SQL COMMANDS

SQL: Data Manipulation Language (DML)

- **Data Manipulation Language (DML)** commands are used **for accessing and manipulating the data stored in the database.**
- The DML commands are *not auto-committed* that means it can't permanently save all the changes in the database. They can be rollback.
- **Data Manipulation Language (DML)**
  - **SELECT** - to retrieve data from the database ⟸ **Data Query language (DQL)**
  - **INSERT** - to insert a new row into a table
  - **UPDATE** - to update existing row in a table — **Modification of Database**
  - **DELETE** - to delete a row from a table

SQL: Data Manipulation Language (DML)

- **Data Manipulation Language (DML)** commands are used **for accessing and manipulating the data stored in the database.**
- The DML commands are *not auto-committed* that means it can't permanently save all the changes in the database. They can be rollback.
- **Data Manipulation Language (DML)**
  - **SELECT** - to retrieve data from the database ⟸ **Data Query language (DQL)**
  - **INSERT** - to insert a new row into a table
  - **UPDATE** - to update existing row in a table — **Modification of Database**
  - **DELETE** - to delete a row from a table

Programmers call these DML operations "CRUD".
CRUD** stands for:* ***Create, Read, Update, Delete
Performed** by:* ***INSERT, SELECT, UPDATE, DELETE

Insert Statements

- Insert command is used to Insert data/record into the database table
- Inserting values for the specific columns in the table. (Always include the columns which are not null)

```
Insert Into <Table-Name> (Fieldname1, FielHR2, Fieldname3,..) Values (value1,
value2,value3,..);
Example:
SQL> insert into dept (deptno,dept_name)values(20,'HR');
If we want to insert values in all columns then no need to specify column values , but
order of column values should be in sync with the column names.
Example:
SQL> insert into dept  values(20,'HR');
```

Insert Statements

```
SQL>  DESC EMPLOYEE;
 Name                      Null?    Type
 ----------------------- -------- ------------
 EMPID                              NUMBER(4)
 NAME                               VARCHAR2(20)
 BIRTHDATE                          DATE
```

```
SQL> INSERT INTO EMPLOYEE
  2  (EMPID,NAME,BIRTHDATE)
  3  VALUES (129,'VIRAT','29-SEP-02');

1 row created.
```

```
SQL> SELECT * FROM EMPLOYEE;

     EMPID NAME                 BIRTHDATE
---------- -------------------- ---------
       230 SACHIN
       340 RAKESH               20-JUN-08
       129 VIRAT                29-SEP-02
```

```
SQL> INSERT INTO EMPLOYEE
  2  VALUES(340,'RAKESH','20-JUN-08');

1 row created.
```

Insert Statements

INSERT ALL statement is used to add multiple rows with a single INSERT statement. The rows can be inserted into one table or multiple tables using only one SQL command.

Syntax

```
INSERT ALL
  INTO mytable (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
  INTO mytable (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
  INTO mytable (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
SELECT * FROM dual;
```

```
SQL> INSERT ALL
  2  INTO EMPLOYEE (EMPID,NAME,BIRTHDATE) VALUES (104,'HARSH','14-NOV-77')
  3  INTO EMPLOYEE (EMPID,NAME,BIRTHDATE) VALUES (105,'VANSH','28-DEC-99')
  4  SELECT * FROM DUAL;

2 rows created.
```

```
SQL> SELECT * FROM EMPLOYEE;

     EMPID NAME                 BIRTHDATE
---------- -------------------- ---------
       230 SACHIN
       340 RAKESH               20-JUN-08
       129 VIRAT                29-SEP-02
       103 RAJ                  29-APR-02
       104 HARSH                14-NOV-77
       105 VANSH                28-DEC-99

6 rows selected.
```

‘Insert- As- Select’ Statement

‘Insert – As –Select’ statement allows to insert into a table using the input from another table. Record from one table will be inserted in another table.

```
INSERT INTO table-name (column-names)
SELECT column-names
FROM table-name
WHERE condition
Syntax:
Insert into <new_table_name> (Select <columnnames> from <old_table_name>);
Example:
SQL> insert into CopyOfEmployee select * from employee where emp_id > 100
```

‘Insert- As- Select’ Statement

```
SQL> SELECT * FROM B1;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981001 DHVANI                18
   1981002 YAASMIN               19
   1981003 SAKSHI                20
   1981004 SAISH                 12
   1981005 NIDHI                 18

SQL> SELECT * FROM B2;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981031 VAISHNAVI             19
   1981032 JHENIL                19
   1981033 BHARGAVI              19
   1981034 ISHITA                20
   1981035 SAKSHI                18
   1981036 MISHITA               19
```

```
SQL> INSERT INTO CSE
  2  SELECT * FROM B1;

5 rows created.

SQL>
SQL> CREATE TABLE CSE
  2  (ROLLNO NUMBER(8),
  3  NAME VARCHAR2(12),
  4
SQL>
SQL> SELECT * FROM CSE;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981001 DHVANI                18
   1981002 YAASMIN               19
   1981003 SAKSHI                20
   1981004 SAISH                 12
   1981005 NIDHI                 18
```

Update Statements

- Update statement updates/modify the existing data in the tables. Using these statements we can update the value of a single column or multiple columns in a single statement
- Updating single column

Syntax:

```
UPDATE <table name> Set <Field Name> = <Value> Where <Condition>;
```

Note:1. Without where clause all the rows will get updated

2. Updating multiple column [while updating more than one column, the column must be separated by comma operator

```
Example: SQL> update dept set loc='Pink City', dept_name= 'HR' where deptno=20;
```

Update Statements

Note: *Always use the WHERE clause with the UPDATE statement to update the selected rows, otherwise all the rows would be affected.*

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ..... columnN = valueN
WHERE condition;
```

Update Statements

```
SQL> SELECT * FROM CSE;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981001 DHVANI                18
   1981002 YAASMIN               19
   1981003 SAKSHI                20
   1981004 SAISH                 12
   1981005 NIDHI                 18

SQL> UPDATE CSE
  2  SET NAME = 'KULSUM';
```

```
5 rows updated.

SQL> SELECT * FROM CSE;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981001 KULSUM                18
   1981002 KULSUM                19
   1981003 KULSUM                20
   1981004 KULSUM                12
   1981005 KULSUM                18
```

```
SQL>  SELECT * FROM B2;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981031 VAISHNAVI              19
   1981032 JHENIL                 19
   1981033 BHARGAVI               19
   1981034 ISHITA                 20
   1981035 SAKSHI                 18
   1981036 MISHITA                19
```

```
SQL> UPDATE B2
  2  SET NAME = 'SAKSHITA'
  3  WHERE ROLLNO = 1981035;
```

```
SQL> SELECT * FROM B2;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981031 VAISHNAVI              19
   1981032 JHENIL                 19
   1981033 BHARGAVI               19
   1981034 ISHITA                 20
   1981035 SAKSHITA               18
   1981036 MISHITA                19
```

```
SQL> UPDATE B2
  2  SET NAME = 'ZENIL',MARKS = 20
  3  WHERE ROLLNO = 1981031;

1 row updated.
```

```
SQL>  SELECT * FROM B2;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981031 ZENIL                 20
   1981032 JHENIL                19
   1981033 BHARGAVI              19
   1981034 ISHITA                20
   1981035 SAKSHITA              18
   1981036 MISHITA               19
```

Delete Statements

Delete commands helps to delete rows/record from database table. Delete Statements can be executed with or without where conditions. Execution of delete commands without where condition will remove all the records/rows from the table

Syntax:

```
Delete from <table name> [where <condition>];
```

```
Example:
a) to delete all rows (Deleting records without where condition):
SQL> delete from dept;
b) conditional deletion (deleting records with where condition):
SQL> delete from dept where loc='Pink City';
```

Delete Statements

- The **DELETE** statement is **used to delete existing records in a table.**
- **Note**: ***Always use the WHERE clause with a DELETE statement to delete the selected rows, otherwise all the records would be deleted.***

Syntax:

```
DELETE FROM table_name
WHERE condition;
```

Delete Statements

```
SQL> SELECT * FROM B1
  2  ;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981001 DHVANI                18
   1981002 YAASMIN               19
   1981003 SAKSHI                20
   1981004 SAISH                 12
   1981005 NIDHI                 18
```

```
SQL> DELETE FROM B1
  2  WHERE NAME = 'NIDHI';
```

```
SQL> SELECT * FROM B1;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981001 DHVANI                18
   1981002 YAASMIN               19
   1981003 SAKSHI                20
   1981004 SAISH                 12
```

```
SQL> DELETE FROM B1;

4 rows deleted.

SQL> SELECT * FROM B1;

no rows selected
```

Select Statement

- **SELECT** statement is **used to select a set of data from a database table.** Or Simply **SELECT** statement is **used to retrieve data from a database table.**
  - It **returns** data in the form of a **result table.** These result tables are called **result-sets.**
- **SELECT** is also called **DQL** because it is used to **query** information from a database table
- **SELECT** statement specifies **column names**, and **FROM** specifies **table name**
- **SELECT** command is used with different **Conditions** and **CLAUSES**.

Basic Syntax:

To retrieve selected fields from the table:

```
SELECT column1, column2, ... columnN
FROM table_name(s);
```

To retrieve all the fields from the table:

```
SELECT *
FROM table_name(s);
```

Select Statement

Example: SELECT

Example 1:

```
SQL> SELECT * FROM B2;

    ROLLNO NAME                 MARKS
---------- ---------- ----------
   1981031 ZENIL                20
   1981032 JHENIL               19
   1981033 BHARGAVI             19
   1981034 ISHITA               20
   1981035 SAKSHITA             18
   1981036 MISHITA              19

6 rows selected.
```

Example 2:

```
SQL> SELECT NAME, MARKS FROM B2;

NAME                 MARKS
---------- ----------
ZENIL                20
JHENIL               19
BHARGAVI             19
ISHITA               20
SAKSHITA             18
MISHITA              19

6 rows selected.
```

Select Statement

Using Alias name for a field:

Syntax:

```
Select <col1> <alias name 1> , <col2> < alias name 2> from < tab1>;
```

```
SQL> SELECT NAME, MARKS AS FCN FROM B2;

NAME                    FCN
------------ ----------
ZENIL                    20
JHENIL                   19
BHARGAVI                 19
ISHITA                   20
SAKSHITA                 18
MISHITA                  19
```

WHERE CLAUSE

- **WHERE Clause** is used **to select a particular record based on a condition**. It is used to filter records.
- **WHERE Clause is used to specify a condition while fetching the data from a single table or by joining with multiple conditions**
- The **WHERE** clause is not only used in the **SELECT** statement, but it is also used in the **UPDATE**, **DELETE** statement.
- ***Syntax: "SELECT with WHERE Clause"***

```
SELECT column1, column2, ... columnN
FROM table_name(s)
WHERE condition;
```

Optional Clauses in SELECT statemer

- **[WHERE Clause]** : It specifies which rows to retrieve by specifying **conditions.**
- **[GROUP BY Clause]** : **Groups rows** that share a **property** so that the **aggregate function** can be applied to each group.
- **[HAVING Clause]** : It **selects** among the **groups** defined by the **GROUP BY** clause by specifying **conditions. .**
- **[ORDER BY]** : It specifies an order in which to return the rows.

- **[DISTINCT Clause]:** It is used to remove duplicates from results set of a **SELECT** statement. (**SELECT DISTINCT**)

SQL Operators

1. SQL Arithmetic Operator
2. SQL Comparison Operators
3. SQL Logical Operators
4. SQL Special Operators

1. SQL Arithmetic Operators

Operator	Description
+	Add
–	Subtract
*	Multiply
/	Divide
%	Modulo

Example:

To retrieve *Emp_ID, Name, Salary plus 3000 from Emp* table

```
SELECT Emp_ID, Name, Salary + 3000
FROM Emp;
```

2. SQL Comparison Operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<> or !=	Not equal to

Example:

To retrieve *Emp_ID, Name* from *Emp* table whose *Salary* is greater than 2000

```
SELECT Emp_ID, Name, Salary
FROM Emp
WHERE Salary >2000;
```

Filtering: Logical Operators

Logical Operators (AND, OR and NOT)

Used in where conditions to join more than two queries. Used to combine the results of two or more conditions to produce single result

AND Logical Operator:

Used to combine two conditions and it fetches the result which satisfy both the conditions

```
SQL> SELECT ROLLNO, NAME, MARKS
  2  FROM B2
  3  WHERE MARKS>15 AND MARKS < 19;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981035 SAKSHITA              18
```

Logical Operator: OR

Logical Operators: OR

OR operators is used to combine two or more conditions and it fetches the result with satisfy any one of the condition in OR statements

```
SQL> SELECT ROLLNO, NAME, MARKS
  2  FROM B2
  3  WHERE MARKS = 20 OR MARKS <19;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981031 ZENIL                20
   1981034 ISHITA               20
   1981035 SAKSHITA             18
```

Logical Operator: NOT

Logical Operator: NOT

NOT operators is used to negate the conditions and it fetches opposite of the result with satisfy the condition.
It is used in combination with other keywords like NOT IN, NOT Between etc

```
SQL> SELECT * FROM B2
  2  WHERE MARKS NOT IN (14,17,19,20);

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981035 SAKSHITA              18
```

```
SQL> SELECT * FROM B2
  2  WHERE MARKS NOT BETWEEN 17 AND 19
  3  ;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981031 ZENIL                  20
   1981034 ISHITA                 20
```

3. SQL Logical Operators

Operator	Description
AND	It displays a record if all the conditions separated by AND are TRUE.
OR	It displays a record if any of the conditions separated by OR is TRUE
NOT	• It displays a record if the condition(s) is NOT TRUE. • It reverses the meaning of any operator with which it is used. This is a negate operator. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, IS NOT NULL, etc.

Example:

To retrieve all records from *Emp* table whose *salary* is greater than 2000 and age is less than 25

```
SELECT *
FROM Emp
WHERE Salary>2000 AND Age<25;
```

Ordering Result

ORDER BY

- Used along with where clause to display the specified column in ascending order or descending order
- Default is ascending order

Syntax:

```
SELECT [distinct] <column(s)>
FROM <table>
[ WHERE <condition> ]
[ ORDER BY <column(s) [asc|desc]>
```

```
SQL> SELECT ROLLNO, NAME, MARKS
  2  FROM B2
  3  ORDER BY MARKS;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981035 SAKSHITA              18
   1981036 MISHITA               19
   1981032 JHENIL                19
   1981033 BHARGAVI              19
   1981031 ZENIL                 20
   1981034 ISHITA                20

6 rows selected.
```

```
SQL> SELECT ROLLNO, NAME, MARKS
  2  FROM B2
  3  ORDER BY MARKS DESC;

    ROLLNO NAME                 MARKS
---------- ------------ ----------
   1981034 ISHITA                20
   1981031 ZENIL                 20
   1981036 MISHITA               19
   1981032 JHENIL                19
   1981033 BHARGAVI              19
   1981035 SAKSHITA              18

6 rows selected.
```

Select Statement: Distinct Values

Distinct Values:

Used to retrieve unique values for a column. Multiple rows can have same values for a column, distinct keyword in select statement helps us to retrieve unique rows for a column

Syntax:

```
Select distinct <col2> from < tab1>;
```

```
SQL> SELECT DISTINCT BATCH FROM CSE;

BATCH
-----
B1
B2
```

4. SQL Special Operators

Operator	Description
ALL	It compares a value to all values in another value set (in subqueries)
ANY	It compares the values in the list according to the condition (in subqueries).
BETWEEN	It is used to search for values that are within a set of values (given minimum and maximum values).
EXISTS	It is used to search for the presence of a row in a specified table (in subqueries).
IN	It compares a value to that specified list value (and in subquries).
LIKE	It compares a value to similar values using wildcard operator (% , _).
IS NULL	It checks for missing data or NULL values

To retrieve all records from *Emp* table with salary between 2000 and 5000

```
SELECT *
FROM Emp
WHERE Salary BETWEEN 2000 AND 5000;
```

Filtering: Comparison Operator

Comparison Operators (=, !=, <>,>=,<=, LIKE, Between , IN): Comparison operator are used in where condition to fetch results from table

```
SQL> select rollno, name from cse
  2  where BIRTHDATE between '01-JAN-04' AND '31-DEC-04';

    ROLLNO NAME
---------- ------------
   1981003 SAKSHI
   1981004 SAISH
   1981008 DEVANG
   1981009 VANSH
   1981013 NINAD
   1981046 HARSH
   1981051 DURVA
   1981256 MAHESH
   1981059 RAJ

9 rows selected.
```

```
SQL> select rollno, name from cse
  2  WHERE AREA IN ('ANDHERI');

    ROLLNO NAME
---------- ------------
   1981005 NIDHi
   1981034 ISHITA
   1981037 DEEP
   1981040 SOHAM
   1981054 SAMARTH
   1981057 DHRUVI
   2081162 ADITI

7 rows selected.
```

Comparison Operator: LIKE condition

LIKE condition to perform wild card searches of valid search string values

- Search conditions can contain either characters or numbers
- % denotes zero or many characters
- _ denotes one character

```
Example:
SQL> select dept_name,loc from dept where loc like 'c%';
Output:
dept_name      LOC
---------   -----------
accounts     Chennai
marketing    Chennai

Example:
SQL> select dept_name,loc from dept where loc like 'chen_ _ _';
Output:
dept_name          LOC
-------------- ------------
accounts       Chennai
marketing      Chennai
Example:
SQL> select dept_name,loc from dept where loc not like 'c%';
```

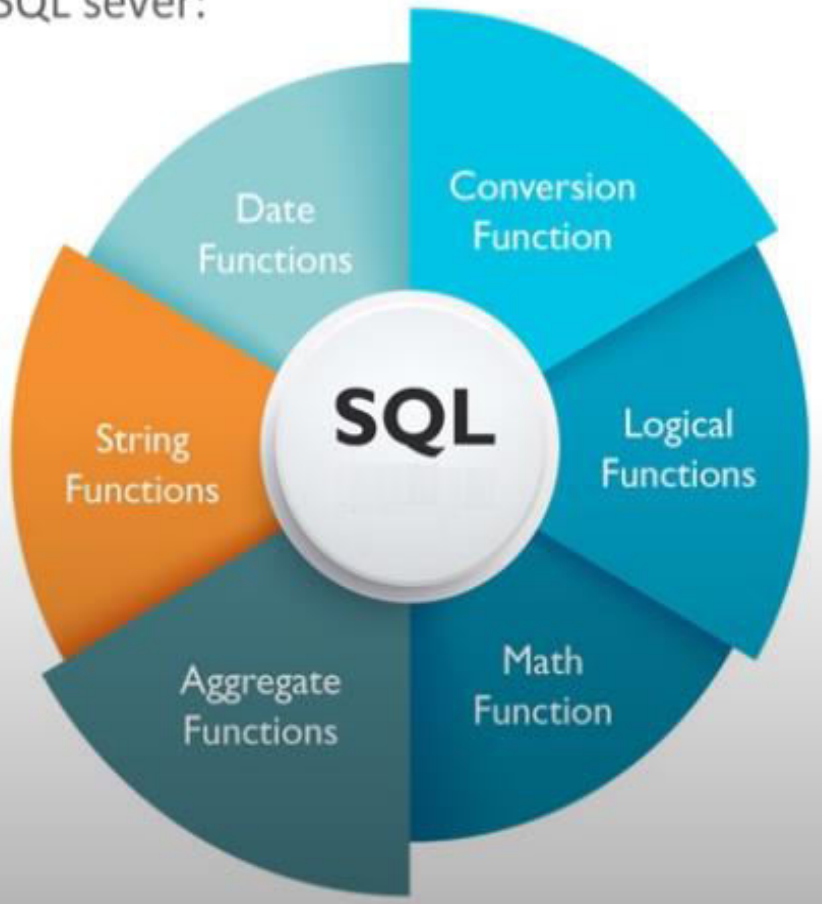
Comparison Operator: LIKE condition

```
SQL> select rollno, name, AREA from cse
  2  WHERE AREA LIKE 'D%';

    ROLLNO NAME          AREA
---------- ------------ ---------------
   1981004 SAISH         DAHISAR
   1981010 ANAND         DADAR
   1981019 PRATHMESH     DAHISAR
   1981022 RAJ           DAHISAR
   1981026 HARSH         DAHISAR
   1981039 VINIT         DADAR
   1981256 MAHESH        DADAR
   1881044 MEET          DOMBIVLI

8 rows selected.
```

```
SQL> select rollno, name, AREA from cse
  2  WHERE AREA LIKE '_I%';

    ROLLNO NAME          AREA
---------- ------------ ---------------
   1981007 DISHA         VILE PARLE
   1981021 PARAM         MIRA ROAD
   1981031 VAISHNAVI     MIRA ROAD
   1981043 KARAN         VILE PARLE
   1981048 NAVANG        VILE PARLE
```

SQL Built-in Functions

- Built-in functions are used to calculate values and manipulate data. These functions can be used anywhere as expressions
- Some common Built-in functions in SQL sever:



Group by Statement

The GROUP BY clause is used in a SELECT statement to group rows into a set of summary rows by values of columns or expressions. The GROUP BY clause returns one row per group.

The GROUP BY clause is often used with aggregate functions such as AVG(), COUNT(), MAX(), MIN(), and SUM().

In this case, the aggregate function returns the summary information per group. For example, given groups of products in several categories, the AVG() function returns the average price of products in each category.

Syntax:

```
SELECT expression1, expression2, ... expression_n,
aggregate_function (aggregate_expression)
FROM tables
WHERE conditions
GROUP BY expression1, expression2, ... expression_n;
```

Group by Statement

Retrieve the minimum marks in batch1 and batch2 from CSE table.

```
SQL> SELECT BATCH, MIN(MARKS)
  2  FROM CSE
  3  GROUP BY BATCH;

BAT MIN
--- ---
B1   10
B2   11
```

Retrieve the average marks area wise from CSE table in ascending order.

```
SQL> SELECT AREA, AVG(MARKS)
  2  FROM CSE
  3  GROUP BY AREA
  4  ORDER BY AREA;

AREA            AVG(MARKS)
--------------- ----------
ANDHERI         17.2857143
BANDRA                  17
BHAYANDAR             15.5
BORIVALI        16.8333333
CHARCHGATE              15
CHARNI ROAD             19
CHEMBUR                 10
DADAR                   19
DAHISAR               14.6
DOMBIVLI                14
GHATKOPAR               15
```

Group by Statement

Group By multiple columns: Group by multiple column is say for example, **GROUP BY column1, column2**.

This means to place all the rows with same values of both the columns **column1** and **column2** in one group. Consider the below query:

```
SQL> SELECT MENTOR, BATCH, COUNT(*)
  2  FROM CSE
  3  GROUP BY MENTOR, BATCH;

MENT BAT   COUNT(*)
---- --- ----------
SRK  B2           4
PPB  B2           6
RVP  B2           6
SUM  B1           3
SRK  B1           3
JSK  B2           1
JSK  B1           3
PPB  B1           6
GS   B1           5
PSA  B1           5
SUM  B2           2
PHS  B1           3
RVP  B1           5
GS   B2           6
PHS  B2           2
PSA  B2           5
```

Having Statement

The HAVING clause is an optional clause of the SELECT statement. It is used to filter groups of rows returned by the GROUP BY clause. This is why the HAVING clause is usually used with the GROUP BY clause.

```
SELECT
  column_list
FROM
  TABLE
GROUP BY
  COLUMN1..
HAVING
  group_condition;
```

```
SQL>  SELECT COURSE, COUNT(*)
  2  FROM CSE
  3   GROUP BY COURSE;

COURS   COUNT(*)
----- ----------
MATHS          12
FCN            13
FOS            14
CG             11
CPP            15
```

```
SQL>  SELECT COURSE, COUNT(*)
  2  FROM CSE
  3   GROUP BY COURSE
  4  HAVING COUNT(*) > 12;

COURS   COUNT(*)
----- ----------
FCN            13
FOS            14
CPP            15
```

Having Statement

EmployeeDetail table

	EmployeeID	FirstName	LastName	Salary	JoiningDate	Department	Gender
1	1	Vikas	Ahlawat	600000.00	2013-02-15 11:16:28.290	IT	Male
2	2	nikita	Jain	530000.00	2014-01-09 17:31:07.793	HR	Female
3	3	Ashish	Kumar	1000000.00	2014-01-09 10:05:07.793	IT	Male
4	4	Nikhil	Sharma	480000.00	2014-01-09 09:00:07.793	HR	Male
5	5	anish	kadian	500000.00	2014-01-09 09:31:07.793	Payroll	Male

ProjectDetail table

	ProjectDetailID	EmployeeDetailID	ProjectName
1	1	1	Task Track
2	2	1	CLP
3	3	1	Survey Managment
4	4	2	HR Managment
5	5	3	Task Track
6	6	3	GRS
7	7	3	DDS
8	8	4	HR Managment
9	9	6	GL Managment

1. Write the query to get the department and department wise total(sum) salary from "EmployeeDetail" table.
2. Write the query to get the department and department wise total(sum) salary, display it in ascending order according to salary.
3. Write the query to get the department and department wise total(sum) salary, display it in descending order according to salary.
4. Write the query to get the department, total no. of departments, total(sum) salary with respect to department from "EmployeeDetail" table.
5. Write down the query to fetch Project name assign to more than one Employee.

Having Statement

Write the query to get the department and department wise total(sum) salary from "EmployeeDetail" table.

```
SELECT Department,SUM(Salary) AS Total Salary
FROM EmployeeDetail
GROUP BY Department;
```

Write the query to get the department and department wise total(sum) salary, display it in ascending order according to salary.

```
SELECT Department, SUM(Salary) AS Total Salary
FROM EmployeeDetail
GROUP BY Department
ORDER BY SUM(Salary) ASC;
```

Write the query to get the department and department wise total(sum) salary, display it in descending order according to salary.

```
SELECT Department, SUM(Salary) AS Total Salary
FROM EmployeeDetail
GROUP BY Department
ORDER BY SUM(Salary) DESC
```

Write the query to get the department, total no. of departments, total(sum) salary with respect to department from "EmployeeDetail" table.

```
SELECT Department, COUNT(*) AS Dept Counts,
 SUM(Salary) AS Total Salary
FROM [EmployeeDetail]
GROUP BY Department;
```

Write down the query to fetch Project name assign to more than one Employee.

```
Select ProjectName,Count(*) AS No_of_Emp
FROM ProjectDetail
GROUP BY ProjectName
HAVING COUNT(*)>1;
```

Subqueries

A subquery is a SELECT statement nested inside another statement such as SELECT, INSERT, UPDATE, or DELETE. Typically, you can use a subquery anywhere that you use an expression.
Oracle allows a maximum nesting of 255 subquery levels in a WHERE clause. There is no limit for nesting subqueries expressed in a FROM clause. In practice, the limit of 255 levels is not really a limit at all because it is rare to encounter subqueries nested beyond three or four levels.

The complete syntax of a subquery is:

```
SELECT [DISTINCT] subquery_select_parameter
FROM {table_name | view_name}
        {table_name | view_name} ...
[WHERE search_conditions]
[GROUP BY column_name [,column_name ] ...]
[HAVING search_conditions]
```

```
SELECT first_name, salary, department_id
FROM employees
WHERE salary = (SELECT MIN (salary)
                FROM employees);
```

```
SELECT first_name, department_id
FROM employees
WHERE department_id IN (SELECT department_id
                        FROM departments
                        WHERE LOCATION_ID = 100)
```

Subqueries

```
SQL> DESC CSE
 Name
 -------------
 ROLLNO
 NAME
 AREA
 BIRTHDATE
 BATCH
 COURSE
 MARKS
 MENTOR
```

```
SQL> DESC COURSE
 Name
 ---------------
 CODE
 CNAME
 FEE
 DURATION
```

```
SQL> DESC MENTORS
 Name
 ----------------
 MCODE
 NAME
 EXPERIENCE
 SPECIALIST
 AREA
```

Retrieve name of the student who are living in the area same as 'JSK' is living.

Retrieve name and area of the student who are being mentored by a mentor living in the area ANDHERI.

Retrieve details of mentors who are mentoring students born in 2004.

Retrieve name and area of the student who are learning the course being taught by a mentor living in the area Thane.

Retrieve name of the student who are learning the course being taught by and mentored by same staff.

Subqueries

Retrieve name of the student who are living in the area same as 'JSK' is living.

```
SQL>  SELECT NAME FROM CSE
  2  WHERE AREA = (SELECT AREA FROM MENTORS
  3  WHERE MCODE = 'JSK');
```

Retrieve details of mentors who are mentoring students born in 2004.

```
SQL> SELECT * FROM MENTORS
  2  WHERE MCODE IN (SELECT MENTOR  FROM CSE
  3   WHERE EXTRACT(YEAR FROM BIRTHDATE) = 2004);
```

Retrieve name and area of the student who are being mentored by a mentor living in the area ANDHERI.

```
SQL> SELECT NAME,AREA
  2  FROM CSE
  3  WHERE MENTOR = (SELECT MCODE
  4  FROM MENTORS
  5   WHERE AREA = 'ANDHERI');
```