Oracle SQL Commands Part - 3

Oracle Database

INTRODUCTION

SQL is divided into the following

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Data Retrieval Language (DRL)
4. Transaction Control Language (TCL)
5. Data Control Language (DCL)

DDL

- create
- alter
- drop
- truncate
- rename

DML

- insert
- update
- delete

DRL

- select

TCL

- commit
- rollback
- savepoint

DCL

- grant
- revoke

CREATE USER

Syntax

```
SQL> create user <user__name_> identified by <password>;

SQL> grant connect, resource, dba to user_name;
```

CREATE TABLE

Syntax

```
Create table <_table_name_> (_col1 datatype1, col2 datatype2,…coln datatypen_);
```

Example

```
SQL> create table student (no number (2), name varchar (10), marks number (3));
```

USING - `INSERT`

Insert command is used to insert the records into a table. We have two methods to do that -

1. Insert by value method
2. Insert by address method

USING THE VALUE METHOD

Syntax

```
insert into <_table_name_> values (_value1, value2, value3, …Valuen_);
```

Example

```
SQL> insert into student values (1, 'Shaheed', 100);
```

INSERTING DATA INTO SPECIFIED COLUMNS USING VALUE METHOD

Syntax

```
insert into <_table_name_>(_col1, col2, col3 … Coln_) values (_value1, value2, value3 …valuen_);
```

Example

```
SQL> insert into student (no, name) values (3, 'Yussef');

SQL> insert into student (no, name) values (4, 'Selamat');
```

USING THE ADDRESS METHOD

To insert a new record again, you need to type the entire insert command. If there are a lot of records, this process can become difficult. To simplify this, you can use the address method.

Syntax

```
insert into <_table_name_> values _(&col1, &col2, &col3 …. &coln_);
```

This will prompt you for the values, but for every insert operation, you have to use a forward slash.

Example

```
SQL> insert into student values (&no, '&name', &marks);
Enter value for no: 1
Enter value for name: Zaheen
Enter value for marks: 300
old 1: insert into student values(&no, '&name', &marks)
new 1: insert into student values(1, 'Zaheen', 300)
SQL> /
Enter value for no: 2
Enter value for name: Nazim
Enter value for marks: 400
old 1: insert into student values(&no, '&name', &marks)
new 1: insert into student values(2, 'Nazim', 400)
SQL> /
```

INSERTING DATA INTO SPECIFIED COLUMNS USING ADDRESS METHOD

Syntax

```
insert into <_table_name_>(_col1, col2, col3 … coln_) values _(&col1, &col2, … &coln_);
```

This will prompt you for the values, but for every insert, you have to use a forward slash.

Example

```
SQL> insert into student (no, name) values (&no, '&name');
Enter value for no: 5
Enter value for name: Vinnie
old 1: insert into student (no, name) values(&no, '&name')
new 1: insert into student (no, name) values(5, 'Vinnie')
SQL> /
Enter value for no: 6
Enter value for name: Rajesh
old 1: insert into student (no, name) values(&no, '&name')
new 1: insert into student (no, name) values(6, 'Rajesh')
SQL> /
```

`SELECT` OPERATION

Syntax

```
select * from <_table_name_>; {-- here * indicates all columns}
```

or

```
select    _col1, col2, … coln_    from <    _table_name_    >;
```

Example

Command:

```
SQL> select * from student;
```

Output:

```
NO  NAME     MARKS
--- ------   --------
 1  Shaheed  100
 4  Selamat  200
 1  Zaheen   300
 2  Nazim    400
```

Command:

```
SQL> select no, name, marks from student;
```

Output:

```
NO  NAME     MARKS
--- ------   --------
 1  Shaheed  100
```

```
4  Selamat  200
1  Zaheen   300
2  Nazim    400
```

Command:

```
SQL> select no, name from student;
```

Output:

```
NO  NAME
--- -------
1   Shaheed
4   Selamat
1   Zaheen
2   Nazeem
```

CONDITIONAL SELECTIONS AND OPERATORS

There are two clause that are used for conditional operations:

1. *WHERE*
2. *ORDER BY*

USING - `WHERE`

Syntax

```
select * from <_table_name_> where <_condition_>;
```

The following are the different types of operators used in the ***WHERE*** clause:

1. Arithmetic operators
2. Comparison operators
3. Logical operators

Arithmetic operators (highest precedence):

```
+, -, *, /
```

Comparison operators:

```
1. =, !=, >, <, >=, <=, <>
2. between, not between
3. in, not in
4. null, not null
5. like
```

Logical operators:

```
And
Or
not
```

USING - `"=, >, <, >=, <=, !=, <>"`

Example

```
SQL> select * from student where no = 2;
NO   NAME   MARKS
---  ------- ---------
2    Nazim   400

SQL> select * from student where no < 2;
NO   NAME   MARKS
---  ------- ---------
1    Shaheed  100
1    Zaheen  300

SQL> select * from student where no > 2;
NO   NAME   MARKS
---  ------- ---------
3    Yussef
4    Selamat
5    Vinnie
6    Rajesh

SQL> select * from student where no <= 2;
NO   NAME   MARKS
---  ------- ---------
1    Shaheed  100
1    Zaheen   300
2    Nazeem   400

SQL> select * from student where no >= 2;
NO   NAME   MARKS
---  ------- ---------
2    Naren   400
3    Yussef
4    Selamat
5    Vinnie
```

```
6    Rajesh
SQL> select * from student where no != 2;
NO   NAME   MARKS
---  ------- ---------
1    Shaheed  100
1    Zaheen  300

SQL> select * from student where no <> 2;
NO   NAME   MARKS
---  ------- ---------
1    Shaheed  100
1    Zaheen  300
```

USING - `AND`

This yields an output when all the conditions become true.

Syntax

```
select * from <_table_name_> where <_condition1_> and <_condition2_> and .. <_conditionn_>;
```

Example

```
SQL> select * from student where no = 2 and marks >= 200;

 NO NAME MARKS
--- ------- --------
 2 Nazim 400
```

USING - `OR`

This yields an output when either of the condition

becomes true.

Syntax

```
select * from <_table_name_> where <_condition1_> and <_condition2_> or.. <_condition_>;
```

Example

```
SQL> select * from student where no = 2 or marks >= 200;

 NO NAME MARKS
--- ------- ---------

 1 Zaheen 300

 2 Nazim 400
```

USING - `BETWEEN`

This yields an output based on a column's lower and upper bound.

Syntax

```
select * from <_table_name_> where <_col_> between <_lower_
_bound_> and <_upper bound_>;
```

Example

```
SQL> select * from student where marks between 200 and 400;

NO NAME MARKS

 --- ------- ---------
 1 Zaheen 300

 2 Nazim 400
```

USING - `NOT BETWEEN`

This yields an output based on the column which values are not in the range between the bounds.

Syntax

```
select * from <_table_name_> where <_col_> not between <_lower bound_> and <_upper bound_>;
```

Example

```
SQL> select * from student where marks not between 200 and 400;

 NO  NAME    MARKS
---- ------- ---------
 1   Shaheed  100
```

USING - `IN`

This yields an output based on the column and its list of values specified.

Syntax

```
select * from <_table_name_> where <_col_> in (_value1,value2, value3…valuen_);
```

Example

```
SQL> select * from student where no in (1, 2, 3);

 NO NAME MARKS

 --- ------- ---------

 1 Shaheed 100

 1 Zaheen 300

 2 Nazim  400
```

USING - `NOT IN`

This yields an output based on the column which values are not in the list of values specified.

Syntax

```
select * from <_table_name_> where <_col_> not in (_value1,value2, value3…valuen_);
```

Example

```
SQL> select * from student where no not in (1, 2, 3);

 NO NAME MARKS
 --- ------- ---------
```

```
     4 Selamat

     5 Vinnie

     6 Rajesh
```

USING - `NULL`

This yields an output based on the null values in the specified column.

Syntax

```
select * from <_table_name_> where <_col_> is null;
```

Example

```
SQL> select * from student where marks is null;

 NO NAME MARKS
 --- ------- ---------

     3 Yussef

     4 Selamat

     5 Vinnie

     6 Rajesh
```

USING - `NOT NULL`

This yields an output based on the not null values in the specified column.

Syntax

```
select * from <_table_name_> where <_col_> is not null;
```

Example

```
SQL> select * from student where marks is not null;

 NO NAME MARKS
```

```
--- ------- ---------
1 Shaheed 100
1 Zaheen 300
2 Nazim 400
```

USING - `LIKE`

This is used to search through the rows of database column based on the pattern you specify.

Syntax

```
select * from <_table_name_> where <_col_> like <_pattern_>;
```

Example

---

1. The following will return the rows where marks is 100.

```
SQL> select * from student where marks like 100;

NO NAME MARKS
--- ------- ---------
1 Shaheed 100
```

2. The following will return the rows where name starts with ‘S’

```
SQL> select * from student where name like 'S%';

 NO  NAME     MARKS
--- -------  ---------
1   Shaheed   100
4   Selamat   null
```

3. The following will return the rows where name ends with ‘h’.

```
SQL> select * from student where name like '%h'

NO  NAME    MARKS
--- ------- ---------
6   Rajesh  null
```

4. The following will return the rows where name’s second letter is ‘a’.

```
SQL> select * from student where name like '_a%';
```

```
NO   NAME    MARKS
---  ------- --------
1    Zaheen  300

2    Nazim   400

6    Rajesh  null
```

5. The following will return the rows where name's third letter is 'j'.

```
SQL> select * from student where name like '__j%';

NO NAME    MARKS
-- ------  -----
6  Rajesh  null
```

6. The following will return the rows where name's second last letter is 'e'.

```
SQL> select * from student where name like '%_e%';
NO  NAME    MARKS
--- ------- ---------
1   Shaheed 100
1   Zaheen  300
```

7. The following will return the rows where name's third last letter is 'e'

```
SQL> select * from student where name like '%e__%';
NO  NAME    MARKS
--- ------- ---------
1   Shaheed 100
1   Zaheen  300
```

8. The following will return the rows where name contains 2 consecutive e's.

```
SQL> select * from student where name like '%ee%';
NO  NAME    MARKS
--- ------- ---------
1   Shaheed 100
1   Zaheen  300
```

You have to specify the patterns in `like` using underscore `_`

USING - `ORDER BY`

1. It's used to order the data according to a specified column (ascending or descending).
2. Oracle uses ascending order by default.
3. In order to use descending order we need to use *`desc`* keyword after the column.

Syntax

```
select * from <_table_name_> order by <_col_> desc;
```

Example

```
SQL> select * from student order by no
NO  NAME      MARKS
--- -------   ---------
1   Shaheed   100
1   Zaheen    300
2   Nazim     400
3   Yussef    null
4   Selamat   null
5   Vinnie    null
6   Rajesh    null
```

Example - Using `desc` keyword

```
SQL> select * from student order by no desc;

 NO  NAME      MARKS
 --- -------   ---------
 6   Rajesh    null
 5   Vinnie    null
 4   Selamat   null
 3   Yussef    null
 2   Nazim     400
 1   Zaheen    300
 1   Shaheed   100
```

USING - `UPDATE`

It's used in order to modify data in a table

Syntax

```
update <_table_name_> set <_col1_> = value1, <_col2_> = value2 where <_condition_>;
```

Example

```
SQL> update student set marks = 500;
```

N.B. *If you are not specifying any condition this will update all the data in the* `marks` *column to* `500`

```
SQL> update student set marks = 500 where no = 2;
SQL> select * from student order by no
NO  NAME      MARKS
--- -------   ---------
1   Shaheed   100
1   Zaheen    300
2   Nazim     500
3   Yussef    null
4   Selamat   null
5   Vinnie    null
6   Rajesh    null

SQL> update student set marks = 200, name = 'Shaeed' where no = 1;
SQL> select * from student order by no
NO  NAME      MARKS
--- -------   ---------
1   Shaheed   200
1   Zaheen    300
2   Nazim     500
3   Yussef    null
4   Selamat   null
5   Vinnie    null
6   Rajesh    null
```

USING - `DELETE`

Syntax

```
delete <_table_name_> where <_condition_>;
```

Example

```
SQL> delete student;
```

If you do not specify any condition this will delete the entire table. So, doing the following would be more appropriate

```
SQL> delete student where no = 2;
SQL> select * from student order by no
NO  NAME      MARKS
--- -------   ---------
1   Shaheed   200
1   Zaheen    300
3   Yussef    null
4   Selamat   null
5   Vinnie    null
6   Rajesh    null
```

USING `ALTER`

This is used to add or remove columns and to modify the precision of the datatype.

ADDING COLUMN - `ADD`

Syntax

```
alter table <_table_name_> add <_col datatype_>;
```

Example

```
SQL> alter table student add sdob date;
```

REMOVING COLUMN - `DROP`

Syntax

```
alter table <_table_name_> drop <_col datatype_>;
```

Example

```
SQL> alter table student drop column sdob;
```

INCREASING OR DECREASING PRECISION OF A COLUMN - `MODIFY`

Syntax

```
alter table <_table_name_> modify <_col datatype_>;
```

Example

```
SQL> alter table student modify marks number(5);
```

MAKING COLUMN - `UNUSED`

The main advantage of setting a column to UNUSED is **to reduce possible high database load when dropping a column from a large table**. To overcome this issue, a column can be marked as unused and then be physically dropped later. To set a column as unused, use the SET UNUSED clause.

Syntax

```
alter table <_table_name_> set unused column <_col_>;
```

Example

```
SQL> alter table student set unused column marks;
```

(Even though the column is unused, it does occupy memory)

DROPPING UNUSED COLUMNS - `DROP UNUSED`

Syntax

```
alter table <_table_name_> drop unused columns;
```

Example

```
SQL> alter table student drop unused columns;
```

(You cannot drop unused columns of a table by addressing them individually, you either drop'em all, or you don't)

RENAMING COLUMN - `RENAME COLUMN`

Syntax

```
alter table <_table_name_> rename column <_old_col_name_> to <_new_col_name_>;
```

Example

```
SQL> alter table student rename column marks to smarks;
```

USING - `TRUNCATE`

- This is used to delete the entire table data permanently

Syntax

```
truncate table <_table_name_>;
```

Example

```
SQL> truncate table student;
```

USING - `DROP`

- This is used to drop the database object

Syntax

```
drop table <_table_name_>;
```

Example

```
SQL> drop table student;
```

USING - `RENAME`

This is be used to rename the database object

Syntax

```
SQL> rename <old__table_name_> to <_new_table_name_>;
```

Example

```
SQL> rename student to stud;
```

End