Chapter 8
Changing Table Contents

Objectives

- After completing this lesson, you should be able to do the following:
  - Describe each DML statement
  - Insert rows into a table
  - Update rows in a table
  - Delete rows from a table
  - Control transactions

Data Manipulation Language

- A DML statement is executed when you:
  - Add new rows to a table
  - Modify existing rows in a table
  - Remove existing rows from a table
- A *transaction* consists of a collection of DML statements that form a logical unit of work.

Adding a New Row to a Table
The INSERT Statement

Add new rows to a table by using the INSERT statement.

INSERT INTO ***table*** **[(*column* [, *column*...])]**
VALUES ***(value*** **[, *value*...])*;***

- Only one row is inserted at a time with this syntax.

Inserting New Rows

- Insert a new row containing values for each column.
- List values in the default order of the columns in the table.
- Optionally list the columns in the INSERT clause.

```
SQL> INSERT INTO dept
     2 (deptno, dname, loc)
     3  VALUES
     4  (50, 'DEVELOPMENT', 'DETROIT');
1 row created.
```

- Enclose character and date values within single quotation marks.

Inserting Rows with Null Values

- Implicit method: Omit the column from the column list.

```
SQL> INSERT INTO dept
     2  (deptno, dname )
     3  VALUES
     4  (60, 'MIS');
1 row created.
```

- Explicit method: Specify the NULL keyword.

```
SQL> INSERT INTO dept
  2  VALUES
  3   (70, 'FINANCE', NULL);
1 row created.
```

Inserting Special Values

- The SYSDATE function records the current date and time.

```
SQL> INSERT INTO       emp
  2                   (empno, ename, job,
  3                    mgr, hiredate, sal, comm,
  4                    deptno)
  5  VALUES
  6                   (7196, 'GREEN', 'SALESMAN',
  7                    7782, SYSDATE, 2000, NULL,
  8                    10);
 1 row created.
```

Inserting Specific Date Values

— Add a new employee.

```
SQL> INSERT INTO emp
  2  VALUES
  3                (2296,'AROMANO','SALESMAN',7782,
  4                TO_DATE('FEB 3, 97', 'MON DD, YY'),
  5                1300, NULL, 10);
1 row created.
```

Inserting Values by Using Substitution Variables

- Create an interactive script by using SQL*Plus substitution parameters.

```
SQL> INSERT INTO dept
  2           (deptno, dname, loc)
  3  VALUES
  4           (&department_id,
  5            '&department_name', '&location');

Enter value for department_id: 80
Enter value for department_name: EDUCATION
Enter value for location: ATLANTA

1 row created.
```

Copying Rows from Another Table

- Write your INSERT statement with a subquery.

```
SQL> INSERT INTO managers(id, name, salary, hiredate)
  2                 SELECT      empno, ename, sal, hiredate
  3                 FROM   emp
  4                 WHERE       job = 'MANAGER';
3 rows created.
```

- Do not use the VALUES clause.
- Match the number of columns in the INSERT clause to those in the subquery.

Changing Data in a Table
The UPDATE Statement

- Modify existing rows with the UPDATE statement.

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE            condition];
```

- Update more than one row at a time, if required.

Updating Rows in a Table

- Specific row or rows are modified when you specify the WHERE clause.

```
SQL> UPDATE       emp
  2  SET      deptno = 20
  3  WHERE    empno = 7782;
1 row updated.
```

- All rows in the table are modified if you omit the WHERE clause.

```
SQL> UPDATE   employee
  2  SET      deptno = 20;
14 rows updated.
```

Updating with Multiple-Column Subquery

- Update employee 7698's job and department to match that of employee 7499.

```
SQL> UPDATE  emp
  2  SET     (job, deptno) =
  3                 (SELECT job, deptno
  4                   FROM    emp
  5                   WHERE   empno = 7499)
  6  WHERE   empno = 7698;
1 row updated.
```

Updating Rows Based on Another Table

- Use subqueries in UPDATE statements to update rows in a table based on values from another table.

```
SQL>  UPDATE      employee
  2   SET         deptno =  (SELECT deptno
  3               FROM   emp
  4               WHERE empno = 7788)
  5   WHERE       job    =  (SELECT job
  6               FROM   emp
  7               WHERE empno = 7788);
2 rows updated.
```

Updating Rows: Integrity Constraint Error

- Department number 55 does not exist

```
SQL> UPDATE       emp
  2  SET          deptno = 55
  3  WHERE        deptno = 10;

UPDATE emp
    *
ERROR at line 1:
ORA-02291: integrity constraint (USR.EMP_DEPTNO_FK) violated - parent key not found
```

Removing a Row from a Table The DELETE Statement

- You can remove existing rows from a table by using the DELETE statement.

```
DELETE [FROM]        table
[WHERE          condition];
```

Deleting Rows from a Table

- Specific rows are deleted when you specify the WHERE clause.

```
SQL> DELETE FROM  department
  2  WHERE        dname = 'DEVELOPMENT';
1 row deleted.
```

- All rows in the table are deleted if you omit the WHERE clause.

```
SQL> DELETE FROM    department;
4 rows deleted.
```

Deleting Rows Based on Another Table

- Use subqueries in DELETE statements to remove rows from a table based on values from another table.

```
SQL> DELETE FROM    employee
  2  WHERE          deptno =
  3                      (SELECT   deptno
  4                       FROM     dept
  5                       WHERE    dname ='SALES');
6 rows deleted.
```

Deleting Rows: Integrity Constraint Error

```
SQL> DELETE FROM    dept
  2  WHERE           deptno = 10;

DELETE FROM dept
            *
ERROR at line 1:
ORA-02292: integrity constraint (USR.EMP_DEPTNO_FK) violated - child record found
```

—You cannot delete a row that contains a primary key that is used as a foreign key in another table.

Truncate Statement

- If just want to empty a table of all rows, can use the DDL statement TRUNCATE.
- Is like a DELETE statement without a WHERE clause.
- Will remove ALL rows from a table.
- Unlike DELETE, cannot rollback.
  - DDL statements will implicitly perform a commit. This includes all pending DML statements.

Truncate vs Delete

- TRUNCATE is very fast as does not generate undo information.
- When a table is truncated, the storage for the table and all indexes can be reset back to the initial size. DELETE will never shrink the size of a table or its indexes.

Using Default Values

- Book says that this was introduced in Oracle9i, but has been around longer than that. Definitely in 8i.
- Can define the default value as part of the CREATE TABLE statement.

MERGE Statement

- Use the MERGE statement to select rows from one table for update or insertion into another table.
- The decision whether to update or insert into the target table is based on a condition in the ON clause.
- This statement is a convenient way to combine at least two operations. Can avoid multiple INSERT and UPDATE DML statements.
- MERGE is a deterministic statement. That is, you cannot update the same row of the target table multiple times in the same MERGE statement.
- Is available in Oracle & SQL Server etc but not MySQL as yet.

- Syntax:

MERGE [hint] INTO [schema .] table [t_alias] USING [schema .]
{ table | view | subquery } [t_alias] ON (condition)
WHEN MATCHED THEN merge_update_clause
WHEN NOT MATCHED THEN merge_insert_clause;

- Assume have a table with the following:

SELECT * FROM bonuses;

EMPLOYEE_ID	BONUS
153	100
154	100
155	100
156	100
158	100
159	100
160	100
161	100
163	100

- *The Human Resources manager decides that all employees should receive a bonus. Those who have not made sales get a bonus of 1% of their salary. Those who already made sales get an increase in their bonus equal to 1% of their salary. The MERGE statement implements these changes in one step:*

```
MERGE INTO bonuses D
    USING (SELECT employee_id, salary, department_id FROM employees WHERE
    department_id = 80) S
    ON (D.employee_id = S.employee_id)
    WHEN MATCHED THEN UPDATE SET D.bonus = D.bonus +     S.salary*.01
    WHEN NOT MATCHED THEN INSERT (D.employee_id, D.bonus) VALUES
    (S.employee_id, S.salary*0.1);
```

SELECT * FROM bonuses;

```
EMPLOYEE_ID BONUS
 ----------- ----------
153          180
154          175
155          170
156          200
158          190
159          180
160          175
161          170
163          195
157          950
145          1400
170          960
179          620
152          900
169          1000
```

Database Transactions

- Consist of one of the following statements:
  - DML statements that make up one consistent change to the data
  - One DDL statement
  - One DCL statement
- Begin when the first executable SQL statement is executed
- End with one of the following events:
  - COMMIT or ROLLBACK is issued
  - DDL or DCL statement executes (automatic commit)
  - User exits
  - System crashes

Advantages of COMMIT and ROLLBACK Statements

- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically related operations

Implicit Transaction Processing

- An automatic commit occurs under the following circumstances:
  - DDL statement is issued
  - DCL statement is issued
  - Normal exit from SQL*Plus, without explicitly issuing COMMIT or ROLLBACK
- An automatic rollback occurs under an abnormal termination of SQL*Plus or a system failure.

State of the Data Before COMMIT or ROLLBACK

- The previous state of the data can be recovered.
- The current user can review the results of the DML operations by using the SELECT statement.
- Other users *cannot* view the results of the DML statements by the current user.
- The affected rows are *locked*; other users cannot change the data within the affected rows.

State of the Data After COMMIT

- Data changes are made permanent in the database.
- The previous state of the data is permanently lost.
- All users can view the results.
- Locks on the affected rows are released; those rows are available for other users to manipulate.

Committing Data

– Make the changes.

```
SQL> UPDATE   emp
  2     SET     deptno = 10
  3     WHERE   empno = 7782;
1 row updated.
```

- Commit the changes.

```
SQL> COMMIT;
Commit complete.
```

State of the Data After ROLLBACK

- Discard all pending changes by using the ROLLBACK statement.
  - Data changes are undone.
  - Previous state of the data is restored.
  - Locks on the affected rows are released.

```
SQL> DELETE FROM   employee;
14 rows deleted.

SQL> ROLLBACK;
Rollback complete.
```

Rolling Back Changes to a Marker

- Create a marker in a current transaction by using the SAVEPOINT statement.
- Roll back to that marker by using the ROLLBACK TO SAVEPOINT statement.

```
SQL> UPDATE...
SQL> SAVEPOINT update_done;
Savepoint created.

SQL> INSERT...
SQL> ROLLBACK TO update_done;
Rollback complete.
```

Statement-Level Rollback

- If a single DML statement fails during execution, only that statement is rolled back.
- The Oracle Server implements an implicit savepoint.
- All other changes are retained.
- The user should terminate transactions explicitly by executing a COMMIT or ROLLBACK statement.

Read Consistency

- Read consistency guarantees a consistent view of the data at all times.
- Changes made by one user do not conflict with changes made by another user.
- Read consistency ensures that on the same data:
  - Readers do not wait for writers
  - Writers do not wait for readers

Locking

- Oracle locks:
  - Prevent destructive interaction between concurrent transactions
  - Require no user action
  - Automatically use the lowest level of restrictiveness
  - Are held for the duration of the transaction
  - Have two basic modes:
    - Exclusive
    - Share

Summary

Statement	Description
INSERT	Adds a new row to the table
UPDATE	Modifies existing rows in the table
DELETE	Removes existing rows from the table
COMMIT	Makes all pending changes permanent
SAVEPOINT	Allows a rollback to the savepoint marker
ROLLBACK	Discards all pending data changes