CHAPTER 7

DATABASE ADMINISTRATION

LEARNING OBJECTIVES

Objectives

- Understand, create, and drop views
- Recognize the benefits of using views
- Use a view to update data
- Grant and revoke users' database privileges
- Understand the purpose, advantages, and disadvantages of using an index
- Create, use, and drop an index
- Understand and obtain information from the system catalog
- Use integrity constraints to control data entry

INTRODUCTION

There are some special issues involved in managing a database. This process, often called **database administration**, is especially important when more than one person uses the database. In a business organization, a person or an entire group known as the **database administrator** is charged with managing the database.

In Chapter 6, you learned about one function of the database administrator: changing the structure of a database. In this chapter, you will see how the database administrator can give each user his or her own view of the database. You will use the GRANT and REVOKE commands to assign different database privileges to different users. You will use indexes to improve database performance. You will learn how a

DBMS stores information about the database structure in an object called the system catalog and how to access that information. Finally, you will learn how to specify integrity constraints that establish rules for the data in the database.

CREATING AND USING VIEWS

Most DBMSs support the creation of views. A **view** is a program's or an individual user's picture of the database. The existing, permanent tables in a relational database are called **base tables**. A view is a derived table because the data in it comes from one or more base tables. To the user, a view appears to be an actual table, but it is not. In many cases, a user can examine table data using a view. Because a view usually includes less information than the full database, its use can represent a great simplification. Views also provide a measure of security, because omitting sensitive tables or columns from a view renders them unavailable to anyone accessing the database through the view.

To help you understand the concept of a view, suppose that Juan is interested in the part number, part description, units on hand, and unit price of parts in item class HW. He is not interested in any other columns in the PART table, nor is he interested in any rows that correspond to parts in other item classes. Viewing this data would be simpler for Juan if the other rows and columns were not even present. Although you cannot change the structure of the PART table and omit some of its rows just for Juan, you can do the next best thing. You can provide him with a view that consists of only the rows and columns that he needs to access.

A view is defined by creating a **defining query**, which indicates the rows and columns to include in the view. The SQL command (or the defining query) to create the view for Juan is illustrated in Example 1.

EXAMPLE 1

Create a view named HOUSEWARES that consists of the part number, part description, units on hand, and unit price of each part in item class HW.

To create a view, use the **CREATE VIEW** command, which includes the words CREATE VIEW, followed by the name of the view, the word AS, and then a query. The CREATE VIEW command shown in Figure 7-1 creates a view of the PART table that contains only the specified columns.

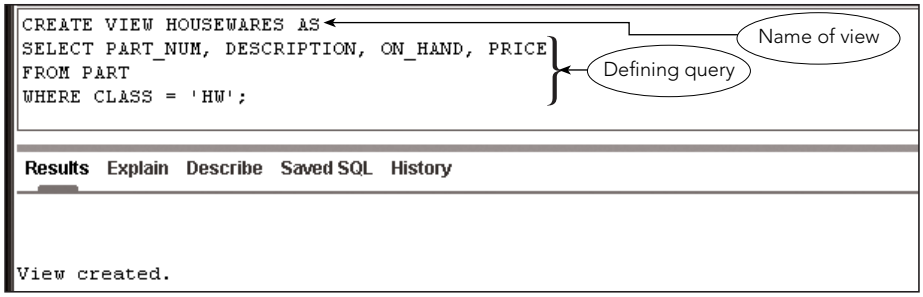

FIGURE 7-1 Creating the HOUSEWARES view

ACCESS USER NOTE

Access does not support the CREATE VIEW command. To create a view in Access, create a query to define the view, and then save the query object in the database using the view's name (for example, HOUSEWARES).

Given the current data in the Premiere Products database, the HOUSEWARES view contains the data shown in Figure 7-2.

HOUSEWARES

PART_NUM	DESCRIPTION	ON_HAND	PRICE
AT94	Iron	50	$24.95
DL71	Cordless Drill	21	$129.95
FD21	Stand Mixer	22	$159.95

FIGURE 7-2 HOUSEWARES view

The data does not actually exist in this form, nor will it *ever* exist in this form. It is tempting to think that when Juan uses this view, the query is executed and produces some sort of temporary table, named HOUSEWARES, that Juan can access, but this is *not* what actually happens. Instead, the query acts as a sort of "window" into the database, as shown in Figure 7-3. As far as Juan is concerned, the entire database is just the darker shaded portion of the PART table. Juan can see any change that affects the darker portion of the PART table, but he is totally unaware of any other changes that are made in the database.

PART

PART_NUM	DESCRIPTION	ON_HAND	CLASS	WAREHOUSE	PRICE
AT94	Iron	50	HW	3	$24.95
BV06	Home Gym	45	SG	2	$794.95
CD52	Microwave Oven	32	AP	1	$165.00
DL71	Cordless Drill	21	HW	3	$129.95
DR93	Gas Range	8	AP	2	$495.00
DW11	Washer	12	AP	3	$399.99
FD21	Stand Mixer	22	HW	3	$159.95
KL62	Dryer	12	AP	1	$349.95
KT03	Dishwasher	8	AP	3	$595.00
KV29	Treadmill	9	SG	2	$1,390.00

FIGURE 7-3 Juan's view of the PART table

When you create a query that involves a view, the DBMS changes the query to one that selects data from the table(s) in the database that created the view. For example, suppose Juan creates the query shown in Figure 7-4.

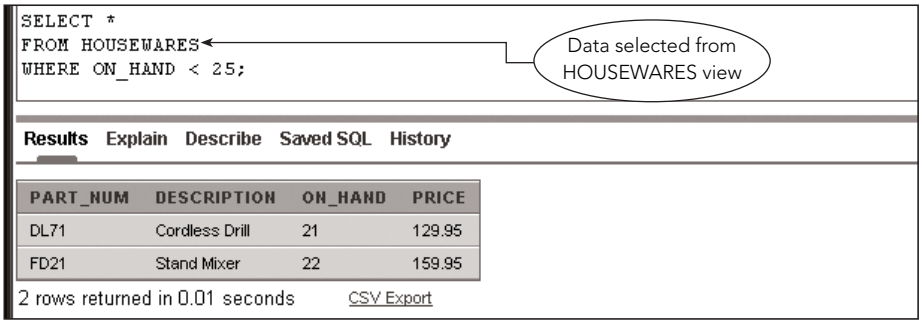

FIGURE 7-4 Using the HOUSEWARES view

The DBMS does not execute the query in this form. Instead, it merges the query Juan entered with the query that creates the view to form the query that is actually executed. When the DBMS merges the query that creates the view with Juan's query to select rows for which the ON_HAND value is less than 25, the query that the DBMS actually executes is:

```
SELECT PART_NUM, DESCRIPTION, ON_HAND, PRICE
FROM PART
WHERE CLASS = 'HW'
AND ON_HAND < 25;
```

In the query that the DBMS executes, the FROM clause lists the PART table rather than the HOUSEWARES view, the SELECT clause lists columns from the PART table instead of * to select all columns from the HOUSEWARES view, and the WHERE clause contains a compound condition to select only those parts in the HW class (as Juan sees in the

HOUSEWARES view) and only those parts with ON_HAND values of less than 25. This new query is the one that the DBMS actually executes.

Juan, however, is unaware that this activity is taking place. To Juan, it seems that he is really using a table named HOUSEWARES. One advantage of this approach is that because the HOUSEWARES view never exists in its own right, any update to the PART table is *immediately* available in the HOUSEWARES view. If the HOUSEWARES view were really a table, this immediate update would not be possible.

You also can assign column names that are different from those in the base table, as illustrated in the next example.

EXAMPLE 2

Create a view named HSEWRES that consists of the part number, part description, units on hand, and unit price of all parts in item class HW. In this view, change the names of the PART_NUM, DESCRIPTION, ON_HAND, and PRICE columns to PNUM, DSC, OH, and PRCE, respectively.

When renaming columns, you include the new column names in parentheses following the name of the view, as shown in Figure 7-5. In this case, anyone accessing the HSEWRES view will refer to PART_NUM as PNUM, to DESCRIPTION as DSC, to ON_HAND as OH, and to PRICE as PRCE.

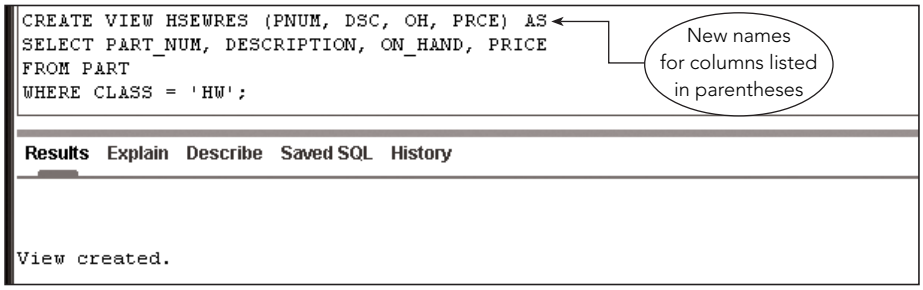

FIGURE 7-5 Renaming columns when creating a view

If you select all columns from the HSEWRES view, the output displays the new column names, as shown in Figure 7-6.

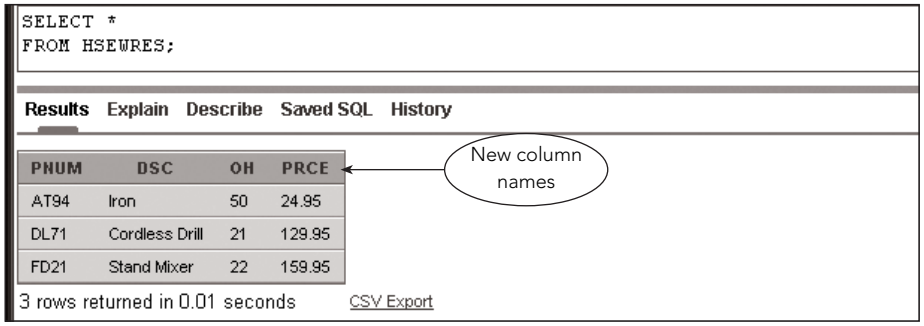

FIGURE 7-6 Data in the HSEWRES view

ACCESS USER NOTE

To change column names in Access, use AS clauses in the query design (for example, SELECT PART_NUM AS PNUM, DESCRIPTION AS DSC, and so on).

The HSEWRES view is an example of a **row-and-column subset view** because it consists of a subset of the rows and columns in some base table—in this case, in the PART table. Because the defining query can be any valid SQL query, a view also can join two or more tables or involve statistics. The next example illustrates a view that joins two tables.

EXAMPLE 3

Create a view named REP_CUST consisting of the sales rep number (named RNUM), sales rep last name (named RLAST), sales rep first name (named RFIRST), customer number (named CNUM), and customer name (named CNAME) for all sales reps and matching customers in the REP and CUSTOMER tables.

The command to create this view appears in Figure 7-7.

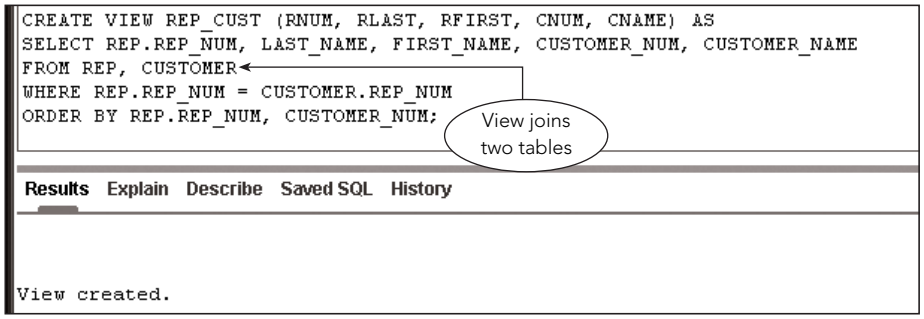

FIGURE 7-7 Creating the REP_CUST view

Given the current data in the Premiere Products database, the REP_CUST view contains the data shown in Figure 7-8.

```
SELECT *
FROM REP_CUST;
```

Results | Explain | Describe | Saved SQL | History

RNUM	RLAST	RFIRST	CNUM	CNAME
20	Kaiser	Valerie	148	Al's Appliance and Sport
20	Kaiser	Valerie	524	Kline's
20	Kaiser	Valerie	842	All Season
35	Hull	Richard	282	Brookings Direct
35	Hull	Richard	408	The Everything Shop
35	Hull	Richard	687	Lee's Sport and Appliance
35	Hull	Richard	725	Deerfield's Four Seasons
65	Perez	Juan	356	Ferguson's
65	Perez	Juan	462	Bargains Galore
65	Perez	Juan	608	Johnson's Department Store

10 rows returned in 0.02 seconds CSV Export

FIGURE 7-8 Data in the REP_CUST view

SQL SERVER USER NOTE

SQL Server does not support the ORDER BY clause in a CREATE VIEW command. When you need to order the query results, insert an ORDER BY clause in the SELECT command when you query the view. For example, the following SELECT command retrieves all records in the REP_CUST view ordered by rep number and customer number:

```
SELECT *
FROM REP_CUST
ORDER BY RNUM, CNUM
```

A view also can involve statistics, as illustrated in Example 4.

EXAMPLE 4

Create a view named CRED_CUST that consists of each credit limit (CREDIT_LIMIT) and the number of customers having this credit limit (NUM_CUSTOMERS). Sort the credit limits in ascending order.

The command shown in Figure 7-9 creates this view.

```
CREATE VIEW CRED_CUST (CREDIT_LIMIT, NUM_CUSTOMERS) AS
SELECT CREDIT_LIMIT, COUNT(*)
FROM CUSTOMER
GROUP BY CREDIT_LIMIT
ORDER BY CREDIT_LIMIT;
```

Results Explain Describe Saved SQL History

View created.

FIGURE 7-9 Creating the CRED_CUST view

The SELECT command shown in Figure 7-10 displays the current data in the Premiere Products database for this view.

```
SELECT *
FROM CRED_CUST;
```

Results Explain Describe Saved SQL History

CREDIT_LIMIT	NUM_CUSTOMERS
5000	2
7500	4
10000	3
15000	1

4 rows returned in 0.01 seconds CSV Export

FIGURE 7-10 Data in the CRED_CUST view

The use of views provides several benefits. First, views provide data independence. When the database structure changes (by adding columns or changing the way objects are related, for example) in such a way that the view still can be derived from existing data, the user can access and use the same view. If adding extra columns to tables in the database is the only change, and these columns are not required by the view's user, the defining query might not even need to be changed for the user to continue using the view. If table relationships are changed, the defining query might be different, but because users are not aware of the defining query, they are unaware of this difference. Users continue accessing the database through the same view, as though nothing has changed. For example, suppose customers are assigned to territories, each territory is assigned to a single sales rep, a sales rep can have more than one territory, and a customer is represented by the sales rep

who covers the customer's assigned territory. To implement these changes, you might choose to restructure the database as follows:

```
REP(REP_NUM, LAST_NAME, FIRST_NAME, STREET, CITY,
     STATE, ZIP, COMMISSION, RATE)
TERRITORY(TERRITORY_NUM, DESCRIPTION, REP_NUM)
CUSTOMER(CUSTOMER_NUM, CUSTOMER_NAME, STREET, CITY,
     STATE, ZIP, BALANCE, CREDIT_LIMIT, TERRITORY_NUM)
```

Assuming that the REP_CUST view created in Figure 7-7 is still required, you could change the defining query as follows:


```
CREATE VIEW REP_CUST (RNUM, RLAST, RFIRST,
     CNUM, CNAME) AS
SELECT REP.REP_NUM, REP.LAST_NAME, REP.FIRST_NAME,
     CUSTOMER_NUM, CUSTOMER_NAME
FROM REP, TERRITORY, CUSTOMER
WHERE REP.REP_NUM = TERRITORY.REP_NUM
AND TERRITORY.TERRITORY_NUM = CUSTOMER.TERRITORY_NUM;
```

This view's user still can retrieve the number and name of a sales rep together with the number and name of each customer the sales rep represents. The user is unaware, however, of the new structure in the database.

The second benefit of using views is that different users can see the same data in different ways through their own views. In other words, you can customize the display of data to meet each user's needs.

The final benefit of using views is that a view can contain only those columns required by a given user. This practice has two advantages. First, because the view usually contains fewer columns than the overall database and is conceptually a single table, rather than a collection of tables, a view greatly simplifies the user's perception of the database. Second, views provide a measure of security. Columns that are not included in the view are not accessible to the view's user. For example, omitting the BALANCE column from a view ensures that the view's user cannot access any customer's balance. Likewise, rows that are not included in the view are not accessible. A user of the HOUSEWARES view, for example, cannot obtain any information about parts in the AP or SG classes.

USING A VIEW TO UPDATE DATA

The benefits of using views hold true only when views are used for retrieval purposes. When updating the database, the issues involved in updating data through a view depend on the type of view, as you will see next.

Updating Row-and-Column Subset Views

Consider the row-and-column subset view for the HOUSEWARES view. There are columns in the underlying base table (PART) that are not present in the view. If you attempt to add a row with the data ('BB99','PAN',50,14.95), the DBMS must determine how to enter the data in those columns from the PART table that are not included in the HOUSEWARES view (CLASS and WAREHOUSE). In this case, it is clear what data to enter in the CLASS column. According to the view definition, all rows are item class HW, but it is not clear what

data to enter in the WAREHOUSE column. The only possibility would be NULL. Therefore, if every column not included in a view can accept nulls, you can add new rows using the INSERT command. There is another problem, however. Suppose the user attempts to add a row to the HOUSEWARES view containing the data ('BV06','Waffle Maker',5,29.95). Because part number BV06 already exists in the PART table, the system *must* reject this attempt. Because this part is not in item class HW (and therefore is not in the HOUSEWARES view), this rejection certainly will seem strange to the user, because there is no such part in the user's view.

On the other hand, updates or deletions cause no particular problem in this view. If the description of part number FD21 changes from Stand Mixer to Pan, this change is made in the PART table. If part number DL71 is deleted, this deletion occurs in the PART table. One surprising change could take place, however. Suppose that the CLASS column is included in the HOUSEWARES view and a user changes the class of part number AT94 from HW to AP. Because this item would no longer satisfy the criterion for being included in the HOUSEWARES view, part number AT94 would disappear from the user's view!

Although there are problems to overcome when updating row-and-column subset views, it seems possible to update the database through the HOUSEWARES view. This does not mean that *any* row-and-column subset view is updatable, however. Consider the REP_CRED view shown in Figure 7-11. (The DISTINCT operator is used to omit duplicate rows from the view.)

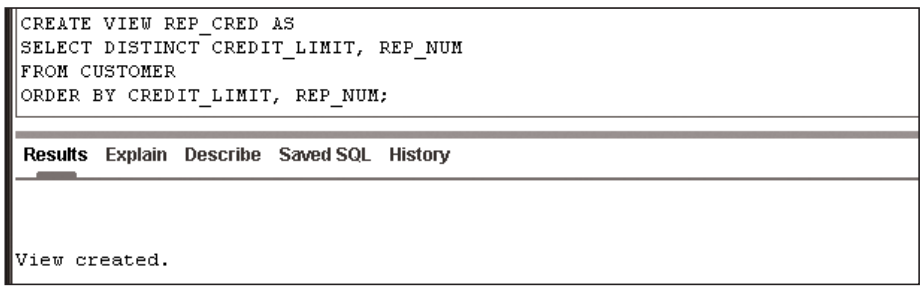

FIGURE 7-11 Creating the REP_CRED view

Figure 7-12 shows the data in the REP_CRED view.

```
SELECT *
FROM REP_CRED;
```

Results Explain Describe Saved SQL History

CREDIT_LIMIT	REP_NUM
5000	35
7500	20
7500	35
7500	65
10000	35
10000	65
15000	20

7 rows returned in 0.01 seconds CSV Export

FIGURE 7-12 Data in the REP_CRED view

How would you add the row 15000,'35' to this view? In the underlying base table (CUSTOMER), at least one customer must be added whose credit limit is $15,000 and whose sales rep number is 35, but which customer is it? You cannot leave the other columns null in this case, because one of them is CUSTOMER_NUM, which is the base table's primary key. What would it mean to change the row 5000,'35' to 15000,'35'? Would it mean changing the credit limit to $15,000 for each customer represented by sales rep number 35 that currently has a credit limit of $5,000? Would it mean changing the credit limit of one of these customers and deleting the rest? What would it mean to delete the row 5000,'35'? Would it mean deleting all customers with credit limits of $5,000 and represented by sales rep number 35, or would it mean assigning these customers a different sales rep or a different credit limit?

Why does the REP_CRED view involve a number of serious problems that are not present in the HOUSEWARES view? The basic reason is that the HOUSEWARES view includes, as one of its columns, the primary key of the underlying base table, but the REP_CRED view does not. A row-and-column subset view that contains the primary key of the underlying base table is updatable (subject, of course, to some of the concerns already discussed).

Updating Views Involving Joins

In general, views that involve joins of base tables can cause problems when updating data. Consider the relatively simple REP_CUST view, for example, described earlier (see Figures 7-7 and 7-8). The fact that some columns in the underlying base tables are not included in this view presents some of the same problems discussed earlier. Assuming that you can overcome these problems by using nulls, there are more serious problems when attempting to update the database through this view. On the surface, changing the row ('35','Hull','Richard', '282','Brookings Direct') to ('35','Baldwin','Sara','282','Brookings Direct'), might not appear to pose any problems other than some inconsistency in the data. (In the new version of the row,

the name of sales rep 35 is Sara Baldwin; whereas in the fourth row in the table, the name of sales rep 35, *the same sales rep*, is Richard Hull.)

The problem is actually more serious than that—making this change is not possible. The name of a sales rep is stored only once in the underlying REP table. Changing the name of sales rep 35 from Richard Hull to Sara Baldwin in this one row of the view causes the change to be made to the single row for sales rep 35 in the REP table. Because the view simply displays data from the base tables, for each row on which the sales rep number is 35, the sales rep name is now Sara Baldwin. In other words, it appears that the same change has been made in the other rows. In this case, this change ensures consistency in the data. In general, however, the unexpected changes caused by an update are not desirable.

Before concluding the topic of views that involve joins, you should note that all joins do not create the preceding problem. When two base tables have the same primary key and the primary key is used as the join column, updating the database using the view is not a problem. For example, suppose the actual database contains two tables (REP_DEMO and REP_FIN) instead of one table (REP). Figure 7-13 shows the data in the REP_DEMO table.

```
SELECT *
FROM REP_DEMO;
```

Results Explain Describe Saved SQL History

REP_NUM	LAST_NAME	FIRST_NAME	STREET	CITY	STATE	ZIP
20	Kaiser	Valerie	624 Randall	Grove	FL	33321
35	Hull	Richard	532 Jackson	Sheldon	FL	33553
65	Perez	Juan	1626 Taylor	Fillmore	FL	33336

3 rows returned in 0.00 seconds CSV Export

FIGURE 7-13 Data in the REP_DEMO table

Figure 7-14 shows the data in the REP_FIN table.

```
SELECT *
FROM REP_FIN;
```

Results Explain Describe Saved SQL History

REP_NUM	COMMISSION	RATE
20	20542.5	.05
35	39216	.07
65	23487	.05

3 rows returned in 0.02 seconds CSV Export

FIGURE 7-14 Data in the REP_FIN table

What was once a single table in the original Premiere Products design has been divided into two separate tables. Users who need to see the rep data in a single table can use a view named SALES_REP that joins these two tables using the REP_NUM column. The defining query for the SALES_REP view appears in Figure 7-15.

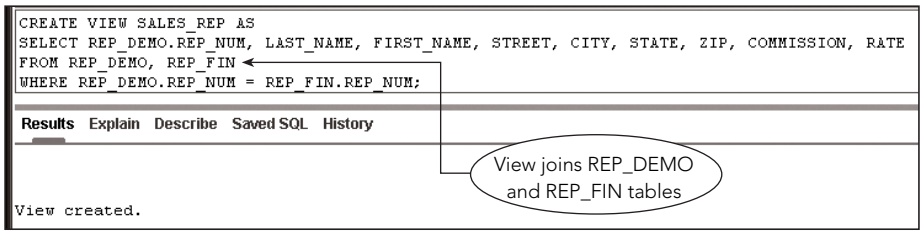

FIGURE 7-15 Creating the SALES_REP view

The data in the SALES_REP view appears in Figure 7-16.

```
SELECT *
FROM SALES_REP;
```

Results Explain Describe Saved SQL History

REP_NUM	LAST_NAME	FIRST_NAME	STREET	CITY	STATE	ZIP	COMMISSION	RATE
20	Kaiser	Valerie	624 Randall	Grove	FL	33321	20542.5	.05
35	Hull	Richard	532 Jackson	Sheldon	FL	33553	39216	.07
65	Perez	Juan	1626 Taylor	Fillmore	FL	33336	23487	.05

3 rows returned in 0.02 seconds CSV Export

FIGURE 7-16 Data in the SALES_REP view

It is easy to update the SALES_REP view. To add a row, use an INSERT command to add a row to each underlying base table. To update data in a row, make the change in the appropriate base table. To delete a row from the view, delete the corresponding rows from both underlying base tables.

Q & A

Question: How would you add the row ('10','Peters','Jean','14 Brink','Holt','FL','46223', 107.50,0.05) to the SALES_REP view?
Answer: Use an INSERT command to add the row ('10','Peters','Jean','14 Brink','Holt','FL','46223') to the REP_DEMO table, and then use another INSERT command to add the row ('10',107.50,0.05) to the REP_FIN table.

Q & A

Question: How would you change the name of sales rep 20 to Valerie Lewis?
Answer: Use an UPDATE command to change the name in the REP_DEMO table.

Q & A

Question: How would you change Valerie's commission rate to 0.06?
Answer: Use an UPDATE command to change the rate in the REP_FIN table.

Q & A

Question: How would you delete sales rep 35 from the REP table?
Answer: Use a DELETE command to delete sales rep 35 from *both* the REP_DEMO *and* REP_FIN tables.

Updates (additions, changes, or deletions) to the SALES_REP view do not cause any problems. The main reason that the SALES_REP view is updatable—and that other views involving joins might not be updatable—is that this view is derived from joining two base tables *on the primary key of each table*. In contrast, the REP_CUST view is created by joining two tables by matching the primary key of one table with a column that is *not* the primary key in the other table. When neither of the join columns in a view is a primary key column, users will encounter even more severe problems when attempting to make updates.

Updating Views Involving Statistics

A view that involves statistics calculated from one or more base tables is the most troublesome view when attempting to update data. Consider the CRED_CUST view, for example (see Figure 7-10). How would you add the row 9000,3 to indicate that there are three customers that have credit limits of $9,000 each? Likewise, changing the row 5000,2 to 5000,5 means you are adding three new customers with credit limits of $5,000 each, for a total of five customers. Clearly these are impossible tasks; you cannot add rows to a view that includes calculations.

DROPPING A VIEW

When a view is no longer needed, you can remove it using the **DROP VIEW** command.

EXAMPLE 5

The HSEWRES view is no longer necessary, so delete it.

The command to delete a view is DROP VIEW as shown in Figure 7-17.

```
DROP VIEW HSEWRES;
```

Results Explain Describe Saved SQL History

```
View dropped.
```

FIGURE 7-17 Dropping a view

ACCESS USER NOTE

Access does not support the DROP VIEW command. To drop a view, delete the query object you saved when you created the view.

SECURITY

Security is the prevention of unauthorized access to a database. Within an organization, the database administrator determines the types of access various users need to the database. Some users might need to retrieve and update anything in the database. Other users might need to retrieve any data from the database but not make any changes to it. Still other users might need to access only a portion of the database. For example, Bill might need to retrieve and update customer data, but does not need to access data about sales reps, orders, order lines, or parts. Valerie might need to retrieve part data and nothing else. Sam might need to retrieve and update data on parts in the HW class, but does not need to retrieve data in any other classes.

After the database administrator has determined the access different users of the database need, the DBMS enforces these access rules by whatever security mechanism the DBMS supports. You can use SQL to enforce two security mechanisms. You already have seen that views furnish a certain amount of security; when users are accessing the database through a view, they cannot access any data that is not included in the view. The main mechanism for providing access to a database, however, is the **GRANT** command.

The basic idea of the GRANT command is that the database administrator can grant different types of privileges to users and then revoke them later, if necessary. These privileges include the right to select, insert, update, and delete table data. You can grant and revoke user privileges using the GRANT and REVOKE commands. The following examples illustrate various uses of the GRANT command when the named users already exist in the database.

NOTE

Do not execute the commands in this section unless your instructor asks you to do so.

EXAMPLE 6

User Johnson must be able to retrieve data from the REP table.

The following GRANT command permits a user named Johnson to execute SELECT commands for the REP table:

```
GRANT SELECT ON REP TO JOHNSON;
```

EXAMPLE 7

Users Smith and Brown must be able to add new parts to the PART table.

The following GRANT command permits two users named Smith and Brown to execute INSERT commands for the PART table. Notice that a comma separates the user names:

```
GRANT INSERT ON PART TO SMITH, BROWN;
```

EXAMPLE 8

User Anderson must be able to change the name and street address of customers.

The following GRANT command permits a user named Anderson to execute UPDATE commands involving the CUSTOMER_NAME and STREET columns in the CUSTOMER table. Notice that the SQL command includes the column names in parentheses before the ON clause:

```
GRANT UPDATE (CUSTOMER_NAME, STREET) ON CUSTOMER TO
ANDERSON;
```

EXAMPLE 9

User Thompson must be able to delete order lines.

The following GRANT command permits a user named Thompson to execute DELETE commands for the ORDER_LINE table:

```
GRANT DELETE ON ORDER_LINE TO THOMPSON;
```

EXAMPLE 10

Every user must be able to retrieve part numbers, part descriptions, and item classes.

The GRANT command to indicate that all users can retrieve data using a SELECT command includes the word PUBLIC, as follows:

```
GRANT SELECT (PART_NUM, DESCRIPTION, CLASS) ON PART TO
PUBLIC;
```


EXAMPLE 11

User Roberts must be able to create an index on the REP table.

You will learn about indexes and their uses in the next section. The following GRANT command permits a user named Roberts to create an index on the REP table:

```
GRANT INDEX ON REP TO ROBERTS;
```

EXAMPLE 12

User Thomas must be able to change the structure of the CUSTOMER table.

The following GRANT command permits a user named Thomas to execute ALTER commands for the CUSTOMER table so he can change the table's structure:

```
GRANT ALTER ON CUSTOMER TO THOMAS;
```

EXAMPLE 13

User Wilson must have all privileges for the REP table.

The GRANT command to indicate that a user has all privileges includes the ALL privilege, as follows:

```
GRANT ALL ON REP TO WILSON;
```

The privileges that a database administrator can grant are SELECT to retrieve data, UPDATE to change data, DELETE to delete data, INSERT to add new data, INDEX to create an index, and ALTER to change the table structure. The database administrator usually assigns privileges. Normally, when the database administrator grants a particular privilege to a user, the user cannot pass that privilege along to other users. When the user needs to be able to pass the privilege to other users, the GRANT command must include the

WITH GRANT OPTION clause. This clause grants the indicated privilege to the user and also permits the user to grant the same privileges (or a subset of them) to other users.

The database administrator uses the **REVOKE** command to revoke privileges from users. The format of the REVOKE command is essentially the same as that of the GRANT command, but with two differences: the word GRANT is replaced by the word REVOKE, and the word TO is replaced by the word FROM. In addition, the clause WITH GRANT OPTION obviously is not meaningful as part of a REVOKE command. Incidentally, the revoke cascades, so if Johnson is granted privileges WITH GRANT OPTION and then Johnson grants these same privileges to Smith, revoking the privileges from Johnson revokes Smith's privileges at the same time. Example 14 illustrates the use of the REVOKE command.

EXAMPLE 14

User Johnson is no longer allowed to retrieve data from the REP table.

The following REVOKE command revokes the SELECT privilege for the REP table from the user named Johnson:

```
REVOKE SELECT ON REP FROM JOHNSON;
```

The database administrator can also apply the GRANT and REVOKE commands to views to restrict access to only certain rows within tables.

INDEXES

When you query a database, you are usually searching for a row (or collection of rows) that satisfies some condition. Examining every row in a table to find the ones you need often takes too much time to be practical, especially in tables with thousands of rows. Fortunately, you can create and use an index to speed up the searching process significantly. An index in SQL is similar to an index in a book. When you want to find a discussion of a given topic in a book, you could scan the entire book from start to finish, looking for references to the topic you need. More than likely, however, you would not have to resort to this technique. If the book has a good index, you could use it to identify the pages on which your topic is discussed.

In a DBMS, the main mechanism for increasing the efficiency with which data is retrieved from the database is the **index**. Conceptually, these indexes are very much like the index in a book. Consider Figure 7-18, for example, which shows the CUSTOMER table for Premiere Products together with one extra column named ROW_NUMBER. This extra column gives the location of the row in the table (customer 148 is the first row in the table and is on row 1, customer 282 is on row 2, and so on). The DBMS—not the user—automatically assigns and uses these row numbers, and that is why you do not see them.

CUSTOMER

ROW_ NUMBER	CUSTOMER_ NUM	CUSTOMER_ NAME	STREET	CITY	STATE	ZIP	BALANCE	CREDIT_ LIMIT	REP_ NUM
1	148	Al's Appliance and Sport	2837 Greenway	Fillmore	FL	33336	$6,550.00	$7,500.00	20
2	282	Brookings Direct	3827 Devon	Grove	FL	33321	$431.50	$10,000.00	35
3	356	Ferguson's	382 Wildwood	Northfield	FL	33146	$5,785.00	$7,500.00	65
4	408	The Everything Shop	1828 Raven	Crystal	FL	33503	$5,285.25	$5,000.00	35
5	462	Bargains Galore	3829 Central	Grove	FL	33321	$3,412.00	$10,000.00	65
6	524	Kline's	838 Ridgeland	Fillmore	FL	33336	$12,762.00	$15,000.00	20
7	608	Johnson's Department Store	372 Oxford	Sheldon	FL	33553	$2,106.00	$10,000.00	65
8	687	Lee's Sport and Appliance	282 Evergreen	Altonville	FL	32543	$2,851.00	$5,000.00	35
9	725	Deerfield's Four Seasons	282 Columbia	Sheldon	FL	33553	$248.00	$7,500.00	35
10	842	All Season	28 Lakeview	Grove	FL	33321	$8,221.00	$7,500.00	20

FIGURE 7-18 CUSTOMER table with row numbers

To access a customer's row using its customer number, you might create and use an index, as shown in Figure 7-19. The index has two columns: the first column contains a customer number, and the second column contains the number of the row on which the customer number is found. To find a customer, look up the customer's number in the first column in the index. The value in the second column indicates which row to retrieve from the CUSTOMER table, then the row for the desired customer is retrieved.

CUSTOMER_NUM Index

CUSTOMER_NUM	ROW_NUMBER
148	1
282	2
356	3
408	4
462	5
524	6
608	7
687	8
725	9
842	10

FIGURE 7-19 Index for the CUSTOMER table on the CUSTOMER_NUM column

Because customer numbers are unique, there is only a single row number in this index. This is not always the case, however. Suppose you need to access all customers with a specific credit limit or all customers represented by a specific sales rep. You might choose to create and use an index on the CREDIT_LIMIT column and an index on the REP_NUM column, as shown in Figure 7-20. In the CREDIT_LIMIT index, the first column contains a credit limit and the second column contains the numbers of *all* rows on which that credit limit appears. The REP_NUM index is similar, except that the first column contains a sales rep number.

CREDIT_LIMIT Index

CREDIT_LIMIT	ROW_NUMBER
$5,000.00	4, 8
$7,500.00	1, 3, 9, 10
$10,000.00	2, 5, 7
$15,000.00	6

REP_NUM Index

REP_NUM	ROW_NUMBER
20	1, 6, 10
35	2, 4, 8, 9
65	3, 5, 7

FIGURE 7-20 Indexes for the CUSTOMER table on the CREDIT_LIMIT and REP_NUM columns

Q & A

Question: How would you use the index shown in Figure 7-20 to find every customer with a $10,000 credit limit?
Answer: Look up $10,000 in the CREDIT_LIMIT index to find a collection of row numbers (2, 5, and 7). Use these row numbers to find the corresponding rows in the CUSTOMER table (Brookings Direct, Bargains Galore, and Johnson's Department Store).

Q & A

Question: How would you use the index shown in Figure 7-20 to find every customer represented by sales rep 35?
Answer: Look up 35 in the REP_NUM index to find a collection of row numbers (2, 4, 8, and 9). Use these row numbers to find the corresponding rows in the CUSTOMER table (Brookings Direct, The Everything Shop, Lee's Sport and Appliance, and Deerfield's Four Seasons).

The actual structure of an index is more complicated than what is shown in the figures. Fortunately, you do not have to worry about the details of manipulating and using indexes because the DBMS manages them for you—your only job is to determine the columns on which to build the indexes. Typically, you can create and maintain an index for any column or combination of columns in any table. After creating an index, the DBMS uses it to speed up data retrieval.

As you would expect, the use of any index has advantages and disadvantages. An important advantage was already mentioned: an index makes certain types of retrieval more efficient.

There are two disadvantages when using indexes. First, an index occupies disk space. Using this space for an index, however, is technically unnecessary because any retrieval that you can make using an index also can be made without the index; the index just speeds up the retrieval. The second disadvantage is that the DBMS must update the index whenever corresponding data in the database is updated. Without the index, the DBMS would not need to make these updates. The main question that you must ask when considering whether to create a given index is this: do the benefits derived during retrieval outweigh the additional storage required and the extra processing involved in update operations? In a very large database, you might find that indexes are essential to decrease the time required to retrieve records. In a small database, however, an index might not provide any significant benefits.

You can add and drop indexes as necessary. You can create an index after the database is built; it does not need to be created at the same time as the database. Likewise, when an existing index is no longer necessary, you can drop it.

Creating an Index

Suppose some users at Premiere Products need to display customer records ordered by balance. Other users need to access a customer's name using the customer's number. In addition, some users need to produce a report in which customer records are listed by credit limit in descending order. Within the group of customers having the same credit limit, the customer records must be ordered by name.

Each of the previous requirements is carried out more efficiently when you create the appropriate index. The command used to create an index is **CREATE INDEX**, as illustrated in Example 15.

EXAMPLE 15

Create an index named BALIND on the BALANCE column in the CUSTOMER table. Create an index named REPNAME on the combination of the LAST_NAME and FIRST_NAME columns in the REP table. Create an index named CREDNAME on the combination of the CREDIT_LIMIT and CUSTOMER_NAME columns in the CUSTOMER table, with the credit limits listed in descending order.

The CREATE INDEX command to create the index named BALIND appears in Figure 7-21. The command lists the name of the index and the table name on which the index is to be created. The column on which to create the index—BALANCE—is listed in parentheses.

```
CREATE INDEX BALIND ON CUSTOMER(BALANCE);

Results  Explain  Describe  Saved SQL  History

Index created.
```

FIGURE 7-21 Creating the BALIND index on the BALANCE column

The CREATE INDEX command to create the index named REPNAME on the combination of the LAST_NAME and FIRST_NAME columns in the REP table appears in Figure 7-22.

```
CREATE INDEX REPNAME ON REP(LAST_NAME, FIRST_NAME);

Results  Explain  Describe  Saved SQL  History

Index created.
```

FIGURE 7-22 Creating the REPNAME index on the LAST_NAME and FIRST_NAME columns

The CREATE INDEX command to create the index named CREDNAME on the combination of the CREDIT_LIMIT and CUSTOMER_NAME columns in the CUSTOMER table appears in Figure 7-23. When you need to index a column in descending order, the column name is followed by the DESC operator.

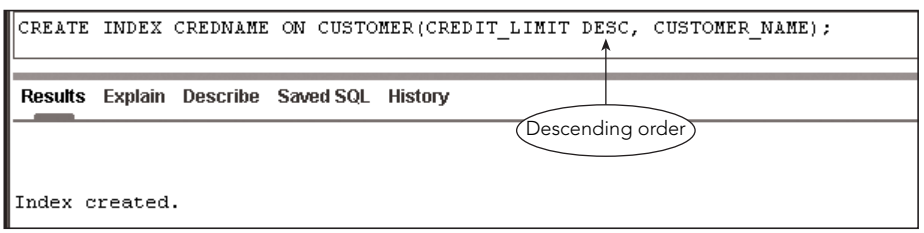

FIGURE 7-23 Creating the CREDNAME index on the CREDIT_LIMIT and CUSTOMER_NAME columns

When customers are listed using the CREDNAME index, the records appear in order by descending credit limit. Within any credit limit, the customers are listed alphabetically by name.

ACCESS USER NOTE

Access supports the creation of indexes in both SQL view and Design view. You can write a CREATE INDEX command in SQL view. You can also open the table containing the column(s) on which you want to create an index in Design view, and click the Indexes button or use the Indexed property to create the index.

Dropping an Index

The command used to drop (delete) an index is **DROP INDEX**, which consists of the words DROP INDEX followed by the name of the index to drop. To delete the CREDNAME index on the CUSTOMER table, for example, the command is:


```
DROP INDEX CREDNAME;
```

The DROP INDEX command permanently deletes the index. CREDNAME was the index the DBMS used when listing customer records in descending order by credit limit order and then by customer name within credit limit. The DBMS still can list customers in this order; however, it cannot do so as efficiently without the index.

SQL SERVER USER NOTE

The SQL Server command to drop an index requires that you qualify the index name. To delete the CREDNAME index on the CUSTOMER table, for example, the command is:

```
DROP INDEX CUSTOMER.CREDNAME;
```

Creating Unique Indexes

When you specify a table's primary key, the DBMS automatically ensures that the values entered in the primary key column(s) are unique. For example, the DBMS rejects an attempt to add a second customer whose number is 148 in the CUSTOMER table because customer 148 already exists. Thus, you do not need to take any special action to make sure that values in the primary key column are unique; the DBMS does it for you.

Occasionally, a nonprimary key column might store unique values. For example, in the REP table, the primary key is REP_NUM. If the REP table also contains a column for Social Security numbers, the values in this column also must be unique because no two people can have the same Social Security number. Because the Social Security number column is not the table's primary key, however, you need to take special action in order for the DBMS to ensure that there are no duplicate values in this column.

To ensure the uniqueness of values in a nonprimary key column, you can create a **unique index** by using the **CREATE UNIQUE INDEX** command. To create a unique index named SSN on the SOC_SEC_NUM column in the REP table, for example, the command is:

```
CREATE UNIQUE INDEX SSN ON REP(SOC_SEC_NUM);
```

This unique index has all the properties of indexes already discussed, along with one additional property: the DBMS rejects any update that causes a duplicate value in the SOC_SEC_NUM column. In this case, the DBMS rejects the addition of a rep whose Social Security number is the same as that of another rep already in the database.

SYSTEM CATALOG

Information about the tables in the database is kept in the **system catalog (catalog)** or the **data dictionary**. This section describes the types of items kept in the catalog and the way in which you can query it to access information about the database structure.

The DBMS automatically maintains the system catalog, which contains several tables. The catalog tables you'll consider in this basic introduction are **SYSTABLES** (information about the tables known to SQL), **SYSCOLUMNS** (information about the columns within these tables), and **SYSVIEWS** (information about the views that have been created). Individual SQL implementations might use different names for these tables. In Oracle, the equivalent tables are named **DBA_TABLES**, **DBA_TAB_COLUMNS**, and **DBA_VIEWS**.

The system catalog is a relational database of its own. Consequently, you can use the same types of queries to retrieve information that you can use to retrieve data in a relational database. You can obtain information about the tables in a relational database, the columns they contain, and the views built on them from the system catalog. The following examples illustrate this process.

NOTE

Most Oracle users need privileges to view system catalog data, so you might not be able to execute these commands. If you are executing the commands shown in the figures, substitute your user name for PRATT to list objects that you own. Your results will differ from those shown in the figures.

ACCESS USER NOTE

In Access, use the Documenter to obtain the information discussed in this section, rather than querying the system catalog.

SQL SERVER USER NOTE

In SQL Server, use stored procedures to obtain the information discussed in this section. To display information about the tables and views in a database, use the sp_tables procedure. For example, the following command displays all the tables and views in the current database:

```
EXEC sp_tables
```

The sp_columns stored procedure displays information about the columns in a particular table. The following command displays the column information for the REP table:

```
EXEC sp_columns REP
```

EXAMPLE 16

List the name of each table for which the owner (creator of the table) is PRATT.

The command to list the table names owned by PRATT is shown in Figure 7-24. The WHERE clause restricts the tables to only those owned by PRATT.

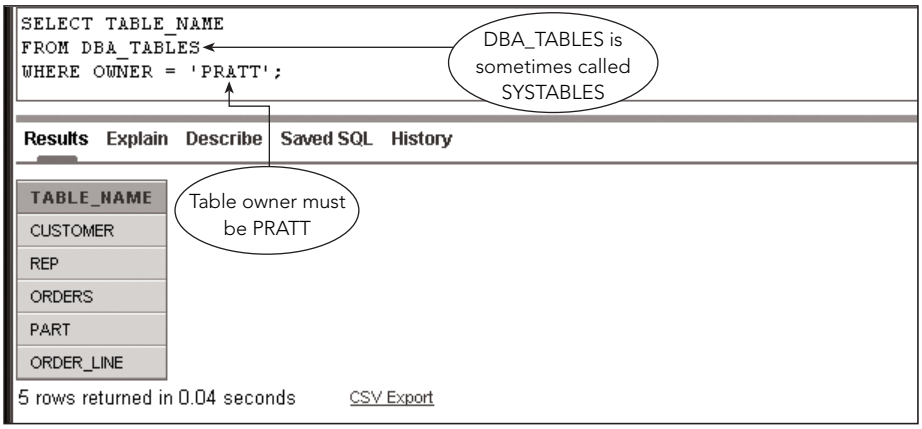

FIGURE 7-24 Tables owned by PRATT

EXAMPLE 17

List the name of each view owned by PRATT.

This command is similar to the command in Example 16. Rather than TABLE_NAME, the column to be selected is named VIEW_NAME. The command appears in Figure 7-25.

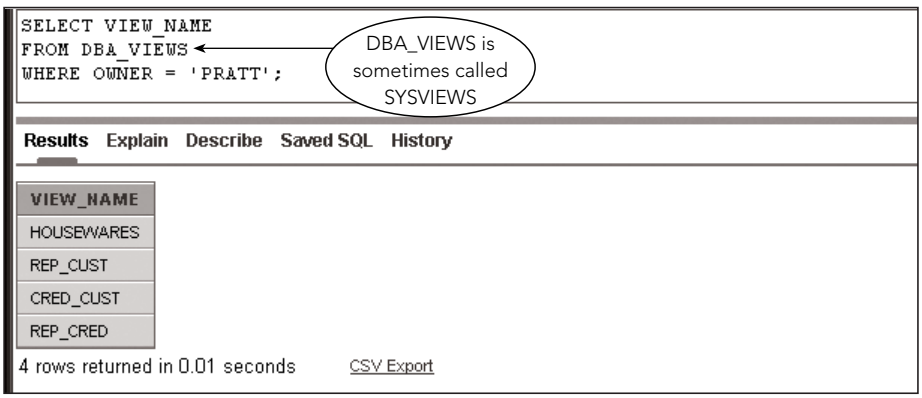

FIGURE 7-25 Views owned by PRATT

EXAMPLE 18

For the CUSTOMER table owned by PRATT, list each column and its data type.

The command for this example appears in Figure 7-26. The columns to select are COLUMN_NAME and DATA_TYPE.

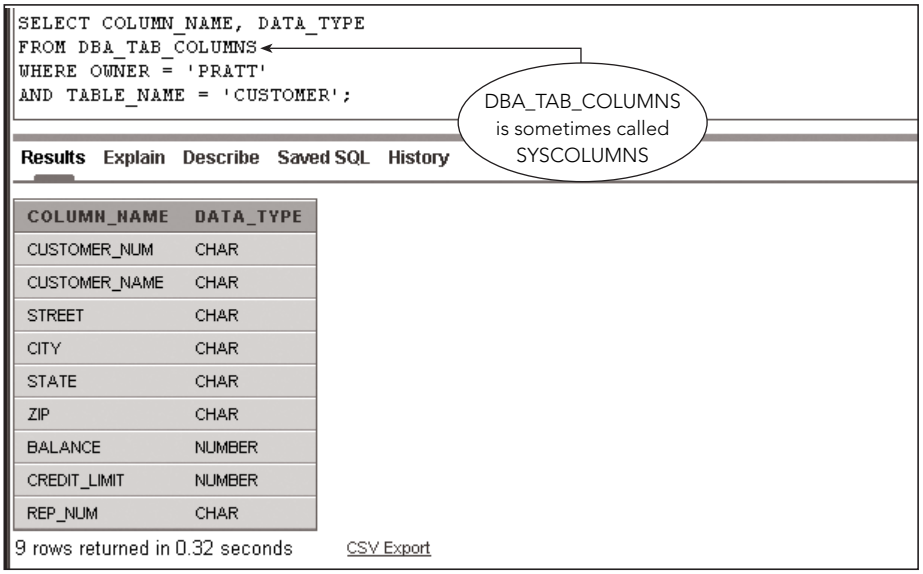

FIGURE 7-26 Columns in the CUSTOMER table

EXAMPLE 19

List each table owned by PRATT that contains a column named CUSTOMER_NUM.

As shown in Figure 7-27, the COLUMN_NAME column is used in the WHERE clause to restrict the rows to those in which the column name is CUSTOMER_NUM. (Extra tables generated by Oracle might appear in your list, as shown in Figure 7-27.)

```
SELECT TABLE_NAME
FROM DBA_TAB_COLUMNS
WHERE OWNER = 'PRATT'
AND COLUMN_NAME = 'CUSTOMER_NUM';
```

Results Explain Describe Saved SQL History

TABLE_NAME
BIN$4wiC+9lRRgy09aROn6UQSQ==$0
BIN$CSd693ZyQKCEjENgJ6NKug==$0
CUSTOMER
ORDERS

4 rows returned in 1.71 seconds CSV Export

FIGURE 7-27 Tables owned by PRATT with CUSTOMER_NUM columns

When users create, alter, or drop tables or create or drop indexes, the DBMS updates the system catalog automatically to reflect these changes. Users should not execute SQL queries to update the catalog directly because this might produce inconsistent information. For example, when a user deletes the CUSTOMER_NUM column in the DBA_TAB_COLUMNS table, the DBMS would no longer have any knowledge of this column, which is the CUSTOMER table's primary key, yet all the rows in the CUSTOMER table would still contain a customer number. The DBMS might now treat those customer numbers as names, because as far as the DBMS is concerned, the column named CUSTOMER_NAME is the first column in the CUSTOMER table.

INTEGRITY CONSTRAINTS IN SQL

An **integrity constraint** is a rule for the data in the database. Examples of integrity constraints in the Premiere Products database are as follows:

- A sales rep's number must be unique.
- The sales rep number for a customer must match the number of a sales rep currently in the database. For example, because there is no sales rep number 11, a customer cannot be assigned to sales rep 11.
- Item classes for parts must be AP, HW, or SG because these are the only valid item classes.

If a user enters data in the database that violates any of these integrity constraints, the database develops serious problems. For example, two sales reps with the same number, a customer with a nonexistent sales rep, or a part in a nonexistent item class would compromise the integrity of data in the database. To manage these types of problems, SQL provides **integrity support**, the process of specifying and enforcing integrity constraints for a database. SQL has clauses to support three types of integrity constraints that you can specify within a CREATE TABLE or an ALTER TABLE command. The only difference between these two commands is that an ALTER TABLE command is followed by the word

ADD to indicate that you are adding the constraint to the list of existing constraints. To change an integrity constraint after it has been created, just enter the new constraint, which immediately takes the place of the original.

The types of constraints supported in SQL are primary keys, foreign keys, and legal values. In most cases, you specify a table's primary key when you create the table. To add a primary key after creating a table, you can use the **ADD PRIMARY KEY** clause of the ALTER TABLE command. For example, to indicate that REP_NUM is the primary key for the REP table, the ALTER TABLE command is:


```
ALTER TABLE REP
ADD PRIMARY KEY (REP_NUM);
```

The ADD PRIMARY KEY clause is ADD PRIMARY KEY followed by the column name that makes up the primary key in parentheses. When the primary key contains more than one column, use commas to separate the column names.

ACCESS USER NOTE

To specify a table's primary key in Access, open the table in Design view, select the column(s) that make up the primary key, and then click the Primary Key button on the Ribbon or the toolbar.

A **foreign key** is a column in one table whose values match the primary key in another table. (One example is the CUSTOMER_NUM column in the ORDERS table. Values in this column are required to match those of the primary key in the CUSTOMER table.)

EXAMPLE 20

Specify the CUSTOMER_NUM column in the ORDERS table as a foreign key that must match the CUSTOMER table.

When a table contains a foreign key, you identify it using the **ADD FOREIGN KEY** clause of the ALTER TABLE command. In this clause, you specify the column that is a foreign key *and* the table it matches. The general form for assigning a foreign key is ADD FOREIGN KEY, the column name(s) of the foreign key, the REFERENCES clause, and then the table name that the foreign key must match, as shown in Figure 7-28.

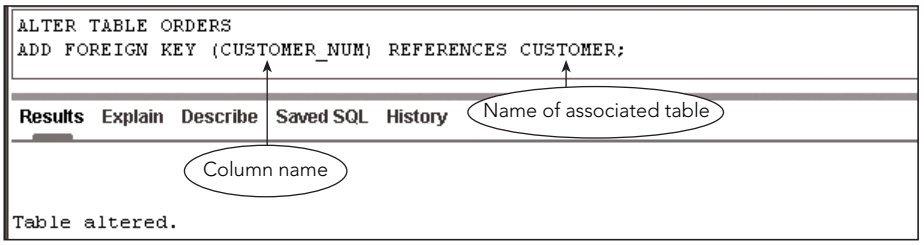

FIGURE 7-28 Adding a foreign key to an existing table

ACCESS USER NOTE

To specify a foreign key in Access, open the Relationships window, relate the corresponding tables on the matching column, and then select the option to enforce referential integrity.

After creating a foreign key, the DBMS rejects any update that violates the foreign key constraint. For example, the DBMS rejects the INSERT command shown in Figure 7-29 because it attempts to add an order for which the customer number (850) does not match any customer in the CUSTOMER table.

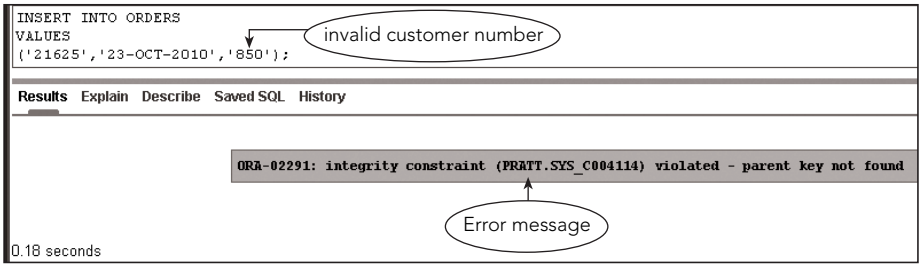

FIGURE 7-29 Violating a foreign key constraint when adding a row

The DBMS also rejects the DELETE command in Figure 7-30 because it attempts to delete customer number 148; rows in the ORDERS table for which the customer number is 148 would no longer match any row in the CUSTOMER table.

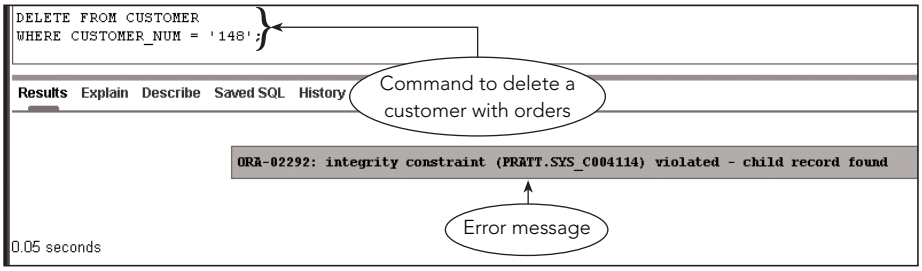

FIGURE 7-30 Violating a foreign key constraint when deleting a row

Note that the error messages shown in Figures 7-29 and 7-30 include the words "parent" and "child." When you specify a foreign key, the table containing the foreign key is the **child**, and the table referenced by the foreign key is the **parent**. For example, the CUSTOMER_NUM column in the ORDERS table is a foreign key that references the CUSTOMER table. For this foreign key, the CUSTOMER table is the parent, and the ORDERS table is the child. The error message shown in Figure 7-29 indicates that there is no parent for the order (there is no customer number 850). The error message shown in Figure 7-30 indicates that there are child records (rows) for customer 148 (customer 148 has orders). The DBMS rejects both updates because they violate referential integrity.

EXAMPLE 21

Specify the valid item classes for the PART table as AP, HW, and SG.

You use the **CHECK** clause of the ALTER TABLE command to ensure that only legal values satisfying a particular condition are allowed in a given column. The general form of the CHECK clause is the word CHECK followed by a condition. If a user enters data that violates the condition, the DBMS rejects the update automatically. For example, to ensure that the only legal values for item class are AP, HW, or SG, use one of the following versions of the CHECK clause:


```
CHECK (CLASS IN ('AP', 'HW', 'SG') )
```

or

```
CHECK (CLASS = 'AP' OR CLASS = 'HW' OR CLASS = 'SG')
```

The ALTER TABLE command shown in Figure 7-31 uses the first version of the CHECK clause.

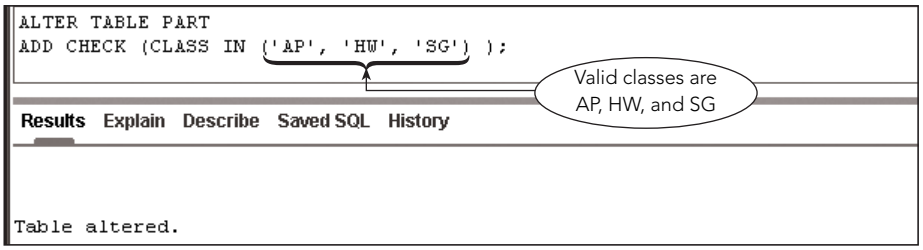

FIGURE 7-31 Adding an integrity constraint to an existing table

Now the DBMS will reject the update shown in Figure 7-32 because the command attempts to change the item class to XX, which is an illegal value.

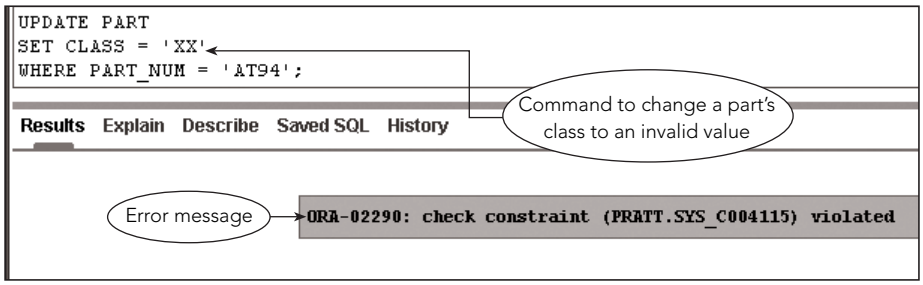

FIGURE 7-32 Update that violates an integrity constraint

ACCESS USER NOTE

Access does not support the CHECK clause. To specify a validation rule in Access, open the table in Design view, and then enter an expression in the column's Validation Rule property to limit the values that users can enter into the column.

Chapter Summary

- A view contains data that is derived from existing base tables when users attempt to access the view.
- To create a view, use the CREATE VIEW command, which includes a defining query that describes the portion of the database included in the view. When a user retrieves data from the view, the DBMS merges the query entered by the user with the defining query and produces the query that the DBMS actually executes.
- Views provide data independence, allow database access control, and simplify the database structure for users.
- You cannot update views that involve statistics and views with joins of nonprimary key columns. Updates for these types of views must be made in the base table.
- Use the DROP VIEW command to delete a view.
- Use the GRANT command to give users access privileges to data in the database.
- Use the REVOKE command to terminate previously granted privileges.
- You can create and use an index to make data retrieval more efficient. Use the CREATE INDEX command to create an index. Use the CREATE UNIQUE INDEX command to enforce a rule so only unique values are allowed in a nonprimary key column.
- Use the DROP INDEX command to delete an index.
- The DBMS, not the user, chooses which index to use to accomplish a given task.
- The DBMS maintains information about the tables, columns, indexes, and other system elements in the system catalog (catalog) or data dictionary. Information about tables is kept in the SYSTABLES table, information about columns is kept in the SYSCOLUMNS table, and information about views is kept in the SYSVIEWS table. In Oracle, these same tables are named DBA_TABLES, DBA_TAB_COLUMNS, and DBA_VIEWS.
- Use the SELECT command to obtain information from the system catalog. The DBMS updates the system catalog automatically whenever changes are made to the database. In Access, use the Documenter to obtain information about the database objects. SQL Server uses stored procedures to obtain information from the system catalog.
- Integrity constraints are rules that the data in the database must follow to ensure that only legal values are accepted in specified columns and that primary and foreign key values match between tables. To specify a general integrity constraint, use the CHECK clause. You usually specify primary key constraints when you create a table, but you can specify them later using the ADD PRIMARY KEY clause. To specify a foreign key, use the ADD FOREIGN KEY clause.

Key Terms

ADD FOREIGN KEY
ADD PRIMARY KEY
base table
catalog
CHECK
child
CREATE INDEX
CREATE UNIQUE INDEX
CREATE VIEW
data dictionary

database administration
database administrator
DBA_TAB_COLUMNS
DBA_TABLES
DBA_VIEWS
defining query
DROP INDEX
DROP VIEW
foreign key
GRANT
index
integrity constraint
integrity support
parent
REFERENCES
REVOKE
row-and-column subset view
security
SYSCOLUMNS
SYSTABLES
system catalog
SYSVIEWS
unique index
view
WITH GRANT OPTION

Review Questions

1. What is a view?
2. Which command creates a view?
3. What is a defining query?
4. What happens when a user retrieves data from a view?
5. What are three advantages of using views?
6. Which types of views cannot be updated?
7. Which command deletes a view?
8. Which command gives users access privileges to various portions of the database?
9. Which command terminates previously granted privileges?
10. What is the purpose of an index?
11. How do you create an index? How do you create a unique index? What is the difference between an index and a unique index?
12. Which command deletes an index?
13. Does the DBMS or the user make the choice of which index to use to accomplish a given task?
14. Describe the information the DBMS maintains in the system catalog. What are the generic names for three tables in the catalog and their corresponding names in Oracle?
15. Use your favorite Web browser and search engine to find information about a data dictionary. Write a one-page paper that describes other types of information that can be stored in a data dictionary. Be sure to cite the URLs that you use.
16. Which command do you use to obtain information from the system catalog in Oracle?
17. How is the system catalog updated?
18. What are integrity constraints?
19. How do you specify a general integrity constraint?

20. When would you usually specify primary key constraints? Can you specify them after creating a table? How?
21. How do you specify a foreign key in Oracle?
22. Use your favorite Web browser and search engine to find information about referential integrity. Write two or three paragraphs that describe what referential integrity is and include an example of how referential integrity is used in the Premiere Products database. Be sure to cite the URLs that you use.

Exercises

Premiere Products

Use SQL to make the following changes to the Premiere Products database (see Figure 1-2 in Chapter 1). After each change, execute an appropriate query to show that the change was made correctly. If directed to do so by your instructor, use the information provided with the Chapter 3 Exercises to print your output. For any exercises that use commands not supported by your version of SQL, write the command to accomplish the task.

1. Create a view named MAJOR_CUSTOMER. It consists of the customer number, name, balance, credit limit, and rep number for every customer whose credit limit is $10,000 or less.
   a. Write and execute the CREATE VIEW command to create the MAJOR_CUSTOMER view.
   b. Write and execute the command to retrieve the customer number and name of each customer in the MAJOR_CUSTOMER view with a balance that exceeds the credit limit.
   c. Write and execute the query that the DBMS actually executes.
   d. Does updating the database through this view create any problems? If so, what are they? If not, why not?
2. Create a view named PART_ORDER. It consists of the part number, description, price, order number, order date, number ordered, and quoted price for all order lines currently on file.
   a. Write and execute the CREATE VIEW command to create the PART_ORDER view.
   b. Write and execute the command to retrieve the part number, description, order number, and quoted price for all orders in the PART_ORDER view for parts with quoted prices that exceed $100.
   c. Write and execute the query that the DBMS actually executes.
   d. Does updating the database through this view create any problems? If so, what are they? If not, why not?
3. Create a view named ORDER_TOTAL. It consists of the order number and order total for each order currently on file. (The order total is the sum of the number of units ordered multiplied by the quoted price on each order line for each order.) Sort the rows by order number. Use TOTAL_AMOUNT as the name for the order total.
   a. Write and execute the CREATE VIEW command to create the ORDER_TOTAL view.
   b. Write and execute the command to retrieve the order number and order total for only those orders totaling more than $1,000.

c. Write and execute the query that the DBMS actually executes.

d. Does updating the database through this view create any problems? If so, what are they? If not, why not?

4. Write, but do not execute, the commands to grant the following privileges:
   a. User Ashton must be able to retrieve data from the PART table.
   b. Users Kelly and Morgan must be able to add new orders and order lines.
   c. User James must be able to change the price for all parts.
   d. User Danielson must be able to delete customers.
   e. All users must be able to retrieve each customer's number, name, street, city, state, and zip code.
   f. User Perez must be able to create an index on the ORDERS table.
   g. User Washington must be able to change the structure of the PART table.
   h. User Grinstead must have all privileges on the ORDERS table.
5. Write, but do not execute, the command to revoke all privileges from user Ashton.
6. Perform the following tasks:
   a. Create an index named PART_INDEX1 on the PART_NUM column in the ORDER_LINE table.
   b. Create an index named PART_INDEX2 on the CLASS column in the PART table.
   c. Create an index named PART_INDEX3 on the CLASS and WAREHOUSE columns in the PART table.
   d. Create an index named PART_INDEX4 on the CLASS and WAREHOUSE columns in the PART table. List item classes in descending order.
7. Delete the index named PART_INDEX3.
8. Write the commands to obtain the following information from the system catalog. Do not execute these commands unless your instructor asks you to do so.
   a. List every table that contains a column named CUSTOMER_NUM.
   b. List every column in the PART table and its associated data type.
9. Add the ORDER_NUM column as a foreign key in the ORDER_LINE table.
10. Ensure that the only values entered into the CREDIT_LIMIT column are 5000, 7500, 10000, and 15000.

Henry Books

Use SQL to make the following changes to the Henry Books database (Figures 1-4 through 1-7 in Chapter 1). After each change, execute an appropriate query to show that the change was made correctly. If directed to do so by your instructor, use the information provided with the Chapter 3 Exercises to print your output. For any exercises that use commands not supported by your version of SQL, write the command to accomplish the task.

1. Create a view named PLUME. It consists of the book code, title, type, and price for every book published by the publisher whose code is PL.
   a. Write and execute the CREATE VIEW command to create the PLUME view.
   b. Write and execute the command to retrieve the book code, title, and price for every book with a price of less than $13.
   c. Write and execute the query that the DBMS actually executes.
   d. Does updating the database through this view create any problems? If so, what are they? If not, why not?

2. Create a view named NONPAPERBACK. It consists of the book code, title, publisher name, and price for every book that is not available in paperback.
   a. Write and execute the CREATE VIEW command to create the NONPAPERBACK view.
   b. Write and execute the command to retrieve the book title, publisher name, and price for every book in the NONPAPERBACK view with a price of less than $20.
   c. Write and execute the query that the DBMS actually executes.
   d. Does updating the database through this view create any problems? If so, what are they? If not, why not?
3. Create a view named BOOK_INVENTORY. It consists of the branch number and the total number of books on hand for each branch. Use UNITS as the name for the count of books on hand. Group and order the rows by branch number.
   a. Write and execute the CREATE VIEW command to create the BOOK_INVENTORY view.
   b. Write and execute the command to retrieve the branch number and units for each branch having more than 25 books on hand.
   c. Write and execute the query that the DBMS actually executes.
   d. Does updating the database through this view create any problems? If so, what are they? If not, why not?
4. Write, but do not execute, the commands to grant the following privileges:
   a. User Rodriquez must be able to retrieve data from the BOOK table.
   b. Users Gomez and Liston must be able to add new books and publishers to the database.
   c. Users Andrews and Zimmer must be able to change the price of any book.
   d. All users must be able to retrieve the book title, book code, and book price for every book.
   e. User Golden must be able to add and delete publishers.
   f. User Andrews must be able to create an index for the BOOK table.
   g. Users Andrews and Golden must be able to change the structure of the AUTHOR table.
   h. User Golden must have all privileges on the BRANCH, BOOK, and INVENTORY tables.
5. Write, but do not execute, the command to revoke all privileges from user Andrews.

6. Create the following indexes:
   a. Create an index named BOOK_INDEX1 on the TITLE column in the BOOK table.
   b. Create an index named BOOK_INDEX2 on the TYPE column in the BOOK table.
   c. Create an index named BOOK_INDEX3 on the CITY and PUBLISHER_NAME columns in the PUBLISHER table.
7. Delete the index named BOOK_INDEX3.
8. Write the commands to obtain the following information from the system catalog. Do not execute these commands unless your instructor asks you to do so.
   a. List every column in the PUBLISHER table and its associated data type.
   b. List every table that contains a column named PUBLISHER_CODE.
   c. List the table name, column name, and data type for the columns named BOOK_CODE, TITLE, and PRICE. Order the results by table name within column name. (That is, column name is the major sort key and table name is the minor sort key.)
9. Add the PUBLISHER_CODE column as a foreign key in the BOOK table.
10. Ensure that the PAPERBACK column in the BOOK table can accept only values of Y or N.

Alexamara Marina Group

Use SQL to make the following changes to the Alexamara Marina Group database (Figures 1-8 through 1-12 in Chapter 1). After each change, execute an appropriate query to show that the change was made correctly. If directed to do so by your instructor, use the information provided with the Chapter 3 Exercises to print your output. For any exercises that use commands not supported by your version of SQL, write the command to accomplish the task.

1. Create a view named LARGE_SLIP. It consists of the marina number, slip number, rental fee, boat name, and owner number for every slip whose length is 40 feet.
   a. Write and execute the CREATE VIEW command to create the LARGE_SLIP view.
   b. Write and execute the command to retrieve the marina number, slip number, rental fee, and boat name for every slip with a rental fee of $3,800 or more.
   c. Write and execute the query that the DBMS actually executes.
   d. Does updating the database through this view create any problems? If so, what are they? If not, why not?
2. Create a view named RAY_4025. It consists of the marina number, slip number, length, rental fee, boat name, and owner's last name for every slip in which the boat type is Ray 4025.
   a. Write and execute the CREATE VIEW command to create the RAY_4025 view.
   b. Write and execute the command to retrieve the marina number, slip number, rental fee, boat name, and owner's last name for every slip in the RAY_4025 view with a rental fee of less than $4,000.

c. Write and execute the query that the DBMS actually executes.

d. Does updating the database through this view create any problems? If so, what are they? If not, why not?

3. Create a view named SLIP_FEES. It consists of two columns: the first is the slip length, and the second is the average fee for all slips in the MARINA_SLIP table that have that length. Use AVERAGE_FEE as the name for the average fee. Group and order the rows by slip length.

   a. Write and execute the CREATE VIEW command to create the SLIP_FEES view.

   b. Write and execute the command to retrieve the slip length and average fee for each length for which the average fee is less than $3,500.

   c. Write and execute the query that the DBMS actually executes.

   d. Does updating the database through this view create any problems? If so, what are they? If not, why not?

4. Write, but do not execute, the commands to grant the following privileges:

   a. User Oliver must be able to retrieve data from the MARINA_SLIP table.

   b. Users Crandall and Perez must be able to add new owners and slips to the database.

   c. Users Johnson and Klein must be able to change the rental fee of any slip.

   d. All users must be able to retrieve the length, boat name, and owner number for every slip.

   e. User Klein must be able to add and delete service categories.

   f. User Adams must be able to create an index on the SERVICE_REQUEST table.

   g. Users Adams and Klein must be able to change the structure of the MARINA_SLIP table.

   h. User Klein must have all privileges on the MARINA, OWNER, and MARINA_SLIP tables.

5. Write, but do not execute, the command to revoke all privileges from user Adams.

6. Create the following indexes:

   a. Create an index named BOAT_INDEX1 on the OWNER_NUM column in the MARINA_SLIP table.

   b. Create an index named BOAT_INDEX2 on the BOAT_NAME column in the MARINA_SLIP table.

   c. Create an index named BOAT_INDEX3 on the LENGTH and BOAT_NAME columns in the MARINA_SLIP table. List the lengths in descending order.

7. Delete the index named BOAT_INDEX3; it is no longer necessary.

8. Write the commands to obtain the following information from the system catalog. Do not execute these commands unless your instructor specifically asks you to do so.

   a. List every column in the MARINA_SLIP table and its associated data type.

   b. List every table and view that contains a column named MARINA_NUM.

9. Add the OWNER_NUM column as a foreign key in the MARINA_SLIP table.

10. Ensure that the LENGTH column in the MARINA_SLIP table can accept only values of 25, 30, or 40.

This page contains answers for this chapter only.

CHAPTER 7—DATABASE ADMINISTRATION

1. A view contains data that is derived from existing base tables when users attempt to access the view.
3. A defining query is the portion of the CREATE VIEW command that describes the data to include in a view.
5. Views provide data independence, allow database access control, and simplify the database structure for users.
7. DROP VIEW
9. REVOKE
11. Use the CREATE INDEX command to create an index. Use the CREATE UNIQUE INDEX command to create a unique index. A unique index allows only unique values in the column (or columns) on which the index is created.
13. The DBMS
15. Answers will vary. Answers should note that a data dictionary is a catalog that stores data about the entities, attributes, relationships, programs, and other

objects in a database. Some items found in a data dictionary include synonyms for attributes, detailed descriptions of each table and attribute in the database, referential integrity constraints, and database schema definitions.

17. The DBMS updates the system catalog automatically when users make change to the database, such as creating, altering, or dropping tables or creating or dropping indexes.
19. Use the CHECK clause of the ALTER TABLE command.
21. Use the ADD FOREIGN KEY clause of the ALTER TABLE command.

This page contains answers for this chapter only.