APPENDIX

2

DBMS Overview and Information for the Database Administrator

Overview of Relational Database Management Systems

A relational DBMS organizes and accesses data according to relationships among data items. The relationships among data items are expressed by tables consisting of columns and rows. The order of the rows and columns is not significant. Each *column* can contain one type of data, and each *row* can hold one data value for each column. See Figure A2.1 on page 250 for a conceptual picture of a table.

In a relational DBMS, you use a high-level language to operate on the data managed by the DBMS. In many cases, that language is SQL. A DBMS enables you, or an application program such as the SAS System, to use SQL statements to read, modify, create, and protect the data it manages.

Databases

A *database* is a collection of objects that includes tables, views, stored procedures, triggers, and indexes. Each database contains several database system files and data files. Databases are usually created by a database administrator with the SQL command CREATE DATABASE. In most databases, you use the GRANT statement to give users privileges on database objects to users. You can create SAS/ACCESS

descriptor files to access the data in the objects, or access the objects directly by using the SAS/ACCESS LIBNAME statement.

Tables

A *table* is a named object that consists of a specific number of columns and some number of rows. The rows have no inherent order, unlike observations in a SAS data set.

In many databases, tables are created with the SQL statement CREATE TABLE. This statement names the table and the columns and defines the *data type* of each column.

Figure A2.1 on page 250 illustrates four columns from the CUSTOMERS table and highlights a column and a row.

Figure A2.1 A DBMS Table

column

CUSTOMER	CITY	STATE	COUNTRY	
14324742	San Jose	CA	USA	
14569877	Memphis	TN	USA	row
14898029	Rockville	MD	USA	
26422096	La Rochelle		France	
38763919	Buenos Aires		Argentina	
46783280	Singapore		Singapore	

Table Privileges

In a database, when you create a table or other database object, you *own it*. The owner of the table can do the following, depending on his or her privileges:

- select columns and rows from the table for viewing
- add or remove columns from the table
- insert, delete, or update rows
- create other objects (indexes and views) based on the table
- alter or drop the table
- grant privileges to other users and applications programs.

If you want to use a table that you do not own, the table's owner must grant you privileges to the table.

Views

A *view* is a named object that usually consists of an SQL SELECT statement that defines a group of data derived from one or more DBMS tables or views. The view itself contains no data. Views are usually created with the SQL statement CREATE VIEW, which names the view and includes an SQL SELECT statement that retrieves the data.

When you reference a view in an SQL statement, the DBMS accesses the table(s) or view(s) on which the view is based. Certain restrictions might apply when a view derives its data from multiple tables or other views. For example, updates, inserts, and deletes might not be allowed. For more information on view restrictions, refer to your DBMS documentation.

Note: To access a view owned by another user, you must be granted privileges on the view and on its underlying tables and views. Having privileges on the table(s) from which a view derives its data does not automatically give you privileges on the view. △

Note: If a table from which a view derives its data is dropped, the view itself is also dropped. △

Indexes

An *index* is a database object that speeds data retrieval by directing the DBMS to the location in storage of a particular row for a given column. Indexes are usually created with the SQL statement CREATE INDEX, which assigns the index a name and identifies the table and column(s) on which the index is created. In many DBMSs, the CREATE INDEX statement enables you to specify the keyword UNIQUE to define a unique index on one or more columns. In some DBMSs, you can also use a CONSTRAINT clause with an ALTER TABLE statement to specify the keywords UNIQUE and USING INDEX, which define a unique index on one or more columns. Unique indexes prevent duplicate values in the column, or combination of columns, on which they are created. Some DBMSs allow you to create different types of indexes, such as sorted or hashed indexes. See your DBMS documentation for details.

An index is keyed on all specified columns unless the KEY option is used. Keys provide a way to identify rows and relate (or join) rows in one table to rows in another table. A *primary key* is the column or combination of columns that uniquely identify a row. A *foreign key* is a column or combination of columns in one table that reference the primary key in another table. The foreign key must have the same attributes as the primary key it references.

If the table on which an index is created is dropped, the index is also dropped.

Most DBMSs automatically determine the most efficient way to process an SQL statement and uses the appropriate indexes if they are available. In addition, the way you specify criteria for selecting rows can affect whether the DBMS can use the indexes. Your database administrator can help you determine whether your selection criteria enable the DBMS to use your tables' indexes.

Stored Procedures

Stored procedures are another type of named object. They usually consist of SQL statements, except for CREATE statements, and include control-of-flow keywords such as IF, ...ELSE, BEGIN, and ...END. Because stored procedures are precompiled, they are more efficient than submitting the same SQL statements individually or in batches.

Stored procedures can generally take parameters, return status values and parameters, and call other procedures. They can also be executed on remote servers. Stored procedures are often created to execute groups of SQL statements that are submitted repeatedly. For example, a stored procedure might be created to list all the orders that were backordered for a particular stock number and to remove the back-order status from that stock number. The stored procedure could then be used whenever a formerly out-of-stock product is back in stock.

See your DBMS documentation for information on whether your DBMS supports stored procedures and how to create them.

Triggers

A *trigger* is a stored procedure that is associated with a table. They are often used to help maintain *referential integrity*–that is, consistency among related data that is

stored in different tables–or to maintain business constraints. Triggers are executed whenever data modification operations (UPDATE, INSERT, or DELETE) are performed on a specified column. You use the CREATE TRIGGER statement to create a trigger. You can use triggers to

- enforce security authorizations or business-specific constraints
- provide auditing and transparent event-logging
- generate derived column values automatically
- maintain replicated asynchronous tables.

A trigger is automatically executed, or "fired," by the DBMS when a specified SQL statement is issued against a table. You can define triggers for SQL DELETE, INSERT, and UPDATE statements. You also define whether a trigger fires before or after a specified SQL statement is executed or once for each row affected by the statement.

The SAS/ACCESS interface view engine generally supports all actions and constraints specified by triggers. For more information, refer to your DBMS documentation.

Constraints

A *constraint* restricts the values that can be stored in a table. A constraint checks the values you insert or update in a table against the conditions specified by the constraint. For example, a constraint could ensure that the value entered for the column GENDER is only **`M`** or **`F`**. Constraints help ensure the referential integrity in a database. See your DBMS documentation for additional information on how your DBMS supports referential integrity.

Information for the Database Administrator

This section explains how SAS/ACCESS works so you can decide how to administer its use at your site.

How SAS/ACCESS Works

When you use the ACCESS procedure to create an access descriptor, the interface issues a SELECT statement to the data dictionary tables in your DBMS. The ACCESS procedure then issues the equivalent of a DESCRIBE statement to gather information about the columns in the specified table. The access descriptor's information about the table and its columns is then copied into the view descriptor when it is created. Therefore, it is not necessary for the SAS System to call the DBMS when it creates a view descriptor.

When you use the DBLOAD procedure to create a table, the procedure issues dynamic SQL statements to create the table and insert data from a SAS data file, DATA step view, PROC SQL view, or view descriptor into the DBMS table.

PROC SQL Pass-Through statements and the CONNECTION TO component are passed directly to the DBMS as soon as they are submitted. They are passed by using Pass-Through functions in the SAS/ACCESS interface view engine.

When you use the DATA step or any SAS procedure with a view descriptor, the SAS/ACCESS interface view engine issues SQL calls to the DBMS.

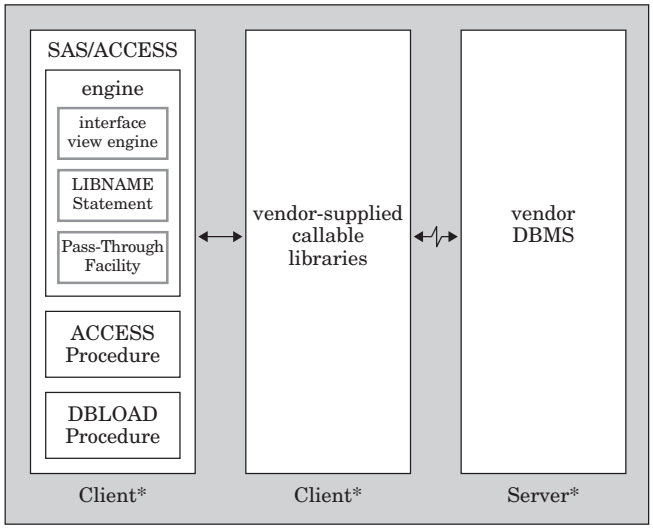
The following sections explain the calls that the SAS System makes to the DBMS when you use the ACCESS and DBLOAD procedures or the SQL Procedure

Pass-Through Facility. The connection between the SAS System and the DBMS is illustrated in Figure A2.2 on page 253.

Note: Currently, the SAS/ACCESS Interface to Informix does not support the ACCESS and DBLOAD procedures. △

Note: Currently, the SAS/ACCESS Interface to Oracle Rdb does not support the SAS/ACCESS LIBNAME statement. △

Figure A2.2 How the SAS System Connects to the DBMS



* In some cases, both client and server software can reside on the same machine.

Using the ACCESS Procedure to Create an Access Descriptor

When you create an access descriptor, the SAS/ACCESS interface view engine requests the DBMS to execute an SQL SELECT statement dynamically by using DBMS-specific calling routines or interface software. The following steps are completed:

1. When you supply the connection information to PROC ACCESS, the SAS/ACCESS interface calls the DBMS to connect to the database.
2. The SAS System constructs a SELECT * FROM *table-name* statement and passes it to the DBMS to retrieve information about the table from the DBMS data dictionary. This SELECT statement is based on the information you supplied to PROC ACCESS. Using a SELECT statement enables the SAS System to determine whether the table exists and can be accessed.
3. The SAS/ACCESS interface calls the DBMS to get table description information, such as the column names, data types (including width, precision, and scale), and whether the columns accept null values.
4. The SAS System closes the connection with the DBMS.

Reading Data

When you use a view descriptor, DATA step, or procedure to read DBMS data, the SAS/ACCESS interface view engine requests the DBMS to execute an SQL SELECT statement. The interface view engine follows these steps:

1. Using the connection information provided during creation of the access descriptor, the SAS/ACCESS interface calls the DBMS to connect to the database.
2. The SAS System constructs a SELECT statement that is based on the information stored in the view descriptor (table name and selected columns and their characteristics) and passes this information to the DBMS.
3. The SAS System fetches the data from the DBMS table and passes it back to the SAS procedures as if it were observations in a SAS data set.
4. The SAS System closes the connection with the DBMS.

For example, if you execute the following SAS program using a view descriptor, the previous steps are executed once for the PRINT procedure, then a second time for the GCHART procedure. (The data used for the two procedures is not necessarily the same because the table might have been updated by another user between procedure executions.)

```
proc print data=vlib.allemp;
run;

proc gchart data=vlib.allemp;
   vbar jobcode;
run;
```

Updating Data

You use a view descriptor, DATA step, or procedure to update DBMS data in much the same way as when reading data. In addition, the following steps might occur:

1. Using the connection information provided during creation of the access descriptor, the SAS/ACCESS interface calls the DBMS to connect to the database.
2. When rows are added to a table, the SAS System constructs an SQL INSERT statement and passes it to the DBMS. When you reference a view descriptor, you can use the ADD command in FSEDIT and FSVIEW, the APPEND procedure, or an INSERT statement in PROC SQL to add data to a DBMS table. (Or, you can use the SQL Procedure Pass-Through Facility's EXECUTE statement to add, delete, or modify DBMS data directly. Literal values must be used when inserting data by using the Pass-Through Facility.)
3. When rows are deleted from a DBMS table, the SAS System constructs an SQL DELETE statement and passes it to the DBMS. (When you reference a view descriptor, you can use the DELETE command in FSEDIT and FSVIEW or a DELETE statement in PROC SQL to delete rows from a DBMS table.)
4. When data in the rows is modified, the SAS System constructs an SQL UPDATE statement and passes it to the DBMS. (When you reference a view descriptor, you can use FSEDIT, the MODIFY command in FSVIEW, or an INSERT statement in PROC SQL to update data in a DBMS table. You can also reference a view descriptor in the DATA step's UPDATE, MODIFY, and REPLACE statements.)
5. The SAS System closes the connection with the DBMS.

Using the DBLOAD Procedure to Create DBMS Tables

You create DBMS tables from SAS data sets by using the DBLOAD procedure. The SAS/ACCESS interface view engine follows these steps:

1. When you supply the connection information to PROC DBLOAD, the SAS/ACCESS interface calls the DBMS to connect to the database.

2 The SAS System uses information that is provided by the DBLOAD procedure to construct a SELECT * FROM *table-name* statement, and it passes the information to the DBMS to determine if the table already exists. PROC DBLOAD continues only if a table with that name does not exist, unless you use the DBLOAD APPEND option.

3 The SAS System uses information that is provided by the DBLOAD procedure to construct an SQL CREATE TABLE statement and passes it to the DBMS.

4 The SAS System constructs an SQL INSERT statement for the current observation and passes it to the DBMS. New INSERT statements are constructed and then executed repeatedly until all of the observations from the input SAS data set are passed to the DBMS. Some DBMSs have a bulkcopy capability that allows a group of observations to be inserted at once. See your DBMS documentation to determine if your DBMS has this capability.

5 Additional nonquery SQL statements specified in the DBLOAD procedure are executed as submitted by the user. The DBMS returns an error message if a statement does not execute successfully.

6 The SAS System closes the connection with the DBMS.

Using the SQL Procedure Pass-Through Facility to Read and Update DBMS Data

To read and update data with the SQL Procedure Pass-Through Facility, the SAS/ACCESS interface view engine passes SQL statements directly to the DBMS for processing. Here are the steps:

- You pass a PROC SQL CONNECT statement to the SAS/ACCESS interface view engine to establish a connection with the specified database.
- You use a CONNECTION TO component in a PROC SQL SELECT statement to read data from a DBMS table or view. The SELECT statement (that is, PROC SQL query) can be stored as a PROC SQL view.
- You use a PROC SQL EXECUTE statement to pass any dynamic, non-query SQL statements (such as INSERT, DELETE, and UPDATE) to a database. INSERT statements must contain literal values.
- In the EXECUTE statement and CONNECTION TO component, all statements are passed to the DBMS exactly as you have typed and submitted them.
- You terminate the connection with the DISCONNECT statement.

Ensuring Data Security

The SAS System preserves the data security provided by your DBMS and the operating system. The DBA controls who has privileges to access or update DBMS objects. The DBA also controls who can create objects, and creators of the objects control who can access the objects. A user cannot use DBMS facilities through the SAS/ACCESS LIBNAME statement, the SAS/ACCESS interface view engine, the ACCESS procedure, or the DBLOAD procedure, unless the user has the appropriate DBMS privileges or authority on those objects. For example, only users who have object privileges for a DBMS table can create an access descriptor on that table.

To secure data from accidental update or deletion, you can take precautionary measures in both your DBMS and the SAS System.

On the DBMS, give users only the privileges they must have. Privileges are granted on whole tables or views. A user must explicitly be granted privileges on the DBMS tables or views underlying a view to use that view.

You can grant privileges on the DBMS side by using the SQL Procedure Pass-Through Facility to submit an SQL GRANT statement, or issue a GRANT statement from the DBLOAD procedure SQL statement.

Using Triggers to Enhance Security

If your DBMS supports triggers, you can use them to enforce security authorizations or business-specific security considerations. When and how triggers are executed is determined by when the SQL statement is executed and how often the trigger executes. Triggers can be executed before an SQL statement is executed, after an ORACLE SQL statement is executed, or can be executed for each row of an SQL statement. Also, triggers can be defined for DELETE, INSERT, and UPDATE statement execution.

Enabling triggers can provide more specific security for delete, insert, and update operations. The SAS/ACCESS interface view engine abides by all constraints and actions that are specified by a trigger. For more information, see your DBMS documentation.

SAS System Security

To secure DBMS data from accidental update or deletion, you can follow these steps on the SAS System side of the interface:

- Create access descriptors yourself and drop columns from the display that contain sensitive data by using the DROP statement.
- Give users read-only access to the SAS data library in which you store the access descriptors.
- Create all view descriptors yourself and control their use.
- Assign SAS System passwords to access descriptors and view descriptors. For more information, see Chapter 9, "ACCESS Procedure Reference".

Note: On CA-OpenIngres, databases are created as public databases, unless the -p flag is used with the CREATEDB command to create a private database. When creating access descriptors, the interface view engine issues PREPARE and DESCRIBE statements to gather information about the table. The DESCRIBE statement uses CA-OpenIngres system catalogs to retrieve the information, and CA-OpenIngres does not check permissions. Therefore, users can create access descriptors and view descriptors on tables on which they have no privileges. This is not caused by any security circumvention by the SAS System; this is the way CA-OpenIngres operates. However, users cannot use the view descriptors to view or extract the data. △

In the SAS/ACCESS LIBNAME statement, you can specify the DBPROMPT= option to defer providing connection information until connection time. When you use DBPROMPT=, you do not need to save connection information in your code.

SAS provides the ability to create SQL views that can be protected from unauthorized access by applying passwords. Also, you can use the DBMS security provided by the DBMS to restrict table access by user ID. See your DBMS documentation for more details.

When you create an access descriptor, the connection information that you provide is stored in the access descriptor and in any view descriptors based on that access descriptor. The password is stored in an encrypted form. When these descriptors are accessed, the connection information that was stored is also used to access the DBMS table or view. To ensure data security, you might want to change the protection on the descriptors to prevent others from seeing the connection information stored in the descriptors.

An alternative is to leave the user name, password, and other connection arguments blank when you create descriptors. In this case, access to the DBMS is denied unless

the correct user and password information is stored in a local environment variable. See your DBMS chapter to determine if this alternative is supported.

You can also use the ACCESS procedure to create access descriptors in which you specify that particular columns be dropped from the descriptor. Columns that are dropped from an access descriptor do not affect the DBMS table and can be reselected for later use.

Note: SAS/ACCESS does not override your DBMS's security. △

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ACCESS® Software for Relational Databases: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS/ACCESS® Software for Relational Databases: Reference, Version 8

ISBN 1–58025–558–2

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999