syntax.

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.

The Master and Access Files for a table identify its primary key. *Describing Tables to FOCUS* on page 55 explains how to describe primary key columns.

You can implement RDBMS referential integrity by defining primary and foreign keys in SQL CREATE TABLE statements (see *Referential Integrity* on page 373). 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 this chapter, the term *primary key* or *key* refers to those columns that compose the unique identifier for each row.

Index Considerations

Indexes enhance the performance of data maintenance routines, especially indexes created on a 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.

Example: Unique Index on a Primary Key

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 MODIFY and Maintain facilities, you can add new rows to a table, update column values for specific rows, or delete specific rows.

The adapter processes a MODIFY or Maintain transaction with the following steps:

1. FOCUS reads the transaction for incoming data values.
2. The adapter 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 adapter either:
   - Performs the update operation (UPDATE or DELETE) on the returned answer set. With MODIFY, the adapter 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 data source appropriately.

In MODIFY, you must use the NEXT command to process a multi-row answer set one row at a time. Each NEXT command 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 Command

In response to a MATCH command, the adapter selects the first row in the table that meets the MATCH criteria.

The MATCH command 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.

Syntax: How to Use the MATCH Command in MODIFY

The syntax of the MATCH command in MODIFY is

```
MATCH field1 [field2...fieldn]
  ON MATCH action_1
  ON NOMATCH action_2
```

where:

field1 ... fieldn

Are fields representing columns. FOCUS compares incoming data values against existing column values. The fields can be any combination of key and/or non-key fields. Specify complete fieldnames. MATCH does not support truncated names.

action_1

Is the operation to perform when the values in a row match the incoming data values.

action_2

Is the operation to perform when the existing values in a row do not match the incoming data values.

Your FOCUS documentation on maintaining databases discusses these actions in detail.

MATCH processing for multi-table Master Files is the same as for a multi-segment FOCUS data source.

In Maintain, you do not have to include an ON NOMATCH command in order to reject a transaction. Maintain automatically rejects a transaction that does not satisfy the MATCH criteria. See your FOCUS documentation on maintaining databases for MATCH syntax in Maintain.

Reference: Acceptable Actions for MATCH

Acceptable actions for MATCH commands 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.

Adapter MATCH Behavior

In MODIFY requests, there are two major differences in the way MATCH commands function for the adapter and for native FOCUS:

- With the adapter, you can change the value of a primary key for a table (subject to RDBMS limitations) using the UPDATE statement. When modifying a FOCUS data source, 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 adapter 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.

Note: In Maintain, MATCH functions identically for the adapter and for native FOCUS.

Example: Using the MODIFY MATCH Command

Consider a MODIFY request that maintains the EMPINFO table. It prompts for an employee ID and for a new salary. Then it processes the incoming data. The annotated request contains the following MATCH commands:

```
    MODIFY FILE EMPINFO
    PROMPT EMP_ID CURRENT_SALARY
1.  MATCH EMP_ID
2.  ON MATCH UPDATE CURRENT_SALARY
3.  ON NOMATCH REJECT
    DATA
```

The incoming transaction contains the following values:

```
EMP_ID = 123456789
CURRENT_SALARY = 20000
```

The request processes as follows:

1. The MATCH command compares the value of the incoming EMP_ID, 123456789, to the EMP_ID values in the rows of the EMPINFO table. Since EMP_ID is the primary key of this table, the RDBMS can return at most one row as a result of this MATCH.
2. If a row exists for EMP_ID 123456789, the MATCH command updates the CURRENT_SALARY value of that row with the incoming value 20000.
3. If no row exists for EMP_ID 123456789, the MATCH command rejects the transaction.

The NEXT Command

In MODIFY, the NEXT command provides a flexible means of processing multi-row answer sets by moving the current position in the answer set from one row to the next.

Syntax: How to Use the NEXT Command in MODIFY

```
NEXT field
  ON NEXT action_1
  ON NONEXT action_2
```

where:

field

Is any field in the table. It does not have to be a primary key.

action_1

Is the operation to perform when there is a subsequent row in the answer set. May be any of the acceptable actions listed for MATCH in *The MATCH Command* on page 353.

action_2

Is the operation to perform when no more rows exist in the answer set.

Reference: Usage Notes for NEXT in MODIFY

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 Tables to FOCUS* on page 55 explains how to specify the KEYORDER parameter. The default is to sort by primary key in ascending order.

Your choice of MATCH and NEXT command combinations determines the contents of the answer set. Subsequent sections explain these choices in more detail:

- **NEXT command without a MATCH command.** The adapter 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 commands.
- **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.

You can also use NEXT commands with multi-table structures (FOCUS views) to modify or display data in either Case Logic or non-Case Logic requests. 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:

- Without Case Logic, you can retrieve all parent rows in the table and only the first descendant row of any specified related table.
- With Case Logic, you can retrieve all rows for each table defined in a multi-table Master File. To do so, first MATCH on the parent. Then, in another case, use NEXT to loop through the related tables (at the lowest level) until there are no more related instances. On NONEXT, return to the parent case for the next parent instance.

  You can trace Case Logic with the FOCUS TRACE facility. To invoke the TRACE facility for the adapter, include the TRACE command on a separate line after the MODIFY FILE command. For complete information about the FOCUS TRACE facility, see your FOCUS documentation. You can also use the adapter trace facilities described in *Tracing Adapter Processing* on page 487.

The following sections illustrate different combinations of MATCH and NEXT commands with annotated examples. The MODIFY requests have been kept simple for purposes of illustration. You can create more sophisticated procedures. Assume KEYORDER=LOW for all of the examples.

Reference: Usage Notes for NEXT in Maintain

- The syntax of the NEXT command includes optional FOR and WHERE phrases that control the number of rows retrieved into a stack. As in MODIFY, the KEYORDER attribute in the Access File determines whether NEXT returns rows in ascending or descending order of the primary key.
- 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.

  Also, as in MODIFY, once you MATCH on a parent segment, a subsequent NEXT on a child segment retrieves descendant rows within the parent established by the MATCH. However, one NEXT command can retrieve all such child instances, without Case Logic.

- 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.

For complete details, see your FOCUS documentation on maintaining databases.

NEXT Processing Without MATCH

If you use a NEXT command without a previous MATCH command in a MODIFY request, the RDBMS returns an answer set consisting of all rows in the table sorted by the primary key. Use the ON NEXT command to view each row in ascending primary key order. In a Maintain request, the FOR and WHERE phrases in the NEXT command determine the number of rows retrieved, sorted by the primary key in KEYORDER sequence.

Example: Using NEXT Without MATCH in MODIFY

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 commands display the following on the screen:

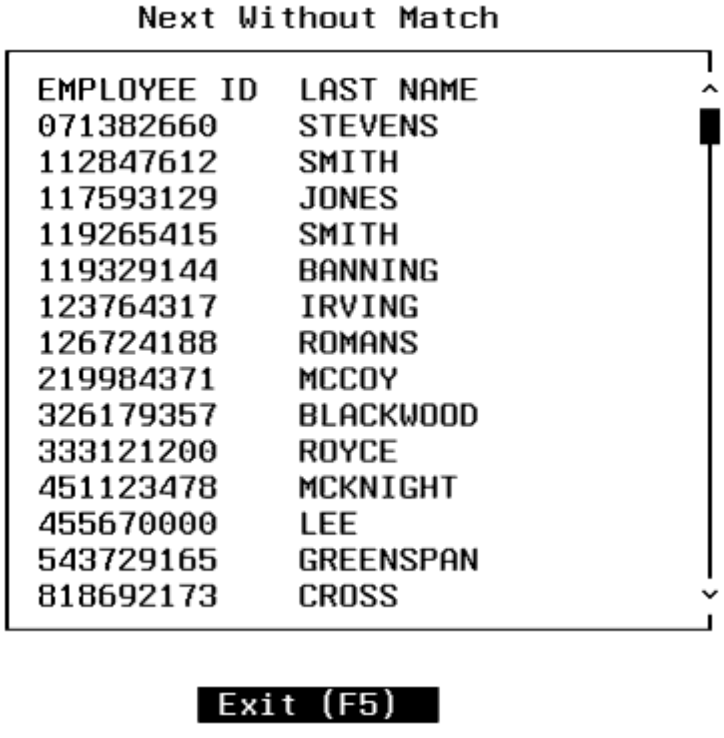
```
EMPLOYEE ID: 071382660 LAST NAME STEVENS
EMPLOYEE ID: 112847612 LAST NAME SMITH
EMPLOYEE ID: 117593129 LAST NAME JONES
EMPLOYEE ID: 119265415 LAST NAME SMITH
EMPLOYEE ID: 119329144 LAST NAME BANNING
EMPLOYEE ID: 123764317 LAST NAME IRVING
EMPLOYEE ID: 126724188 LAST NAME ROMANS
EMPLOYEE ID: 219984371 LAST NAME MCCOY
EMPLOYEE ID: 326179357 LAST NAME BLACKWOOD
EMPLOYEE ID: 333121200 LAST NAME ROYCE
EMPLOYEE ID: 451123478 LAST NAME MCKNIGHT
EMPLOYEE ID: 455670000 LAST NAME LEE
EMPLOYEE ID: 543729165 LAST NAME GREENSPAN
EMPLOYEE ID: 818692173 LAST NAME CROSS
```

Example: Using NEXT in Maintain

The following Maintain procedure named NEXT1 retrieves the same answer set into a stack named INSTACK and displays the retrieved values on a Winform named WIN1 (see your FOCUS documentation on maintaining databases for instructions on creating Winforms):

```
MAINTAIN FILE EMPINFO
FOR ALL NEXT EMP_ID INTO INSTACK
WINFORM SHOW WIN1
END
```

Executing the NEXT1 procedure displays a Winform similar to the following:



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 command causes the RDBMS to retrieve all rows following the matched row in key sequence.

Example: Using NEXT After MATCH on a Full Primary Key in MODIFY

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, 219984371.
2. The MATCH command 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 command.
4. The NEXT command retrieves the next logical row in EMP_ID sequence. 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: 219984371                                          (line 3)

EMP_ID:  326179357   LAST_NAME:  BLACKWOOD               (line 10)

EMP_ID:  333121200   LAST_NAME:  ROYCE                  (line 10)

EMP_ID:  451123478   LAST_NAME:  MCKNIGHT               (line 10)

EMP_ID:  455670000   LAST_NAME:  LEE                    (line 10)

EMP_ID:  543729165   LAST_NAME:  GREENSPAN              (line 10)

EMP_ID:  818692173   LAST_NAME:  CROSS                  (line 10)
```

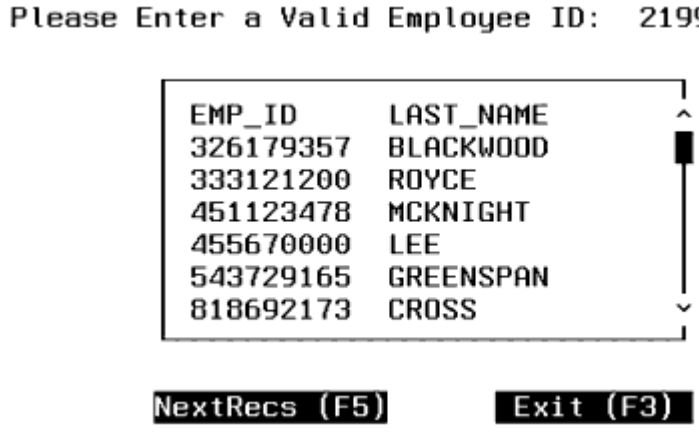
Because of the NEXT command, all employees after 219984371 display one at a time on the screen. Notice that the rows are retrieved in key sequence.

Example: Using NEXT on a Full Primary Key in Maintain

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 219984371 into a stack named TRANS and presses PF5 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
```

Executing the NEXT2 procedure displays a Winform similar to the following:



NEXT Processing After MATCH on a Non-Key Field or Partial Key

In a MODIFY request processed by the adapter, 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.

Example: Using MATCH on a Non-Key Field in MODIFY

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 command 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:  112847612   LAST_NAME:  SMITH
EMP_ID:  119265415   LAST_NAME:  SMITH
```

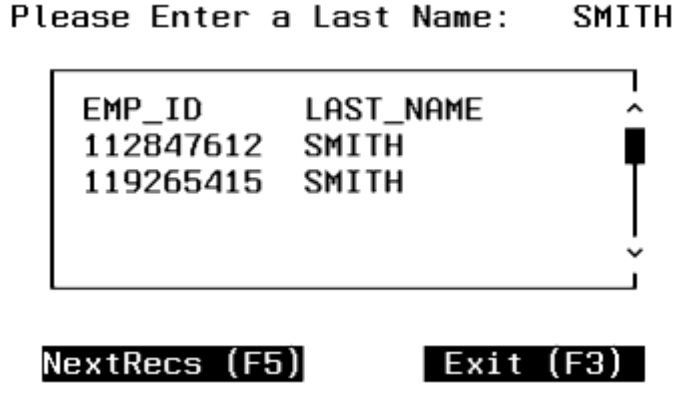
A line displays on the screen for each employee with the last name SMITH. Employee ID 112847612 is the result of the MATCH operation. Employee ID 119265415 is the result of the NEXT operation.

Example: Using NEXT on a Non-Key Field in Maintain

The following Maintain procedure retrieves the answer set into a stack named EMPSTACK. Assume that when Maintain displays the Winform named WINA, the user enters the transaction value (SMITH) into the first row of a stack named TRANS and presses PF5 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
```

Executing the NEXT3 procedure displays a Winform similar to the following:



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 commands 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 command determines the number of rows affected.

Example: Updating Rows With MODIFY

Suppose you want to display all the employees in a department and 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 command causes the RDBMS to search the table for the first row with the value PRODUCTION and return it 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.

The lines displayed by this MODIFY procedure follow:

```
 PLEASE ENTER A VALID DEPARTMENT
 DEPARTMENT:  PRODUCTION
EMP_ID:  071382660   SALARY:   11000.00
```

You can change the salary, or leave it as is. Each time you press *Enter*, the current salary is updated and the next employee ID displays:

```
EMP_ID:  119265415   SALARY:    9500.00
EMP_ID:  119329144   SALARY:   29700.00
EMP_ID:  123764317   SALARY:   26862.00
EMP_ID:  126724188   SALARY:   21120.00
EMP_ID:  451123478   SALARY:   16100.00
```

Example: Updating Rows With Maintain

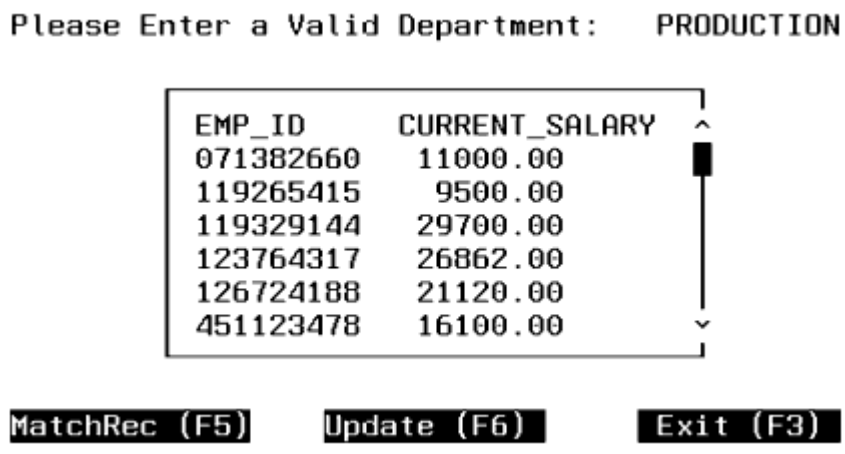
In Maintain, you can use stack columns as turnaround values to update a table. The following annotated Maintain request named UPDATE1 updates the same rows as the preceding MODIFY request:

```
    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 opens. 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. See your FOCUS documentation on maintaining databases for instructions on creating Winforms.
2. The user enters a DEPARTMENT value for the search and presses a PF key or 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 PF key or button to invoke case UPDSAL, which updates all salaries.

Executing the UPDATE1 procedure displays a Winform similar to the following:



RDBMS Transaction Control Within MODIFY

The adapter 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 within the LUW can operate on the same or separate tables.

DB2, IDMS/SQL, and Oracle define a transaction as all actions taken since the application first accessed the RDBMS, last issued a COMMIT WORK, or last issued a ROLLBACK WORK.

The Adapter for **Teradata** defines a transaction as either explicit or implicit:

- An explicit transaction consists of all actions enclosed within DBC/SQL BEGIN TRANSACTION and END TRANSACTION statements.
- An implicit transaction consists of one action that is not enclosed within DBC/SQL BEGIN TRANSACTION and END TRANSACTION statements. An implicit transaction is treated as a single unit of work.

Within 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 within the LUW:

- FOCUS issues a COMMIT WORK statement for DB2, IDMS/SQL, and Oracle. The changes indicated by the updates within the transaction are recorded in the table.
- The RDBMS releases locks on the target data. (See *Isolation Levels and Locks* on page 385.)

- Data source changes become available for other tasks.

In response to unsuccessful execution of any statement in the transaction, the adapter:

- 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 adapter provide a level of automatic transaction management but, in many cases, this level of management alone is not sufficient. In MODIFY, the Adapters for **DB2**, **IDMS/SQL**, and **Oracle** support explicit control of RDBMS transactions with the commands SQL COMMIT WORK and SQL ROLLBACK WORK. The Adapter for **Teradata** supports explicit transaction control with the commands SQLDBC BEGIN TRANSACTION, SQLDBC END TRANSACTION, and SQL ROLLBACK WORK. (The Adapter for Teradata also supports the SQL COMMIT WORK command). In Maintain, the equivalent commands are COMMIT and ROLLBACK.

Note: SQL COMMIT WORK and SQL ROLLBACK WORK are native SQL commands—commands that the adapter 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 data sources only. The adapter ignores COMMIT WORK and ROLLBACK WORK without the SQL qualifier.

Note: For DB2, IDMS/SQL, and Oracle:

With the default AUTOCOMMIT setting (see *Controlling Connection Scope* on page 295), 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 command constitute a single LUW. If the procedure completes successfully, the adapter automatically transmits a COMMIT WORK command to the RDBMS, and the changes become permanent. If the procedure terminates abnormally, the adapter 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 *Controlling Connection Scope* on page 295). An automatic COMMIT is issued at the end of every procedure. All called procedures issue a COMMIT before returning to the calling procedure. The COMMIT issued by a called procedure commits all uncommitted changes, even those in the calling procedure.

  For more information, consult your FOCUS documentation on maintaining databases.

- You cannot use the FOCUS CHECK facility when updating a table. The adapter ignores the CHECK command. You must use the SQL COMMIT WORK statement in MODIFY or the COMMIT command in Maintain.

  You can COMMIT after each transaction, or you can use a counter within 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. (See *Change Verify Protocol: AUTOCOMMIT ON CRTFORM* on page 391.)

- The actions described in this section 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, although the SQLCODE is negative, processing continues normally to the next transaction record. Check the FOCERROR variable (discussed in *Using the Return Code Variable: FOCERROR* on page 371) to determine whether or not to ROLLBACK within the MODIFY procedure.

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 within cases of a Case Logic request.

Note: In Maintain, you must use the Maintain facility COMMIT command to transmit an SQL COMMIT WORK to the RDBMS.

Example: Using COMMIT WORK in a MODIFY Procedure

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 data source.

Teradata Transaction Termination: BEGIN/END TRANSACTION

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. Teradata releases data locks when it encounters the END TRASACTION statement or the ROLLBACK WORK statement discussed in *RDBMS Transaction Termination (ROLLBACK WORK)* on page 369.

Example: Teradata Transaction Control Using BEGIN/END TRANSACTION

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. Teradata locks the rows until the update is complete and the END TRANSACTION statement is processed.

```
MODIFY FILE ...
 TRACE
    .
    .
    .
CASE STARTIT
  SQLDBC BEGIN TRANSACTION;
  GOTO PROCESS
ENDCASE

CASE PROCESS
  CRTFORM ...
  MATCH keyfields
  ON MATCH UPDATE ...
  ON MATCH GOTO ENDIT
  ON NOMATCH INCLUDE
  ON NOMATCH GOTO ENDIT
ENDCASE

CASE ENDIT
  SQLDBC END TRANSACTION;
  GOTO STARTIT
ENDCASE
```

This example illustrates multiple update transaction control. It also illustrates the use of the 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
```

RDBMS Transaction Termination (ROLLBACK WORK)

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, or for **Teradata**, since the last BEGIN TRANSACTION 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 unacceptable 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 within cases of a Case Logic request.

For Teradata, the END TRANSACTION statement is not required when the ROLLBACK WORK statement is used. The END TRANSACTION statement is implied and, if it is encountered, produces a message that you can ignore.

Note: In Maintain, you must use the Maintain facility ROLLBACK command to transmit an SQL ROLLBACK WORK to the RDBMS.

The adapter 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 adapter 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.

Example: Using ROLLBACK WORK in a MODIFY Procedure

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 command. 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
```

Example: RDBMS Transaction Control

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> "
  "JOBCODE <T.CURR_JOBCODE> "
  "PLEASE CHANGE THE SALARY AND JOB CODE"
  ON MATCH COMPUTE
      OLDSAL/D12.2 = D.CURRENT_SALARY;
  ON MATCH UPDATE CURRENT_SALARY CURR_JOBCODE
  ON NOMATCH REJECT


MATCH DAT_INC
  ON NOMATCH COMPUTE
    SALARY=CURRENT_SALARY;
    JOBCODE=CURR_JOBCODE;
    PCT_INC = (SALARY - OLDSAL)/OLDSAL;
  ON NOMATCH INCLUDE
  ON NOMATCH SQL COMMIT WORK
  ON MATCH SQL ROLLBACK WORK
  ON MATCH REJECT
DATA
END
```

Using the Return Code Variable: FOCERROR

The RDBMS produces a return code or SQLCODE 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 machine, automatically terminate the 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 commands to determine whether to continue or terminate processing. For example, if FOCERROR is -803 for DB2, 2801 for Teradata, 1058 for IDMS SQL, or 1481 for Oracle, 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 list of common SQL return codes, see *SQL Codes and Adapter Messages* on page 453. For a complete list, see the Messages and Codes manual for your RDBMS.

Using the Adapter 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 within 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 the 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 negative SQLCODE or non-zero return code 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.

The DB2 Resource Limit Facility

The DB2 Resource Limit Facility, also known as the Governor, sets a limit on the resources a dynamic SQL query may use. All SQL queries generated by the adapter are dynamic. (For a discussion of static SQL, see *Static SQL (DB2)* on page 401.) A DB2 database administrator can set limits on application plans, individual users, or both.

If an SQL request generated by a MODIFY or Maintain procedure fails because a resource limit has been reached, the adapter posts the DB2 SQLCODE -905 to the FOCERROR variable.

Since SQLCODE -905 does not terminate a procedure, you need not SET ERRORRUN ON to continue MODIFY processing.

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 data source. 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 within 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 (or WHO field) 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:
  - **INCLUDE Referential Integrity.** 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.
  - **DELETE 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.

The RDBMS can define and enforce referential integrity rules (constraints).

FOCUS can also provide referential integrity for those tables described in a multi-table Master File and Access File pair. The following sections discuss both types of referential integrity constraints.

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 adapter 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 unless you can add primary and foreign key definitions to such tables.

Violations of RDBMS referential integrity rules result in an error. The adapter posts the return code it receives from the RDBMS to the FOCERROR variable. Your MODIFY or Maintain procedure can test this value.

Referential integrity violations do not terminate MODIFY or Maintain procedures, so you need not use the FOCUS SET ERRORRUN ON command to continue MODIFY 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 Master Files and let the RDBMS take care of all referential integrity enforcement.

You may also choose to describe the tables as related (using multi-table Master 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 adapter provides some level of automatic referential integrity for tables described in a multi-table Master File.

The following sections describe the rules and techniques for ensuring or inhibiting FOCUS INCLUDE and DELETE referential integrity. The examples use the ECOURSE Master File, a multi-table description that relates the EMPINFO and COURSE tables. (See *File Descriptions and Tables* on page 459.)

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 Master and Access File. 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 Master File, 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.

Example: Using FOCUS INCLUDE Referential Integrity

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 command 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 command.
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 10
      "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 command 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 command 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 command only identifies CNAME. FOCUS automatically equates the value of EMP_ID with WHO, 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 Master and Access File 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.

Example: **Using FOCUS DELETE Referential Integrity**

When you delete an employee from the EMPINFO table in the ECOURSE Master File, 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 command 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 Master File. COMBINE of single tables does not invoke FOCUS referential integrity.

The MODIFY 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 Master File, you can use the COMBINE facility to modify them as if they are one.

Note: In Maintain, you do not issue a COMBINE command to modify unrelated tables. Instead, you reference multiple tables in the MAINTAIN FILE command. 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 Master Files. However, each Master File can describe more than one table, for a total of 64 per procedure, minus one for the virtual 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 data source with anything except other FOCUS data sources.

Note: In Maintain, you can modify FOCUS data sources 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 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 (see *The FIND Function* on page 384).

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 Master Files. Your procedure should check for and enforce referential integrity, if necessary.

Syntax: How to Create a COMBINE Structure

```
COMBINE FILES file1 [PREFIX pref1|TAG tag1] [AND]
   .
   .
   .
              filen [PREFIX prefn|TAG tagn] AS asname
```

where:

file1 - filen

Are the Master File names of the tables you want to modify. You can specify up to 16 Master Files.

pref1 - prefn

Are prefix strings for each file, up to four characters. They provide uniqueness for field names. You cannot mix TAG and PREFIX in a COMBINE structure. Refer to your FOCUS documentation 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

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.

How FOCUS Creates a COMBINE Structure

The EMPINFO table contains employee number, last name, first name, hire date, department code, current job code, 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 job code. (*File Descriptions and Tables* on page 459 provides the Master and Access Files for these two tables.)

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 job code, this application is appropriate for the COMBINE facility. You can update both tables at the same time without having to define multi-table Master and Access Files.

The following figures represent the tables as separate entities.

```
EMPINFO table          PAYINFO table
           EMPINFO                 PAYINFO
01         S0          01          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 FOCEXEC:

```
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 empspay 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 your FOCUS documentation.

SET INCLUDE SUBTREE

In early releases, the adapter used segment activation logic in MODIFY write operations that differed from that of standard FOCUS against FOCUS files.

In a multi-path file structure involving FOCUS files (including structures generated by the COMBINE command), INCLUDE or DELETE actions operate only on active segments in the subtree of the segment specified in the previous MATCH or NEXT command.

However, with external DBMS files, INCLUDE or DELETE actions would operate not only on active segments in the subtree of the segment specified in the MATCH or NEXT command, but also on any active segments to the right of that segment in the hierarchy (as displayed by the CHECK FILE PICTURE command).

Currently, the default adapter behavior is consistent with the standard FOCUS behavior.

Note: You can control whether the adapter uses the FOCUS standard behavior for external DBMS files using the SET INCLUDE SUBTREE subcommand. Place the subcommand on the line immediately following the MODIFY command:

```
SQL SET INCLUDE {SUBTREE|LATERAL}
```

where:

SUBTREE

Institutes the standard FOCUS behavior. INCLUDE or DELETE actions operate only on active segments in the subtree of the segment specified in the previous MATCH or NEXT command. This is the default value

LATERAL

INCLUDE or DELETE actions operate not only on active segments in the subtree of the segment specified in the MATCH or NEXT command, but also on any active segments to the right of that segment in the hierarchy (as displayed by the CHECK FILE PICTURE command).

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 commands.

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 value of the field 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 data sources 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 your FOCUS documentation 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 command or a MAINTAIN FILE command. With COMBINE, if the FIND is for a field in a VSAM 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 data source.

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 command.

dbfield

For MODIFY, is the field in another COMBINE file structure or an indexed field in a VSAM 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 field name from one of the files listed in the MAINTAIN FILE command, qualified with its file name.

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 VSAM file in which *dbfield* resides. In Maintain, is ignored.

The FIND function is only supported within MODIFY or Maintain procedures. For more information, consult your FOCUS documentation on maintaining databases.

Isolation Levels and Locks

While you are changing data in a table, the table is in an unstable state. The changes may eventually become permanent, or they may be backed out. In order for reports to contain meaningful results, users should not see changes until they are permanent.

To protect data integrity, the RDBMS must guarantee to the person updating a table that no other user will change the selected values before the update made by the first user is submitted.

The RDBMS provides a locking system for concurrency management. With native SQL, you can specify the length of time to hold a lock. This section discusses the duration of the lock, called the *isolation level*.

DB2 Isolation Levels

During installation, every DB2 application plan is bound with a default isolation level. The installation procedures for the adapter specify that the default isolation level is Cursor Stability. (Static MODIFY procedures are an exception. See *Static SQL (DB2)* on page 401 for information about isolation levels in static procedures.)

With the Cursor Stability setting, shared (read) locks on a data row or page are released as your cursor moves off that location. For example, if a report reads many data pages, the shared lock acquired on each data page is released as the shared lock on the next data page is acquired.

Use Cursor Stability for read-only applications such as TABLE requests or read-only MODIFY or Maintain procedures that browse data without updating. You may not use Cursor Stability with a MODIFY or Maintain procedure that changes values in the database. Doing so would leave the data susceptible to change by other tasks in the interim between the initial selection (MATCH) and the update or inclusion.

The isolation level setting named Repeatable Read provides the highest level of protection. With Repeatable Read, any lock acquired is held until the transaction boundary (COMMIT WORK or ROLLBACK WORK). Therefore, any data the MODIFY or Maintain procedure displays on the screen remains unchanged until the procedure submits the update and executes a COMMIT WORK command.

This higher level of protection is provided at the expense of concurrency, since all locks, including shared locks, remain in effect until the end of a logical unit of work. Therefore, you should commit transactions on a regular basis throughout the MODIFY or Maintain procedure. Periodic transaction termination is especially important for NEXT processing. All rows retrieved by NEXT remain locked even if they are not all updated. These retrieved rows are not available for update by another user until your procedure releases them (using SQL COMMIT WORK or SQL ROLLBACK WORK in MODIFY, or using COMMIT or ROLLBACK in Maintain).

Because of these concurrency considerations, all read/write MODIFY and Maintain procedures require the Repeatable Read isolation level, including those MODIFY procedures that invoke the Change Verify Protocol described in *Change Verify Protocol: AUTOCOMMIT ON CRTFORM* on page 391.

In DB2, there are two additional isolation levels for read-only access. Uncommitted Read (UR) provides read-only access to records even if they are locked. However, these records may not yet be committed to the database. The other isolation level is called Read Stability (RS). For more information, see the *DB2 Command and Utility Reference*.

Note: The adapter AUTOCOMMIT ON CRTFORM feature is designed to eliminate some concurrency problems. (See *Change Verify Protocol: AUTOCOMMIT ON CRTFORM* on page 391 for more information.) However, it also requires an isolation level of Repeatable Read.

The following sections describe how to change the isolation setting.

Changing the DB2 Isolation Level

The SET ISOLATION command allows you to dynamically change the isolation level for DB2.

For DB2 on z/OS, you can also switch to a plan with a different isolation level. See *Changing the DB2 Isolation Level by Switching to Another Plan* on page 387.

Syntax: How to Change the DB2 Isolation Level

You can change the isolation level by issuing the SET ISOLATION command in a MODIFY procedure or at the FOCUS command level. (For Maintain, you must issue the SET ISOLATION command at the command level prior to invoking the Maintain procedure.) The setting remains in effect for the FOCUS session or until you reset it.

From the FOCUS command level, issue

```
{ENGINE|SQL} [DB2] SET ISOLATION level
```

where:

level

Indicates the isolation level. Valid values are:

CS Cursor Stability, the default. Releases shared locks as the cursor moves on in the table. Use for read-only requests.

RR Repeatable Read. Use for MODIFY and Maintain read/write routines. Locks the retrieved data until it is released by an SQL COMMIT WORK or SQL ROLLBACK WORK.

UR Uncommitted Read. Provides read-only access to records even if they are locked. However, these records may not yet be committed to the database. Use for read-only requests.

RS Read Stability. Use for read-only requests. For more information, see the DB2 Command and Utility Reference.

blank A blank value resets the level to the adapter default.

Note:

- Omit the target RDBMS qualifier to issue the command in MODIFY procedures or if you previously issued the SET SQLENGINE command.
- The adapter does not validate the isolation level values. If you issue the SET ISOLATION command with a level not supported for the version of the RDBMS you are using, the RDBMS will return an SQL code of -104, signifying an SQL syntax error.
- The SET ISOLATION command is enabled for only SELECT requests created as a result of FOCUS TABLE requests.
- This setting cannot override the required isolation level of RR for MODIFY and Maintain requests.

For more information about adapter commands, consult *Adapter Commands* on page 309.

Changing the DB2 Isolation Level by Switching to Another Plan

The Adapter for DB2 SET PLAN command allows you, within a FOCUS session, to switch to a plan bound with a different isolation level. The SET PLAN command is only available if the Adapter for DB2 was installed using the Call Attachment Facility (CAF).

The systems group responsible for adapter installation must generate two DB2 application plans for the adapter, one bound with the Cursor Stability isolation level and the other with Repeatable Read. You can then use the adapter SET PLAN command to switch between the plans for the desired isolation level. After you issue the SET PLAN command, all FOCUS TABLE, Maintain, or MODIFY procedures take advantage of the new isolation level.

DB2 has two additional isolation levels appropriate for read-only access. Uncommitted Read (UR) provides read-only access to records even if they are locked. However, these records may not yet be committed to the database. The other isolation level is Read Stability (RS). For more information, see the *DB2 Command and Utility Reference*.

Syntax: How to Dynamically Change the DB2 Plan

Issue the SET PLAN command from the FOCUS command level

```
{ENGINE|SQL} [DB2] SET PLAN planname
```

where:

planname

Is the name of your application plan. The default is DSQL unless your site changed the default at installation time.

Note:

- Omit the DB2 qualifier if you previously issued the SET SQLENGINE command for DB2.
- In non-CAF versions of the Adapter for DB2, you must use a separate JCL procedure or CLIST to access each plan.

Isolation Levels in IDMS/SQL

Use Cursor Stability or Transient Read for read-only applications such as TABLE requests or read-only MODIFY procedures that browse data without updating. You may not use Transient Read with a MODIFY procedure that changes values in the database. Doing so would leave the data susceptible to change by other tasks in the interim between the initial selection (MATCH) and the update or inclusion.

Cursor Stability (the adapter default) provides the highest level of protection. With Cursor Stability, *any* lock acquired is held until the transaction boundary (COMMIT WORK or ROLLBACK WORK) or until an updateable cursor is closed. Therefore, any data the MODIFY procedure displays on the screen remains unchanged until the procedure submits the update and executes a COMMIT WORK command.

This higher level of protection is provided at the expense of concurrency, since all locks, including shared locks, remain in effect until the end of a Logical Unit of Work. Therefore, you should COMMIT transactions on a regular basis throughout the MODIFY procedure. Periodic transaction termination is especially important for NEXT processing. All rows retrieved by NEXT remain locked even if they are not all updated, since the cursor used is a retrieval cursor. These retrieved rows are not available for update by another user until your procedure releases them (using SQL COMMIT WORK, SQL ROLLBACK WORK, or another IDMS SQL transaction or session ending command such as RELEASE in MODIFY).

Because of these concurrency considerations, all read/write MODIFY procedures require the Cursor Stability Isolation Level, including those MODIFY procedures that invoke the Change Verify Protocol described in *Change Verify Protocol: AUTOCOMMIT ON CRTFORM* on page 391.

Note: The adapter AUTOCOMMIT ON CRTFORM feature is designed to eliminate some concurrency problems. However, it also requires an Isolation Level of Cursor Stability.

Syntax: How to Change the IDMS SQL Isolation Level

The isolation level must be Cursor Stability for all read/write MODIFY procedures. You must issue the IDMS/SQL SET TRANSACTION command at the FOCUS command level prior to invoking the MODIFY procedure. The setting remains in effect for the FOCUS session or until you reset it.

From the FOCUS command level, issue

```
SQL [SQLIDMS] SET TRANSACTION level
```

where:

level

Can be one of the following:

CURSOR STABILITY is Cursor Stability, the default. Provides the maximum amount of concurrency while guaranteeing the integrity of the data selected.

`TRANSIENT READ` is Transient Read. Allows the reading of records locked by other users (allows dirty reads). Recommended for TABLE only. Transient Read prevents the SQL transaction from performing updates. Use this only when you do not need the data retrieved to be absolutely consistent and accurate. If you specify Transient Read, IDMS assumes Read Only.

`READ ONLY` allows data to be retrieved, but does not allow the database to be updated.

`READ WRITE` allows data to be retrieved, and allows the data source to be updated.

Note: Omit the SQLIDMS target RDBMS qualifier from the command when issuing it in a MODIFY procedure or if you previously issued the SET SQLENGINE command for SQLIDMS.

Oracle Locks

The method of concurrency control used by Oracle is implemented by locking the shared data. Locks may be of the following types:

- **Write** enables a single user or program to lock out all other users from the data it is currently reading or modifying, except those users acquiring Access locks. This type of lock is automatically acquired by Oracle for data involved in FOCUS UPDATE, DELETE or INCLUDE operations.
- **Read** is used to ensure consistency during read operations (for example, MATCH, NEXT and TABLE FILE). Several users may hold read locks on a table to retrieve data, at which time no modification of that data is permitted by Oracle.

Issuing SQL Commands in MODIFY

The adapter allows you to execute a wide range of native SQL commands 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—from within 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* on page 265.)

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 TSO or MVS environmental qualifiers. 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.

Example: Issuing SQL Commands in MODIFY

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.

Change Verify Protocol: AUTOCOMMIT ON CRTFORM

The adapter 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.

Note:

- This protocol is not supported for Teradata, which implicitly commits each SQL statement individually.
- Maintain does not support AUTOCOMMIT ON CRTFORM.

Without Change Verify Protocol, the adapter 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. Within 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 within the boundaries of the LUW, the RDBMS will undo, or ROLLBACK, any outstanding updates within 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. Data source 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 adapter 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 adapter retrieves the row from the table again. The adapter 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 (see *The FOCURRENT Variable* on page 393).
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 (see *The FOCURRENT Variable* on page 393). 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 the run-time path of an application, 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 within 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.

Syntax: How to Invoke the Change Verify Protocol

```
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) for **DB2**, Cursor Stability for **IDMS/SQL**, and Write for **Oracle**. For a discussion of isolation levels, see *Isolation Levels and Locks* on page 385.

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.
- The FOCURRENT variable is not supported for Teradata.

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, within the logic of your application, you wish to re-retrieve a VALIDATE-rejected record from the data source 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.

Example: Testing FOCURRENT

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
ENDCASE                         refreshed data on the screen.
-*
CASE AT START
  SQL SET AUTOCOMMIT ON CRTFORM  <== Releases record after display
ENDCASE
-*
DATA
END
```

Note: If the application (and not the adapter) sponsors the second MATCH, the MODIFY application itself must check FOCURRENT (see *The FOCURRENT Variable* on page 393). The second MATCH does not automatically perform the check. Any maintenance action issued subsequent to the second MATCH subcommand still initiates an automatic check.

Loading Tables Faster: The MODIFY Fastload Facility

The adapter Fastload facility increases the speed of loading data into tables with MODIFY. It is especially effective when using FIXFORM to load large volumes of data into a table.

For Oracle and DB2, you should issue the INSERTSIZE command in conjunction with Fastload to take advantage of blocked inserts. For more information, see *DB2 and Oracle Array Blocking for INSERT Requests* on page 397.

Use the following command to invoke the Fastload option from a MODIFY procedure:

```
SQL SET LOADONLY
```

For example:

```
MODIFY FILE table
SQL SET LOADONLY
      .
      .
      .
MATCH key1
    ON NOMATCH INCLUDE
    ON MATCH REJECT
      .
      .
      .
END
```

A standard MODIFY procedure uses a MATCH command to test for the existence of a single row whose primary key value matches that supplied by the input transaction. When the underlying table is an RDBMS table, the adapter uses an SQL SELECT statement to perform this test. After examining the return code resulting from the SELECT, FOCUS determines whether or not the row exists and directs MODIFY processing to the appropriate MATCH logic.

Fastload eliminates this SELECT operation. Its sole responsibility is to load rows into the RDBMS. Fastload does so without first determining whether the row already exists within the table. The RDBMS ensures the uniqueness of stored rows using its unique index. However, FOCUS messages about the existence of duplicates display online or go into a LOG file if one exists.

Fastload can only insert (ON NOMATCH INCLUDE) rows into a table. Any attempt to use other MODIFY maintenance functions produces a FOC1404 error message, and the procedure terminates.

DB2 and Oracle Array Blocking for INSERT Requests

The Adapters for DB2 and Oracle support buffered insertion of rows into tables. This technique substantially reduces network traffic and CPU utilization.

High INSERTSIZE values increase the efficiency of requests involving many rows, at the cost of higher virtual storage requirements. A value higher than 100 is not recommended because the increased efficiency it would provide is generally negligible.

Syntax: How to Set INSERTSIZE for DB2 and Oracle

```
SQL [sqlengine] SET INSERTSIZE n
```

where:

sqlengine

Indicates the Adapter. Valid values are DB2 and SQLORA. You can omit this value if you previously issued the SET SQLENGINE command.

n

Is the number of rows to be transmitted to the RDBMS at once. Accepted values are 1 to 5000. The default is 1.

Note: 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 the buffer is not full. This restriction has several ramifications:

- To use INSERTSIZE in a MODIFY request, you must use the SQL SET LOADONLY command. Without LOADONLY, the adapter does not insert a row without first issuing a SELECT statement to check that the row does not already exist.
- To use INSERTSIZE with Direct SQL Passthru, the INSERT statement must be parameterized and use the PREPARE, BIND, and EXECUTE command set within a BEGIN SESSION/END SESSION command pair. In this case, the BIND is not optional. See *Direct SQL Passthru* on page 265, for a discussion of parameterized Passthru.

Example: Sample Oracle Session Using INSERTSIZE

```
SET SQLENGINE = SQLORA
SQL SET CONNECTION_ATTRIBUTES /USER1,PASS1
SQL SET AUTOCOMMIT ON COMMAND
SQL SET INSERTSIZE 3
SQL BEGIN SESSION
SQL PREPARE ABC FOR INSERT INTO USER1.PAYROLL
  VALUES (:001,:002);
END
SQL BIND ABC USING CHAR(5), DECIMAL(11,2);
END
SQL EXECUTE ABC USING '11111', 50000; (Row is buffered)
END
SQL EXECUTE ABC USING '22222', 65000; (Row is buffered)
END
```

```
SQL EXECUTE ABC USING '33333', 30000; (Row is buffered, block transmitted)
END

SQL END SESSION                       (LUW is committed)
```

Chapter 13

Static SQL (DB2)

In early releases of the adapter, FOCUS access to the RDBMS was entirely through dynamic SQL requests. While dynamic SQL is ideal for applications like ad hoc reporting, in which you do not know in advance what SQL statements you will execute at run time, it is less desirable for applications that do not require such flexibility.

In contrast, static SQL is ideal when you know beforehand all the SQL statements a procedure may execute. You register the procedure itself, including the SQL statements, with the RDBMS through a process known as binding. The resulting database object, called a DB2 application plan, is stored in the database and retrieved whenever you run the procedure. *Introduction to Adapters for Relational Data Sources* on page 17, introduces bind concepts.

Static SQL provides an additional degree of security not available with dynamic SQL. Authorization is granted to the static plan, not the underlying database objects, restricting the SQL statements a user can execute with these objects.

Another important aspect of the static SQL process is that the RDBMS optimizes each SQL statement in the program and chooses an access path for it. A data access path consists of low-level data access requests that the RDBMS formulates and stores in internal format. When you execute the procedure, the RDBMS retrieves these requests and executes them immediately. The net effect is that SQL statements in the static request are not reinterpreted at run time. The access path for each statement is pre-selected and reused each time you run the procedure.

In this chapter:

Static SQL Overview

The Adapter for DB2 comes with a Static SQL facility for quickly and easily creating application plans or packages for MODIFY requests.

The Static SQL for MODIFY facility creates a static SQL module for a compiled MODIFY procedure. The procedure can contain only one MODIFY request, which can contain SQL commands. If the procedure references Dialogue Manager variables, you supply all of the values at compile time and cannot substitute new values at run-time.

When you run static procedures, you take advantage of the following security and performance benefits of static SQL:

- Procedure-level security.

  Privileges (for example, SELECT, UPDATE, or DELETE) for operations on a table are available only to authorized users of the application plan (or package) associated with the static procedure. Users have no access to underlying tables outside of the procedure. Under dynamic SQL, privileges must be granted on the actual tables. Many sites do not allow dynamic access to tables because users can then access the same tables using other programs, such as QMF□. Static SQL specifically addresses this security concern.

- Performance improvements.

  Static SQL offers potentially substantial performance improvements for MODIFY procedures. The PREPARE cursor operation that checks syntax and authorization and selects access paths is removed from the run-time environment.

  MODIFY procedures tend to have many more SQL statements than TABLE requests, and each statement executed must first be prepared. Furthermore, the access path is lost each time the procedure executes a COMMIT or ROLLBACK WORK statement, even if the SQL statement has already been prepared. In procedures that COMMIT after each transaction, the same SQL statements must be prepared prior to every transaction. Many interactive programs COMMIT after each transaction to release locks and minimize the impact of a system or program failure. These programs benefit most from static SQL.

  If you do not normally compile your MODIFY procedures, you will probably notice that it takes less time to initialize the static procedure. FOCUS has already interpreted the MODIFY commands, and the time required to initialize the procedure is a function of compiled MODIFY rather than of static SQL.

  Static SQL does not increase the actual speed of data retrieval or update. In a static SQL procedure, the time required to retrieve and/or update a given set of rows is identical to that of the same procedure executing dynamically, minus the cost of access path selection and authorization checking for the SQL statements.

Note: If RDBMS table statistics change, or if table or index structures are altered after the static procedure is bound, the procedure should be rebound to ensure optimum performance. Refer to *Optionally BIND the Plan for the FOCEXEC* on page 408 for a discussion of BIND. (*Introduction to Adapters for Relational Data Sources* on page 17 includes a discussion of basic bind concepts.)

- Ease of Use.

  The FOCUS implementation of static SQL offers you a far less complex and time-consuming development cycle than is normally the case with static SQL applications. No third-generation language or SQL skills are required. In fact, you can write a FOCUS application to use static SQL without coding a single SQL statement. Additionally, you can develop, test, and revise your FOCEXEC using dynamic SQL without going through the cumbersome static SQL preparation process every time you make a change. You do not have to create the application plan or package until the end of the development cycle.

Static SQL Requirements

You can implement many existing dynamic applications as static procedures with little preparatory work. A FOCEXEC does not require special coding (for example, embedded SQL) to use static SQL. All current MODIFY procedures that update DB2 tables (and can currently be compiled), can be registered as static SQL procedures with no alterations.

A MODIFY procedure to be compiled with the Static SQL for MODIFY facility can also contain SQL commands (for example, SQL COMMIT WORK, INSERT, or UPDATE). The FOCEXEC can also invoke other FOCEXECs; however, to run these additional FOCEXECs statically, you must convert them separately to static SQL.

You must satisfy the following requirements to create a static SQL procedure:

- A MODIFY procedure can contain only one MODIFY request. It cannot contain any other FOCUS or operating system commands. If the FOCEXEC contains Dialogue Manager commands, FOCUS prompts you for the values of any variables at compilation time. You cannot substitute new values at run time.
- All tables used in a MODIFY procedure must have primary keys. That is, you may not include any table whose Access File specifies KEYS=0.
- A MODIFY procedure cannot use the WITH-UNIQUES method for processing unique segments.
- The Adapter for DB2 must have been installed to use the Call Attachment Facility (CAF). You can verify this by using the SQL DB2 ? command to display adapter settings (Call Attachment Facility should be ON).

- You cannot compile a MODIFY FOCEXEC as a static procedure if it accesses a DB2 view created with a GROUP BY or HAVING clause.

  The DB2 environment prohibits the type of static SQL syntax the adapter uses against a DB2 view created with a GROUP BY or HAVING clause. If you issue a static COMPILE for a FOCEXEC that operates against such a view, an SQLCODE of -815 results either at BIND or RUN time, depending on your current level of DB2 maintenance.

- Static SQL is not supported for tables with date-time columns.

Note:

- Certain resource restrictions exist for any procedure that uses static SQL. Under some circumstances they may make it impossible to compile a MODIFY procedure to use static SQL. *Resource Restrictions* on page 415 explains these restrictions and how to determine whether your procedure may be affected.

- If your table contains a TEXT (LONG VARCHAR) field, place the description of that field at the end of the segment declaration for that table in the Master File. No change has to be made to the RDBMS table. If you have more than one TEXT field in a table, place one of them last. It does not matter where you place the other TEXT field.

- The FOCCOMP library member for a MODIFY procedure contains a record of the fact that the procedure uses static SQL. To have both static and dynamic versions of the compiled procedure, you need two FOCCOMP libraries, one compiled with STATIC ON or NOBIND, and the other compiled with STATIC OFF.

- The DB2 default value of 100 for the FETCHSIZE parameter is incompatible with static MODIFY procedures. To run a compiled MODIFY procedure, you must SET FETCHSIZE to 1.

Creating a Static Procedure for DB2

Creating a static MODIFY is an extension to the usual procedure for compiling a FOCEXEC with the FOCUS COMPILE facility. Invoke the static creation process after you complete the FOCEXEC or MODIFY procedure.

You must allocate some additional DDNAMEs, you may need to issue the adapter SET STATIC command, and you must execute the FOCUS COMPILE command. In response, the adapter automatically creates an Assembler program with the embedded SQL required by the procedure. It then precompiles, assembles, link-edits, and, optionally, binds the program. It accomplishes all steps, including the automatic bind, from within FOCUS.

You must give some thought to the issue of plan management discussed in *Plan Management in DB2* on page 413 as it will affect your choice of bind option.

This section outlines the steps required to create a static procedure, and provides a description of FOCUS processing for each step. *DB2 Static MODIFY Example* on page 410 provides an annotated example.

Creating a static module for DB2 consists of the following steps:

1. Write the procedure.
2. Allocate the required DDNAMEs.
3. Optionally issue the SET STATIC command.
4. Optionally issue the SET SSID command.
5. Compile the FOCEXEC.
6. Optionally bind the plan for the FOCEXEC.
7. Authorize users to run the plan.

Write the FOCEXEC

The MODIFY procedure must be compilable. Consult *Static SQL Requirements* on page 403 and your FOCUS documentation on maintaining databases for more information on compiled MODIFY requirements.

Allocate the Required DDNAMEs

The following chart lists the DDNAMEs you must allocate before compiling a FOCEXEC to use static SQL. You can add these allocations to your FOCUS CLIST, batch JCL, or PROFILE FOCEXEC. Note that these allocations are additions to the normal allocations for running FOCUS with DB2:

DDNAME	DCB Parameters	Description
ASMSQL	DSORG(PO) RECFM(FB) LRECL(80)	Target data set for the assembler source code generated by the compilation. The size of each member varies depending on the size of the MODIFY procedure and the number of columns in each table the MODIFY procedure references.
DBRMLIB	DSORG(PO) RECFM(FB) LRECL(80)	Target data set for the database request module generated by the compilation.

DDNAME	DCB Parameters	Description
DB2LOAD	n/a	DB2 load library. The name for this library is site-specific, but usually follows the form DSNxy0.SDSNLOAD (where xy0 is DB2 Version x Release y). The DB2 load library should be the same one used by the DB2 subsystem where the bind will take place. **Note:** The STEPLIB allocation of the DB2 load library is still necessary.
STUBLIB	DSORG(PO) RECFM(U)	Contains the load module created for the procedure. This data set is also required at run time.
SQLERR1	DSORG(PS) RECFM(FB) LRECL(130)	Contains the output of the IBM precompiler. DCB parameters are set by the precompiler. To route output to the terminal, use DA(*).
SQLERR2	DSORG(PS) RECFM(FM) LRECL(121)	Contains the output of the IBM assemble operation. DCB parameters are set by the assembler. To route output to the terminal, use DA(*).
SQLERR3	DSORG(PS) RECFM(FA) LRECL(81)	Contains the output of the IBM linkage editor. DCB parameters are set by the linkage editor. To route output to the terminal, use DA(*).
FOCCOMP	DSORG(PO) RECFM(VB) LRECL(32756) BLKSIZE(32760)	Contains the compiled MODIFY procedure. This data set is also required at run time. **Note:** The FOCCOMP library member for a MODIFY procedure contains a record of the fact that the procedure uses static SQL. If you have both static and dynamic versions of the compiled procedure, you need two FOCCOMP libraries, one compiled with STATIC ON or NOBIND, and the other compiled with STATIC OFF.

Optionally Issue the SET STATIC Command

The syntax for the SET STATIC command is

```
{ENGINE|SQL} [DB2] SET STATIC process
```

where:

process

Indicates the command that invokes static processing. Valid values are:

OFF

Is the default setting. Reverses ON or NOBIND settings. Does not invoke static processing for any subsequent COMPILE commands issued. For information about the FOCUS COMPILE command, see your FOCUS documentation on maintaining databases.

ON

invokes static processing with automatic bind. This option does not support extended plan management (see *Plan Management in DB2* on page 413) or modification of any of the default bind parameters. If you require more flexibility, use the NOBIND option.

NOBIND

invokes static processing without bind. Use this option if you need extended plan management, to change the default bind parameters, or to bind your programs at another time (for example, during an off peak period).

Note:

- If FOCUS performs the bind, the isolation level is always RR for MODIFY requests. To change the isolation level, you must issue the BIND outside of FOCUS, as described in Optionally BIND the Plan for the FOCEXEC on page 13-9.
- Before requesting an automatic bind, make sure the DB2 subsystem is operational. In addition, the bind operation itself requires DB2 privileges that you may not possess. Your DB2 database administrator can tell you if you are authorized to use BIND.
- Omit the DB2 qualifier if you previously issued the SET SQLENGINE command for DB2.
- Batch considerations:

  You can create or run static procedures in batch. The DSN command processor cannot be invoked in batch. Therefore, you must compile your programs using the NOBIND option and bind the programs separately.

Optionally Issue the SET SSID Command

If you request the automatic bind option and the DB2 subsystem in which you want the bind to occur is different from your installation default, you must issue the SET SSID command (see *Adapter Commands* on page 309). You can obtain the current setting for the DB2 subsystem by issuing the SQL DB2 ? command.

Note: The DB2 subsystem referenced by the SET SSID command should use the DB2 load library allocated to DDNAME DB2LOAD.

Compile the FOCEXEC

To compile the MODIFY procedure, use the FOCUS COMPILE facility.

Syntax: How to Compile a Static MODIFY Procedure on z/OS

Before compiling the request, be sure to allocate all input and output files, such as transaction files or log files, that the MODIFY will use at run time. Any COMBINE structures must be in effect both before compiling the MODIFY and at run time.

Issue the COMPILE command

```
COMPILE focexec
```

where:

focexec

Is the name of the MODIFY procedure to compile.

You do not have to recompile or rebind FOCUS applications for different releases of a FOCUS Version. This saves time and effort.

For example, a FOCUS static MODIFY procedure compiled and bound in Version 7.6 Release 11, does not have to be recompiled or rebound for FOCUS Version 7.6 Release 12, or vice versa. However, the procedure must be recompiled and rebound for a different FOCUS Version (for example, FOCUS Version 7.7).

Note: The ‘AS module’ extension to the COMPILE command is not supported for static SQL procedures. The module name must be identical to the FOCEXEC name.

FOCEXECs compiled with the Static SQL facility do not support the use of the LOAD command. To execute them you must use the RUN command.

Optionally BIND the Plan for the FOCEXEC

This step is only necessary if you issued the SET STATIC command with the NOBIND option.

You may choose to bind one application plan for each FOCEXEC (basic plan management) or to combine several FOCEXECs into one application plan (extended plan management). Also, use this option to supply bind parameters (such as isolation level) other than the default. *Plan Management in DB2* on page 413 discusses plan management, and *Introduction to Adapters for Relational Data Sources* on page 17 introduces bind concepts.

Issue the BIND command outside the FOCUS environment, using one of the methods supplied by IBM. Your BIND may include one or more FOCEXEC DBRMs.

For example:

```
BIND PLAN (BIGPLAN) MEM(FEX1 FEX2 FEX3 FEX4) ACTION(ADD) ISOLATION(CS)
```

For a full explanation of bind methods and parameters, consult the *IBM DB2 Command and Utility Reference*.

Authorize Users to Run the Plan

Issue the SQL GRANT EXECUTE command to authorize users to execute the application plan created with the previous steps. This example shows how to issue the command from within a FOCUS session:

```
SQL DB2 GRANT EXECUTE ON PLAN BIGPLAN TO USER1, USER2
```

For more information on GRANT, consult the *IBM DB2 SQL Reference*.

Run-time Requirements

In addition to the allocations required for FOCUS and the adapter outlined in *Invoking Relational Adapters* on page 25, you must include allocations for the FOCCOMP and STUBLIB libraries allocated to DDNAMEs FOCCOMP and STUBLIB.

If you use extended plan management (see *Extended Plan Management* on page 414), you must issue the adapter SET PLAN command before running any of the procedures included in the DB2 application plan (see *Maintaining Tables With FOCUS* on page 349):

```
SQL DB2 SET PLAN BIGPLAN
```

This setting overrides the plan for the FOCUS dynamic adapter. *Extended Plan Management* on page 414 explains options for resetting the plan at the conclusion of your procedures.

You can use the SQL DB2 ? command to view all current adapter plan settings, for example:

```
SQL DB2 SET PLAN BIGPLAN
SQL DB2 ?
>
     .
     .
     .
(FOC1448) ACTIVE PLAN FOR CALL ATTACH IS     - :
(FOC1459) USER SET PLAN FOR CALL ATTACH IS   - : BIGPLAN
(FOC1460) INSTALLATION DEFAULT PLAN IS       - : P7029910
     .
     .
     .
```

The most recently issued request to DB2 sets the Active Plan. If the thread is closed or marked inactive, no value displays for Active Plan. Your SET PLAN command determines the User Set Plan.

Note: To execute a static MODIFY procedure, you do not need the Access File library (allocated to DDNAME FOCSQL) at run time if you will not be accessing DB2 dynamically during the FOCUS session. The Master File library (allocated to DDNAME MASTER) is required for all static and dynamic access.

Processing and Security Overview

When you run a compiled FOCEXEC procedure, the adapter searches the STUBLIB load library for a load module of the same name and loads it, if it exists.

When you run a compiled MODIFY procedure, FOCUS loads the FOCCOMP library member into virtual memory. If the flag setting in this member indicates that this is a static procedure, the adapter searches the STUBLIB load library for a load module of the same name. The adapter compares timestamps in the FOCCOMP and STUBLIB members for consistency.

Next, FOCUS connects to DB2 and opens a thread to the application plan of the same name as the static procedure (you can use the SET PLAN command to override this). DB2 compares the owner, program name, and timestamps in the STUBLIB member and the DB2 application plan for consistency.

This process verifies that the compiled MODIFY (FOCCOMP member), the program load module (STUBLIB member), and the DB2 application plan are valid and that no substitutions have been made.

DB2 Static MODIFY Example

The following annotated example illustrates the creation and automatic bind of a static MODIFY procedure named MOD1. The numbers 1 through 7 refer to the explanatory notes that follow the example.

Also included for your information are the options FOCUS uses for each step (for example, Assembler options). Please refer to the appropriate IBM manual for an explanation of these parameters and their possible settings.

```
    SQL DB2 SET SSID DB2P
    SQL DB2 SET STATIC ON
1.  COMPILE MOD1
    EMPLOYEE    ON 02/27/91 at 14.26.57
2.  (FOC1472) STATIC SQL PROGRAM CREATED SUCCESSFULLY:
3.  (FOC1473) STATIC SQL PROGRAM PREPROCESSED. RETURN CODE IS: 4
4.  (FOC1474) STATIC SQL PROGRAM ASSEMBLED   . RETURN CODE IS: 0
5.  (FOC1476) STATIC SQL PROGRAM LINKED      . RETURN CODE IS: 0
    DSN SYSTEM(DB2P)
6.  BIND PLAN(MOD1) MEM(MOD1) ACTION(ADD) ISOLATION(RR)
    DSNT252I - BIND OPTIONS FOR PLAN MOD1
               ACTION    ADD
               OWNER     PMSEF
               VALIDATE  RUN
               ISOLATION RR
               ACQUIRE   USE
               RELEASE   COMMIT
               EXPLAIN   NO
    DSNT253I - BIND OPTIONS FOR PLAN MOD1
               NODEFER         PREPARE
    DSNT200I -  BIND FOR PLAN MOD1     SUCCESSFUL
    END

7.  COMPILED...
```

The steps in the process are:

1. Issue the COMPILE command.

   The user invokes the FOCUS COMPILE facility for MODIFY procedures.

2. Create the Assembler program.

   In this initial step, FOCUS reads the MODIFY procedure and creates an Assembler program with embedded SQL statements representing all SQL statements the MODIFY will execute. This program is the input file for the next step.

3. Precompile the Assembler program.

   FOCUS invokes the IBM precompiler for the Assembler program created in Step 2. The precompiler comments out the embedded SQL statements and replaces them with Assembler calls to DB2. It places the SQL statements in a Database Request Module (DBRM) created as a member in the data set allocated to DBRMLIB. This member is part of the input to the bind process. *Introduction to Adapters for Relational Data Sources* on page 17, introduces bind concepts. FOCUS places the modified source program into a temporary data set. It writes the IBM precompiler listing to the data set allocated to DDNAME SQLERR1 or to the terminal.

This step uses the Assembler precompiler included in the DB2 load library allocated to DDNAME STEPLIB. The adapter SET SSID command has no influence on the precompiler used. If you have more than one DB2 subsystem, be sure that the DB2 load library used in the precompilation step is the same one used by the DB2 subsystem in which the program will run. The DB2 subsystem does not have to be active during this step.

The precompiler options for this step are HOST(ASM), DATE(ISO).

Note: The precompilation process generates the statement WARNINGS HAVE BEEN SUPPRESSED DUE TO LACK OF TABLE DECLARATIONS. This message is normal and may be ignored. The precompilation should complete with a return code of four (4).

4. Assemble the program.

   FOCUS assembles the modified source program using IBM's Assembler H and places the output into a temporary data set. It writes the resulting listing to the data set allocated to DDNAME SQLERR2 or to the terminal.

   The assembler options are DECK, NOOBJECT, NOALIGN and TERM. This step should complete with a return code of zero (0).

5. Link-edit the program.

   FOCUS links the assembled source program using IBM's linkage editor, placing the run-time module into the data set allocated to DDNAME STUBLIB. It writes the linkage editor output to the data set allocated to DDNAME SQLERR3 or to the terminal.

   The linkage editor options are TERM, RENT, AMODE(31), RMODE(ANY), and LIST. This step should complete with a return code of zero (0).

6. Optional automatic BIND.

   Since the SET STATIC ON command requests an automatic BIND, FOCUS invokes the DB2 DSN command processor and submits the following bind request:

```
DSN SYSTEM(DB2P)
BIND PLAN (MOD1) MEM(MOD1) ACTION(REPLACE) ISOLATION(RR)
```

   Note:

   - The SSID of the DB2 subsystem in which this plan will be bound is DB2P. The target subsystem for the bind was established by the SET SSID command.
   - The plan name will be the same as that of the FOCEXEC (MOD1). The MEM parameter means that application plan MOD1 will include member MOD1 from the data set allocated to DDNAME DBRMLIB. Member MOD1 was created during the precompilation step. The plan owner is the DB2 primary authorization ID in effect at the time of the bind.

7. Compile the FOCEXEC.

   FOCUS compiles the FOCEXEC and creates member MOD1 in the data set allocated to DDNAME FOCCOMP. It sets a flag in the FOCCOMP member to indicate that this is a static procedure.

   FOCEXECs compiled with the Static SQL facility do not support the use of the LOAD command. To execute them you must use the RUN command.

Plan Management in DB2

Using extended plan management, you can create application plans that contain multiple procedures. This section discusses both the basic and extended plan management options. *Introduction to Adapters for Relational Data Sources* on page 17 introduces application plans, packages, and bind concepts.

Basic Plan Management

With basic plan management, you create a separate DB2 application plan for every MODIFY procedure. You do not have to issue the SET PLAN command, and the adapter automatically establishes and terminates the necessary DB2 threads for each procedure invoked.

The following examples assume that MOD1 and MOD2 are static MODIFY procedures and that a corresponding DB2 application plan exists for each of them:

Step	MODIFY
1.	`RUN MOD1`
2.	`RUN MOD2`
3.	`RUN MOD2`
4.	`EX RPT1`

The adapter takes the following actions:

1. First, it closes the thread to a prior application plan, if one exists. It deallocates that plan with a Call Attachment Facility (CAF) CLOSE.

   Then it opens a thread to plan MOD1. If no prior connection existed, it first issues a CAF CONNECT. It establishes the thread to MOD1 with a CAF OPEN.

2. It opens a thread to plan MOD2. First it deallocates the thread to MOD1 with a CAF CLOSE, then issues a CAF OPEN to establish the thread to MOD2.

3. No action. The adapter recalls that it already has a thread to MOD2 and takes no action.
4. It opens a thread to the FOCUS dynamic plan. First it deallocates the thread to MOD2 with a CAF CLOSE. Then it issues a CAF OPEN to establish the thread to the installation default application plan for the dynamic SQL adapter.

Note: The settings for AUTOCLOSE and AUTODISCONNECT affect thread retention across invocations of the same procedure, as in items 2 and 3. Some settings sever a thread and/or connection to DB2, either at the conclusion of any procedure, or within the procedure itself. More information is available in *Controlling Connection Scope* on page 295.

Extended Plan Management

With extended plan management, you can create a DB2 application plan that contains more than one procedure. You must issue the SET PLAN command to establish an umbrella plan that includes each FOCEXEC to be invoked.

The primary purpose of extended plan management is to limit the amount of overhead involved in plan switching. Therefore, this method of binding is not consistent with the AUTOCLOSE or AUTODISCONNECT options, both of which terminate threads and/or connections to DB2.

The following example assumes that MOD1 and MOD2 have been bound into a plan called BIGPLAN:

Step	MODIFY
1.	`SQL DB2 SET PLAN BIGPLAN`
2.	`RUN MOD1`
3.	`RUN MOD2`
4.	`RUN MOD2`
5.	`SQL DB2 SET PLAN`
6.	`EX RPT1`

The adapter takes the following actions:

1. It sets the application plan. The SET PLAN command tells the adapter that all required SQL statements for subsequent FOCEXECs are contained in BIGPLAN.
2. It opens a thread to plan BIGPLAN. It establishes the thread to BIGPLAN with a CAF OPEN. If there is no prior connection to DB2, it executes a CONNECT before the OPEN.

3. No action. The adapter recalls that the user-set plan is BIGPLAN and that the thread has already been established.
4. No action. The adapter recalls that the user-set plan is BIGPLAN and that the thread has already been established.
5. It returns control to the dynamic adapter. Setting the plan to blank instructs the adapter to return control to the installation default application plan. If you wish to return control to a plan other than the default plan, specify that plan name.

   Another option would be to include the dynamic adapter in BIGPLAN by including the DBRM for RSQL as part of the member list when binding BIGPLAN. This procedure would eliminate the need to reset the plan for dynamic access to DB2. If using this option, skip steps 5 and 6.
6. It opens a thread to the FOCUS dynamic plan. It executes a CAF OPEN to establish the thread to the installation default application plan.

Resource Restrictions

Static compilation creates an Assembler routine that includes SQL statements. Each SQL statement uses a certain number of Assembler resource units (base registers) from the finite number of base registers available. Exceeding this limit might, in some cases, prohibit a large FOCEXEC from compiling successfully. These cases, however, should be extremely rare, and should only involve extremely large generated SQL SELECT statements. The vast majority of applications will be unaffected. These limitations are not unique to FOCUS. They apply to any Assembler program with embedded SQL.

The adapter uses an optimization algorithm to allocate resources from the available pool of 16 registers. If it determines that a limit has been exceeded, it issues an error message during static compilation. Therefore, if your FOCEXEC already exists, the easiest way to determine if it exceeds any limits is to statically compile it.

If the adapter determines that a base register limit has been exceeded during the creation of the Assembler program, it issues the FOC1355 and FOC1359 error messages and terminates compilation. In some cases, the adapter cannot determine whether limits have been exceeded until the program is actually assembled. It flags these errors as addressability errors during the assembly and adds them to the Assembler listing file that it generates.

Appendix A

Additional Topics

This appendix describes the Dialogue Manager status return variable, &RETCODE, stored procedure support, the FOCUS default date and its implications for reporting and modifying an RDBMS table, differences between the relational adapters and standard FOCUS, and remote segment descriptions.

It also discusses additional topics relevant to specific adapters.

In this appendix:

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). See *SQL Codes and Adapter Messages* on page 453 for a list of common SQL return codes.

In a Dialogue Manager request, you can use a -IF command to test the &RETCODE value against a specified SQL or DBC return code. You can then take corrective actions based on the result of the -IF test. An SQL return code of zero (0) indicates a successful execution, a positive return code indicates a warning, and a negative return code indicates an error. For Teradata, any non-zero return code is either a warning or an error.

Example: Testing the Return Code Using &RETCODE

The following FOCEXEC issues the FOCUS CREATE FILE command to create the DB2 EMPINFO table. The FOCEXEC tests for an SQL return code of -601, generated when a DB2 table already exists. The -TYPE command displays message text. The first -TYPE command displays the &RETCODE value. The second explains that the table exists.

```
CREATE FILE EMPINFO
-RUN
-TYPE RETCODE IS  &RETCODE
-IF &RETCODE EQ -601 GOTO DUPL;
-EXIT
-DUPL
-TYPE  THIS TABLE ALREADY EXISTS. SPECIFY ANOTHER TABLE NAME.
-EXIT
```

When you execute the FOCEXEC, the following messages display:

```
>  (FOC1400) SQLCODE IS  : -601/FFFFFDA7
(FOC1421) TABLE EXISTS ALREADY. DROP IT OR USE ANOTHER TABLENAME
(FOC1414) EXECUTE IMMEDIATE ERROR.
RETCODE IS     -601
THIS TABLE ALREADY EXISTS. SPECIFY ANOTHER TABLE NAME
>
```

Since the table already exists, the RDBMS generates the SQL return code -601, and the &RETCODE variable stores the code. The expression in the -IF command is true, and the text messages display.

Note: Another useful Dialogue Manager variable, &FOCERRNUM, stores the last FOCUS or adapter (not RDBMS) error number generated by the execution of a FOCEXEC. See your FOCUS documentation for information about &FOCERRNUM and other statistical variables.

Standard FOCUS and Adapter Differences

In the design of the adapter, every effort has been made to retain compatibility with the FOCUS mainframe product. Concepts described in your FOCUS documentation become portable across environments. 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, the FOCUS FIELDTYPE keyword (FIELDTYPE=I) is not required for the cross-referencing field of the SQL table. The adapter ignores the keyword.

- You may not use the logical relations INCLUDES and EXCLUDES with non-FOCUS data sources in a FOCUS IF or WHERE test.
- GROUP fields are not supported.
- The FOCUS CREATE FILE command requires the DROP option in order to write over an existing table.
- HOLD FORMAT SQL*engine* 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.
- HOLD tables will not be overwritten when a HOLD command is issued with the same name, unless the DROP option is specified in the HOLD command.
- JOIN results (embedded or dynamic) are a function of Master Files or a JOIN specification and an OPTIMIZATION setting. All variations are discussed in *Advanced Reporting Techniques* on page 213.
- Since MODIFY FILE does not create a table if it does not exist, you must create the table prior to executing the MODIFY.
- Some RDBMS views may not be updated. See the documentation for your RDBMS for update restrictions concerning views.
- The maximum number of bytes per Direct SQL Passthru request is 32764 bytes.
- A COMBINE command in a MODIFY procedure may link several non-FOCUS data sources. However, you cannot use the COMBINE command to link a FOCUS data source with an RDBMS table.
- The adapter allows you to change the value of the primary key of a table with the UPDATE command in a MODIFY MATCH request. When modifying a FOCUS data source, you cannot change key field values. Also, with the adapter 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 data sources including RDBMS tables, you must code the keywords TRACE or ECHO on the line following the MODIFY FILE command in order to invoke tracing or echoing.
- **Teradata** and **Oracle** do not support the FOCUS field formats F and Z.
- The adapter does not support the **Oracle** LONG RAW data type, which has been deprecated in Oracle 8.

Adapter for DB2 Stored Procedure Support (CLI Only)

DB2 stored procedures are procedures that are compiled and stored in the DB2 database. These procedures must be developed within DB2 using the CREATE PROCEDURE command. After creating the procedure, you can use the CREATE SYNONYM command to generate a Master and Access File for the stored procedure. You can then run requests against the generated synonym to report against the stored procedure, if you are using the CLI version of the adapter.

The adapter supports stored procedures with IN, OUT, and INOUT parameters.

The output parameter values that are returned by stored procedures are available as result sets. These values form a single-row result set that is transferred to the client after all other result sets are returned by the invoked stored procedure. The names of the output parameters (if available) become the column titles of that result set.

Note that only the output parameters (and the returned value) referenced in the invocation string are returned to the client. As a result, users have full control over which output parameters have values displayed.

FOCUS supports invocation of stored procedures written according to the rules of the underlying DBMS. Note that the examples shown in this section are SQL-based. See the DBMS documentation for rules, languages, and additional programming examples.

Example: Sample Stored Procedure

The following stored procedure uses IN and OUT parameters:

```
CREATE PROCEDURE SPSAMP (IN CNAME CHAR(30), OUT OUTVAL
CHAR(50))
RESULT SETS
1
LANGUAGE
SQL


BEGIN

DECLARE GETTBINFO
CHAR(100);
DECLARE TRIMCNAME
VARCHAR(35);
DECLARE C1 CURSOR WITH RETURN FOR
S1;


SET TRIMCNAME = '''%' || TRIM(CNAME) ||
'%''';
SET GETTBINFO
=
 'SELECT COLCOUNT,NAME FROM SYSIBM.SYSTABLES WHERE CREATOR LIKE ' ||
TRIMCNAME;
SET OUTVAL='TABLES FOR USER ' ||
CNAME;


PREPARE S1 FROM
GETTBINFO;
OPEN
C1;


END
```

The following CREATE SYNONYM command generates a Master File and Access File for the stored procedure. Note that the supplied input parameter is a value, while the output parameter is represented by a question mark (?). For information about the CREATE SYNONYM command, see *Generating a Master and Access File Using the CREATE SYNONYM Command*.

```
CREATE SYNONYM FSPSAMP DROP
  FOR USER1.SPSAMP
  DBMS DB2
  AT CON1
  STOREDPROCEDURE
  PARMS "'USER1',?"
END
```

The following Master File is generated as a result of the CREATE SYNONYM command.

```
FILENAME=FSPSAMP, SUFFIX=DB2     , $
  SEGMENT=INPUT, SEGTYPE=S0, $
    FIELDNAME=CNAME, ALIAS=P0001, USAGE=A30, ACTUAL=A30,
      MISSING=ON, ACCESS_PROPERTY=(NEED_VALUE), $
  SEGMENT=OUTPUT, SEGTYPE=S0, PARENT=INPUT, $
    FIELDNAME=OUTVAL, ALIAS=P0002, USAGE=A50, ACTUAL=A50,
      MISSING=ON, $
  SEGMENT=ANSWERSET1, SEGTYPE=S0, PARENT=INPUT, $
    FIELDNAME=COLCOUNT, ALIAS=COLCOUNT, USAGE=I6, ACTUAL=I2, $
    FIELDNAME=NAME, ALIAS=NAME, USAGE=A128V, ACTUAL=A128V, $
```

The following Access File is generated as a result of the CREATE SYNONYM command.

```
SEGNAME=INPUT,
   CONNECTION=CON1,
   STPNAME=USER1.SPSAMP, $
 SEGNAME=OUTPUT,
   STPRESORDER=0, $
 SEGNAME=ANSWERSET1,
   STPRESORDER=1, $
```

The following request invokes the stored procedure, specifying a value for the input parameter, CNAME.

```
TABLE FILE FSPSAMP
PRINT NAME COLCOUNT
WHERE CNAME EQ 'SYSIBM'
HEADING
"<OUTVAL "
" "
END
```

The output is:

```
PAGE     1

TABLES FOR USER SYSIBM

COLCOUNT  NAME
--------  ----
      10  SYSOBDS
      16  SYSCONTEXT
       5  SYSCTXTTRUSTATTRS
       7  SYSCONTEXTAUTHIDS
      33  SYSCOPY
      16  SYSCOLAUTH
      41  SYSCOLUMNS
       7  SYSFOREIGNKEYS
      59  SYSINDEXES
      35  SYSINDEXPART
       8  SYSKEYS
      16  SYSRELS
       9  SYSSYNONYMS
      30  SYSTABAUTH
      46  SYSTABLEPART
      59  SYSTABLES
```

Adapter for Oracle Stored Procedure Support

The Adapter for Oracle allows you to use Direct SQL Passthru to call an Oracle stored procedure using any necessary input parameters. These procedures need to be developed within Oracle using the CREATE PROCEDURE command. They can either return an answer set or a return code indicating the result of the invocation when no data is returned. When an answer set is returned, the adapter creates a temporary Master File named SQLOUT that can be used to generate reports against the answer set.

Note: Oracle stored procedure support only allows the processing of one answer set per invocation. Attempts to retrieve multiple answer sets generate an error message.

Syntax: How to Invoke an Oracle Stored Procedure

```
{ENGINE|SQL} SQLORA EX spname parm1,parm2,...,parmn;
```

where:

spname

Is the Oracle stored procedure, it is of the form:

packagename.procedurename

parm1,parm2,...,parmn

Are the parameter values to be supplied as input for the procedure and must be scalar in type. That is, only single values are acceptable as opposed to vector or array types. Non-supported parameter types will generate an error message.

Reference: Rules for Oracle Stored Procedures

- Only scalar input parameters are allowed, but not required.
- Output parameters are not allowed, except for optional cursor type parameters.
- A cursor used for output parameters must be defined with:
  - The TYPE statement in a PACKAGE or PROCEDURE.
  - An associated record layout of the answer set to be returned.
- The cursor must be opened in the procedure.
- No fetching is allowed from the output parameter cursors in the stored procedure. The Adapter for Oracle fetches the answer set.
- Any error messages must be issued using the RAISE APPLICATION ERROR method.

Note: Any application error that is issued by the stored procedure is available in the variable &ORAMSGTXT. The variable &ORAMSGTXT can be used in conjunction with &RETCODE to process errors and/or messages that are returned by the stored procedure.

Syntax: How to Set a Maximum Number of Input Parameters for Oracle Stored Procedures

An adapter parameter can be used to set the maximum number of input parameters.

```
{ENGINE|SQL} SQLORA SET SPMAXPRM value
```

where:

value

Is a numeric value indicating the maximum number of input parameters that may be entered for stored procedures. The default value is 256. This value is displayed by the SQL SQLORA ? query.

Example: **Sample Oracle Stored Procedure**

```
CREATE OR REPLACE PACKAGE pack1 AS
TYPE nfrectype IS RECORD (
employee NF29005.EMPLOYEE_ID5%TYPE,
ssn5     NF29005.SSN5%TYPE,
l_name   NF29005.LAST_NAME5%TYPE,
f_name   NF29005.FIRST_NAME5%TYPE,
birthday NF29005.BIRTHDATE5%TYPE,
salary   NF29005.SALARY5%TYPE,
joblevel NF29005.JOB_LEVEL5%TYPE);
TYPE nfcurtype IS REF CURSOR RETURN nfrectype ;
PROCEDURE proc1(c_saltable IN OUT nfcurtype);
END pack1 ;
/
CREATE OR REPLACE PACKAGE BODY pack1 AS
PROCEDURE proc1 (c_saltable IN OUT nfcurtype)
IS
BEGIN
OPEN c_saltable FOR SELECT
EMPLOYEE_ID5,SSN5,LAST_NAME5,FIRST_NAME5,BIRTHDAT
E5,SALARY5,JOB_LEVEL5 FROM NF29005;
END proc1 ;  -- end of procedure
END pack1; -- end of package body
/
```

Example: **Invoking an Oracle Stored Procedure That Returns an Answer Set**

This example invokes a stored procedure that returns an answer set. The results can then be displayed using a TABLE request against the Master File SQLOUT that is created by the adapter to describe the returned answer set:

```
SQL SQLORA EX  pkg1.tblproc  parma;
TABLE FILE SQLOUT
PRINT *
END
```

Example: **Invoking an Oracle Stored Procedure That Returns a Return Code**

In the following example, the return code from the Oracle stored procedure is inspected to determine the flow of control.

```
SQL SQLORA EX pkg1.insrtproc parm1;
-IF &RETCODE EQ 0 GOTO OK;
-TYPE &ORAMSGTXT
-OK;
```

Adapter for Teradata Stored Procedure and Macro Support

SQL Passthru is supported for Teradata macros and stored procedures.

Macros need to be developed within Teradata using the CREATE or REPLACE MACRO command. Procedures need to be developed within Teradata using the CREATE PROCEDURE command.

You must call a macro in the same transaction mode in which it was compiled. To find out which transaction mode is currently in effect, issue the HELP SESSION command:

```
SQL SQLDBC HELP SESSION;
TABLE FILE SQLOUT PRINT TRANSACTION_SEMANTICS
END
```

Before you can call a stored procedure or a macro, you must set the connection accordingly, by issuing the SET MACRO command

```
SQL SQLDBC SET MACRO {ON|OFF}
```

where:

ON

Enables one to call a macro. This is the default.

OFF

Enables one to call a stored procedure.

Example: **Calling a Macro**

This is an example of the syntax for calling a macro:

```
ENGINE SQLDBC
EX SAMPLE PARM1,PARM2,PARM3...;
TABLE FILE SQLOUT
END
```

Example: **Calling a Stored Procedure**

The supported syntax to call a stored procedure is shown below.

```
ENGINE SQLDBC
EX SAMPLE PARM1,PARM2,PARM3...;
TABLE FILE SQLOUT
END
```

When using the adapter with:

- **ODBC,** all scalar parameters (IN, OUT, and INOUT) are supported.
- **CLI,** scalar IN parameters, and INOUT parameters in IN mode, are supported.

Example: **Sample Teradata Stored Procedure**

```
CREATE OR REPLACE PACKAGE pack1 AS
TYPE nfrectype IS RECORD (
employee NF29005.EMPLOYEE_ID5%TYPE,
ssn5     NF29005.SSN5%TYPE,
l_name   NF29005.LAST_NAME5%TYPE,
f_name   NF29005.FIRST_NAME5%TYPE,
birthday NF29005.BIRTHDATE5%TYPE,
salary   NF29005.SALARY5%TYPE,
joblevel NF29005.JOB_LEVEL5%TYPE);
TYPE nfcurtype IS REF CURSOR RETURN nfrectype ;
PROCEDURE proc1(c_saltable IN OUT nfcurtype);
END pack1 ;
/
CREATE OR REPLACE PACKAGE BODY pack1 AS
PROCEDURE proc1 (c_saltable IN OUT nfcurtype)
IS
BEGIN
OPEN c_saltable FOR SELECT
EMPLOYEE_ID5,SSN5,LAST_NAME5,FIRST_NAME5,BIRTHDAT
E5,SALARY5,JOB_LEVEL5 FROM NF29005;
END proc1 ;  -- end of procedure
END pack1; -- end of package body
/
```

Default Date Considerations

The default date value the adapter uses for the RDBMS DATE data type changed starting with FOCUS Version 6.8. If you use the FOCUS MODIFY facility to maintain RDBMS tables containing DATE columns, this change may have some impact on your applications. If you have a read-only version of the adapter, or if your site does not use MODIFY, you will not be affected.

You can control the default date value with the data adapter SET DEFDATE command.

The Default Date Value

The adapter uses the default date value in conjunction with RDBMS DATE columns described in a Master File as ACTUAL=DATE. Under certain circumstances, if your MODIFY procedure does not provide a value for a DATE column, the adapter substitutes the default value. In FOCUS Version 6.8 and up, the default value changed to make adapter date behavior more closely resemble that of the FOCUS DBMS.

In releases of FOCUS prior to 6.8, the adapter 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 (or base) value has always been '1900-12-31'.

With the FOCUS DBMS, base date values print as blanks in report output by default. (This discussion assumes that the FOCUS DATEDISPLAY parameter is OFF, the default. The SET DATEDISPLAY = ON command displays the base date in FOCUS reports. See your FOCUS documentation for details.) The old adapter default DATE value always displayed on reports.

When the DEFDATE value is NEW, the adapter base date value is identical to the FOCUS DBMS base date: 1900-12-31, and it displays the same way in reports.

The Adapter SET DEFDATE Command

You can control the adapter default date with the following adapter SET command

```
{ENGINE|SQL} [sqlengine] SET DEFDATE {NEW|OLD}
```

where:

sqlengine

Indicates the target RDBMS. Valid values are DB2, SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

OLD

The adapter supplies the old default date, '1901-01-01'.

NEW

The adapter supplies the new default date, '1900-12-31'. NEW is the default value.

Effects of DEFDATE on Existing Applications

With the new default date, TABLE requests no longer require DEFINE or COMPUTE commands in order to display blanks instead of default dates. Applications that test for default values require updating to reflect the new default date.

For the sake of consistency, you may wish to update your databases to change old default values to the new default. You can use the following code to change old default values

```
SQL [sqlengine] UPDATE creator.tablename
    SET date_column = '1900-12-31' WHERE date_column = '1901-01-01'
END
```

where:

sqlengine

Is DB2,SQLDBC, SQLIDMS, or SQLORA. Omit if you previously issued the SET SQLENGINE command.

The change in the default date value affects only the selection of default values supplied by MODIFY procedures under certain circumstances, and whether default values appear on a FOCUS report. In all other respects (for example, in screening conditions), the default value is a valid date value. If you use a report writer other than FOCUS, any default date values in the RDBMS table will display as either '1901-01-01' or '1900-12-31', depending on the FOCUS release that placed them in the table and the value of the DEFDATE parameter.

Chart: FOCUS Date Values for User Input Values

The following chart summarizes how FOCUS stores date values in response to specific user input values for non-conditional and conditional data entry in MODIFY:

			If DEFDATE is NEW:		
Non-conditional data entry (<date_column>) in MODIFY					
ON	blank	blank	1901/01/01	blank	1900-12-31
ON	'.'	NODATA	NODATA	NODATA	null
ON	0 (zero)	blank	1901/01/01	blank	1900-12-31
OFF	blank	blank	1901/01/01	blank	1900-12-31
OFF	'.'	input value rejected by MODIFY			
OFF	0 (zero)	blank	1901/01/01	blank	1900-12-31
Conditional data entry (<date_column) in MODIFY					
ON	blank	NODATA	NODATA	NODATA	null
ON	'.'	NODATA	NODATA	NODATA	null
ON	0 (zero)	blank	1901/01/01	blank	1900-12-31
OFF	blank	blank	1901/01/01	blank	1900-12-31
OFF	'.'	input value rejected by MODIFY			
OFF	0 (zero)	blank	1901/01/01	blank	1900-12-31

Note:

- FOCUS displays the NODATA value whenever a column contains a null value. The default NODATA value is the period ('.').

- Column 1 shows the MISSING parameter in the Master File for the column described as ACTUAL=DATE.
- Column 2 shows the value the MODIFY user enters for that column.
- Column 3 shows, for the FOCUS DBMS, how that value would display in FOCUS report requests. If a blank appears on the report, the stored value is '1900-12-31'.
- Column 4 shows, for an RDBMS table, how the entered value displays in a FOCUS report if DEFDATE is OLD. You can see the value FOCUS stored, since it appears on the report instead of a blank.
- Column 5 shows, again for an RDBMS table, how the entered value displays on a FOCUS report provided DEFDATE is NEW. The report output is now identical to the report produced for the FOCUS DBMS (Column 3).
- Column 6 shows the actual default value stored in the RDBMS table for values entered when DEFDATE is NEW.

The chart shows how values are stored and displayed for conditional (single caret) and unconditional (double caret) fields.

Note: If you enter a blank for a conditional field, and if a value already exists for that field, the field is not updated. The chart shows what happens when you enter blanks for conditional fields that have no prior values.

If you want to be sure that FOCUS does not store a default value regardless of the user's input, have the application program check the entered value. If the user enters 0 (zero) or blank, COMPUTE the date field as 'MISSING' to make FOCUS set the column to NULL in UPDATE or INSERT statements. This technique works only if the date column allows nulls and is described to FOCUS as MISSING=ON.

See your FOCUS documentation if you are not familiar with the terms in the preceding discussion.

Remote Segment Descriptions

Remote segment descriptions simplify the process of describing hierarchies of RDBMS tables to FOCUS. You can use them for Master and Access Files that provide read-only access to RDBMS tables. You cannot use the same descriptions for MODIFY procedures.

The adapter allows you to create multi-table Master and Access Files that define RDBMS tables or views as segments in the description (see *Multi-Table Structures* on page 99). Each Master 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 Master Files to contain a segment description for the same RDBMS table. If a table's description is detailed in one Master File, you can automatically incorporate that description in other Master Files. The syntax is

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

where:

segname

Is identical to the SEGNAME in the Master File that contains the full description of the columns in the RDBMS table (the remote Master File).

parent

Is the parent of the segment.

KL

Describes one-to-many relationships.

KLU

Describes unique relationships.

filename

Is the name of the remote Master File that contains the full description of the fields in the segment.

Note:

- You may not use the attributes CRKEY and CRSEGNAME.
- If a Master File that contains a CRFILE cross-reference to a segment in another Master File does not contain a declaration for that segment in its own Access File, the adapter issues a FOC1351 message as a warning. The adapter 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 Master File, the adapter 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 Master Files containing remotely-described segments in CREATE FILE commands or MODIFY procedures.

SEGTYPEs KL and KLU describe segments whose field attributes are described in other Master Files that FOCUS may read at run-time. You can use remote segment descriptions for situations in which several Master Files introduce different views on the same collection of RDBMS tables. You describe the fields of one or several tables in one Master File, and refer to this first file from other Master Files 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 Master File that contains the complete segment definition. For example:

```
SEGNAME=DEPT, PARENT=EMP, SEGTYPE=KL, CRFILE=MAINFILE, $
```

The adapter obtains the descriptions of fields in the DEPT segment in this file from the DEPT segment in the MAINFILE Master File, 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 Master File, that specification becomes the default for subsequently described segments. If you later wish to describe a segment locally (using the traditional method), you must re-specify 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 Master File 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 Master File 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 Master File 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 within the local Master File (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 Master File that contains remote segment declarations must have its own Access File for defining the relationships between all of its segments (including the remote segments). This local Access File overrides the Access File corresponding to the CRFILE Master File, if one exists.

Example: Using a Remote Segment Description

The following example shows the ECOURSE Master File with the COURSE segment described remotely:

```
FILENAME=ECOURSE     ,SUFFIX=DB2, $
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=DPT         ,USAGE=A10   ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL        ,USAGE=P9.2  ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE  ,ALIAS=CJC         ,USAGE=A3    ,ACTUAL=A3,$
 FIELD=ED_HRS        ,ALIAS=OJT         ,USAGE=F6.2  ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN    ,ALIAS=BONUS_PLAN  ,USAGE=I4    ,ACTUAL=I4,$
SEGNAME=COURSE ,SEGTYPE=KL  ,PARENT=EMPINFO, CRFILE=COURSE,$
```

The corresponding Access File is:

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1,
   WRITE = YES,  DBSPACE = PUBLIC.SPACE0,$
SEGNAME = COURSE, TABLENAME = "USER1"."COURSE", KEYS = 2,
   WRITE = YES,     DBSPACE = PUBLIC.SPACE0,
     KEYFLD = EMP_ID, IXFLD = WHO,$
```

Long Field Name Considerations

For a complete discussion of long and qualified field name support, consult your FOCUS documentation.

Note: If a request written for a prior release of FOCUS contains truncated field names, and if you rerun AUTODB2 or AUTODBC on the table, the AUTO facility may generate field names in the new Master File that are different from the names it generated before long field names were supported.

Limitations on Long Field Names

The following limitations apply to long field names and aliases:

- In a Master File, the FIELDNAME and ALIAS attributes may not be qualified.
- Long field names are truncated to 12 characters in CHECK FILE PICTURE operations.
- In ModifyTalk, field name, title, and field length may not exceed 80 characters.

Describing a Long Field Name in the Access File (AUTODBC)

An alternative method describes field names greater than 12 characters in the Access File.

For this method, specify the field name declaration with a 12-character (or less) field name in the Master File; leave the ALIAS value blank. In the corresponding Access File, specify a long field name declaration after the appropriate segment declaration.

A long field name declaration consists of two attributes, FIELD and ALIAS. The FIELD value is the 12-character field name from the Master File. The ALIAS value identifies the full RDBMS column name, up to 30 characters. (See *Describing Tables to FOCUS* on page 55 for naming conventions.) The long field name declarations must follow their associated segment declarations and correspond to the field order of the Master File.

For example, this EMPINFO Master File for a Teradata data source contains blank ALIAS values for the DEPARTMENT and SALARY fields. The column names DEPARTMENT_CD and CURRENT_SALARY are longer than 12 characters

```
FILENAME=EMPINFO, SUFFIX=SQLDBC, $
SEGNAME=EMPINFO, SEGTYPE=S0, $
    FIELD=EMP_ID     ,ALIAS=EID      ,USAGE=A9 ,ACTUAL=A9 ,$
    FIELD=LAST_NAME  ,ALIAS=LNAME    ,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=P9.2,ACTUAL=P8 ,$
    FIELD=CURR_JOBCODE,ALIAS=CJC      ,USAGE=A3 ,ACTUAL=A3 ,$
    FIELD=ED_HRS     ,ALIAS=OJT      ,USAGE=D6.2,ACTUAL=D8 ,
                MISSING=ON, $
    FIELD=BONUS_PLAN ,ALIAS=BONUS_PLAN,USAGE=I4 ,ACTUAL=I4 ,$
```

The long field name declarations in the Access File are specified:

```
SEGNAME=EMPINFO, TABLENAME=EMPINFO, KEYS=1, WRITE=YES,FALLBACK=YES,$
    FIELD=DEPARTMENT, ALIAS=DEPARTMENT_CD, $
    FIELD=SALARY   , ALIAS=CURRENT_SALARY, $
```

Determining DB2 Decimal Notation at Run-time

The Adapter for DB2 generates the appropriate SQL for whichever decimal point notation was selected (period or comma) when the IBM DB2 software was installed, regardless of the FOCUS Continental Decimal Notation (CDN) parameter setting. You can then set the CDN parameter to control how FOCUS displays the numbers.

Continental decimal notation uses a comma to mark the decimal position in a number and uses periods, blanks, or single quotation marks for separating significant digits into groups of three. In FOCUS, to set the CDN parameter, issue the following

```
SET CDN = cdnvalue
```

where:

cdnvalue

Can be one of the following:

OFF uses a period as the decimal separator and a comma as the thousands separator. OFF is the default.

`ON` uses a comma as the decimal separator and a period as the thousands separator.

`SPACE` uses a comma as the decimal separator and a space as the thousands separator.

`QUOTE` uses a comma as the decimal separator and a single quotation mark as the thousands separator.

Regardless of the CDN setting, the adapter generates the appropriate SQL for the decimal point notation (comma or period) selected when the IBM DB2 software was installed using the supplied IBM Application Programming Defaults Panel DSNTIPF. For more information on setting IBM DB2 application defaults, refer to the *IBM DB2 Administration Guide*. The SET CDN command controls how numbers are displayed in the FOCUS environment.

For example, given the number 3,045,000.76:

`SET CDN = ON` Displays the number as 3.045.000,76.

`SET CDN = OFF` Displays the number as 3,045,000.76.

Note: In order for this feature to function correctly, the proper DB2 DSNEXIT data set (for example, DSN910.SDSNEXIT) must be first in the sequence of data sets allocated to DDNAME STEPLIB. If this is not possible at your site, you can place the DSNEXIT data set in the FOCLIB concatenation instead, as FOCUS searches FOCLIB before STEPLIB.

CALLDB2: Invoking Subroutines Containing Embedded SQL

CALLDB2, a FOCUS subroutine included in the Adapter for DB2, provides a standard method for invoking user-written subroutines that make embedded SQL calls to DB2.

Prior to CALLDB2, you had to bind subroutines containing embedded SQL with the application plan for the adapter, so they generated increased administrative overhead. The application programmer was responsible for run-time plan management (making sure to invoke the correct application plan with the adapter SET PLAN command), and for instructing the adapter to establish a thread to DB2 before invoking the subroutine.

CALLDB2 eliminates the requirement to include adapter Database Request Modules (DBRMs) in the application plan for the subroutine. In addition, CALLDB2 ensures that subroutines do not disrupt the current Adapter for DB2 thread. The adapter automatically closes any existing thread to DB2 and opens a new thread to the application plan for the subroutine. After the subroutine completes, the adapter restores the original environment. With CALLDB2, you can invoke subroutines when no prior thread to DB2 exists.

Note: The CALLDB2 feature requires that the Adapter for DB2 be installed to use the Call Attachment Facility (CAF). You can verify this by issuing the SQL DB2 ? command. Call Attach should be ON.

For additional documentation, consult your FOCUS documentation. Read it before writing subroutines that use CALLDB2.

Note: Any additions, differences, or limitations described in this topic override that document.

Syntax: How to Invoke CALLDB2

CALLDB2 is an Information Builders-supplied subroutine you invoke from a Dialogue Manager program. The syntax is

```
-SET &var = CALLDB2('subrtine','planname',input1,input2,...,['format']);
```

where:

&var

Is the Dialogue Manager variable to receive the value returned by the subroutine.

subrtine

Is a subroutine that uses embedded SQL. The subroutine name can be up to eight characters long (unless you are writing the subroutine in a language that allows less) and must be enclosed in single quotation marks. The first character must be a letter (A-Z). Each additional character can be a letter or a number. You must pad the name on the right with blanks if it is less than eight characters long.

planname

Is the DB2 application plan for your subroutine. The plan name can be up to eight characters long and must be enclosed in single quotation marks. You must pad the name on the right with blanks if it is less than eight characters long.

input1...

Are the input arguments. You must know what arguments the subroutine requires, their formats, and the order in which to specify them. You can include up to 26 input arguments (normally, user written subroutines allow 28, but CALLDB2 uses two of them for subroutine name and plan name).

format

Is the format of the output value, enclosed in single quotation marks. This parameter is optional. If you do not provide a format, the default format is determined by the format of the last element in the calling list. If your program does not return an output value, the default return value is the last element in the calling list. In cases of error, the returned value is '*ERROR* '.

Although Dialogue Manager variables contain only alphanumeric data, they can serve as numeric arguments. The -SET command converts their alphanumeric values to double-precision format before passing them to the subroutine. However, if a subroutine returns a numeric value and you set a Dialogue Manager variable to this value, FOCUS truncates the output to an integer and converts it to a character string before storing it in the variable.

Note: You can invoke CALLDB2 only from Dialogue Manager -SET, -IF, or -TSO RUN control statements. You cannot invoke it from DEFINE, COMPUTE, MODIFY VALIDATE or IF commands, or from Financial Modeling Language (also known as Extended Matrix Reporting) RECAP commands.

Creating CALLDB2-Invoked Subroutines

The steps for creating a CALLDB2-invoked subroutine are:

1. Write the subroutine program logic.
2. Precompile the subroutine.
3. Compile or assemble the subroutine.
4. Link-edit the subroutine.
5. BIND the subroutine.

Introduction to Adapters for Relational Data Sources on page 17, includes a discussion of bind concepts.

Procedure: How to Write the Subroutine

Your subroutine should assume that the adapter will do all thread handling. This includes connection to DB2, disconnection from DB2, and the opening and closing of threads. In addition, do not code COMMIT WORK or ROLLBACK WORK statements within the subroutine itself. The adapter tracks open logical units of work (LUWs), and it cannot detect a COMMIT WORK or ROLLBACK WORK in a subroutine.

Since CALLDB2 is an adapter command, its behavior is affected by the adapter SET AUTO*action* ON *event* command (see *Controlling Connection Scope* on page 295 for more information about this command). If AUTOCOMMIT is set ON COMMAND, the default setting, the adapter assumes there is an open LUW and automatically issues a COMMIT WORK at the end of a CALLDB2 subroutine.

If your subroutine requires the adapter to issue a conditional COMMIT or ROLLBACK WORK:

1. Set AUTOCOMMIT on FIN immediately before entering the subroutine to prevent the adapter from generating an automatic COMMIT WORK at the end of the CALLDB2 command.
2. Exit the subroutine and return a value to the Dialogue Manager procedure indicating which action, COMMIT or ROLLBACK, the procedure should take on behalf of the subroutine.
3. After your Dialogue Manager procedure issues the COMMIT or ROLLBACK WORK command, reset AUTOCOMMIT on COMMAND to restore the default adapter environment.

The example in *Sample Procedure for Invoking CALLDB2* illustrates this technique.

A CALLDB2 subroutine is also subject to the rules of adapter plan management. Two types of plan management are defined for the adapter, basic and extended. If you use basic plan management, the adapter automatically switches plans as required. The adapter SET PLAN command invokes the alternative, extended plan management. In this case, the application program, not the adapter, manages plans. The discussion in *Plan Management in DB2* on page 413 in *Static SQL (DB2)* on page 401, applies to CALLDB2 subroutines.

Example: Preparing a Static Subroutine for use With CALLDB2

The following very simple COBOL program illustrates the steps involved in preparing a static subroutine for use with CALLDB2:

```
    ID DIVISION.
    PROGRAM-ID. TESTDB2.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
        EXEC SQL INCLUDE SQLCA END-EXEC.
    LINKAGE SECTION.
    01  VALUE1                PIC X(8).
    01  VALUE2                PIC X(8).
    01  RETCODE               PIC S9(9) COMP SYNC.
1.  PROCEDURE DIVISION USING VALUE1, VALUE2, RETCODE.
2.      EXEC SQL WHENEVER SQLERROR GO TO PROGRAM-EXIT
        END-EXEC.
3.      EXEC SQL INSERT INTO USER1.TESTDB2
           (COL1)
           VALUES(:VALUE1)
        END-EXEC.
        EXEC SQL INSERT INTO USER1.TESTDB2
           (COL1)
           VALUES(:VALUE2)
        END-EXEC.
4.  PROGRAM-EXIT.
        MOVE SQLCODE TO RETCODE.
        GOBACK.
    //**
```

The subroutine:

1. Accepts two input values and returns a value named RETCODE.
2. Checks the SQLCODE after each SQL command. If it encounters a negative SQLCODE, the subroutine branches to PROGRAM-EXIT.
3. Issues two SQL INSERT commands that place the input values into a single-column DB2 table.
4. Passes the value of the SQLCODE back to FOCUS. If the procedure encountered an error, it returns the negative SQLCODE from that error back to the Dialogue Manager routine. Otherwise, it passes the last SQLCODE, which should be zero, back to the Dialogue Manager routine. The Dialogue Manager routine will check the value of the returned SQLCODE and issue an SQL COMMIT WORK or ROLLBACK WORK depending on the returned value.

The following steps prepare the sample subroutine, TESTDB2, for execution. JCL for each step is provided for illustrative purposes. In general, these steps apply to any user-written subroutine containing embedded SQL. Your JCL will vary depending on such factors as choice of language and site-specific requirements or conventions. Consult the appropriate IBM manual for help with each step.

Example: Precompiling the Subroutine

This example includes the sample program as in-stream input to the precompilation step:

```
//Job card goes here...
//*****************************************************************
//*** PRECOMPILE COBOL II CODE CONTAINING EMBEDDED (STATIC) SQL ***
//*****************************************************************
//PC       EXEC PGM=DSNHPC,
//          PARM='HOST(COB2),OPTIONS,SOURCE,XREF'
//STEPLIB  DD DISP=SHR,DSN=DSN910.SDSNLOAD
//DBRMLIB  DD DISP=SHR,DSN=prefix.DBRMLIB.DATA(TESTDB2)
//SYSCIN   DD DSN=prefix.PRECOMP.OUTPUT,DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,SPACE=(TRK,(20,10),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=4080)
//SYSPRINT DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//SYSUT1   DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT2   DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT3   DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT4   DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT5   DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSIN    DD *
ID DIVISION.
  PROGRAM-ID. TESTDB2.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
    EXEC SQL INCLUDE SQLCA END-EXEC.

LINKAGE SECTION.
01  VALUE1                PIC X(8).
01  VALUE2                PIC X(8).
01  RETCODE               PIC S9(9) COMP SYNC.
PROCEDURE DIVISION USING VALUE1, VALUE2, RETCODE.
    EXEC SQL WHENEVER SQLERROR GO TO PROGRAM-EXIT
    END-EXEC.
    EXEC SQL INSERT INTO USER1.TESTDB2
         (COL1)
         VALUES(:VALUE1)
    END-EXEC.
    EXEC SQL INSERT INTO USER1.TESTDB2
         (COL1)
         VALUES(:VALUE2)
    END-EXEC.
PROGRAM-EXIT.
    MOVE SQLCODE TO RETCODE.
    GOBACK.
//**
```

The precompilation process produces a modified source program (the data set allocated to DDNAME SYSCIN) and a database request module or DBRM (member TESTDB2 in the data set allocated to DDNAME DBRMLIB).

Example: Compiling or Assembling the Subroutine

After precompilation, compile or assemble the modified source program:

```
//Job card goes here...
//************************************
//*** COBOL II COMPILE OF SOURCE CODE ***
//************************************
//COB2     EXEC PGM=IGYCRCTL,PARM='OBJECT',REGION=1024K
//STEPLIB  DD DSN=VSCOBOL.COBCOMP,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLIN   DD DSN=prefix.OBJECT.MODULE,DISP=(NEW,CATLG,DELETE),
//            UNIT=SYSDA,SPACE=(TRK,(20,10),RLSE)
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN    DD DSN=prefix.PRECOMP.OUTPUT,DISP=(OLD,DELETE)
//**
```

The output of the compilation in this example is the data set allocated to DDNAME SYSLIN.

Example: Link-Editing the Subroutine

Link-edit the output of the compilation or assembly to produce a run-time module:

```
//Job card goes here...
//*********************************
//*** LINK EDIT OBJECT MODULE(S) ***
//*********************************
//LKED     EXEC PGM=IEWL,PARM='LIST,XREF,LET,MAP',REGION=512K
//SYSLIN   DD DSN=prefix.OBJECT.MODULE,DISP=(OLD,DELETE)
//         DD *
 INCLUDE DB2(DSNALI)
 INCLUDE DB2(DSNHADDR)
 MODE AMODE(31),RMODE(ANY)
 NAME TESTDB2(R)
/*
//SYSLMOD  DD DSN=prefix.TESTDB2.LOAD,DISP=SHR
//SYSLIB   DD DSN=VSCOBOL.COBLIB,DISP=SHR
//DB2      DD DSN=DSN910.SDSNLOAD,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPRINT DD SYSOUT=*
//
/*
```

The run-time module in this example is the data set allocated to DDNAME SYSLMOD. You must include DSNALI, the language interface, during link-edit. DSNALI is in the DB2 load library (DSN910.SDSNLOAD in the example). The link-edit of the sample program also includes DSNHADDR, used to move values to and from the SQL Descriptor Area (SQLDA). DSNHADDR is required only for COBOL programs.

Example: Binding the Subroutine

Now, bind the DBRM created during precompilation into an application plan, and store it in the DB2 database:

```
//Job card goes here...
//*********************************
//*** BIND THE PLAN               ***
//*********************************
//BIND EXEC   PGM=IKJEFT01,DYNAMNBR=10
//STEPLIB   DD DSN=DSN910.SDSNLOAD,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSTSPRT  DD SYSOUT=*
//DBRMLIB   DD DSN=prefix.DBRMLIB.DATA,DISP=SHR
//SYSTSIN   DD *
DSN SYSTEM(DB2P)
BIND PLAN(TESTDB2) MEMBER(TESTDB2) -
                   ACTION(REPLACE) ISOLATION(RR)
END
//*
```

The BIND subcommand creates an application plan called TESTDB2 that contains SQL statements from the DBRM named TESTDB2.

Refer to the *DB2 Command and Utility Reference* for an explanation of possible BIND parameters.

Note: If you make any changes to your subroutine, you have to repeat the precompilation, compilation or assembly, link-edit, and BIND steps.

CALLDB2 Run-time Requirements

```
ALLOC F(USERLIB)  DA('prefix.TESTDB2.LOAD' -
                     'prefix.FOCSQL.LOAD'  -
                     'prefix.FOCLIB.LOAD'  -
                     'prefix.FUSELIB.LOAD') SHR REU
```

or:

```
//STEPLIB   DD DSN=prefix.TESTDB2.LOAD,DISP=SHR
//          DD DSN=prefix.FOCSQL.LOAD,DISP=SHR
//          DD DSN=prefix.FOCLIB.LOAD,DISP=SHR
//          DD DSN=prefix.FUSELIB.LOAD,DISP=SHR
```

Example: Sample Procedure for Invoking CALLDB2

This sample Dialogue Manager procedure invokes the TESTDB2 subroutine:

```
1.  SQL DB2 SET AUTOCOMMIT ON FIN
    -RUN
    -*
2.  -SET &RUNOK = CALLDB2('TESTDB2 ','TESTDB2 ','BLUE','RED','I4') ;
    -RUN


3.  -SET &LUW = IF &RUNOK NE 0 THEN 'SQL DB2 ROLLBACK WORK' ELSE
    -           'SQL DB2 COMMIT WORK' ;
4.  &LUW
    -RUN


5.  SQL DB2 SET AUTOCOMMIT ON COMMAND
    -RUN
```

The procedure:

1. Sets AUTOCOMMIT on FIN to prevent the adapter from issuing an automatic COMMIT WORK at the end of the subroutine.
2. Invokes CALLDB2 for subroutine TESTDB2. The return code from the subroutine will be stored in Dialogue Manager variable &RUNOK.
3. Tests the value of the return code and, depending on the result of the test, stores either the SQL COMMIT WORK or the SQL ROLLBACK WORK command in Dialogue Manager variable &LUW.
4. Issues the COMMIT or ROLLBACK command.
5. Sets AUTOCOMMIT on COMMAND to restore data adapter default behavior.

The DB2 Distributed Data Facility

The adapter fully supports the DB2 Distributed Data Facility (DDF). The following sections describe adapter support for DB2 DDF.

File Descriptions for DDF

To identify the DB2 subsystem, include the DB2 LOCATION attribute in the TABLENAME parameter of the Access File. This identifies the table to FOCUS as a remote table. The syntax is

```
TABLENAME=[location.][creator.]table
```

where:

location

Is the DB2 subsystem location name; 16 characters maximum.

creator

Is the authorization ID of the table's creator, 8 characters maximum.

table

Is the name of the RDBMS table or view, 18 characters maximum.

Note:

- The TABLENAME value can have be a maximum of 44 characters (including the required periods).
- AUTODB2 can create Access Files that specify three-part table names. For information, see *Automated Procedures* on page 113.

Accessing Tables at Different Locations

The adapter allows you to join tables at different locations. FOCUS performs the join since DB2 does not permit a single SQL statement to reference data at more than one location.

The adapter determines that FOCUS must manage the join based on the presence and value of the LOCATION attribute in the TABLENAME parameter of one or more of the tables referenced in the request. If all remote tables have the same LOCATION attribute, the adapter optimizes the join to the RDBMS. If you use local aliases for remote tables (and, therefore, have no LOCATION attribute in the table name), you must SET OPTIMIZATION=OFF for each request that joins tables from more than one location.

FOCUS automatically appends a "FOR FETCH ONLY" clause to SELECT statements generated by TABLE requests. This clause assists the DB2 optimizer in access path selection and offers substantial performance improvements.

DB2 DRDA Support

The Adapter for DB2 supports IBM's Distributed Relational Database Architecture (DRDA), including:

- The Level 1 DRDA CONNECT command.
- Level 2 commands such as SET CONNECTION and RELEASE, and the SET CURRENT PACKAGESET command.

Level 1 DRDA Support: CONNECT

Users can change their default database server with the CONNECT command.

The administrator (or authorized person) may establish one ID and password for a group of users. Assigning CONNECT authority to the group ID, rather than the individual users, simplifies the task of administering privileges (see *Connection, Authentication, and Security* on page 45).

Syntax: How to Issue the CONNECT Command

The syntax of the CONNECT command is

```
{ENGINE|SQL} [DB2] CONNECT [TO dbname|RESET]
```

where:

dbname

Is the DB2 location identifier.

RESET

Reconnects to the local DB2 server.

You can include the CONNECT command in a FOCEXEC such as the PROFILE FOCEXEC. The FOCEXEC may be encrypted.

Note:

- The adapter must be installed using the DB2 BIND PACKAGE command. Consult with your on-site DBA.
- For more information, consult the *IBM DB2 SQL Reference* manual.

Level 2 DRDA Support

Level 2 DRDA commands include:

- The SET CONNECTION and RELEASE commands for controlling connection to a remote server.
- The SET CURRENT PACKAGESET command.

Syntax: How to Dynamically Connect to a DB2 Server or Release a Connection

The syntax for the SET CONNECTION and RELEASE commands is

```
SQL [DB2] SET CONNECTION server_name

SQL [DB2] RELEASE {server_name|option}
```

where:

server_name

Is the location name of any valid DRDA database server.

option

Can be one of the following:

```
CURRENT
ALL [SQL]
```

Note: Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.

Syntax: How to Switch Application Packages Dynamically

You can use the SET CURRENT PACKAGESET command to switch application packages during a session. The effect is similar to the SET PLAN command, but does not require that the DB2 thread be closed and reopened (for example, when switching isolation levels).

```
SQL [DB2] SET CURRENT PACKAGESET {packageset_name|USER}
```

where:

packageset_name

Is the name of any package set previously bound into the current DB2 plan.

USER

Indicates the primary authorization ID.

Use of DRDA and the SET CURRENT PACKAGESET command requires that the adapter be installed using the DB2 BIND PACKAGE command. For more information, consult with the DBA at your site or refer to the IBM *DB2 Application Programming and SQL Guide*.

For a complete description of these commands, refer to the IBM *DB2 SQL Reference*.

Read-only Access to IMS Data From DB2 MODIFY Procedures

The Adapter for DB2 not only allows FOCUS MODIFY procedures to view and/or update data residing in DB2 tables, it also gives them read-only access to IMS data sources. You can issue FIND and LOOKUP commands against IMS data sources and use MATCH and NEXT commands to display IMS data. FOCUS can access data from both DB2 and IMS transparently, postponing or eliminating the need to physically move IMS files to DB2.

Prerequisites for DB2 Access to IMS Data

Read-only access to IMS data from DB2 MODIFY procedures requires installation of the Adapter for IMS/DB in addition to the Adapter for DB2. Both products are available from Information Builders. See the appropriate installation documentation for installation instructions.

Note:

- This feature requires that the Adapter for DB2 be installed to use the Call Attachment Facility (CAF). You can verify the attachment facility by issuing the SQL DB2 ? command. Call Attach should be ON.
- To issue MODIFY write commands (INCLUDE, UPDATE, DELETE) against DB2 tables, you must install the DB2 Read/Write Adapter. If your MODIFY procedure only displays data (MATCH, NEXT, FIND, LOOKUP), the DB2 Read-only Adapter is sufficient.

Implementation of DB2 Access to IMS Data

MODIFY commands accessing IMS data behave similarly to MODIFY commands accessing the FOCUS DBMS. If you are not familiar with FOCUS MODIFY commands, consult *Maintaining Tables With FOCUS* on page 349 and your FOCUS documentation on maintaining databases.

If an IMS data source participates in a COMBINE structure with a DB2 table, the MODIFY MATCH, NEXT, REPOSITION, and FIND commands can access the IMS data. If you dynamically JOIN the IMS file to a DB2 Master File, you can LOOKUP data values in the IMS file. The following sections discuss these two techniques.

Syntax: How to Issue MODIFY Subcommands With a COMBINE Structure

If you COMBINE the IMS data source with a DB2 table, several MODIFY subcommands can access the IMS data. The syntax is

```
COMBINE FILES  file1 [PREFIX pref1|TAG tag1] [AND]
   .
   .
   .
               filen [PREFIX prefn|TAG tagn] AS asname
```

where:

file1 - filen

Are the Master File names of the tables you want to modify. You can specify up to 16 Master Files.

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 your FOCUS documentation on maintaining databases 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

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.

Note:

- Any attempt to perform an INCLUDE, UPDATE, or DELETE operation on an IMS segment results in an error message.
- Avoid using CRTFORM * on IMS segments.

Once you issue the COMBINE FILE command, you can access the IMS files in the structure with the following MODIFY commands:

MATCH

The MATCH command selects specific segment instances based on their values. It compares field values in the instances with incoming data values.

The adapter passes a MATCH on a full key that is defined as .IMS, .HKY, or .KEY, directly to the IMS DBMS. This is the most efficient access method. The Adapter for IMS/DB issues a GET UNIQUE.

A MATCH on a non-key or partial key field repositions to the beginning of the chain and searches forward for the specified value. This search is not passed to the IMS DBMS and, therefore, is less efficient. The adapter issues GET NEXT commands until the match condition is met or no records remain.

When you specify a MATCH with both a key and a non-key field, the MATCH condition for the key is passed to the IMS DBMS for retrieval. Then the adapter does the sequential forward search for the non-key value. The adapter issues a GET UNIQUE for the qualified key, applies the condition for the non-key value based on the qualified key, and then issues GET NEXT commands for the qualified value.

`NEXT`

The NEXT command selects the next segment instance after the current position, making the selected instance the new current position. The current position depends on the execution of the MATCH and NEXT commands:

- If a MATCH or NEXT command selects a segment instance, that instance becomes the current position within the segment.
- If a MATCH or NEXT command selects a parent instance of a segment chain, the current position is before the first instance in that chain.
- At the beginning of a request, the current position is in the root segment before the first instance.

NEXT processing with a MATCH command is identical to NEXT processing without a MATCH command. If a current position has not been established, the adapter issues a GET UNIQUE to retrieve the first record and then issues GET NEXT calls to retrieve remaining records.

NEXT processing after MATCH on a non-key or partial key produces the same results. It is important to realize that the MATCH on the non-key is not passed to the IMS DBMS. The adapter repositions from the beginning of the chain and searches forward until the condition is met. Therefore, for maximum efficiency, consider matching on the primary key.

`REPOSITION`

The REPOSITION command sets the current position to the beginning of the IMS segment chain you are traversing or to the beginning of the chain of any of the parent instances along the segment path. Refer to your FOCUS documentation on maintaining databases for additional information.

`FIND`

The FIND function tests for the existence of indexed values in IMS data sources. COMBINE must be in effect in order to use FIND. Refer to your FOCUS documentation on maintaining databases for additional information.

You can also use the following MODIFY commands for DB2 tables in the COMBINE structure (refer to *Maintaining Tables With FOCUS* on page 349, for additional information):

- MATCH, NEXT, REPOSITION, INCLUDE, DELETE.
- FIND. You cannot use LOOKUP, which requires a dynamic JOIN, in the same MODIFY because you cannot issue a JOIN when a COMBINE is in effect.

Reference: The LOOKUP Function With a Dynamic JOIN

The LOOKUP function retrieves data values from data sources joined dynamically by the JOIN command (see *Advanced Reporting Techniques* on page 213). When you join a DB2 Master File to an IMS file, you can LOOKUP either DB2 or IMS data.

You cannot issue LOOKUP if the MODIFY contains commands that require a COMBINE (for example, FIND of IMS or DB2 data), or if it contains MATCH and/or NEXT commands against IMS data.

LOOKUP for IMS data sources supports the extended syntax parameters GE and LE, while LOOKUP for DB2 data does not. Refer to your FOCUS documentation for additional information. Also consult *Advanced Reporting Techniques* on page 213, and *Maintaining Tables With FOCUS* on page 349, of this manual, and the FOCUS IMS/DB Data Adapter documentation.

Run-time Requirements for DB2 Access to IMS

After the Adapters for IMS/DB and DB2 are installed, you must create a CLIST or JCL to invoke this feature. Subsequent sections outline JCL and CLIST preparation.

Example: JCL Preparation for DB2 Access to IMS in Batch

The following JCL runs a batch FOCUS job that uses both the Adapters for IMS/DB and DB2 to provide read-only access to IMS data sources during update of DB2 tables. You must concatenate the FOCLIB.LOAD library into the allocations for DDNAME STEPLIB since the IMS software program, DFSRRC00, searches STEPLIB only for libraries that are called. You can concatenate the Adapter for IMS/DB module, IMS, with DDNAME STEPLIB or USERLIB.

This JCL is only a model. Before executing it, you must create an appropriate job card and modify the JCL to conform to your site's specifications

```
//JOB card goes here
//BATIMS   EXEC PGM=DFSRRC00,PARM='DLI,FOCUS,PSBNAME'
//STEPLIB  DD   DISP=SHR,DSN=prefix.FOCLIB.LOAD
                DISP=SHR,DSN=DSN910.SDSNLOAD
//USERLIB  DD   DISP=SHR,DSN=prefix.IMS.LOAD
            DD   DISP=SHR,DSN=prefix.FOCSQL.LOAD
            DD   DISP=SHR,DSN=prefix.FOCLIB.LOAD
            DD   DISP=SHR,DSN=prefix.FUSELIB.LOAD
//DFSRESLB DD   DISP=SHR,DSN=IMSVS.RESLIB
//ERRORS   DD   DISP=SHR,DSN=prefix.ERRORS.DATA
                DISP=SHR,DSN=prefix.IMS.DATA
//IMS      DD   DISP=SHR,DSN=user.DBDLIB
            DD   DISP=SHR,DSN=user.PSBLIB
//FOCPSB   DD   DISP=SHR,DSN=user.FOCPSB(PSBNAME)
//MASTER   DD   DISP=SHR,DSN=user.MASTER.DATA
//FOCSQL   DD   DISP=SHR,DSN=user.FOCSQL.DATA
//FOCEXEC  DD   DISP=SHR,DSN=user.FOCEXEC.DATA
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   *
           FOCUS request goes here.  For example,
TABLE FILE imsfile
PRINT field1
END
FIN
```

where:

prefix

Is the high-level qualifier for your FOCUS production libraries.

user

Is the qualifier for a private version of a library.

Note: For a description of other IMS environments and their corresponding JCL requirements, see the Adapter for IMS/DB documentation.

Example: CLIST Preparation for DB2 Access to IMS in Interactive Environments

You can use this feature interactively if you call it from a CLIST or REXX EXEC. You can allocate the FOCUS load libraries directly in your CLIST or REXX EXEC. This CLIST is only a model. Edit it to conform to your site's standards

```
PROC 0
CONTROL MSG NOLIST NOFLUSH
ALLOC F(STEPLIB)      DA('prefix.FOCLIB.LOAD')    SHR REUSE
ALLOC F(USERLIB)      DA('prefix.IMS.LOAD'        -
                         'prefix.FOCSQL.LOAD'     -
                         'prefix.FOCLIB.LOAD'     -
                         'prefix.FUSELIB.LOAD')   SHR REUSE
ALLOC F(DFSRESLB)     DA('IMSVS.RESLIB')          SHR REUSE
ALLOC F(FOCEXEC)      DA('user.FOCEXEC.DATA')     SHR REUSE
ALLOC F(MASTER)       DA('user.MASTER.DATA')      SHR REUSE
ALLOC F(FOCSQL)       DA('user.FOCSQL.DATA')      SHR REUSE
ALLOC F(ERRORS)       DA('prefix.ERRORS.DATA'     -
                         'prefix.IMS.DATA')       SHR REUSE
ALLOC F(FOCPSB)       DA('user.FOCPSB(PSBNAME)')  SHR REUSE
ALLOC F(IMS)          DA('user.PSBLIB'            -
                         'user.DBDLIB')           SHR REUSE
CALL  'IMSVS.RESLIB(DFSRRC00)'  'DLI,FOCUS,PSB'
```

where:

prefix

Is the high-level qualifier for your FOCUS production libraries.

user

Is the high-level qualifier for a private version of a library.

Note: For a description of other IMS environments and their corresponding CLIST requirements, see the Adapter for IMS/DB documentation.

Appendix B

SQL Codes and Adapter Messages

This appendix lists common RDBMS return codes, describes common errors and solutions, and explains how to access adapter messages.

In this appendix:

Common SQL Return Codes for DB2

+100	No row meets the search conditions specified in a DELETE, UPDATE, or FETCH operation; or table is empty.
0	Successful execution.
-104	SQL syntax error. Run trace to determine nature of translation error.
-204	The table specified by TABLENAME in the FOCUS Access File does not exist.
-205	The SQL column name, defined by the ALIAS parameter in either the FOCUS Master or Access File, does not exist within the target TABLENAME.
-301 -302	The USAGE type or length defined for a field in the FOCUS Master does not conform to the corresponding column definition in the SQL table.
-309 -407	There is a null value within a predicate (-309) or within an update statement (-407) where nulls are not allowed.
-551 -552	You tried to perform an action within SQL for which you are not authorized.
-601	You tried to create a table, index, or view using a name that already exists.

-612	You tried to create a table that contains duplicate SQL column names. Check each Master File segment for unique ALIAS values.
-803	You tried to include or update a value within a column for which a UNIQUE INDEX exists, and that value is already present.
-904	A resource is unavailable. Refer to the DB2 messages and codes manual in order to evaluate the situation.
-905	A resource limit based on the DB2 RLST Governor has been exceeded. There are rows in the SYSIBM.DSNRLST00 table that pertain to this resource and the limits placed on it. Consult your DB2 DBA to find out what action is necessary.
-911 -913	The application was the victim in a deadlock or experienced a timeout.
-923	DB2 is not operational.

Note: If the adapter is using the DB2 Call Attachment Facility (CAF), and an attempt to communicate with a DB2 subsystem results in a CAF error (for example, the subsystem specified does not exist), the adapter will return the decimal representation of the CAF error code to FOCUS. The hexadecimal representations of these codes range from 00C10002 (decimal 12648450) to 00C10824 (decimal 12650532) and may be found, accompanied by their explanations, in the IBM DB2 Messages and Codes manual. Consult this IBM manual for information concerning any of these error codes that are returned by the adapter.

Common DBC Return Codes for Teradata

Some common mainframe host messages are:

CLI026	HSISPB unavailable. Failed to issue GLOBAL TXTLIB CLI. (CLI TXTLIB found on the TERADATA disk contains HSISPB).
CLI036	Invalid TDPID in logon text string. The TDP ID preceding the logon string does not conform to the rules for a valid TDP ID.
CLI040	Invalid logon string. The logon string passed to DBCLGN is invalid.
CLI282	Requested TDP not available. The TDP ID specified is valid but the requested TDP is not active.

CLI426	TDP unavailable – connect failed – TDP not logged on. The TDP is unavailable because the TDP machine is not logged on.
CLI521	Failure due to security violation. An attempt to log on to Teradata was made and an invalid logon id was given.

Some general return codes, which may be tested with the Dialogue Manager &RETCODE variable or the MODIFY FOCERROR variable, include:

0	No error. Successful execution.
2978	Table to be created already exists.
2980	Duplicate primary key. An insert was attempted using a unique primary key, a row with the key value specified was found already to exist.
3510	Too many END TRANSACTION statements. You issued an END TRANSACTION statement that does not correspond to a BEGIN TRANSACTION statement.
3523	User does not have specified operation access to object. (For example, you tried to perform an action within Teradata for which you are not authorized.)
3604	Cannot place a null value in a NOT NULL field.
3807	Table/view table name does not exist. You referenced a table or view that does not exist.

Common User Errors and Corrections

The following is a partial list of common data adapter error messages that you may encounter and their solutions.

`(FOC1351 or FOC2567) ACCESS FILE NOT FOUND`

Ensure that each Master File has an associated Access File. If this is the case, check that the Access File exists in a library allocated to ddname FOCSQL. The member names for both Master and Access Files must be identical.

```
(FOC1356 or FOC2566) NO TABLE NAME FOR THE SEGMENTIN THE ACCESS FILE
```

Check that both the database and table names specified by the TABLENAME keyword in the Access File are valid. Verify that the SEGNAME values in the Master and Access Files match.

```
(FOC1377) TERADATA TABLE MUST NOT HAVE KEYS=0
```

Each Teradata table used in a MODIFY procedure must have a primary key. Edit the Master and Access Files for this table to define the primary key (see *Describing Tables to FOCUS* on page 55) and resubmit the procedure.

```
(FOC1388 or FOC2554) INTEGER FIELDS FOR SQL MUST HAVEACTUAL FORMAT OF I2
OR I4
```

Only ACTUAL=I2 or ACTUAL=I4 may be used in describing an integer in the Master File. The Adapter for Teradata supports the Teradata data types BYTEINT, SMALLINT, and INTEGER.

```
(FOC2555) ACTUAL FORMAT F IS INVALID FOR SQL, USE FORMAT
D(FOC2557)ACTUAL FORMAT Z IS INVALID FOR SQL, USE FORMAT A
```

The field type equivalent of the Teradata floating-point data type, FLOAT, is D. Currently, there are no Teradata equivalents for the single-precision floating-point field type F or the zoned-decimal field type Z. Therefore, these field types (F and Z) may not be used.

```
(FOC1410) SQL COLUMN NOT FOUND. CHECK FIELDNAME ANDALIAS KEYWORDS
```

Check the spelling of the ALIAS value in the Master File. Also, verify this in the long field name declaration in the Access File. Check with your database administrator to make sure the column exists in the table or re-run the AUTODBC facility to refresh the description.

```
(FOC1413) USER HAS INSUFFICIENT AUTHORITY FOR REQUESTED OPERATION
```

The RDBMS has determined that you do not possess the authority to perform the requested operation. Notify the database administrator to resolve the error. A trace of the requested operation is recommended to identify the RDBMS object and the type of operation causing the violation.

Accessing Adapter Messages

If you need to see the text or explanation for any adapter message, you can display it online in your FOCUS session or find it in a standard FOCUS ERRORS file, ERRNLS.DATA.

To display an error message online, issue the following query command at the FOCUS command level

```
? n
```

where:

n

Is the message number.

This command displays the message number and text along with a detailed explanation of the message (if available).

Example: Displaying a Message

Issue the following command to get the text and an explanation of the error:

```
? 1369
```

The output is:

```
(FOC1369) STATIC LOAD MODULE ANDFOCCOMP OUT OF SYNCH %1%2%3%4
```

When a FOCEXEC which was compiled using the static SQL option was run, the timestamps for the FOCCOMP and its corresponding static SQL load module did not match. Check current FOCCOMP and STUBLIB allocations, and recompile the FOCEXEC with the static SQL option activated if necessary.

Appendix C

File Descriptions and Tables

This appendix consists of sample tables and views cited in previous chapters.

In this appendix:

Samples Overview

This appendix consists of sample tables and views cited in previous chapters. It provides single-table Master and Access Files and CHECK FILE diagrams for:

- ADDRESS
- COURSE
- DEDUCT
- EMPINFO
- FUNDTRAN
- PAYINFO
- SALINFO

In addition, it includes the following multi-table Master and Access Files and CHECK FILE diagrams:

- ECOURSE
- EMPADD
- EMPFUND
- EMPPAY
- SALDUCT

An OCCURS segment is described in the following Master File and Access File with a CHECK FILE diagram:

- SALARY

This appendix also illustrates sample tables used in examples for the Direct SQL Passthru facility:

- DPBRANCH
- DPINVENT
- DPVENDOR

As discussed in *Direct SQL Passthru* on page 265, the in-memory SQLOUT Master Files and the FOCUS views created with SQL PREPARE vary depending on the specified SELECT statements. They are not accessible for the above tables and are not provided in this appendix.

ADDRESS Sample

The ADDRESS table contains employees' home and bank addresses.

ADDRESS MASTER

For DB2:

```
FILENAME=ADDRESS  ,SUFFIX=DB2
SEGNAME=ADDRESS  ,SEGTYPE=S0,$
  FIELDNAME=ADDEID       ,ALIAS=EID     ,USAGE=A9    , ACTUAL = A9,$
  FIELDNAME=TYPE         ,ALIAS=AT      ,USAGE=A4    , ACTUAL = A4,$
  FIELDNAME=ADDRESS_LN1  ,ALIAS=LN1     ,USAGE=A20   , ACTUAL = A20,$
  FIELDNAME=ADDRESS_LN2  ,ALIAS=LN2     ,USAGE=A20   , ACTUAL = A20,$
  FIELDNAME=ADDRESS_LN3  ,ALIAS=LN3     ,USAGE=A20   , ACTUAL = A20,$
  FIELDNAME=ACCTNUMBER    ,ALIAS=ANO     ,USAGE=I9L   , ACTUAL = I4,$
```

Note: For Teradata, change the suffix value to SQLDBC, for CA-IDMS, change the suffix value to SQLIDMS, and for Oracle, change the suffix value to SQLORA.

ADDRESS FOCSQL

For DB2:

```
SEGNAME = ADDRESS, TABLENAME = "USER1"."ADDRESS", KEYS = 2, WRITE = YES,
    DBSPACE = PUBLIC.SPACE0,$
```

For Teradata:

```
SEGNAME = ADDRESS,TABLENAME = USER1.ADDRESS, KEYS = 2, WRITE = YES,
    FALLBACK=YES,$
```

For IDMS SQL:

```
SEGNAME = ADDRESS, TABLENAME = ADDSCHEM.ADDRESS, KEYS = 2, WRITE = YES,
    DBSPACE = ADDSEG.ADDAREA,$
```

For Oracle:

```
SEGNAME=ADDRESS, TABLENAME=USER1.ADDRESS, KEYS=2, WRITE=YES,
    DBSPACE=SPACE1, $
```

ADDRESS Diagram

```
check file address pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   1  ( REAL=    1  VIRTUAL=   0 )
  NUMBER OF FIELDS=     6  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   77
 SECTION 01
             STRUCTURE OF SQLDS    FILE ADDRESS  ON 06/17/93 AT 13.51.17
          ADDRESS
  01      S0
 **************
 *ADDEID      **
 *TYPE        **
 *ADDRESS_LN1 **
 *ADDRESS_LN2 **
 *            **
 ***************
  ***************
```

COURSE Sample

The COURSE table lists the courses that the employees attended.

COURSE MASTER

For DB2 :

```
FILENAME=COURSE ,SUFFIX=DB2,$
SEGNAME=COURSE  ,SEGTYPE=S0  ,$
 FIELD=CNAME   ,ALIAS=COURSE_NAME ,USAGE=A30, ACTUAL=A30,$
 FIELD=WHO     ,ALIAS=EMP_NO      ,USAGE=A9,  ACTUAL=A9,$
 FIELD=GRADE   ,ALIAS=GRADE       ,USAGE=A1,  ACTUAL=A1, MISSING=ON,$
 FIELD=YR_TAKEN,ALIAS=YR_TAKEN    ,USAGE=A2,  ACTUAL=A2,$
 FIELD=QTR     ,ALIAS=QUARTER      ,USAGE=A1,  ACTUAL=A1,$
```

Note: For Teradata, change the suffix value to SQLDBC, for CA-IDMS, change the suffix value to SQLIDMS, and for Oracle, change the suffix value to SQLORA.

COURSE FOCSQL

For DB2:

```
SEGNAME = COURSE, TABLENAME = "USER1"."COURSE", KEYS = 2, WRITE = YES,
    DBSPACE = PUBLIC.SPACE0, $
```

For Teradata:

```
SEGNAME=COURSE, TABLENAME=USER1.COURSE, KEYS=2, WRITE=YES, FALLBACK=YES,
$
```

For IDMS SQL:

```
SEGNAME = COURSE, TABLENAME = CRSSCHEM.COURSE, KEYS = 2, WRITE = YES,
    DBSPACE = CRSSEG.CRSAREA, $
```

For Oracle:

```
SEGNAME=COURSE, TABLENAME=USER1.COURSE, KEYS=2, WRITE=YES,
  DBSPACE=SPACE1, $
```

COURSE Diagram

```
check file course pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   1  ( REAL=    1  VIRTUAL=   0 )
  NUMBER OF FIELDS=     5  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   28
 SECTION 01
             STRUCTURE OF SQLDS    FILE COURSE   ON 06/17/93 AT 13.51.41
          COURSE
  01      S0
 **************
 *CNAME       **
 *WHO         **
 *GRADE       **
 *YR_TAKEN    **
 *            **
 ***************
  **************
```

DEDUCT Sample

The DEDUCT table contains data on monthly pay deductions.

DEDUCT MASTER

For DB2:

```
FILENAME=DEDUCT ,  SUFFIX=DB2,$
SEGNAME=DEDUCT   ,SEGTYPE=S0,$
  FIELDNAME=DEDEID  ,ALIAS=EID ,USAGE=A9       ,ACTUAL = A9,$
  FIELDNAME=DEDDATE ,ALIAS=PD  ,USAGE=YMD      ,ACTUAL = DATE,$
  FIELDNAME=DED_CODE,ALIAS=DC  ,USAGE=A4       ,ACTUAL = A4,$
  FIELDNAME=DED_AMT ,ALIAS=DA   ,USAGE=D12.2M   ,ACTUAL = D8,$
```

Note:

- For Teradata, change the suffix value to SQLDBC, the USAGE format for DED_AMT to P9.2, and the ACTUAL to P8.
- For CA-IDMS, change the suffix value to SQLIDMS.
- For Oracle, change the suffix value to SQLORA.

DEDUCT FOCSQL

For DB2:

```
SEGNAME = DEDUCT, TABLENAME = "USER1"."DEDUCT",  KEYS = 3, WRITE = YES,
    KEYORDER = HIGH, DBSPACE = PUBLIC.SPACE0,$
```

For Teradata:

```
SEGNAME = DEDUCT, TABLENAME = USER1.DEDUCT,  KEYS = 3, WRITE = YES,
    KEYORDER=HIGH, FALLBACK=YES,$
```

For IDMS SQL:

```
SEGNAME = DEDUCT, TABLENAME = DEDSCHEM.DEDUCT,  KEYS = 3, WRITE = YES,
    KEYORDER = HIGH, DBSPACE = DEDSEG.DEDAREA,$
```

For Oracle:

```
SEGNAME=DEDUCT, TABLENAME=USER1.DEDUCT, KEYS=3, WRITE=YES,
    KEYORDER = HIGH, DBSPACE=SPACE1, $
```

DEDUCT Diagram

```
check file deduct pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   1  ( REAL=    1  VIRTUAL=   0 )
  NUMBER OF FIELDS=     4  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   25
 SECTION 01
              STRUCTURE OF SQLDS    FILE DEDUCT   ON 06/16/93 AT 09.42.45
          DEDUCT
  01      S0
 **************
 *DEDEID      **
 *DEDDATE     **
 *DED_CODE    **
 *DED_AMT     **
 *            **
 ***************
  **************
```

EMPINFO Sample

The EMPINFO table contains employee IDs, names, positions, and current salary information.

EMPINFO MASTER

For DB2:

```
FILENAME=EMPINFO       ,SUFFIX=DB2,$
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=DPT          ,USAGE=A10    ,ACTUAL=A10,
    MISSING=ON,$
 FIELD=CURRENT_SALARY  ,ALIAS=CSAL         ,USAGE=P9.2   ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE    ,ALIAS=CJC          ,USAGE=A3     ,ACTUAL=A3,$
 FIELD=ED_HRS          ,ALIAS=OJT          ,USAGE=F6.2   ,ACTUAL=F4,
    MISSING=ON,$
 FIELD=BONUS_PLAN      ,ALIAS=BONUS_PLAN   ,USAGE=I4     ,ACTUAL=I4,$
 FIELD=HIRE_DATE_TIME  ,ALIAS=HDTT         ,USAGE=HYYMDm,ACTUAL=HYYMDm ,
    MISSING=ON,$
 FIELD=HIRE_TIME       ,ALIAS=HT           ,USAGE=HHIS   ,ACTUAL=HHIS   ,
    MISSING=ON,$
```

Note:

- For Teradata:
  - Change the suffix value to SQLDBC.
  - Change the ALIAS for LAST_NAME to LNAME.
  - Change the ACTUAL format for CURRENT_SALARY to P8.
  - Change the USAGE format for ED_HRS to D6.2 and the ACTUAL to D8.
  - Remove the fields HIRE_DATE_TIME and HIRE_TIME.
- For CA-IDMS, change the suffix value to SQLIDMS. Also, remove the fields HIRE_DATE_TIME and HIRE_TIME.
- For Oracle, change the suffix value to SQLORA. Also, change the USAGE and ACTUAL formats for HIRE_DATE_TIME to HYYMDS and remove field HIRE_TIME.

EMPINFO FOCSQL

For DB2:

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1,
 WRITE = YES,  DBSPACE = PUBLIC.SPACE0,$
```

For Teradata:

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

For IDMS SQL:

```
SEGNAME = EMPINFO, TABLENAME = EMPSCHEM.EMPINFO, KEYS = 1,
 WRITE = YES, DBSPACE=EMPSEG.EMPAREA, $
```

For Oracle:

```
SEGNAME=EMPINFO, TABLENAME=USER1.EMPINFO, KEYS=1, WRITE=YES,
DBSPACE=SPACE1, $
```

EMPINFO Diagram

```
check file empinfo pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   1  ( REAL=    1  VIRTUAL=   0 )
  NUMBER OF FIELDS=     9  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   63
 SECTION 01
             STRUCTURE OF SQLDS    FILE EMPINFO  ON 06/16/93 AT 09.44.03
          EMPINFO
  01      S0
 **************
 *EMP_ID      **
 *LAST_NAME   **
 *FIRST_NAME  **
 *HIRE_DATE   **
 *            **
 ***************
  **************
```

FUNDTRAN Sample

The FUNDTRAN table contains data about the employees' direct deposit accounts.

FUNDTRAN MASTER

For DB2:

```
FILENAME=FUNDTRAN ,SUFFIX=DB2
SEGNAME=FUNDTRAN, SEGTYPE=S0,$
 FIELDNAME=WHO           ,ALIAS=EID    ,USAGE=A9     ,ACTUAL=A9,$
 FIELDNAME=BANK_NAME     ,ALIAS=BN     ,USAGE=A20    ,ACTUAL=A20,$
 FIELDNAME=BANK_CODE     ,ALIAS=BC     ,USAGE=I6S    ,ACTUAL=I4,$
 FIELDNAME=BANK_ACCT     ,ALIAS=BA     ,USAGE=I9S    ,ACTUAL=I4,$
 FIELDNAME=EFFECT_DATE    ,ALIAS=EDATE ,USAGE=YMD    ,ACTUAL=DATE,$
```

Note: For Teradata, change the suffix value to SQLDBC, for CA-IDMS, change the suffix value to SQLIDMS, and for Oracle, change the suffix value to SQLORA.

FUNDTRAN FOCSQL

For DB2:

```
SEGNAME = FUNDTRAN, TABLENAME = "USER1"."FUNDTRAN", KEYS = 1,
    WRITE = YES, DBSPACE = PUBLIC.SPACE0,$
```

For Teradata:

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

For IDMS SQL:

```
SEGNAME = FUNDTRAN, TABLENAME = FUNDSCHM.FUNDTRAN, KEYS = 1,
    WRITE = YES, DBSPACE = FUNDSEG.FUNDAREA,$
```

For Oracle:

```
SEGNAME=FUNDTRAN, TABLENAME=USER1.FUNDTRAN, KEYS=1, WRITE=YES,
    DBSPACE=SPACE1, $
```

FUNDTRAN Diagram

```
check file fundtran pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   1  ( REAL=    1  VIRTUAL=   0 )
  NUMBER OF FIELDS=     5  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   41
 SECTION 01
             STRUCTURE OF SQLDS    FILE FUNDTRAN ON 06/17/93 AT 13.53.15
          FUNDTRAN
  01      S0
 **************
 *WHO          **
 *BANK_NAME    **
 *BANK_CODE    **
 *BANK_ACCT    **
 *             **
 ***************
  **************
```

PAYINFO Sample

The PAYINFO table contains the employees' salary history.

PAYINFO MASTER

For DB2:

```
FILENAME=PAYINFO,  SUFFIX=DB2,$
SEGNAME=PAYINFO  ,SEGTYPE=S0,$
  FIELDNAME=PAYEID  ,ALIAS=EID ,USAGE=A9        ,ACTUAL=A9,$
  FIELDNAME=DAT_INC ,ALIAS=DI  ,USAGE=YMD       ,ACTUAL=DATE,$
  FIELDNAME=PCT_INC ,ALIAS=PI  ,USAGE=F6.2      ,ACTUAL=F4,$
  FIELDNAME=SALARY  ,ALIAS=SAL ,USAGE=D12.2M    ,ACTUAL=D8,$
  FIELDNAME=JOBCODE  ,ALIAS=JBC ,USAGE=A3        ,ACTUAL=A3,$
```

Note:

- For Teradata:
  - Change the suffix value to SQLDBC.
  - Change the ALIAS for PCT_INC to PINC, the USAGE format to D6.2, and the ACTUAL format to D8.
  - Change the USAGE format for SALARY to P9.2 and the ACTUAL to P8.
- For CA-IDMS, change the suffix value to SQLIDMS.
- For Oracle, change the suffix value to SQLORA.

PAYINFO FOCSQL

For DB2:

```
SEGNAME = PAYINFO, TABLENAME = "USER1"."PAYINFO", KEYS = 2, WRITE = YES,
    DBSPACE = PUBLIC.SPACE0,$
```

For Teradata:

```
SEGNAME=PAYINFO,TABLENAME=USER1.PAYINFO,KEYS=2,WRITE=YES,FALLBACK=YES,$
```

For IDMS SQL:

```
SEGNAME = PAYINFO, TABLENAME = PAYSCHEM.PAYINFO, KEYS = 2, WRITE = YES,
    DBSPACE = PAYSEG.PAYAREA,$
```

For Oracle:

```
SEGNAME=PAYINFO, TABLENAME=USER1.PAYINFO, KEYS=2, WRITE=YES,
   DBSPACE=SPACE1, $
```

PAYINFO Diagram

```
check file payinfo pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   1  ( REAL=    1  VIRTUAL=   0 )
  NUMBER OF FIELDS=     5  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   28
 SECTION 01
             STRUCTURE OF SQLDS    FILE PAYINFO  ON 06/16/93 AT 09.44.31
          PAYINFO
  01      S0
 **************
 *PAYEID       **
 *DAT_INC      **
 *PCT_INC      **
 *SALARY       **
 *             **
 ***************
  **************
```

SALINFO Sample

The SALINFO table contains data on the employees' monthly pay.

SALINFO MASTER

For DB2:

```
FILENAME=SALINFO,  SUFFIX=DB2,$
SEGNAME=SALINFO  ,SEGTYPE=S0,$
 FIELDNAME=SALEID   ,ALIAS=EID     ,USAGE=A9      ,ACTUAL=A9,$
 FIELDNAME=PAY_DATE ,ALIAS=PD      ,USAGE=YMD     ,ACTUAL=DATE,$
 FIELDNAME=GROSS     ,ALIAS=MO_PAY  ,USAGE=D12.2M  ,ACTUAL=D8,$
```

Note:

- For Teradata:
  - Change the suffix value to SQLDBC.
  - Change the USAGE format for GROSS to P9.2 and the ACTUAL to P8.
- For CA-IDMS, change the suffix value to SQLIDMS.
- For Oracle, change the suffix value to SQLORA.

SALINFO FOCSQL

For DB2:

```
SEGNAME = SALINFO, TABLENAME = "USER1"."SALINFO", KEYS = 2,
 WRITE = YES,  KEYORDER = HIGH, DBSPACE = PUBLIC.SPACE0,$
```

For Teradata:

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

For IDMS SQL:

```
SEGNAME = SALINFO, TABLENAME = SALSCHEM.SALINFO, KEYS = 2,
 WRITE = YES,  KEYORDER = HIGH, DBSPACE = SALSEG.SALAREA,$
```

For Oracle:

```
SEGNAME=SALINFO, TABLENAME=USER1.SALINFO, KEYS=2, WRITE=YES,
    KEYORDER=HIGH, DBSPACE=SPACE1, $
```

SALINFO Diagram

```
check file salinfo pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   1  ( REAL=    1  VIRTUAL=   0 )
  NUMBER OF FIELDS=     3  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   21
 SECTION 01
             STRUCTURE OF SQLDS    FILE SALINFO  ON 06/17/93 AT 13.52.04
          SALINFO
  01      S0
 **************
 *SALEID      **
 *PAY_DATE    **
 *GROSS       **
 *            **
 *            **
 ***************
  **************
```

ECOURSE Sample

The ECOURSE view accesses the EMPINFO and COURSE tables.

ECOURSE MASTER

For DB2:

```
FILENAME=ECOURSE     ,SUFFIX=DB2, $
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=DPT          ,USAGE=A10    ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL         ,USAGE=P9.2   ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE  ,ALIAS=CJC          ,USAGE=A3     ,ACTUAL=A3,$
 FIELD=ED_HRS        ,ALIAS=OJT          ,USAGE=F6.2   ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN    ,ALIAS=BONUS_PLAN   ,USAGE=I4     ,ACTUAL=I4,$
SEGNAME=COURSE ,SEGTYPE=S0   ,PARENT=EMPINFO,$
 FIELD=CNAME    ,ALIAS=COURSE_NAME ,USAGE=A30, ACTUAL=A30,$
 FIELD=WHO      ,ALIAS=EMP_NO      ,USAGE=A9,  ACTUAL=A9,$
 FIELD=GRADE    ,ALIAS=GRADE       ,USAGE=A1,  ACTUAL=A1, MISSING=ON,$
 FIELD=YR_TAKEN,ALIAS=YR_TAKEN     ,USAGE=A2,  ACTUAL=A2,$
 FIELD=QTR      ,ALIAS=QUARTER      ,USAGE=A1,  ACTUAL=A1,$
```

Note:

- For Teradata:
  - Change the suffix value to SQLDBC.
  - Change the ALIAS for LAST_NAME to LNAME.
  - Change the ACTUAL format for CURRENT_SALARY to P8.
  - Change the USAGE format for ED_HRS to D6.2 and the ACTUAL to D8.
- For CA-IDMS, change the suffix value to SQLIDMS.
- For Oracle, change the suffix value to SQLORA.

ECOURSE FOCSQL

For DB2:

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1,
   WRITE = YES,  DBSPACE = PUBLIC.SPACE0,$
SEGNAME = COURSE, TABLENAME = "USER1"."COURSE", KEYS = 2,
   WRITE = YES,     DBSPACE = PUBLIC.SPACE0,
     KEYFLD = EMP_ID, IXFLD = WHO,$
```

For Teradata:

```
SEGNAME=EMPINFO, TABLENAME=USER1.EMPINFO, KEYS=1, WRITE=YES,
   FALLBACK=YES,$
SEGNAME=COURSE,  TABLENAME=USER1.COURSE,  KEYS=2  WRITE=YES,
   FALLBACK=YES, KEYFLD=EMP_ID,IXFLD=WHO,$
```

For IDMS SQL:

```
SEGNAME = EMPINFO, TABLENAME = EMPSCHEM.EMPINFO, KEYS = 1,
   WRITE = YES,  DBSPACE = EMPSEG.EMPAREA,$
SEGNAME = COURSE, TABLENAME = CRSSCHEM.COURSE, KEYS = 2,
   WRITE = YES,     DBSPACE = CRSSEG.CRSAREA,
     KEYFLD = EMP_ID, IXFLD = WHO,$
```

For Oracle:

```
SEGNAME=EMPINFO, TABLENAME=USER1.EMPINFO, KEYS=1, WRITE=YES,
    DBSPACE=SPACE1,
SEGNAME = COURSE, TABLENAME = USER1.COURSE, KEYS = 2, WRITE=YES,
    DBSPACE=SPACE1,
     KEYFLD = EMP_ID, IXFLD = WHO,$
```

ECOURSE Diagram

```
check file ecourse pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   2  ( REAL=    2  VIRTUAL=   0 )
  NUMBER OF FIELDS=    14  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   91
 SECTION 01
             STRUCTURE OF SQLDS    FILE ECOURSE  ON 06/16/93 AT 09.43.01
          EMPINFO
  01      S0
 **************
 *EMP_ID      **
 *LAST_NAME   **
 *FIRST_NAME  **
 *HIRE_DATE   **
 *            **
 ***************
  **************
        I
        I
        I
        I COURSE
  02    I S0
 **************
 *CNAME       **
 *WHO         **
 *GRADE       **
 *YR_TAKEN    **
 *            **
 ***************
  **************
```

EMPADD Sample

The EMPADD view accesses the EMPINFO and ADDRESS tables.

EMPADD MASTER

For DB2:

```
FILENAME=EMPADD,    SUFFIX=DB2,$
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=DPT           ,USAGE=A10      ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL           ,USAGE=P9.2     ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE   ,ALIAS=CJC           ,USAGE=A3       ,ACTUAL=A3,$
 FIELD=ED_HRS         ,ALIAS=OJT           ,USAGE=F6.2     ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN     ,ALIAS=BONUS_PLAN    ,USAGE=I4       ,ACTUAL=I4,$
SEGNAME=ADDRESS   ,SEGTYPE=S0, PARENT = EMPINFO,$
  FIELD=ADDEID            ,ALIAS=EID   ,USAGE=A9      ,ACTUAL=A9,$
  FIELD=TYPE              ,ALIAS=AT    ,USAGE=A4      ,ACTUAL=A4,$
  FIELD=ADDRESS_LN1       ,ALIAS=LN1   ,USAGE=A20     ,ACTUAL=A20,$
  FIELD=ADDRESS_LN2       ,ALIAS=LN2   ,USAGE=A20     ,ACTUAL=A20,$
  FIELD=ADDRESS_LN3       ,ALIAS=LN3   ,USAGE=A20     ,ACTUAL=A20,$
  FIELD=ACCTNUMBER         ,ALIAS=ANO   ,USAGE=I9L     ,ACTUAL=I4,$
```

Note:

- For Teradata:
  - Change the suffix value to SQLDBC.
  - Change the ALIAS for LAST_NAME to LNAME.
  - Change the ACTUAL format for CURRENT_SALARY to P8.
  - Change the USAGE format for ED_HRS to D6.2 and the ACTUAL to D8.
- For CA-IDMS, change the suffix value to SQLIDMS.
- For Oracle, change the suffix value to SQLORA.

EMPADD FOCSQL

For DB2:

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1, WRITE = YES,
     DBSPACE = PUBLIC.SPACE0,$
SEGNAME = ADDRESS, TABLENAME = "USER1"."ADDRESS", KEYS = 2, WRITE = YES,
     DBSPACE = PUBLIC.SPACE0,
     KEYFLD = EMP_ID, IXFLD = ADDEID,$
```

For Teradata:

```
SEGNAME=EMPINFO, TABLENAME=USER1.EMPINFO, KEYS=1, WRITE=YES,
FALLBACK=YES,$
SEGNAME=ADDRESS, TABLENAME=USER1.ADDRESS, KEYS=2, WRITE=YES,
FALLBACK=YES,
    KEYFLD=EMP_ID, IXFLD=ADDEID,$
```

For IDMS SQL:

```
SEGNAME = EMPINFO, TABLENAME = EMPSCHEM.EMPINFO, KEYS = 1, WRITE = YES,
     DBSPACE = EMPSEG.EMPAREA,$
SEGNAME = ADDRESS, TABLENAME = ADDSCHEM.ADDRESS, KEYS = 2, WRITE = YES,
     DBSPACE = ADDSEG.ADDAREA,
     KEYFLD = EMP_ID, IXFLD = ADDEID,$
```

For Oracle:

```
SEGNAME=EMPINFO, TABLENAME=USER1.EMPINFO, KEYS=1, WRITE=YES,
    DBSPACE=SPACE1,
SEGNAME = ADDRESS, TABLENAME = USER1.ADDRESS, KEYS = 2, WRITE=YES,
    DBSPACE=SPACE1,
     KEYFLD = EMP_ID, IXFLD = ADDEID,$
```

EMPADD Diagram

```
check file empadd pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   2  ( REAL=    2  VIRTUAL=   0 )
  NUMBER OF FIELDS=    15  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=  140
 SECTION 01
             STRUCTURE OF SQLDS    FILE EMPADD   ON 06/16/93 AT 09.43.16
          EMPINFO
  01      S0
 **************
 *EMP_ID      **
 *LAST_NAME   **
 *FIRST_NAME  **
 *HIRE_DATE   **
 *            **
 ***************
  **************
        I
        I
        I
        I ADDRESS
  02    I S0
 **************
 *ADDEID      **
 *TYPE        **
 *ADDRESS_LN1 **
 *ADDRESS_LN2 **
 *            **
 ***************
  **************
```

EMPFUND Sample

The EMPFUND view accesses the EMPINFO and FUNDTRAN tables.

EMPFUND MASTER

For DB2:

```
FILENAME=EMPFUND ,SUFFIX=DB2,$
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=DPT           ,USAGE=A10      ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL          ,USAGE=P9.2     ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE  ,ALIAS=CJC           ,USAGE=A3       ,ACTUAL=A3,$
 FIELD=ED_HRS        ,ALIAS=OJT           ,USAGE=F6.2     ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN    ,ALIAS=BONUS_PLAN    ,USAGE=I4       ,ACTUAL=I4,$
SEGNAME=FUNDTRAN, SEGTYPE=U, PARENT=EMPINFO,$
 FIELDNAME=WHO          ,ALIAS=EID   ,USAGE=A9    ,ACTUAL=A9,$
 FIELDNAME=BANK_NAME    ,ALIAS=BN    ,USAGE=A20   ,ACTUAL=A20,$
 FIELDNAME=BANK_CODE    ,ALIAS=BC    ,USAGE=I6S   ,ACTUAL=I4,$
 FIELDNAME=BANK_ACCT    ,ALIAS=BA    ,USAGE=I9S   ,ACTUAL=I4,$
 FIELDNAME=EFFECT_DATE   ,ALIAS=EDATE ,USAGE=YMD   ,ACTUAL=DATE,$
```

Note:

- For Teradata:
  - Change the suffix value to SQLDBC.
  - Change the ALIAS for LAST_NAME to LNAME.
  - Change the ACTUAL format for CURRENT_SALARY to P8.
  - Change the USAGE format for ED_HRS to D6.2 and the ACTUAL to D8.
- For CA-IDMS, change the suffix value to SQLIDMS.
- For Oracle, change the suffix value to SQLORA.

EMPFUND FOCSQL

For DB2:

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1, WRITE = YES,
     DBSPACE = PUBLIC.SPACE0,$
SEGNAME = FUNDTRAN, TABLENAME = "USER1"."FUNDTRAN", KEYS = 1,
    WRITE = YES, DBSPACE = PUBLIC.SPACE0,
    KEYFLD = EMP_ID, IXFLD = WHO,$
```

For Teradata:

```
SEGNAME=EMPINFO, TABLENAME=USER1.EMPINFO, KEYS=1, WRITE=YES,
FALLBACK=YES,$
SEGNAME=FUNDTRAN,TABLENAME=USER1.FUNDTRAN,KEYS=1, WRITE=YES,
FALLBACK=YES,
    KEYFLD=EMP_ID, IXFLD=WHO,$
```

For IDMS SQL:

```
SEGNAME = EMPINFO, TABLENAME = EMPSCHEM.EMPINFO, KEYS = 1, WRITE = YES,
     DBSPACE = EMPSEG.EMPAREA,$
SEGNAME = FUNDTRAN, TABLENAME = FUNDSCHM.FUNDTRAN, KEYS = 1,
    WRITE = YES, DBSPACE = FUNDSEG.FUNDAREA,
    KEYFLD = EMP_ID, IXFLD = WHO,$
```

For Oracle:

```
SEGNAME=EMPINFO, TABLENAME=USER1.EMPINFO, KEYS=1, WRITE=YES,
    DBSPACE=SPACE1,
SEGNAME = FUNDTRAN, TABLENAME = USER1.FUNDTRAN, KEYS = 1, WRITE=YES,
    DBSPACE=SPACE1,
     KEYFLD = EMP_ID, IXFLD = WHO,$
```

EMPFUND Diagram

```
check file empfund pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   2  ( REAL=    2  VIRTUAL=   0 )
  NUMBER OF FIELDS=    14  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=  104
 SECTION 01
             STRUCTURE OF SQLDS    FILE EMPFUND  ON 06/16/93 AT 09.43.45
          EMPINFO
  01      S0
 **************
 *EMP_ID      **
 *LAST_NAME   **
 *FIRST_NAME  **
 *HIRE_DATE   **
 *            **
 ***************
  **************
        I
        I
        I
        I FUNDTRAN
  02    I U
 **************
 *WHO         *
 *BANK_NAME   *
 *BANK_CODE   *
 *BANK_ACCT   *
 *            *
 **************
```

EMPPAY Sample

The EMPAY view accesses the EMPINFO and PAYINFO tables.

EMPPAY MASTER

For DB2:

```
FILENAME=EMPPAY  ,SUFFIX=DB2,$
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=DPT          ,USAGE=A10      ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL         ,USAGE=P9.2     ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE  ,ALIAS=CJC          ,USAGE=A3       ,ACTUAL=A3,$
 FIELD=ED_HRS        ,ALIAS=OJT          ,USAGE=F6.2     ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN    ,ALIAS=BONUS_PLAN   ,USAGE=I4       ,ACTUAL=I4,$
SEGNAME=PAYINFO  ,SEGTYPE=S0, PARENT=EMPINFO, $
  FIELDNAME=PAYEID  ,ALIAS=EID ,USAGE=A9        ,ACTUAL=A9,$
  FIELDNAME=DAT_INC ,ALIAS=DI  ,USAGE=YMD       ,ACTUAL=DATE,$
  FIELDNAME=PCT_INC ,ALIAS=PI  ,USAGE=F6.2      ,ACTUAL=F4,$
  FIELDNAME=SALARY  ,ALIAS=SAL ,USAGE=D12.2M    ,ACTUAL=D8,$
  FIELDNAME=JOBCODE  ,ALIAS=JBC ,USAGE=A3        ,ACTUAL=A3,$
```

Note:

- For Teradata:
  - Change the suffix value to SQLDBC.
  - Change the ALIAS for LAST_NAME to LNAME.
  - Change the ACTUAL format for CURRENT_SALARY to P8.
  - Change the USAGE format for ED_HRS to D6.2 and the ACTUAL to D8.
  - Change the USAGE format for PCT_INC to D6.2 and the ACTUAL to D8.
  - Change the USAGE format for SALARY to P9.2 and the ACTUAL to P8.
- For CA-IDMS, change the suffix value to SQLIDMS.
- For Oracle, change the suffix value to SQLORA.

EMPPAY FOCSQL

For DB2:

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1, WRITE = YES,
     DBSPACE = PUBLIC.SPACE0,$
SEGNAME = PAYINFO, TABLENAME = "USER1"."PAYINFO", KEYS = 2, WRITE = YES,
     DBSPACE = PUBLIC.SPACE0,
     KEYFLD = EMP_ID, IXFLD = PAYEID, $
```

For Teradata:

```
SEGNAME=EMPINFO, TABLENAME=USER1.EMPLOYEE, KEYS=1, WRITE=YES,
    FALLBACK=YES,$
SEGNAME=PAYINFO, TABLENAME=USER1.PAYINFO,  KEYS=2, WRITE=YES,
    FALLBACK=YES,
    KEYFLD=EMP_ID, IXFLD=PAYEID,$
```

For IDMS SQL:

```
SEGNAME = EMPINFO, TABLENAME = EMPSCHEM.EMPINFO, KEYS = 1, WRITE = YES,
     DBSPACE = EMPSEG.EMPAREA.SPACE0,$
SEGNAME = PAYINFO, TABLENAME = PAYSCHEM.PAYINFO, KEYS = 2, WRITE = YES,
     DBSPACE = PAYSEG.PAYAREA,
     KEYFLD = EMP_ID, IXFLD = PAYEID, $
```

For Oracle:

```
SEGNAME=EMPINFO, TABLENAME=USER1.EMPINFO, KEYS=1, WRITE=YES,
    DBSPACE=SPACE1,
SEGNAME = PAYINFO, TABLENAME = USER1.PAYINFO, KEYS = 2, WRITE=YES,
    DBSPACE=SPACE1,
     KEYFLD = EMP_ID, IXFLD = PAYEID,$
```

EMPPAY Diagram

```
check file emppay pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   2  ( REAL=    2  VIRTUAL=   0 )
  NUMBER OF FIELDS=    14  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   91
 SECTION 01
             STRUCTURE OF SQLDS    FILE EMPPAY   ON 06/16/93 AT 09.44.14
          EMPINFO
  01      S0
 **************
 *EMP_ID      **
 *LAST_NAME   **
 *FIRST_NAME  **
 *HIRE_DATE   **
 *            **
 ***************
  **************
        I
        I
        I
        I PAYINFO
  02    I S0
 **************
 *PAYEID      **
 *DAT_INC     **
 *PCT_INC     **
 *SALARY      **
 *            **
 ***************
  **************
```

SALDUCT Sample

The SALDUCT view accesses the SALINFO and DEDUCT tables.

SALDUCT MASTER

For DB2:

```
FILENAME=SALDUCT,  SUFFIX=DB2,$
SEGNAME=SALINFO  ,SEGTYPE=S0,$
 FIELDNAME=SALEID   ,ALIAS=EID     ,USAGE=A9      ,ACTUAL=A9,$
 FIELDNAME=PAY_DATE ,ALIAS=PD      ,USAGE=YMD     ,ACTUAL=DATE,$
 FIELDNAME=GROSS    ,ALIAS=MO_PAY  ,USAGE=D12.2M  ,ACTUAL=D8,$
SEGNAME=DEDUCT   ,SEGTYPE=S0, PARENT =SALINFO,$
  FIELDNAME=DEDEID  ,ALIAS=EID ,USAGE=A9       ,ACTUAL = A9,$
  FIELDNAME=DEDDATE ,ALIAS=PD  ,USAGE=YMD      ,ACTUAL = DATE,$
  FIELDNAME=DED_CODE,ALIAS=DC  ,USAGE=A4       ,ACTUAL = A4,$
  FIELDNAME=DED_AMT  ,ALIAS=DA  ,USAGE=P9.2     ,ACTUAL = P4,$
```

Note:

- For Teradata:
  - Change the suffix value to SQLDBC.
  - Change the USAGE format for GROSS to P9.2 and the ACTUAL to P8.
  - Change the ACTUAL format for DED_AMT to P9.2 and the ACTUAL to P8.
- For CA-IDMS, change the suffix value to SQLIDMS.
- For Oracle, change the suffix value to SQLORA.

SALDUCT FOCSQL

For DB2:

```
SEGNAME = SALINFO, TABLENAME = "USER1"."SALINFO", KEYS = 2, WRITE = YES,
    KEYORDER = HIGH, DBSPACE = PUBLIC.SPACE0,$
SEGNAME = DEDUCT, TABLENAME = "USER1"."DEDUCT",  KEYS = 3, WRITE = YES,
    KEYORDER = HIGH, DBSPACE = PUBLIC.SPACE0,
    KEYFLD = SALEID/PAY_DATE , IXFLD = DEDEID/DEDDATE,$
```

For Teradata:

```
SEGNAME = SALINFO,TABLENAME = USER1.SALINFO, KEYS=2, WRITE=YES,
    KEYORDER=HIGH, FALLBACK=YES,$
SEGNAME = DEDUCT, TABLENAME = USER1.DEDUCT,  KEYS=3, WRITE=YES,
    KEYORDER=HIGH, FALLBACK=YES,
    KEYFLD=SALEID/PAY_DATE, IXFLD=DEDEID/DEDDATE,$
```

For IDMS SQL:

```
SEGNAME = SALINFO, TABLENAME = SALSCHEM.SALINFO, KEYS = 2, WRITE = YES,
    KEYORDER = HIGH, DBSPACE = SALSEG.SALAREA,$
SEGNAME = DEDUCT, TABLENAME = DEDSCHEM.DEDUCT,  KEYS = 3, WRITE = YES,
    KEYORDER = HIGH, DBSPACE = DEDSEG.DEDAREA,
    KEYFLD = SALEID/PAY_DATE, IXFLD = DEDEID/DEDDATE,$
```

For Oracle:

```
SEGNAME=SALINFO, TABLENAME=USER1.SALINFO, KEYS=2, WRITE=YES,
    KEYORDER = HIGH, DBSPACE=SPACE1,
SEGNAME = DEDUCT, TABLENAME = USER1.DEDUCT, KEYS = 3, WRITE=YES,
    KEYORDER = HIGH, DBSPACE=SPACE1,
     KEYFLD = SALEID/PAY_DATE, IXFLD = DEDEID/DEDDATE,$
```

SALDUCT Diagram

```
check file salduct pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   2  ( REAL=    2  VIRTUAL=   0 )
  NUMBER OF FIELDS=     7  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   42
 SECTION 01
             STRUCTURE OF SQLDS    FILE SALDUCT  ON 06/17/93 AT 13.52.47
          SALINFO
  01      S0
 **************
 *SALEID      **
 *PAY_DATE    **
 *GROSS       **
 *            **
 *            **
 ***************
  **************
        I
        I
        I
        I DEDUCT
  02    I S0
 **************
 *DEDEID      **
 *DEDDATE     **
 *DED_CODE    **
 *DED_AMT     **
 *            **
 ***************
  **************
```

SALARY Sample

The SALARY table contains data on salary and monthly pay deductions.

SALARY MASTER

For DB2:

```
FILENAME=SALARY, SUFFIX=DB2,$
SEGNAME=SALARY,  SEGTYPE=S0,$
 FIELD=EMPID,    ALIAS=EMPID,    USAGE=A7,   ACTUAL=A7,$
 FIELD=EMPNAME,  ALIAS=EMPNAME,  USAGE=A10,  ACTUAL=A10,$
 FIELD=SALARY,   ALIAS=PAY,      USAGE=P9.2, ACTUAL=P8,$
 FIELD=DEDUCT1,  ALIAS=DEDUCT1,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT2,  ALIAS=DEDUCT2,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT3,  ALIAS=DEDUCT3,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT4,  ALIAS=DEDUCT4,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT5,  ALIAS=DEDUCT5,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT6,  ALIAS=DEDUCT6,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT7,  ALIAS=DEDUCT7,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT8,  ALIAS=DEDUCT8,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT9,  ALIAS=DEDUCT9,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT10, ALIAS=DEDUCT10, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT11, ALIAS=DEDUCT11, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT12, ALIAS=DEDUCT12, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
SEGNAME=OCC, PARENT=SALARY, POSITION=DEDUCT1, OCCURS=12,$
 FIELD=TAX,      ALIAS=TAXDEDUC, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=ORDER,    ALIAS=ORDER,     USAGE=I4,   ACTUAL=I4,$
```

Note: For Teradata, change the suffix value to SQLDBC, for CA-IDMS, change the suffix value to SQLIDMS, and for Oracle, change the suffix value to SQLORA.

SALARY FOCSQL

For DB2:

```
SEGNAME = SALARY, TABLENAME = "USER1"."SALARY", KEYS = 1,
WRITE = NO,   DBSPACE = PUBLIC.SPACE0,$
```

For Teradata:

```
SEGNAME=SALARY, TABLENAME=USER1.SALARY, KEYS=1, WRITE= NO, FALLBACK=YES,$
```

For IDMS SQL:

```
SEGNAME = SALARY, TABLENAME = SALRSCHEM.SALARY, KEYS = 2,
 WRITE = YES,  DBSPACE = SALRSEG.SALRAREA,$
```

For Oracle:

```
SEGNAME=SALARY, TABLENAME=USER1.SALARY, KEYS=1, WRITE = NO,
   DBSPACE=SPACE1, $
```

SALARY Diagram With OCCURS Segment

```
check file salary pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   2  ( REAL=    2  VIRTUAL=   0 )
  NUMBER OF FIELDS=     6  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=  129
 SECTION 01
             STRUCTURE OF SQLDS    FILE SALARY   ON 06/16/93 AT 09.44.51
          SALARY
  01      S0
 **************
 *EMPID        **
 *EMPNAME      **
 *SALARY       **
 *DEDUCT       **
 *             **
 ***************
  **************
        I
        I
        I
        I OCC
  02    I S0
 **************
 *TAX          **
 *ORDER        **
 *             **
 *             **
 *             **
 ***************
  **************
```

DPBRANCH Sample

The DPBRANCH table contains branch information.

DPBRANCH Table Definition

```
CREATE TABLE DPBRANCH
  (BRANCH_NUMBER    INTEGER      NOT NULL,
   BRANCH_NAME      CHAR(5)      NOT NULL,
   BRANCH_MANAGER   CHAR(5)      NOT NULL,
   BRANCH_CITY      CHAR(5)      NOT NULL);
```

DPBRANCH Contents

```
SELECT * FROM DPBRANCH;
BRANCH_NUMBER BRANCH_NAME BRANCH_MANAGER BRANCH_CITY
------------- ----------- -------------- -----------
            1 WEST        PIAF           NY
            2 EAST        SMITH          NY
            3 NORTH       AMES           NY
            4 EAST        ALVIN          MIAMI
            5 WEST         FIELD          MIAMI
```

DPINVENT Sample

The DPINVENT table contains inventory information.

DPINVENT Table Definition

```
CREATE TABLE DPINVENT
  (BRANCH_NUMBER    INTEGER      NOT NULL,
   VENDOR_NUMBER    INTEGER      NOT NULL,
   PRODUCT          CHAR(5)      NOT NULL,
   NUMBER_OF_UNITS  INTEGER      NOT NULL,
   PER_UNIT_VALUE    DECIMAL(9,2) NOT NULL);
```

DPINVENT Contents

```
SELECT * FROM DPINVENT;
BRANCH_NUMBER VENDOR_NUMBER PRODUCT NUMBER_OF_UNITS PER_UNIT_VALUE
------------- ------------- ------- --------------- --------------
            1             1 RADIO                 5          12.95
            2             1 RADIO                 5          12.95
            3             1 RADIO                 5          12.95
            4             1 RADIO                 5          12.95
            1             3 MICRO                 6         110.00
            4             3 MICRO                 2         110.00
            2             3 MICRO                 1         110.00
            2             3 MICRO                 2          110.00
```

DPVENDOR Sample

The DPVENDOR table contains vendor information.

DPVENDOR Table Definition

```
CREATE TABLE DPVENDOR
  (VENDOR_NUMBER    INTEGER      NOT NULL,
   VENDOR_NAME      CHAR(5)      NOT NULL,
   VENDOR_CITY      CHAR(5)      NOT NULL);
```

DPVENDOR Contents

```
SELECT * FROM DPVENDOR;
VENDOR_NUMBER VENDOR_NAME VENDOR_CITY
------------- ----------- -----------
            1 ACME        NY
            2 STAR        OMAHA
            3 MMT         NY
            4 PIKE        MIAMI
```

Appendix D

Tracing Adapter Processing

The adapter communicates with the RDBMS through SQL statements it generates on your behalf. You can view these SQL statements with the trace facility. Traces are helpful for debugging procedures or for adapter performance analysis. The trace facility is easy to invoke, requires no changes to either the adapter or FOCUS request, and have no effect on how the adapter functions.

The trace facility can provide several types of information, organized into trace components and levels.

Using a SET command, you must turn on each trace component and level you want to generate prior to issuing the request to be traced.

Note: The trace facility is intended for use in query optimization and problem debugging. Application programs should not be written to depend on the format or content of any trace, as they may change in later releases of the adapter.

In this appendix:

Available Traces

There are four types of trace available for the relational adapters. The following chart lists the associated trace component name and function for each trace:

Component	Trace Function
SQLDI	Records SQL statements, RDBMS return codes, COMMIT and ROLLBACK commands, and SQL cursor operations such as PREPARE, OPEN, FETCH, and CLOSE. You can use this trace component with all FOCUS report requests, MODIFY requests, and with native SQL commands. Has levels 1, 2, and 3.
SQLAGGR	Displays adapter-to-RDBMS aggregation and join analysis. Indicates whether the adapter successfully transferred aggregation and join operations to the RDBMS. You can use it only for FOCUS reporting operations such as TABLE, GRAPH, and MATCH FILE. When the adapter is able to pass all join, sort, and aggregation operations to the RDBMS, it does not populate the SQLAGGR trace. In this case, the message "AGGREGATION DONE..." appears either at the terminal or in the job output, whichever is appropriate. Has level 1 only.
STMTRACE	Records SQL SELECT statements generated by the adapter for FOCUS report requests, MODIFY procedures, or Direct SQL Passthru SELECT requests. It also records the SQL Data Definition Language (DDL) statements generated by the CREATE FILE command. You can display the trace information online or store it in a file or sequential data set. This technique can save you time and effort when you create tables and indexes. The adapter terminates its generated SQL SELECT statements with a semicolon. Therefore, you can submit them to the RDBMS for processing, interactively or in batch, without any modifications. Thus, you can use STMTRACE for debugging, performance tuning, and for capturing SQL Data Definition and Data Manipulation statements to reuse. Has levels 1 and 2.
SQLCALL	Traces commands and data exchange between the physical and the logical layers of the adapter.

You can activate all or any combination of these traces during your FOCUS session or in batch. You can display the results online or store them in a file or sequential data set.

Activating Trace Components

You must activate the trace destination as well as the trace components. For information on activating the trace destination, see *Activating the Trace Destination* on page 490.

Syntax: How to Activate Trace Components

Activate specific trace components prior to running the request for which you want the trace generated:

```
SET TRACEON = component / [level] / destination
```

where:

component

Is the component to activate. Components for the relational adapters are:

SQLDI traces the SQL physical layer (formerly FSTRACE)

SQLAGGR provides optimization information (formerly FSTRACE3)

STMTRACE displays generated SQL statements (formerly FSTRACE4)

SQLCALL traces commands and data exchange between the physical and the logical layers of the adapter.

level

If a component is associated with multiple levels, identifies the trace level to activate. Trace levels are not necessary for the relational components. You must indicate their absence with a double slash (//).

destination

Can be one of the following:

FSTRACE returns the trace output to the destination indicated in the allocation for DDNAME FSTRACE. You must explicitly allocate this DDNAME.

CLIENT displays the trace output on the screen of an online session. No explicit allocation is necessary.

Syntax: How to Control Trace Timestamps

By default, a timestamp is prepended to each line of trace output. You can turn off the trace timestamps using the SET TRACESTAMP command.

```
SET TRACESTAMP = {ON|OFF|TLEFT}
```

where:

ON

Prepends a timestamp to each line of trace output. ON is the default value.

OFF

Omits the timestamp from the trace output.

TLEFT

Prepends a timestamp to each line of trace output. Includes microseconds in the timestamp.

Activating the Trace Destination

Traces can go to the screen (CLIENT) or to a file (allocated to DDNAME FSTRACE). You must activate the trace destination as well as activating each trace component.

Syntax: How to Activate the Trace Destination

You activate the tracing facility by issuing the following command at the command level, in a FOCEXEC, or in any FOCUS-supported profile:

```
SET TRACEUSER={OFF|ON|FSTRACE}
```

where:

OFF

Does not activate any trace destination. OFF is the default value.

ON

Activates the screen as the trace destination for any trace component you activate using CLIENT as the component destination.

FSTRACE

Activates a file as the trace destination for any trace component you activate using FSTRACE as the destination. You must allocate DDNAME FSTRACE to a file (for information, see *Allocating FSTRACE* on page 492).

Note: You must issue the SET TRACEUSER=FSTRACE command after activating all of the trace components you want to go to the allocated file. In the SET TRACEON commands for those components, specify FSTRACE as the destination. For example:

```
DYNAM ALLOC DD FSTRACE DA USER1.FSTRACE.DATA SHR REU
SET TRACEON = STMTRACE//FSTRACE
SET TRACEON = SQLDI//FSTRACE
SET TRACEUSER = FSTRACE
```

Deactivating Trace Components

```
SET TRACEOFF=ALL
SET TRACEOFF = component [/ [level] [/ [destination] ] ]
```

where:

ALL

Deactivates all traces for all components.

component

Deactivates traces for the named component. If omitted, the command applies to all components.

level

Identifies the trace level to be deactivated. If omitted, all trace levels are deactivated.

destination

Turns off the trace levels associated with DDNAME. Valid values for DDNAME are FSTRACE or CLIENT.

Tip: To make sure that only the traces you want are activated, issue the following command prior to activating any components:

```
SET TRACEOFF = ALL
```

Trace Activation and Deactivation Examples

The following commands activate the trace facility to the screen, make sure that only the trace components we want are activated, and activates the SQLDI trace component to the screen:

```
SET TRACEUSER = ON
SET TRACEOFF = ALL
SET TRACEON = SQLDI//CLIENT
```

The following command activates the STMTRACE component and writes the trace output to the file allocated to DDNAME FSTRACE. Note that the SET TRACEUSER=FSTRACE command is issued after all trace components are activated:

```
SET TRACEOFF = ALL
SET TRACEON = STMTRACE//FSTRACE
SET TRACEUSER = FSTRACE
```

The following command turns off all trace components allocated to DDNAME FSTRACE:

```
SET TRACEOFF = //FSTRACE
```

Querying Traces

The trace query commands tell you which trace components and levels are either activated or deactivated for *every* component. Be aware that these commands may show a voluminous amount of information that does not pertain to the components in which you are interested.

To list all of the trace level/component combinations currently active, issue the following command:

```
SET TRACEON = ?
```

To list all of the trace level/component combinations not currently active, issue the following command:

```
SET TRACEOFF = ?
```

Example: Querying Traces

The following command queries the active traces. None are activated:

```
>  > set traceon = ?
Name     Level Description                    Set  Comp.ID
>  >
```

The next commands activate the SQLCALL and STMTRACE traces:

```
SET TRACEON = SQLCALL//CLIENT
SET TRACEON = STMTRACE//CLIENT
```

Now, issuing the query command generates a list of all active trace components:

```
>  > set traceon = ?
Name     Level Description                    Set  Comp.ID
STMTRACE  1    SQL/MDX Generated Statement Trace
STMTRACE  2    SQL/MDX Generated Substatement Trc
SQLCALL   1    SQL Call to Physical Layer
```

Allocating FSTRACE

You can allocate FSTRACE during your session or in batch. You can store the results in a file or sequential data set.

Tip: The trace facilities are intended for use in query optimization and problem debugging. Application programs should not be written to depend on the format or content of any trace, as they may change in later releases.

How to Allocate FSTRACE Online

You can allocate FSTRACE to a z/OS sequential data set. To capture trace data in a sequential file, issue the appropriate command from the command level. For example:

```
{MVS|TSO} ALLOC F(FSTRACE) DA('userid.FSTRACE.DATA') SHR REU
```

or

```
DYNAM ALLOC DD FSTRACE DATASET userid.FSTRACE.DATA SHR REUSE
```

Note: DCB attributes are LRECL=80 and RECFM=F.

To view the trace information, use the system editor or the FOCUS TED editor.

How to Allocate FSTRACE in Batch

You can write trace results to SYSOUT. BLKSIZE information is optional, but should be compatible with other FSTRACE formats. For example, to allocate DDNAME FSTRACE:

```
//FSTRACE   DD   SYSOUT=*,DCB=(LRECL=80,BLKSIZE=80,RECFM=F)
```

You can also write trace results to a z/OS sequential data set. First, allocate the FSTRACE data set in a prior batch step (as shown) or in ISPF:

```
//ALLOC    EXEC PGM=IEFBR14
//FSTRACE  DD   DISP=(,CATLG),DSN=userid.FSTRACE.DATA,
//         UNIT=SYSDA,VOL=SER=USERM1,SPACE=(TRK,(5,5)),
//         DCB=(LRECL=80,BLKSIZE=80,RECFM=F)
                     .
                     .
                     .
```

Then, allocate the trace data set with DISP=MOD in the batch FOCUS JCL:

```
                     .
                     .
                     .
//FOCBATCH EXEC PGM=FOCUS
//FSTRACE  DD   DISP=(MOD,KEEP,KEEP),DSN=userid.FSTRACE.DATA
```

How to Free Trace Allocations

To clear the FSTRACE allocation:

```
{MVS|TSO} FREE F(FSTRACE)
```

or

```
DYNAM FREE FILE FSTRACE
```

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

TIBCO, the TIBCO logo, the TIBCO O logo, FOCUS, iWay, Omni-Gen, Omni-HealthData, and WebFOCUS are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

Index

B

C

D

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z