

MANIPULATING DATA

-- CHAPTER 9 --

- A Data Manipulation Language (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 is accomplished using the INSERT statement

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

- Because you can insert a new row that contains values for each column, the column list is not required in the INSERT clause. However, if you do not use the column list, the values must be listed according to the default order of the columns in the table.
 - You can insert NULL values by simply omitting the column value, or by specifying either (“”) or NULL as the item to be inserted.
- Here’s an example of an insert...

```
SQL> insert into emp
  2  values (2296, 'AROMANO', 'SALESMAN', 7782,
  3  TO_DATE('FEB 3, 1997', 'MON DD, YYYY'),
  4  1300, NULL, 10);

1 row created.
```

- Note that the TO_DATE() function formats the string into a DATE datatype.
- Creating a Script with Customized Prompts
 - ACCEPT stores the value in the variable
 - PROMPT displays your customized text.
- Let’s look at the following example...first, we create the following script and save it with the following name scriptsWithCustomizedPrompts.sql:

```
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 dept (deptno, dname, loc)
VALUES (&department_id, '&department_name', '&location');
```

- We then run the script using the START keyword...

```

SQL> START scriptsWithCustomizedPrompts
Please enter the          department number:90
Please enter          the department name:PAYROLL
Please enter the          location:HOUSTON
old  2: VALUES (&department_id, '&department_name',
'&location')
new  2: VALUES (90, 'PAYROLL', 'HOUSTON')

1 row created.

```

- Notice that the ACCEPT keyword allows us to accept the value entered by the user and to store it in the variable name that follows it, which, by the way, does not require the substitution parameter (&). However, when we do make use of its contents in the INSERT statement, we must include the ampersand! When the script is run, the user is asked to provide the values for all three variables here defined: department_id, department_name, and location.
- Copying Rows from another Table: Write your INSERT statement with a subquery. Do not use the VALUES clause. Match the number of columns in the INSERT clause to those in the subquery.

```

SQL> create table managers(id number(4), name varchar2(10), salary
number(7,2), hiredate date)
2 /

```

Table created.

```

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

```

3 rows created.

```

SQL> select *
2 from managers
3 /

```

| ID | NAME | SALARY | HIREDATE |
|------|-------|--------|-----------|
| 7566 | JONES | 2000 | 02-APR-81 |
| 7698 | BLAKE | 2000 | 01-MAY-81 |
| 7782 | CLARK | 2000 | 09-JUN-81 |

- For changing date in a table, we make use of the UPDATE statement...

```

SQL> update managers
2 set id = 3434
3 where name=' JONES';

```

1 row updated.

```

SQL> select *
2 from managers;

```

| ID | NAME | SALARY | HIREDATE |
|------|-------|--------|-----------|
| 3434 | JONES | 2000 | 02-APR-81 |
| 7698 | BLAKE | 2000 | 01-MAY-81 |
| 7782 | CLARK | 2000 | 09-JUN-81 |

- Notice that we make use of the keyword SET to modify the contents of a particular row. In this particular case, we updated only one row by specifically singling out the record for JONES using the WHERE clause. However, if left out, the UPDATE statement would have modified all rows. Note that if ID is a primary key, the update would fail since no two rows may have the same value. It is, however, possible to modify all ID values at once, so as long as the update statement allows for all of them to be different.
- Updating with Multiple-Column Subquery

```
SQL> UPDATE managers
  2  SET (name, salary) =
  3  (select ename, sal
  4  from emp
  5  where empno = 7698)
  6  WHERE ID = 7782;
```

1 row updated.

```
SQL> select * from managers;
```

| ID | NAME | SALARY | HIREDATE |
|------|-------|--------|-----------|
| 3434 | JONES | 2000 | 02-APR-81 |
| 7698 | BLAKE | 2000 | 01-MAY-81 |
| 7782 | BLAKE | 2000 | 09-JUN-81 |

- Note here that the name CLARK (bottom) was changed to BLAKE and the salary seems to have remained unchanged since 2000 was replaced by 2000...however, the value was technically modified. Note that the information used to modify the managers table came from table emp.
- Removing a row from a table: Use the DELETE statement as follows:

```
SQL> delete from managers
  2  where id = 7782;
```

1 row deleted.

```
SQL> select * from managers;
```

| ID | NAME | SALARY | HIREDATE |
|------|-------|--------|-----------|
| 3434 | JONES | 2000 | 02-APR-81 |
| 7698 | BLAKE | 2000 | 01-MAY-81 |

- Again, it is possible to delete rows based on the contents of a second table by using subqueries. This is likely a useful procedure.
- 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
- Transactions consist of DML statements that make up one consistent change to the date. For example, a transfer of funds between two accounts should include the debit to one account and the credit to another account in the same amount. Both actions should either fail or succeed together. The credit should not be committed without the debit.
- Database Transactions: Begin when the first executable SQL statement is executed and end with one of the following events: Commit or Rollback is issued.
- Controlling Transactions: You can control the logic of transaction by using the COMMIT, SAVEPOINT, and ROLLBACK statements.
 - COMMIT - Ends the current transaction by making all pending data changes permanent.
 - SAVEPOINT name – Marks a savepoint within the current transaction
 - ROLLBACK [TO SAVEPOINT name] – ROLLBACK ends the current transaction by discarding all pending data changes; ROLLBACK TO SAVEPOINT rolls back the current transaction to the specified savepoint, thereby discarding the savepoint and any subsequent changes. If you omit this clause, the ROLLBACK statement rolls back to the entire transaction.
- 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.
- Committing Changes: Every data change made during the transaction is temporary until the transaction is committed. Other users cannot view the results of the data manipulation operations made by the current user. The Oracle Server institutes read consistency to ensure that each user sees data as it existed at a the last commit.
- 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: 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.
- Statement-Level-Rollback
 - If a single DML statement fails during execution, only the 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
- Implementation of Read Consistency: Read consistency is an automatic implementation. It keeps a partial copy of the database in rollback segments. When insert, update, or delete operation is made to the database, the Oracle Server takes a copy of the data before it is changed and writes it to a rollback segment. All readers, except the one who issued the change, still see the database as it existed before the changes started; they view the rollback segments' "snapshot" of the data.
- 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 – Prevents a resource from being shared. The first transaction to lock a resource exclusively, is the only transaction that can alter the resource until the exclusive lock is released.
 - Share – Allows the resource to be shared. Multiple users reading data can share the data, holding share locks to prevent concurrent access by a writer (who needs an exclusive lock) . Several transactions can acquire share locks on the same resource.