CHAPTER 6

UPDATING DATA

LEARNING OBJECTIVES

Objectives

- Create a new table from an existing table
- Change data using the UPDATE command
- Add new data using the INSERT command
- Delete data using the DELETE command
- Use nulls in an UPDATE command
- Change the structure of an existing table
- Use the COMMIT and ROLLBACK commands to make permanent data updates or to reverse updates
- Understand transactions and the role of COMMIT and ROLLBACK in supporting transactions
- Drop a table

INTRODUCTION

In this chapter, you will learn how to create a new table from an existing table and make changes to the data in a table. You will use the UPDATE command to change data in one or more rows in a table, and use the INSERT command to add new rows. You will use the DELETE command to delete rows. You will learn how to change the structure of a table in a variety of ways and use nulls in update operations. You will use the COMMIT command to make changes permanent and use the ROLLBACK command to undo changes, and understand how to use these commands in transactions. Finally, you will learn how to delete a table and its data.

CREATING A NEW TABLE FROM AN EXISTING TABLE

You can create a new table using data in an existing table, as illustrated in the following examples.

EXAMPLE 1

Create a new table named LEVEL1_CUSTOMER that contains the following columns from the CUSTOMER table: CUSTOMER_NUM, CUSTOMER_NAME, BALANCE, CREDIT_LIMIT, and REP_NUM. The columns in the new LEVEL1_CUSTOMER table should have the same characteristics as the corresponding columns in the CUSTOMER table.

You describe the new table named LEVEL1_CUSTOMER by using the CREATE TABLE command shown in Figure 6-1.

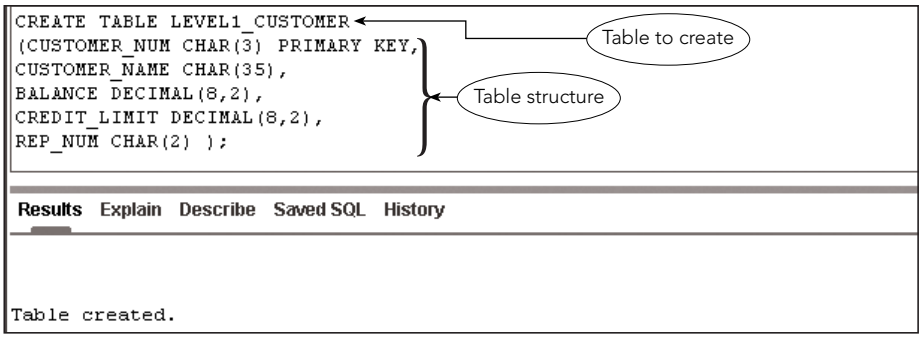


FIGURE 6-1 Creating the LEVEL1_CUSTOMER table

ACCESS USER NOTE

If you are using Access to create the LEVEL1_CUSTOMER table, use the CURRENCY data type instead of the DECIMAL data type for the BALANCE and CREDIT_LIMIT fields. (Access does not support the DECIMAL data type.) You do not need to enter the field size and number of decimal places when using the CURRENCY data type.

EXAMPLE 2

Insert into the LEVEL1_CUSTOMER table the customer number, customer name, balance, credit limit, and rep number for customers with credit limits of $7,500.

You can create a SELECT command to select the desired data from the CUSTOMER table, just as you did in Chapter 4. By placing this SELECT command in an INSERT command, you can add the query results to a table. The INSERT command appears in Figure 6-2; this command inserts four rows into the LEVEL1_CUSTOMER table.

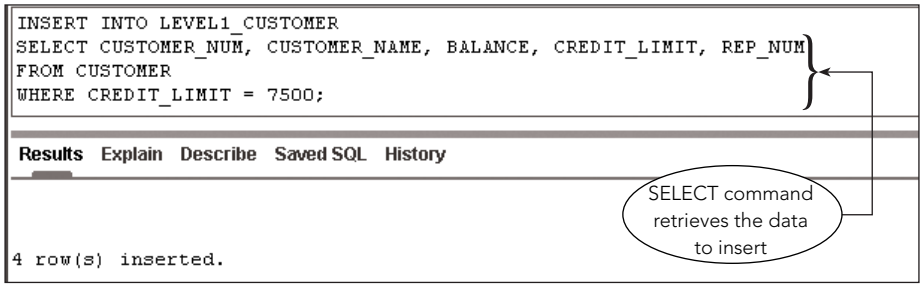


FIGURE 6-2 INSERT command to add data to the LEVEL1_CUSTOMER table

The SELECT command shown in Figure 6-3 displays the data in the LEVEL1_CUSTOMER table. Notice that the data comes from the new table you just created (LEVEL1_CUSTOMER), and not from the CUSTOMER table.

```
SELECT *
FROM LEVEL1_CUSTOMER;
```

Results Explain Describe Saved SQL History

CUSTOMER_NUM	CUSTOMER_NAME	BALANCE	CREDIT_LIMIT	REP_NUM
148	Al's Appliance and Sport	6550	7500	20
356	Ferguson's	5785	7500	65
725	Deerfield's Four Seasons	248	7500	35
842	All Season	8221	7500	20

4 rows returned in 0.15 seconds CSV Export

FIGURE 6-3 LEVEL1_CUSTOMER data

CHANGING EXISTING DATA IN A TABLE

The data stored in tables is subject to constant change; prices, addresses, commission amounts, and other data in a database change on a regular basis. To keep data current, you must be able to make these changes to the data in your tables. You can use the **UPDATE** command to change rows for which a specific condition is true.

EXAMPLE 3

Change the name of customer 842 in the LEVEL1_CUSTOMER table to "All Season Sport."

The format for the UPDATE command is the word UPDATE, followed by the name of the table to be updated. The next portion of the command consists of the word SET, followed by the name of the column to be updated, an equals sign, and the new value. When necessary, include a WHERE clause to indicate the row(s) on which the change is to occur. The UPDATE command shown in Figure 6-4 changes the name of customer 842 to All Season Sport.

```
UPDATE LEVEL1_CUSTOMER
SET CUSTOMER_NAME = 'All Season Sport'
WHERE CUSTOMER_NUM = '842';
```

Results Explain Describe Saved SQL History

```
1 row(s) updated.
```

FIGURE 6-4 UPDATE command to change the name of customer 842

The SELECT command shown in Figure 6-5 shows the data in the table after the change has been made. It is a good idea to use a SELECT command to display the data you changed to verify that the correct update was made.

```
SELECT *
FROM LEVEL1_CUSTOMER;
```

Results Explain Describe Saved SQL History

CUSTOMER_NUM	CUSTOMER_NAME	BALANCE	CREDIT_LIMIT	REP_NUM
148	Al's Appliance and Sport	6550	7500	20
356	Ferguson's	5785	7500	65
725	Deerfield's Four Seasons	248	7500	35
842	All Season Sport	8221	7500	20

4 rows returned in 0.06 seconds CSV Export

FIGURE 6-5 LEVEL1_CUSTOMER table after update

EXAMPLE 4

For each customer in the LEVEL1_CUSTOMER table that is represented by sales rep 20 and also has a balance that does not exceed the credit limit, increase the customer's credit limit to $8,000.

The only difference between Examples 3 and 4 is that Example 4 uses a compound condition to identify the row(s) to be changed. The UPDATE command appears in Figure 6-6.


```
UPDATE LEVEL1_CUSTOMER
SET CREDIT_LIMIT = 8000
WHERE REP_NUM = '20'
AND BALANCE < CREDIT_LIMIT;
```

Results Explain Describe Saved SQL History

```
1 row(s) updated.
```

FIGURE 6-6 Using a compound condition in an update

The SELECT command shown in Figure 6-7 shows the table after the update.

```
SELECT *
FROM LEVEL1_CUSTOMER;
```

Results Explain Describe Saved SQL History

CUSTOMER_NUM	CUSTOMER_NAME	BALANCE	CREDIT_LIMIT	REP_NUM
148	Al's Appliance and Sport	6550	8000	20
356	Ferguson's	5785	7500	65
725	Deerfield's Four Seasons	248	7500	35
842	All Season Sport	8221	7500	20

4 rows returned in 0.08 seconds CSV Export

FIGURE 6-7 Credit limit increased for customer number 148

You also can use the existing value in a column and a calculation to update a value. For example, when you need to increase the credit limit by 10 percent instead of changing it to a specific value, you can multiply the existing credit limit by 1.10. The following SET clause makes this change:

```
SET CREDIT_LIMIT = CREDIT_LIMIT * 1.10
```

ADDING NEW ROWS TO AN EXISTING TABLE

In Chapter 3, you used the INSERT command to add the initial rows to the tables in the database. You also can use the INSERT command to add additional rows to tables.

EXAMPLE 5

Add customer number 895 to the LEVEL1_CUSTOMER table. The name is Peter and Margaret's, the balance is 0, the credit limit is $8,000, and the rep number is 20.

The appropriate INSERT command is shown in Figure 6-8. Because the name "Peter and Margaret's" contains an apostrophe, you type two single quotation marks to create the apostrophe.

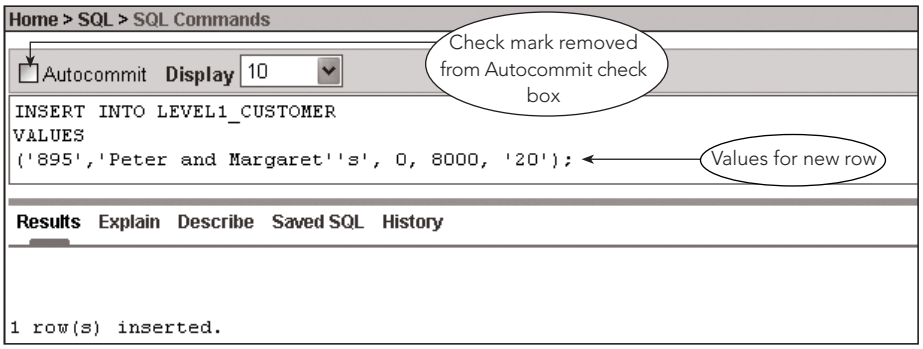
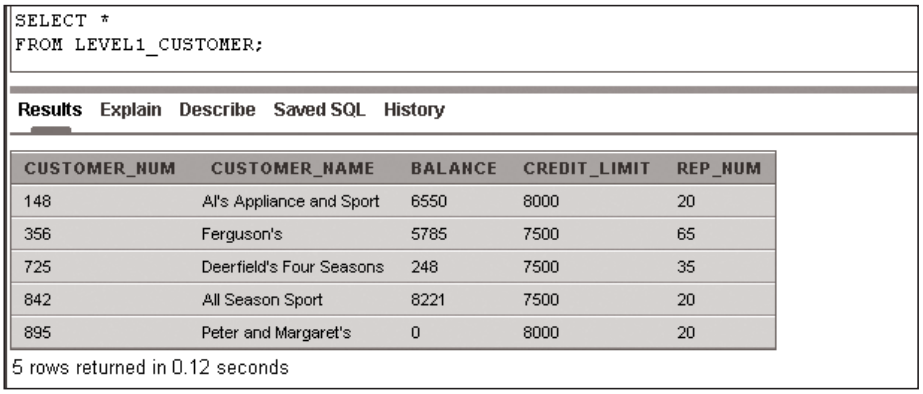


FIGURE 6-8 Inserting a row

The SELECT command in Figure 6-9 shows that the row was successfully added.



CUSTOMER_NUM	CUSTOMER_NAME	BALANCE	CREDIT_LIMIT	REP_NUM
148	Al's Appliance and Sport	6550	8000	20
356	Ferguson's	5785	7500	65
725	Deerfield's Four Seasons	248	7500	35
842	All Season Sport	8221	7500	20
895	Peter and Margaret's	0	8000	20

FIGURE 6-9 Customer 895 added to LEVEL1_CUSTOMER table

NOTE

Your output might be sorted in a different order from what is shown in Figure 6-9. If you need to sort the rows in a specific order, use an ORDER BY clause with the desired sort key(s).

COMMIT AND ROLLBACK

Figure 6-8 shows that the user cleared the check mark from the Autocommit check box before running the query. **Autocommit** is the default transaction mode and commits (makes permanent) each action query (INSERT, UPDATE, DELETE) as soon as the user executes the query. Although the Autocommit transaction mode is fine for most action queries, there are times when the user needs better control over when a transaction is committed. This is particularly important in multi-user database applications when more than one person can update the database and in applications when users are running script files that contain multiple updates. When you need more control over when transactions are committed, you should disable the Autocommit feature by clearing its check box before executing a query.

If you do not use Autocommit, queries that include updates to table data are only temporary and you can reverse (cancel) them at any time during your current work session. Updates become permanent automatically when you exit from the DBMS. If you are not using Autocommit during your current work session, however, you can still **commit** (save) your changes immediately by executing the **COMMIT** command.

If you decide that you do not want to save the changes you have made during your current work session, you can **roll back** (reverse) the changes by executing the **ROLLBACK** command. Any updates made since you ran the most recent COMMIT command will be reversed when you run the ROLLBACK command. If you have not run the COMMIT command, executing the ROLLBACK command will reverse all updates made during the current work session. You should note that the ROLLBACK command reverses only changes made to the data; it does not reverse changes made to a table's structure. For example, if you change the length of a character column, you cannot use the ROLLBACK command to return the column length to its original state.

If you determine that an update was made incorrectly, you can use the ROLLBACK command to return the data to its original state. If, on the other hand, you have verified that the update you made is correct, you can use the COMMIT command to make the update permanent. You do this by typing COMMIT; after running the update. However, you should note that the COMMIT command is permanent; after executing a COMMIT command, running the ROLLBACK command cannot reverse the update.

ACCESS USER NOTE

Access does not support the COMMIT or ROLLBACK commands.

SQL SERVER USER NOTE

In SQL Server, the commands used to commit and roll back data are COMMIT TRANSACTION and ROLLBACK TRANSACTION. By default, SQL Server is in Autocommit transaction mode. To turn off the Autocommit feature, execute the following command:

```
SET XACT_ABORT ON
```

To turn the Autocommit feature back on, execute the following command:

```
SET XACT_ABORT OFF
```


TRANSACTIONS

A **transaction** is a logical unit of work. You can think of a transaction as a sequence of steps that accomplish a single task. When discussing transactions, it is essential that the entire sequence is completed successfully.

For example, to enter an order, you must add the corresponding order to the ORDERS table, and then add each order line in the order to the ORDER_LINE table. These multiple steps accomplish the "single" task of entering an order. Suppose you have added the order and the first order line, but you are unable to enter the second order line for some reason; perhaps the part on the order line does not exist. This problem would leave the order in a partially entered state, which is unacceptable. To prevent this problem, you would execute a rollback, thus reversing the insertion of the order and the first order line.

You can use the COMMIT and ROLLBACK commands to support transactions as follows:

- Before beginning the updates for a transaction, commit any previous updates by executing the COMMIT command.
- Complete the updates for the transaction. If any update cannot be completed, execute the ROLLBACK command and discontinue the updates for the current transaction.
- If you can complete all updates successfully, execute the COMMIT command after completing the final update.

CHANGING AND DELETING EXISTING ROWS

As you learned in Chapter 3, you use the DELETE command to remove rows from a table. In Example 6, you will change data and then use the DELETE command to delete a customer from the LEVEL1_CUSTOMER table. In Example 7, you will execute a rollback to reverse the updates made in Example 6. In this case, the rollback will return the row to its previous state and reinstate the deleted record.

EXAMPLE 6

In the LEVEL1_CUSTOMER table, change the name of customer 356 to "Smith Sport," and then delete customer 895.

To delete data from the database, use the DELETE command. The format for the **DELETE** command is the word DELETE followed by the name of the table containing the row(s) to be deleted. Next, use a WHERE clause with a condition to select the row(s) to delete. All rows satisfying the condition will be deleted.

The first part of Example 6 requests a name change for customer 356; the command shown in Figure 6-10 makes this change.


```
UPDATE LEVEL1_CUSTOMER
SET CUSTOMER_NAME = 'Smith Sport'
WHERE CUSTOMER_NUM = '356';
```

Results Explain Describe Saved SQL History

```
1 row(s) updated.
```

FIGURE 6-10 Using an UPDATE command to change the name of customer 356

The second part of Example 6 requires deleting customer 895; this command is shown in Figure 6-11.

```
DELETE FROM LEVEL1_CUSTOMER
WHERE CUSTOMER_NUM = '895';
```

Results Explain Describe Saved SQL History

```
1 row(s) deleted.
```

FIGURE 6-11 Using a DELETE command to delete customer 895

The command shown in Figure 6-12 displays the data in the table, verifying the change and the deletion.

```
SELECT *
FROM LEVEL1_CUSTOMER;
```

Results Explain Describe Saved SQL History

CUSTOMER_NUM	CUSTOMER_NAME	BALANCE	CREDIT_LIMIT	REP_NUM
148	Al's Appliance and Sport	6550	8000	20
356	Smith Sport	5785	7500	65
725	Deerfield's Four Seasons	248	7500	35
842	All Season Sport	8221	7500	20

4 rows returned in 0.06 seconds CSV Export

FIGURE 6-12 Results of update and delete

Q & A

Question: What happens when you run a DELETE command that does not contain a WHERE clause?
Answer: Without a condition to specify which row(s) to delete, the query will delete all rows from the table.

Executing a Rollback

The following example executes a rollback.

EXAMPLE 7

Execute a rollback and then display the data in the LEVEL1_CUSTOMER table.

To execute a rollback, execute the ROLLBACK command, as shown in Figure 6-13.

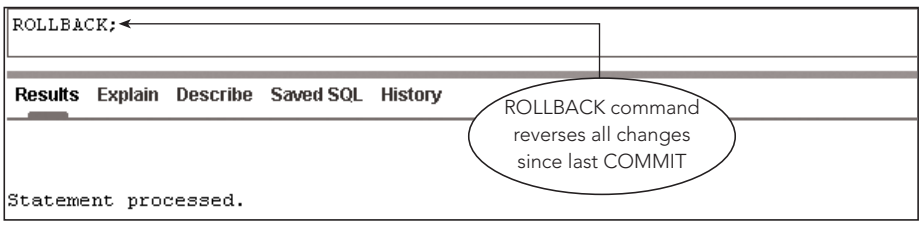


FIGURE 6-13 Executing a rollback

Figure 6-14 shows a SELECT command for the LEVEL1_CUSTOMER table after executing the rollback. Notice that the name of customer 356 has changed back to Ferguson's and

the row for customer 895 has been reinstated. All updates made prior to the previous commit are still reflected in the data.

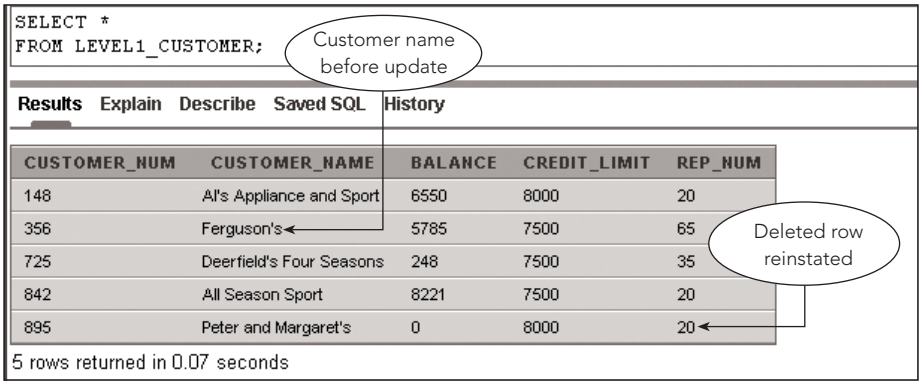


FIGURE 6-14 Data in the LEVEL1_CUSTOMER table after executing a rollback

ACCESS USER NOTE

If you are using Access to complete these steps, you will not be able to execute the ROLLBACK command. Consequently, your data for the remaining examples in this chapter will differ slightly from the data shown in the figures—customer 356 will be named Smith Sport and customer 895 will not be included.

NOTE

In the remaining examples in this chapter, the Autocommit feature is enabled, that is, there is a check mark in the Autocommit check box. All updates are committed immediately without requiring any special action on your part. In addition, it will no longer be possible to roll back updates.

CHANGING A VALUE IN A COLUMN TO NULL

There are some special issues involved when dealing with nulls. You already have seen how to add a row in which some of the values are null and how to select rows in which a given column is null. You also must be able to change the value in a column in an existing row to null, as shown in Example 8. Remember that to make this type of change, the affected column must accept nulls. If you specified NOT NULL for the column when you created the table, then changing a value in a column to null is prohibited.

EXAMPLE 8

Change the balance of customer 725 in the LEVEL1_CUSTOMER table to null.

The command for changing a value in a column to null is exactly what it would be for changing any other value. You simply use the value NULL as the replacement value, as shown in Figure 6-15. Notice that the value NULL is *not* enclosed in single quotation marks. If it were, the command would change the balance to the word NULL.

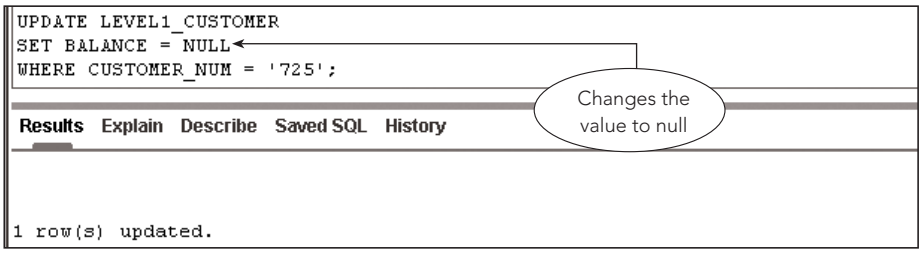


FIGURE 6-15 Changing a value in a column to null

Figure 6-16 shows the data in the LEVEL1_CUSTOMER table after changing the BALANCE column value for customer 725 to null. In Oracle 10*g*, a null value is displayed as a hyphen, as shown in Figure 6-16.

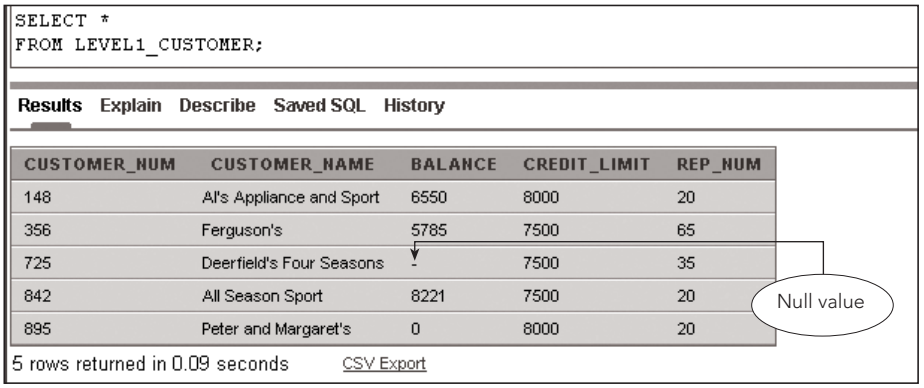


FIGURE 6-16 BALANCE column for customer 725 is null

SQL SERVER USER NOTE

In SQL Server, the word "NULL" appears in the results (without the quotation marks) when a column contains a null value.

CHANGING A TABLE'S STRUCTURE

One of the nicest features of a relational DBMS is the ease with which you can change table structures. In addition to adding new tables to the database and deleting tables that are no longer required, you can add new columns to a table and change the physical characteristics of existing columns. Next, you will see how to accomplish these changes.

You can change a table's structure in SQL by using the **ALTER TABLE** command, as illustrated in the following examples.

EXAMPLE 9

Premiere Products decides to maintain a customer type for each customer in the database. These types are R for regular customers, D for distributors, and S for special customers. Add this information in a new column named CUSTOMER_TYPE in the LEVEL1_CUSTOMER table.

To add a new column, use the **ADD clause** of the ALTER TABLE command. The format for the ALTER TABLE command is the words ALTER TABLE followed by the name of the table to be altered and an appropriate clause. The ADD clause consists of the word ADD followed by the name of the column to be added, followed by the characteristics of the column. Figure 6-17 shows the appropriate ALTER TABLE command for this example.

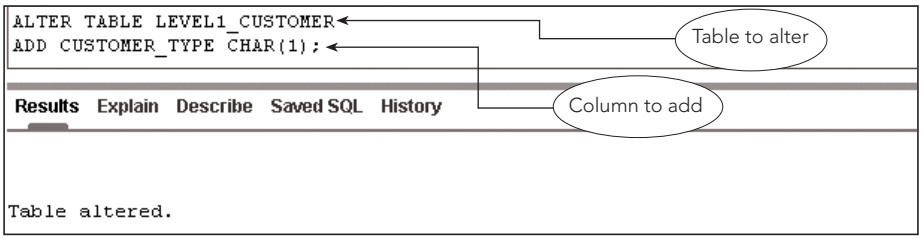


FIGURE 6-17 Adding a column to an existing table

The LEVEL1_CUSTOMER table now contains a column named CUSTOMER_TYPE, a CHAR column with a length of 1. Any new rows added to the table must include values for the new column. Effective immediately, all existing rows also contain this new column. The data in any existing row will contain the new column the next time the row is updated. Any time a row is selected for any reason, however, the system treats the row as though the column is actually present. Thus, to the user, it seems as though the structure was changed immediately.

For existing rows, you must assign some value to the CUSTOMER_TYPE column. The simplest approach (from the point of view of the DBMS, not the user) is to assign the value NULL as a CUSTOMER_TYPE in all existing rows. This process requires the CUSTOMER_TYPE column to accept null values, and some systems actually insist on this. The default for Oracle, Access, and SQL Server is to accept null values.

To change the values in a new column that was added using an ALTER TABLE command, follow the ALTER TABLE command with an UPDATE command like the one shown in Figure 6-18, which sets the CUSTOMER_TYPE value for all rows to R.

```
UPDATE LEVEL1_CUSTOMER
SET CUSTOMER_TYPE = 'R';
```

Omitting the WHERE clause updates all rows

Results Explain Describe Saved SQL History

```
5 row(s) updated.
```


FIGURE 6-18 Making the same update for all rows

The SELECT command shown in Figure 6-19 verifies that the value in the CUSTOMER_TYPE column for all rows is R.

```
SELECT *
FROM LEVEL1_CUSTOMER;
```

Results Explain Describe Saved SQL History

CUSTOMER_NUM	CUSTOMER_NAME	BALANCE	CREDIT_LIMIT	REP_NUM	CUSTOMER_TYPE
148	Al's Appliance and Sport	6550	8000	20	R
356	Ferguson's	5785	7500	65	R
725	Deerfield's Four Seasons	-	7500	35	R
842	All Season Sport	8221	7500	20	R
895	Peter and Margaret's	0	8000	20	R

5 rows returned in 0.18 seconds CSV Export

FIGURE 6-19 CUSTOMER_TYPE set to R for all rows

EXAMPLE 10

Two customers in the LEVEL1_CUSTOMER table have a type other than R. Change the types for customers 842 and 148 to S and D, respectively.

Example 9 used an UPDATE command to assign type R to every customer. To change individual types to something other than type R, use the UPDATE command. Figure 6-20 shows the UPDATE command to change customer 842 to customer type S.

```
UPDATE LEVEL1_CUSTOMER
SET CUSTOMER_TYPE = 'S'
WHERE CUSTOMER_NUM = '842';
```

Results Explain Describe Saved SQL History

```
1 row(s) updated.
```


FIGURE 6-20 Updating customer 842 to customer type S

Figure 6-21 shows the UPDATE command to change customer 148 to customer type D.

```
UPDATE LEVEL1_CUSTOMER
SET CUSTOMER_TYPE = 'D'
WHERE CUSTOMER_NUM = '148';
```

Results Explain Describe Saved SQL History

```
1 row(s) updated.
```

FIGURE 6-21 Updating customer 148 to customer type D

The SELECT command shown in Figure 6-22 shows the results of these UPDATE commands. The customer type for customer 842 is S and the type for customer 148 is D. The type for all other customers is R.

```
SELECT *
FROM LEVEL1_CUSTOMER;
```

Results Explain Describe Saved SQL History

CUSTOMER_NUM	CUSTOMER_NAME	BALANCE	CREDIT_LIMIT	REP_NUM	CUSTOMER_TYPE
148	Al's Appliance and Sport	6550	8000	20	D
356	Ferguson's	5785	7500	65	R
725	Deerfield's Four Seasons	-	7500	35	R
842	All Season Sport	8221	7500	20	S
895	Peter and Margaret's	0	8000	20	R

5 rows returned in 0.00 seconds CSV Export

FIGURE 6-22 Customer types in the LEVEL1_CUSTOMER table after updates

Figure 6-23 uses the DESCRIBE command to display the structure of the LEVEL1_CUSTOMER table, which now includes the CUSTOMER_TYPE column.

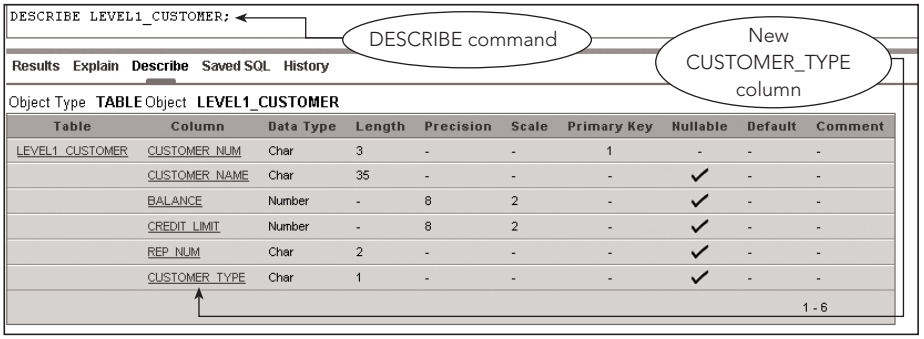


FIGURE 6-23 Structure of the LEVEL1_CUSTOMER table

ACCESS USER NOTE

In Access, use the Documenter tool to show the layout of a table.

SQL SERVER USER NOTE

In SQL Server, execute the following command to list all the columns in the LEVEL1_CUSTOMER table:

```
Exec sp_columns LEVEL1_CUSTOMER
```

EXAMPLE 11

The length of the CUSTOMER_NAME column in the LEVEL1_CUSTOMER table is too short. Increase its length to 50 characters. In addition, change the CREDIT_LIMIT column so it cannot accept nulls.

You can change the characteristics of existing columns by using the **MODIFY clause** of the ALTER TABLE command. Figure 6-24 shows the ALTER TABLE command that changes the length of the CUSTOMER_NAME column from 35 to 50 characters.

```
ALTER TABLE LEVEL1_CUSTOMER
MODIFY CUSTOMER_NAME CHAR(50);
```

Results Explain Describe Saved SQL History

```
Table altered.
```

FIGURE 6-24 Changing the length of the CUSTOMER_NAME column in the LEVEL1_CUSTOMER table

SQL SERVER USER NOTE

To change the length of the CUSTOMER_NAME column in SQL Server, use the following ALTER COLUMN clause:

```
ALTER TABLE LEVEL1_CUSTOMER
ALTER COLUMN CUSTOMER_NAME CHAR(50);
```

ACCESS USER NOTE

The version of the ALTER TABLE command shown in Figure 6-24 is not available in Access; to modify the table's structure, make the changes in Design view and save the table.

NOTE

You also can decrease the length of columns, but you might lose some data currently in the column. For example, if you decrease the length of the CUSTOMER_NAME column from 35 to 20 characters, only the first 20 characters of the current customer names will be included. Any characters from position 21 on will be lost. Thus, you should only decrease column lengths when you are positive that you will not lose any data stored in the column.

You can change the length of DECIMAL columns in the same manner that you change the length of CHAR columns.

Figure 6-25 shows the ALTER TABLE command to change the CREDIT_LIMIT column so it does not accept null values.

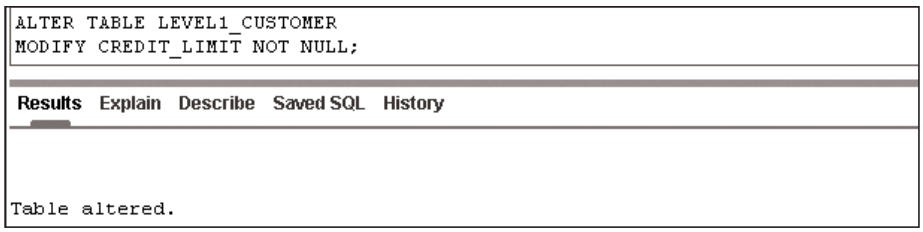


FIGURE 6-25 Changing the CREDIT_LIMIT column in the LEVEL1_CUSTOMER table to reject null values

The DESCRIBE command shown in Figure 6-26 shows the revised structure of the LEVEL1_CUSTOMER table. The length of the CUSTOMER_NAME column is 50 characters.

The dash in the Nullable column for the CREDIT_LIMIT column (instead of a check mark) indicates that the CREDIT_LIMIT column no longer accepts null values.

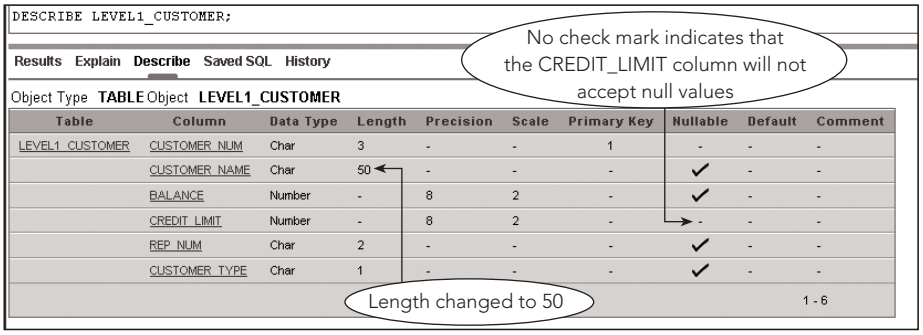


FIGURE 6-26 Revised structure of the LEVEL1_CUSTOMER table

NOTE

You also can use the MODIFY clause of the ALTER TABLE command to change a column that currently rejects null values so that it accepts null values by using NULL in place of NOT NULL in the ALTER TABLE command.

NOTE

If there were existing rows in the LEVEL1_CUSTOMER table in which the CREDIT_LIMIT column was already null, the DBMS would reject the modification to the CREDIT_LIMIT column shown in Figure 6-25 and display an error message indicating that this change is not possible. In this case, you first must use an UPDATE command to change all values that are null to some other value. Then you could alter the table's structure as shown in the figure.

Making Complex Changes

In some cases, you might need to change a table's structure in ways that are either beyond the capabilities of SQL or that are so complex that it would take longer to make the changes than to re-create the table. Perhaps you need to eliminate multiple columns, rearrange the order of several columns, or combine data from two tables into one. For example, if you try to change a column with a data type of VARCHAR to CHAR, SQL still uses VARCHAR when the table contains other variable-length columns. In these situations, you can use a CREATE TABLE command to describe the new table (which must use a different name than the existing table), and then insert values from the existing table into it using the INSERT command combined with an appropriate SELECT command.

DROPPING A TABLE

As you learned in Chapter 3, you can delete a table that is no longer needed by executing the DROP TABLE command.

EXAMPLE 12

Delete the LEVEL1_CUSTOMER table because it is no longer needed in the Premiere Products database.

The command to delete the table is shown in Figure 6-27.

```
DROP TABLE LEVEL1_CUSTOMER;
```

Results Explain Describe Saved SQL History

```
Table dropped.
```

FIGURE 6-27 DROP TABLE command to delete the LEVEL1_CUSTOMER table

When the command shown in Figure 6-27 is executed, the LEVEL1_CUSTOMER table and all its data are permanently removed from the database.

Chapter Summary

- To create a new table from an existing table, first create the new table by using the CREATE TABLE command. Then use an INSERT command containing a SELECT command to select the desired data to be included from the existing table.
- Use the UPDATE command to change existing data in a table.
- Use the INSERT command to add new rows to a table.
- Use the DELETE command to delete existing rows from a table.
- Use the COMMIT command to make updates permanent; use the ROLLBACK command to reverse any updates that have not been committed.
- To change all values in a column to null, use the SET clause followed by the column name, an equal sign, and the word NULL. To change a specific value in a column to null, use a condition to select the row.
- To add a column to a table, use the ALTER TABLE command with an ADD clause.
- To change the characteristics of a column, use the ALTER TABLE command with a MODIFY clause.
- Use the DROP TABLE command to delete a table and all its data.

Key Terms

ADD clause
ALTER TABLE
Autocommit
commit
COMMIT
DELETE
MODIFY clause
roll back
ROLLBACK
transaction
UPDATE

Review Questions

1. Which command creates a new table?
2. Which command and clause adds an individual row to a table?
3. How do you add data from an existing table to another table?
4. Which command changes data in a table?
5. Which command removes rows from a table?
6. In Oracle and in SQL Server, which command makes updates permanent?
7. In Oracle and in SQL Server, which command reverses updates? Which updates are reversed?
8. How do you use the COMMIT and ROLLBACK commands to support transactions?
9. What is the format of the SET clause that changes the value in a column to null in an UPDATE command?
10. Which command and clause adds a column to an existing table?

11. In Oracle and in SQL Server, which command and clause changes the characteristics of an existing column in a table?
12. Which command deletes a table and all its data?
13. Microsoft Access supports make-table queries. What is a make-table query? What SQL statement(s) are equivalent to make-table queries?
14. Use your favorite Web browser and Web search engine to find the SQL command to delete a column in a table. Write the SQL command in Oracle to delete the CUSTOMER_TYPE column from the LEVEL1_CUSTOMER table. Would you use the same command in SQL Server to delete the column? If no, write the command to use in SQL Server.

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.

1. Create a NONAPPLIANCE table with the structure shown in Figure 6-28.

NONAPPLIANCE

Column	Type	Length	Decimal Places	Nulls Allowed?	Description
PART_NUM	CHAR	4		No	Part number (primary key)
DESCRIPTION	CHAR	15			Part description
ON_HAND	DECIMAL	4	0		Number of units on hand
CLASS	CHAR	2			Item class
PRICE	DECIMAL	6	2		Unit price

FIGURE 6-28 NONAPPLIANCE table layout

2. Insert into the NONAPPLIANCE table the part number, part description, number of units on hand, item class, and unit price from the PART table for each part that is *not* in item class AP.
3. In the NONAPPLIANCE table, change the description of part number AT94 to "Steam Iron."
4. In the NONAPPLIANCE table, increase the price of each item in item class SG by three percent. (*Hint*: Multiply each price by 1.03.)
5. Add the following part to the NONAPPLIANCE table: part number: TL92; description: Edge Trimmer; number of units on hand: 11; class: HW; and price: 29.95.
6. Delete every part in the NONAPPLIANCE table for which the class is SG.
7. In the NONAPPLIANCE table, change the class for part FD21 to null.
8. Add a column named ON_HAND_VALUE to the NONAPPLIANCE table. The on-hand value is a seven-digit number with two decimal places that represents the product of the number of units on hand and the price. Then set all values of ON_HAND_VALUE to ON_HAND * PRICE.

9. In the NONAPPLIANCE table, increase the length of the DESCRIPTION column to 30 characters.
10. Remove the NONAPPLIANCE table from the Premiere Products database.

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.

1. Create a FICTION table with structure shown in Figure 6-29.

FICTION

Column	Type	Length	Decimal Places	Nulls Allowed?	Description
BOOK_CODE	CHAR	4		No	Book code (primary key)
TITLE	CHAR	40			Book title
PUBLISHER_CODE	CHAR	3			Publisher code
PRICE	DECIMAL	4	2		Book price

FIGURE 6-29 FICTION table layout

2. Insert into the FICTION table the book code, book title, publisher code, and price from the BOOK table for only those books having type FIC.
3. The publisher with code LB has decreased the price of its fiction books by four percent. Update the prices in the FICTION table accordingly.
4. Insert a new book into the FICTION table. The book code is 9946, the title is *Cannery Row*, the publisher is PE, and the price is 11.95.
5. Delete the book in the FICTION table having the book code 9883.
6. The price of the book entitled *To Kill a Mockingbird* has been increased to an unknown amount. Change the value in the FICTION table to reflect this change.
7. Add to the FICTION table a new character column named BEST_SELLER that is one character in length. Then set the default value for all columns to N.
8. Change the BEST_SELLER column in the FICTION table to Y for the book entitled *Song of Solomon*.
9. Change the length of the TITLE column in the FICTION table to 50 characters.
10. Change the BEST_SELLER column in the FICTION table to reject nulls.
11. Delete the FICTION table from the database.

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.

1. Create a LARGE_SLIP table with the structure shown in Figure 6-30. (*Hint*: If you have trouble creating the primary key, see Figure 3-31 in Chapter 3.)

LARGE_SLIP

Column	Type	Length	Decimal Places	Nulls Allowed?	Description
MARINA_NUM	CHAR	4		No	Marina number (primary key)
SLIP_NUM	CHAR	4		No	Slip number in the marina (primary key)
RENTAL_FEE	DECIMAL	8	2		Annual rental fee for the slip
BOAT_NAME	CHAR	50			Name of boat currently in the slip
OWNER_NUM	CHAR	4			Number of boat owner renting the slip

FIGURE 6-30 LARGE_SLIP table layout

2. Insert into the LARGE_SLIP table the marina number, slip number, rental fee, boat name, and owner number for those slips whose length is 40 feet.
3. Alexamara has increased the rental fee of each large slip by $150. Update the rental fees in the LARGE_SLIP table accordingly.
4. After increasing the rental fee of each large slip by $150 (Exercise 3), Alexamara decides to decrease the rental fee of any slip whose fee is more than $4,000 by one percent. Update the rental fees in the LARGE_SLIP table accordingly.
5. Insert a new row into the LARGE_SLIP table. The marina number is 1, the slip number is A4, the rental fee is $3,900, the boat name is *Bilmore*, and the owner number is FE82.
6. Delete all slips in the LARGE_SLIP table for which the owner number is TR72.
7. The name of the boat in marina 1 and slip A1 is in the process of being changed to an unknown name. Change the name of this boat in the LARGE_SLIP table to null.
8. Add to the LARGE_SLIP table a new character column named CHARTER that is one character in length. (This column will indicate whether the boat is available for chartering.) Set the value for the CHARTER column on all rows to N.
9. Change the CHARTER column in the LARGE_SLIP table to Y for the slip containing the boat named *Our Toy*.
10. Change the length of the BOAT_NAME column in the LARGE_SLIP table to 60 characters.
11. Change the RENTAL_FEE column in the LARGE_SLIP table to reject nulls.
12. Delete the LARGE_SLIP table from the database.

This page contains answers for this chapter only.

CHAPTER 6—UPDATING DATA

1. CREATE TABLE
3. Use the INSERT command with a SELECT clause.
5. DELETE
7. In Oracle, use the ROLLBACK command. In SQL Server, use the ROLLBACK TRANSACTION command. Any updates made since the most recent COMMIT command (or COMMIT TRANSACTION command in SQL Server) are reversed.
9. The clause is SET followed by the column name, followed by an equals sign (=) and the word NULL.
11. In Oracle, use the ALTER TABLE command with a MODIFY clause. In SQL Server, use the ALTER TABLE command with an ALTER COLUMN clause.
13. Use a make-table query to create a table from another table. The equivalent SQL commands to the Access make-table query are CREATE TABLE, SELECT, and INSERT.

This page contains answers for this chapter only.