USING MANIPULATING DATA

Oracle provide Data Manipulation Language commands to exercise data operations in the database.Data operations can be populating the database tables with the application or business data,modifying the data and removing the data from the database,whenever required. Besides the data operations,there are set of commands which are used to control these operations.These commands are grouped as Transaction Control Language.

There are three types of DML statements involved in a logical SQL transaction namely, Insert, Update, Delete and Merge.A transaction is the logical collection of DML actions within a database session.

INSERT statement

The INSERT command is used to store data in tables. The INSERT command is often used in higher-level programming languages such as Visual Basic.NET or C++ as an embedded SQL command; however,this command can also be executed at the SQL*PLUS prompt in command mode.There are two different forms of the INSERT command. The first form is used if a new row will have a value inserted into each column of the row. The second form of the INSERT command is used to insert rows where some of the column data is unknown or defaulted from another business logic.This form of the INSERT command requires that you specify column names for which data are being stored.

Syntax:

The below syntax can be followed if the values for all the columns in the table is definite and known.

```
INSERT INTO table
VALUES (column1 value, column2 value,
...);
```

The below syntax can be used if only few columns from the table have to be populated with a value. Rest of the columns can deduce their values either as NULL or from a different business logic.

```
INSERT INTO table (column1 name, column2 name, . . .)
VALUES (column1 value, column2 value, . . .);
```

The INSERT statement below creates a new employee record in the EMPLOYEES table. Note that it inserts the values for the primary columns EMPLOYEE_ID, FIRST_NAME, SALARY and DEPARTMENT_ID.

```
INSERT INTO employees (EMPLOYEE_ID, FIRST_NAME, SALARY, DEPARTMENT_ID)
VALUES (130, 'KEMP', 3800, 10);
```

Otherwise, complete employee data can be inserted in the EMPLOYEES table without specifying the column list using the below INSERT statement - provided the values are known beforehand and must be in compliance with the data type and position of columns in the table.

```
INSERT INTO employees
VALUES (130, 'KEMP','GARNER', 'kemp.garner@xxx.com', '48309290',TO_DATE ('01-JAN-2012'),
'SALES', 3800, 0, 110, 10);
```

Values to be inserted must be compatible with the data type of the column. Literals, fixed values and special values like functions, SYSDATE, CURRENT_DATE, SEQ.CURRVAL $NEXTVAL$, or USER can be used as column values. Values specified must follow the generic rules. String literals and date values must be enclosed within quotes. Date value can be supplied in DD-MON-RR or D-MON-YYYY format, but YYYY is preferred since it clearly specifies the century and does not depend on internal RR century calculation logic.

INSERT-AS-SELECT *IAS* statement

Data can be populated into the target table from the source table using INSERT..AS..SELECT *IAS* operation. Its a direct path read operation.Its a simple way of creating copy of the data from one table to another or creating a backup copy of the table which the source table operations are online.

For example, data can be copied from EMPLOYEES table to EMP_HISTORY table.

```
INSERT INTO EMP_HISTORY
SELECT EMPLOYEE_ID, EMPLOYEE_NAME, SALARY, DEPARTMENT_ID
FROM employees;
```

UPDATE statement

The UPDATE command modifies the data stored in a column.It can update single or multiple rows at a time depending on the result set filtered by conditions specified in WHERE clause. Note that Updating columns is different from altering columns. Earlier in this chapter, you studied the ALTER command.The ALTER command changes the table structure, but leaves the table data unaffected.The UPDATE command changes data in the table, not the table structure.

Syntax:

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

From the syntax,

The SET column = expression can be any combination of characters, formulas, or functions that will update data in the specified column name.The WHERE clause is optional, but if it is included, it specifies which rows will be updated.Only one table can be updated at a time with an UPDATE command.

The UPDATE statement below updates the salary of employee JOHN to 5000.

```
UPDATE employees
SET salary = 5000
WHERE UPPER (first_name) = 'JOHN';
```

Though WHERE predicates are optional, but must be logically appended so as to modify only the required row in the table. The UPDATE statement below updates the salaries of all the employees in the table.

```
UPDATE employees
SET salary = 5000;
```

Multiple columns can also be updated by specifying multiple columns in SET clause separated by a comma. For example, if both salary and job role has to be changed to 5000 and SALES respectively for JOHN, the UPDATE statement looks like,

```
UPDATE employees
SET SALARY = 5000,
 JOB_ID = 'SALES'
WHERE UPPER (first_name) = 'JOHN';

1 row updated.
```

Another way of updating multiple columns of the same row shows the usage of subquery.

```
UPDATE employees
SET (SALARY, JOB_ID) = (SELECT 5000, 'SALES' FROM DUAL)
WHERE UPPER (ENAME) = 'JOHN'
```

DELETE statement

The DELETE command is one of the simplest of the SQL statements. It removes one or more rows from a table. Multiple table delete operations are not allowed in SQL.The syntax of the DELETE command is as below.

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

The DELETE command deletes all rows in the table that satisfy the condition in the optional WHERE clause. Since the WHERE clause is optional, one can easily delete all rows from a table by omitting a WHERE clause since the WHERE clause limits the scope of the DELETE operation.

The below DELETE statement would remove EDWIN's details from EMP table.

```
DELETE employees
WHERE UPPER (ENAME) = 'EDWIN'

1 row deleted.
```

Note: DELETE [TABLE NAME] and DELETE FROM [TABLE NAME] hold the same meaning.

The WHERE condition in the conditional delete statements can make use of subquery as shown below.

```
DELETE FROM employees
WHERE DEPARTMENT_ID IN (SELECT DEPARTMENT_ID
        FROM LOCATIONS
        WHERE LOCATION_CODE = 'SFO')
```

TRUNCATE

Truncate is a DDL command which is used to flush out all records from a table but retaining the table structure. It does not supports WHERE condition to remove the selected records.

Syntax:

```
TRUNCATE [table name]
```

It is Auto Commit i.e. it commits the current active transaction in the session. Truncating the table does not drops dependent indexes, triggers or table constraints. If a table A is parent of a reference constraint of a table B in the database, the table A could not be truncated.

Transaction

A transaction is a logical unit of work done in database. It can either contain -

- Multiple DML commands ending with a TCL command i.e. COMMIT or ROLLBACK
- One DDL command
- One DCL command

Beginning of a transaction is marked with the first DML command. It ends with a TCL, DDL or DCL command. A TCL command i.e. COMMIT or ROLLBACK is issues explicitly to end an active transaction. By virtue of their basic behavior, if any of DDL or DCL commands get executed in a database session, commit the ongoing active transaction in the session. If the database instance crashes abnormally, the transaction is stopped.

COMMIT, ROLLBACK and SAVEPOINT are the transaction control language. COMMIT applies the data changes permanently into the database while ROLLBACK does anti-commit operation. SAVEPOINT controls the series of a transaction by setting markers at different transaction stages. User can roll back the current transaction to the desired save point, which was set earlier.

COMMIT - Commit ends the current active transaction by applying the data changes permanently into the database tables. COMMIT is a TCL command which explicitly ends the transaction. However, the DDL and DCL command implicitly commit the transaction.

SAVEPOINT - Savepoint is used to mark a specific point in the current transaction in the session. Since it is logical marker in the transaction, savepoints cannot be queried in the data dictionaries.

ROLLBACK - The ROLLBACK command is used to end the entire transaction by discarding the data changes. If the transaction contains marked savepoints, ROLLBACK TO SAVEPOINT [name] can be used to rollback the transaction upto the specified savepoint only. As a result, all the data changes upto the specified savepoint will be discarded.

Demonstration

Consider the EMPLOYEES table which gets populated with newly hired employee details during first quarter of every year. The clerical staff appends each employee detail with a savepoint, so as to rollback any faulty data at any moment during the data feeding activity. Note that he keeps the savepoint names same as the employee names.

```
INSERT INTO employees (employee_id, first_name, hire_date, job_id, salary,
department_id)
VALUES (105, 'Allen',TO_DATE ('15-JAN-2013','SALES',10000,10);

SAVEPOINT Allen;

INSERT INTO employees (employee_id, first_name, hire_date, job_id, salary,
department_id)
VALUES (106, 'Kate',TO_DATE ('15-JAN-2013','PROD',10000,20);

SAVEPOINT Kate;

INSERT INTO employees (employee_id, first_name, hire_date, job_id, salary,
department_id)
VALUES (107, 'McMan',TO_DATE ('15-JAN-2013','ADMIN',12000,30);

SAVEPOINT McMan;
```

Suppose, the data feeding operator realises that he has wrongly entered the salary of 'Kate' and 'McMan'. He rolls back the active transaction to the savepoint Kate and re-enters the employee details for Kate and McMan.

```
ROLLBACK TO SAVEPOINT Kate;

INSERT INTO employees (employee_id, first_name, hire_date, job_id, salary,
department_id)
VALUES (106, 'Kate',TO_DATE ('15-JAN-2013','PROD',12500,20);

SAVEPOINT Kate;

INSERT INTO employees (employee_id, first_name, hire_date, job_id, salary,
department_id)
VALUES (107, 'McMan',TO_DATE ('15-JAN-2013','ADMIN',13200,30);

SAVEPOINT McMan;
```

Once he is done with the data entry, he can commit the entire transaction by issuing COMMIT in the current session.

Read Consistency

Oracle maintains consistency among the users in each session in terms of data access and read/write actions.

When a DML occurs on a table, the original data values changed by the action are recorded in the database undo records. As long as transaction is not committed into database, any user in other session that later queries the modified data views the original data values. Oracle uses current

information in the system global area and information in the undo records to construct a read-consistent view of a table's data for a query. Only when a transaction is committed, the changes of the transaction made permanent. The transaction is the key to Oracle's strategy for providing read consistency.

Start point for read-consistent views is generated on behalf of readers

Controls when modified data can be seen by other transactions of the database for reading or updating