DML Statements

Objectives

- Insert rows into a table
- Update rows in a table
- Delete rows from a table
- Controlling the 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.

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.
- Enclose character and date values within single quotation marks.

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

Insert Rows with Null Values

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

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

- Explicit method: Specify the NULL keyword.

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

Inserting Special Values

- The SYSDATE and USER function records the current date and time.

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

Inserting Specific Date Values

- Add a new employee.

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

- Verify your addition.

```
EMPNO ENAME    JOB       MGR   HIREDATE  SAL COMM DEPTNO
----- -------- --------- ----  --------- ---- ---- ------
 2296 AROMANO  SALESMAN  7782  03-FEB-97 1300           10
```

Substitution Variables (&)

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

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

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

1 row created.
```

Substitution Variables (&&)

- Use the double –ampersand(&&) if you want to reuse the variable value without prompting the user each time.

```
SQL> SELECT empno,ename,job, &&column_name
  2  FROM employee
  3  ORDER BY &column_name;
```

```
Enter value for column_name: deptno
  EMPNO    ENAME                JOB                DEPTNO
-------    ------------------   --------------   --------
   7839    KING                 PRESIDENT              10
   7782    CLARK                MANAGER                10
   7934    MILLER               CLERK                  10
. . .
14 rows selected.
```

Customized Prompts

- ACCEPT stores the value into a variable.
- PROMPT displays your customized text.

```
ACCEPT        department_id PROMPT 'Please enter the -
              department number:'
ACCEPT        department_name PROMPT 'Please enter -
              the department name:'
ACCEPT        location PROMPT 'Please enter the -
              location:'
INSERT INTO   department (deptno, dname, loc)
VALUES        (&department_id, '&department_name',
              '&location');
```

Copying 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   employee
  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.

UPDATE Statement

- Modify existing rows with the UPDATE statement.

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

Updating Rows in a Table

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

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

Updating Rows:

- Integrity Constraint Error

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

Department number 55 does not exist

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

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 row or 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:

- Integrity Constraint Error

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

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

- You ca… primary key … another table.

Database Transaction

A database transaction consists of one of the following:

- **DML statements which constitute one consistent change to the data**
- **One DDL statement**
- **One DCL statement**

Oracle Transaction Types

Type	Description
Data manipulation language (DML)	Consists of any number of DML statements that the Oracle server treats as a single entity or a logical unit of work
Data definition language (DDL)	Consists of only one DDL statement
Data control language (DCL)	Consists of only one DCL statement

Transaction boundaries

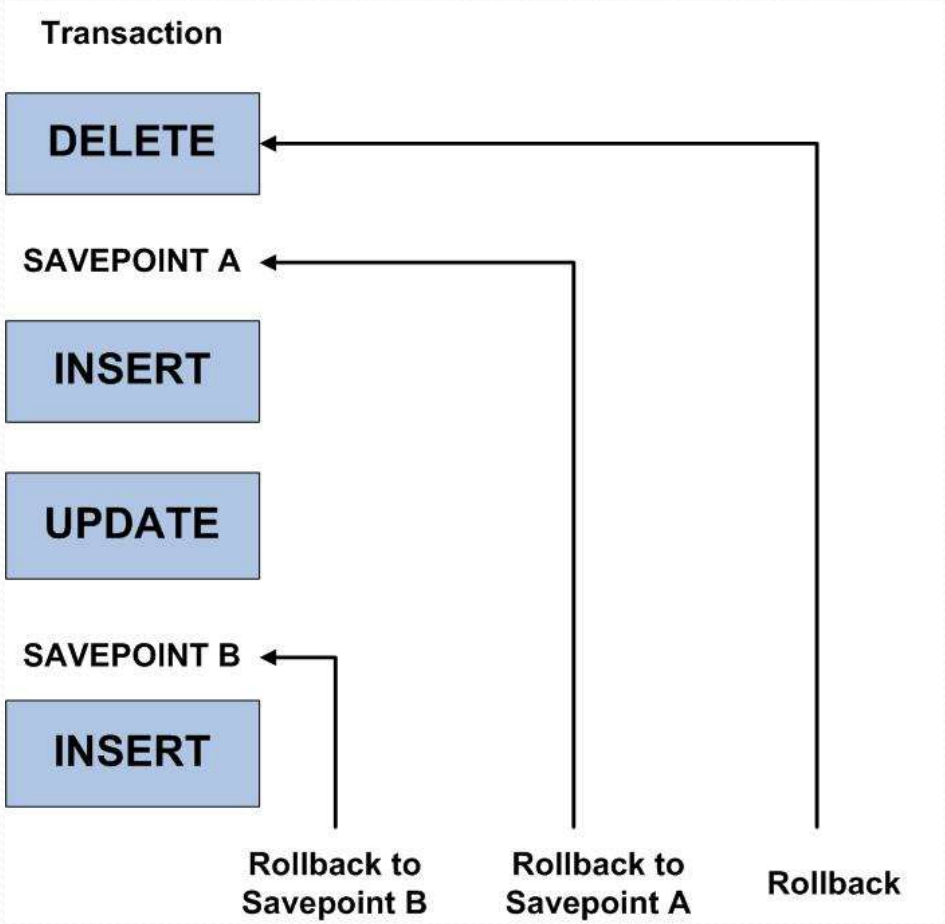
- **A transaction begins with the first DML statement is executed.**
- **A transaction ends with one of the following events:**
  - **A COMMIT or ROLLBACK statement is issued**
  - **A DDL or DCL statement executes (automatic commit)**
  - **The user exits *i*SQL*Plus**
  - **The system crashes**

Advantages of COMMIT and ROLLBACK

With COMMIT and ROLLBACK statements, you can:

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

Controlling transaction



COMMIT transaction

Before COMMIT

- Generated rollback segment records in buffers in the SGA
- Generated redo log entries in the redo log buffer of the SGA.
- The changes have been made to the database buffers of the SGA.

After COMMIT

- The internal transaction table for the associated rollback segment records updated with SCN
- LGWR writes SGA redo log entries to the online redo log file
- Oracle releases locks
- Oracle marks the transaction complete.

ROLLBACK transaction

ROLLBACK

- Oracle undoes all transaction changes using the undo tablespace or rollback segments
- Oracle releases all the transaction's locks of data
- The transaction ends

ROLLBACK to SAVEPOINT

- Oracle rolls back only the statements run after the savepoint.
- Oracle preserves the specified savepoint, but all savepoints that were established after the specified one are lost
- Oracle releases all table and row locks acquired since that savepoint

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 *can not 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.
- All savepoints *are erased*.

State of the Data after ROLLBACK

- The previous state of the data *restored*.
- All users *can view* the results.
- Locks on the affected rows *are released*; those rows are available for other users to manipulate.
- All savepoints *are erased*.

AUTOCOMMIT

- Finally, can turn AUTOCOMMIT on:
  - Oralce: SQL> SET AUTOCOMMIT ON;
- Then each statement is auto-committed as its own transaction
  - Not just DDL statements

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