# MySQL

MySQL is Structured Query Language and this SQL is further divided into 4 parts:

1. **DML**
   **Data Manipulation Language -**

   DML statements affect the record in the table and perform basic operations like selecting any record, inserting any new record, deleting any record or updating or modifying existing records.
   Under this SELECT, INSERT, UPDATE, DELETE commands fall.

2. **DDL**
   **Data Definition Language -**

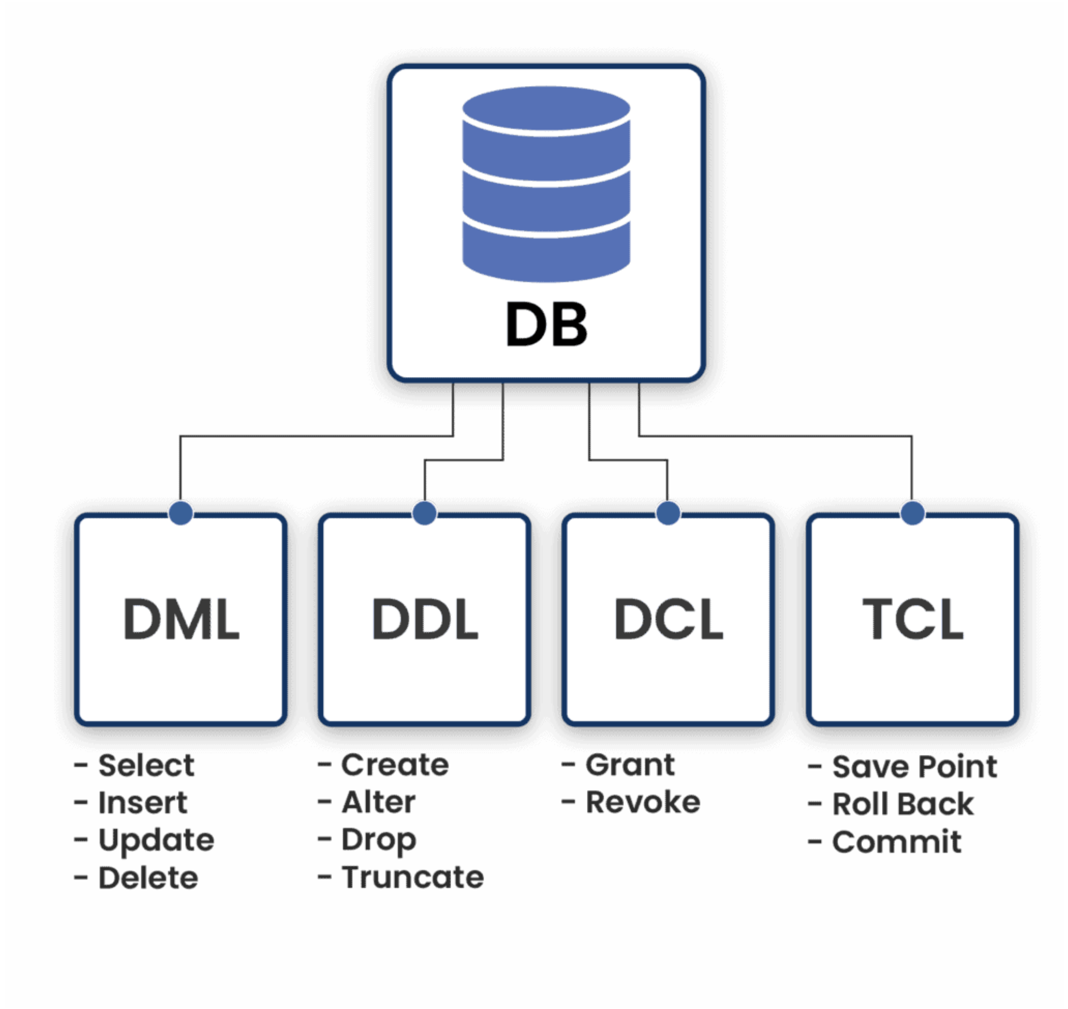
   DDL is used to alter/modify a database or table structure and schema. These statements are used to handle storage of database objects or design.
   Under these CREATE, ALTER, DROP commands fall.

3. **DCL**
   **Data Control Language -**

   DCL statements control the level of access that users have on database objects.
   Commands like GRANT (allows the user to read/write on certain database objects), REVOKE (take back the permission from the user to read/write on database objects)

4. **TCL**
   **Transaction Control Language -**

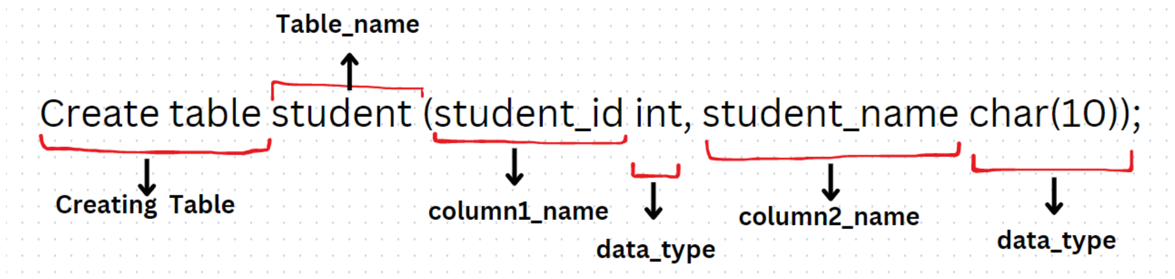
   TCL statements allow you to control and manage transactions to maintain the integrity of data within the system.
   Commands like BEGIN (this opens the transaction), COMMIT (to commit any transaction) , ROLLBACK (undo any transaction statement in case of any error).



## Data Types in MySQL:

A data type is a type/category to which data belongs to.
Data types define the type of data to be stored in each column of the table. Each column/attribute has its own type, we need to specify the type during creation of any table.
Eg. Create table table_name (column1 datatype, column2 datatype);

Create table student (student_id int, student_name char(10));



**String and character data types:**

1. **Characters(char) :-**

   These data types can hold alphabets, numbers, and special characters. These are of fixed length, if we don't specify any (value) Then the default size is 1. It will occupy space and memory according to the size of the parameter. Suppose we have given char(10), it will occupy 10 bytes of memory.
   Length of any value can range from 0 to 255.
   Length is fixed when we declare a table.

2. **Varchar :-**

   This stands for **variable character**, this means it can store variable length strings. The range of characters this can hold lies between 0 - 65,535.
   You cannot increase the length of the string defined using the size parameter. Its default size is also 1.
   Suppose you have declared varchar(20), and you have assigned a value "Ram". As the value size is of 3 bytes it will occupy only 3 bytes and release all the other spaces.

| varchar | char |
|---|---|
| It is variable length data type | It is fixed length data type |
| No padding is necessary (white spaces) as it is variable in size | Padding is done to the right to store the string when its size is less than declared value |
| Value can range between 0 - 65,535 | Value can range from 0 to 255 |

3. **Blob:**

   This stands for Binary Large Object. This can hold a variable amount of data. It can store binary data such as images, pdf, videos, and etc.
   Suppose your data contains 50 binary values then BLOB will occupy 52 bytes in memory it will add 2 bytes overhead to each value specified.

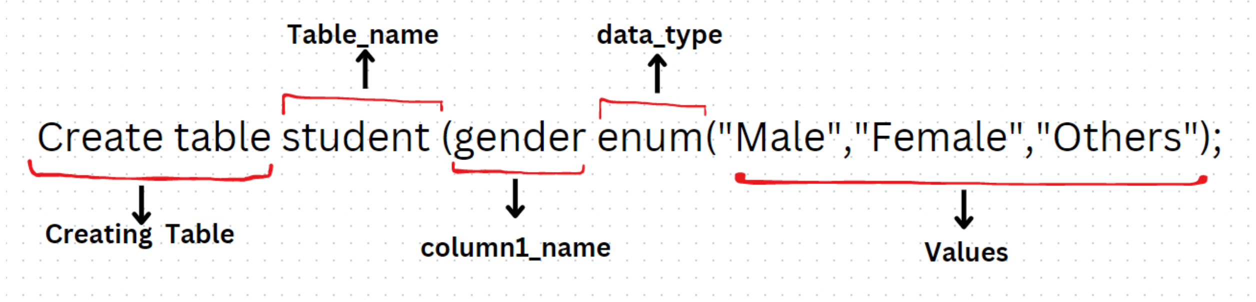
4. **Text:**

   Text is useful for storing long format text strings such as articles, blogs, etc. It can hold data up to 4GB.
   Suppose your data contains 50 binary values then BLOB will occupy 52 bytes in memory it will add 2 bytes overhead to each value specified.

5. **Enum:**

This is a string object whose value is chosen from a list of permitted values defined at the time of table creation. This provides compact storage of data. If a value is inserted that is not in the list, a blank value will be inserted.
Eg. Create table table_name (column enum(“val1, “val2”, “val3”))

Create table student (gender enum(“Male”, “Female”, “Others”))


Table_name
data_type
Create table student (gender enum("Male","Female","Others");
Creating Table
column1_name
Values

6. **Numeric:**

**int(size) -** int is used for storing integer type values. The size parameter specifies the maximum length of the number which you can store.

**float(p) -** Here **p** is used to determine whether to use float or double for the resulting data type. If the value of **p** ranges from 0-24, this means that the data type is float. If the range is from 25-53, then it becomes double.

**float(size,d)** - The length of digits is specified using the size parameter, the number of digits after decimal point is specified by d parameter.

Eg. float(3,2) - This means you can store the size of the number till 3 and 2 digits after decimal.

7. **Boolean:**

   False is stored as **0** and True is stored as **1**.

8. **Date:**

   You can store data in "**YYYY-MM-DD**" format. Date within the range from 1000-1-1 to 9999-12-31 can be stored.

9. **Time:**

   Time is used for storing time of a day. Time can be stored in the format of "**HH-MM-SS**".

10. **Datetime:**

    It can store date and time in the format of "**YYYY-MM-DD HH-MM-SS**".

11. **Timestamp:**

    This stores the data in the same format as date and time. You can automatically update/store the current date and time by using default current_timestamp.

12. **Year:**

    It is used to store a year in 4 digit format.

The default size of int is 4 bytes, big int 8 bytes, float 4 bytes, double 8 bytes

| | |
|---|---|
| int | 4 bytes |
| big int | 8 bytes |
| float | 4 bytes |
| double | 8 bytes |

We can view a table using the SELECT statement. To use any data from table first we need to create a database, if created then we need to use the database command is
USE database_name

SELECT * FROM table_name
***** - Will give the output, the whole records and attributes will be displayed.

desc command is used to see the description of any table. This gives you the structure or schema of any table which includes column name/attribute name, data type, constraints, indexes applied on it.

desc table_name

Alternate way to show the structure of table, we can use

show columns from table_name

show tables This command is used to display all the tables in current database

show tables
- to use this command first we need to use the database.

```
use database_name;
show tables;
```

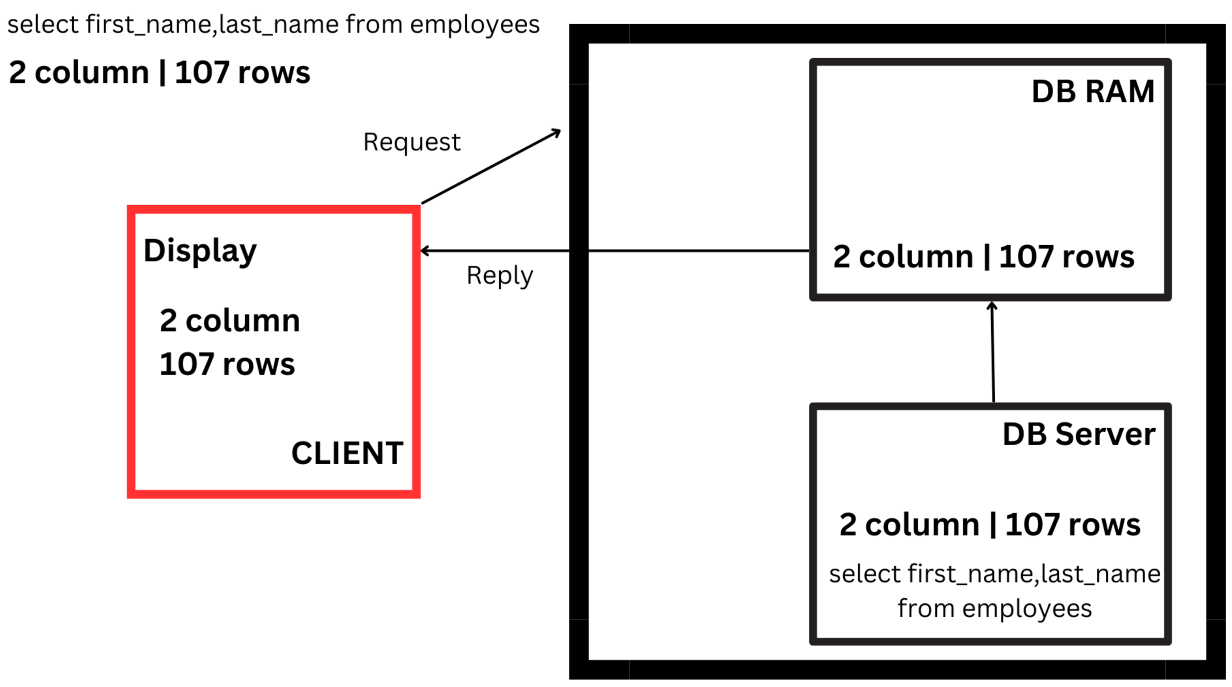
Alternate way to use show table is

```
show table status;
```

- This command will display information about each table in the database, table size, number of rows, creation date.

```
show databases
```

- This command displays the list of databases on your DB server.
  ```
  show databases
  ```
- If you want to get details about particular database we write
  ```
  show database database_name
  show databases like '%hr%'
  ```



# Alias:

Aliases are used to give temporary names to any column or table. It is used to make the statement more readable.
We can create aliases by using the **AS** keyword. This keyword is used to create an alias for any table or any columns.

```
-- Column Alias
        SELECT column_name AS alias_name
-- Table Alias
        select e.first_name from employees e;
-- Shorthand alias | without using AS keyword
        select first_name fname from employees;
select concat(first_name," ",last_name) as Emp from employees;
```

**Creating temporary table using Alias**

—

- **Where clause:**

  Where clause is used to filter the row written by select statement. It allows you to specify a condition that each row must meet in order to be included in the result set.

select column_name from table where condition

This condition can include one or more expressions that compare column value to literal value or other column. We can use operators like relational operators, logical operators in where condition.

- **Relational Operator:**

  Precedence High to Low
  1. >
  2. >=
  3. <=
  4. != or <>
  5. =
- **Logical operator:**

  Precedence high to low.
  1. NOT

     It will give the output which is not present in the condition.
  2. AND

     It will give output where both the conditions match.
  3. OR

     It will give output where any of the conditions match.

select * from employees where department_id=100 and SALARY>6000;

select * from employees where department_id=100 or SALARY>6000;

**AND** operator will get the precedence over **OR** operator.

- **Query 1**

select * from employees where department_id = 60 or department_id = 20 and salary =17000;

In Query 1, the condition department_id = 60 **or** department_id = 20 will return employees who either belong to 60 or 20. The condition salary = 17000 will be connected to the previous condition that was department_id, which means the condition will be only applied to the employee whose department_id is 60 hence This will lead to incorrect results as the condition is applied to employees who belong to either of the departments (20 or 60).

- **Query 2**

select * from employees where (department_id = 60 or department_id = 20) and salary =17000;

In Query 2, in this condition department_id = 60 or department_id = 20 are grouped together within the parenthesis. This ensures that **OR** condition will be evaluated first and then only condition salary = 17000 will be applied to employees who belong to either department_id = 20 or 60.

**Note:**

1. **Always use parentheses when you are using multiple logical operators to ensure that the conditions are evaluated correctly.**

- **IN operator:**

  In operator allow you to specify multiple values in where clause. **IN** operator is a shorthand for **OR** operator.

- **ANY operator:**

  **ANY** operator will return boolean value as a result. If returns **TRUE** if any of the values meets the condition.
  These are mostly used in subqueries to specify that the value(result should match any value returned by this subquery)
  We can use **WHERE** condition in **ANY** operator

**IN** operator can be used in a smaller list of values, **ANY** operator will be used when you have a large list or when a subquery generates a result.

- **ALL operator:**
  **ALL** operator is also used in subquery to specify that the result should match all the values returned by subquery.
  In **ALL** operator we need to specify a condition using the WHERE clause.

  ```
  select * from employees where department_id = any(select department_id
  from departments where department_id =90);
  ```

  ```
  select * from employees where department_id = all(select department_id
  from departments where department_id =90);
  ```

  **ALL** operator can be replaced with the combination of **NOT** and **ANY.**

- **Arithmetic operator:**
  Precedence high to low.
  1. ( ) -> Parenthesis
  2. ** -> Exponential
  3. /
  4. *
  5. +
  6. -

## Predefined Functions:

These predefined functions are divided into categories:

1. Aggregate function
2. Comparison function
3. Control Flow function
4. Date function
5. String function
6. Windows function
7. Math function

- **String functions**

1. **Concat** - Concat is used to concatenate two columns.
   select concat(first_name," ",last_name) as Employees from employees;

2. **INSTR** - When we want to find the position of the first occurrence of any substring then we use this function.
   select instr(first_name,"ar") as pos_of_ar from employees;

3. **Length**- If we want to find the length of any column in bytes and characters we use the length function.

   **-- Length in bytes -** the length function returns number of bytes given in a string
   select length(first_name) as length_bytes from employees;
   **-- Length in character -** the char_length function returns the number of characters in a string
   select char_length(first_name) as length_bytes from employees;

   **The difference between two functions is that the number of bytes may not be the same as the number of characters, but you may get different results while using the character set (UTF-8).**

4. **Left**- If you want to get first 3 characters of any column, we will use the left function
   As we have given the example of 3, we have written 3.
   select left(first_name,3) as three_char from employees;
   First parameter is the column name and second parameter is first n characters in the string/number of characters you want from the string.

5. **Substring** - When you want to extract a substring with their position value and with specified length then we use this command.

   select substring(column_name,start_position,number_of_char) from table_name;

   select substring(column_name,start_position)  from table_name;

- ❖ Realtime example of polymorphism can be the **substring** function.

6. **Substr()** - Works similar to substring

   select substr(column_name,start_position)  from table_name;

7. **Substring_index** - This function is used to extract substring from a string based on a delimiter that separates the substring based on the string. It takes 3 arguments
   - Original String
   - Delimiter
   - Counter parameter that specify which occurrence of delimiter to use for substring extraction

   select substring_index(street_address,' ',-1) as street from locations;

8. **Ltrim**  - It will remove the unwanted space from the left side.

   select ltrim(column_name) from employees;
   select ltrim(column_name,*) from employees;

9. **Rtrim**  - It will remove the unwanted space from the right side.

   select rtrim(column_name) from employees;
   select rtrim(column_name,*) from employees;

10. **Trim**  - It will remove the unwanted space/tabs.
    select trim(column_name) from employees;

11. **Right**- If you want to get last n characters of any column, we will use the right function
    select right(first_name,n) as last_char from employees;

12. **Lower** - This will convert the string to lower case.
    select lower(column_name) from table_name;

13. **Upper** - This will convert the string to upper case.

    select upper(column_name) from table_name;

14. **Lpad** - This is used for right justification.

    select lpad(column_name,number_of_character,delimiter) from table_name;

15. **Rpad** - This is used for left justification.

    select rpad(column_name,number_of_character,delimiter) from table_name;

16. **intcap** - This is used to make the initial character in capital.

    select intcap(column_name) from table_name;

select concat(upper(substr(first_name,1,1)),lower(substr(first_name,2))) from employees;

17. **Replace** - This function is used to replace all occurrences of substring within a string with new substring.
    - ❖ Case sensitive

select replace(column_name,character_to_be_replaced,new_character) from table_name

18. **Reverse** - This function is used to reverse a string.

select reverse(column_name) from table_name

19. **Locate** - This function will find the position of substring within a string. This will return the value at which the sub string was found.

select locate(substring_you_want_to_locate,column_name) from table_name

20. **Find_in_set** - This function is used to find the position of a string within a list of strings.

select find_in_set(string_to_be_searched,comma_separated_string_list) from table_name

## Aggregate functions

1. **Avg**- This function is used to calculate the average of the column.

   select avg(column_name) from table_name

2. **Sum**- This function is used to calculate the summation of all the records in the column.

   select sum(column_name) from table_name

3. **Count** - This function is used to count all the records in the column.

   select count(*/column_name) from table_name

4. **Max** - This function is used to find the maximum value from the column.

   select max(column_name) from table_name

5. **Min** - This function is used to find the minimum value from the column.

   select min(column_name) from table_name

## Math functions

In math function there are functions like:

- abs() - Returns absolute value of a number
- floor() - Returns the largest integer value that is greater than the argument passed

- ceil() - Returns the smallest integer value that is greater than or equal to the input passed
- round() - This will round off the decimal
- truncate() - This returns a specified number of decimal
- mod() - Returns the mod
- pow() - This will return the power

## Comparison functions

The comparison function compares different values.
Operators like **in**, **between**, fall under this.

- isnull() - This checks for the value whether it is present in the list or not.
- least(column_name,value) - This will return the least values from the column which are less than the specified value.

```
least(column1,column2,column3)
least(num1,num2,num3)
least(str1,str2,str3)
least(date1,date2,date3)
```

This works for any data type and returns the least among the values.

This can be used in the scenario where we have to give a discount of a certain percentage then you will use this command.

We can pass up to **255** values inside the parenthesis.

- greatest(column_name,value) - This will return the greater values from the column which are greater than the specified value.

Whenever you do a comparison with **null** it will always return **null**. These are also referred to as pessimistic queries which means that we are searching for **null** values.

```
select salary from employees where salary=null;
```

- like() - This operator is used to search the specified pattern in the column. Like operator supports 2 wildcard characters
  - % - Represents 0 or more character
  - _ - Underscore represents single character

```
select first_name from employees where first_name like 's%';
```
The names starting with ‘s’ will be printed.
```
select first_name from employees where first_name like '%a';
```
The names ending with ‘a’ will be printed.
```
select first_name from employees where first_name like '%a%';
```
The names which contain ‘a’ in between will be printed.
```
select first_name from employees where first_name like '__a';
```
The names which have ‘a’ and two characters before a will be printed.

## MySQL Case Expression:

Case statement sees the condition and returns the value of the condition that matches, so if one condition is true it will stop reading and return the result. If there's no else or no conditions are true it will return **NULL.**

This accepts 2 parameters:

- Condition
- Result

It works by evaluating a series of conditions and returns a result based on first condition that may be True.

```
case value
    when condition1 then result1
    when condition2 then result2
    when condition3 then result3
    else result
end
```

**-- Write a query, To show the bonus of employees who are working in a particular department. If the employee is working in the Marketing department and earns less than 6000 then add a bonus of 5 %.**
**If the employee is working in the HR department and has a salary < 5000 then add a bonus of 6%.**
**If the the employee is working in IT dept. and earns salary > 10000 then add bonus of 10%**
->

```
select first_name,last_name,salary,
case
        when salary <=6000 and department_id = 20 then salary*.05
   when salary <=5000 and department_id = 40 then salary*.06
   when salary >=10000 and department_id = 60 then salary*.10
end as bonus
from employees;
```

---

## Limit

**limit** clause is used to restrict the number of rows returned by the **select** statement.

`select column_name from table_name limit 10`(the number of rows to return)

`select column_name from table_name limit 5 offset 10`(the number of rows to return(5) after the offset value(10))

**row offset -** This specify the number of row to skip before starting to return the row

## Distinct

**Distinct** keyword is used to return distinct/remove duplicate rows based on specific columns.

## Order by

**Order by** is used to sort the result of the query in ascending or descending order.

```
select column_name from table order by column_name (asc/desc)
```

asc stands for ascending order, desc stands for descending order.
It will  sort by default, if we don't specify, it will sort in ascending order.

```
select first_name from employees order by first_name desc;
select first_name from employees order by first_name;
```

Sorting will be done at server ram, **order by** does sorting only so this will be done at server ram before sending the response to the client.

**Order by** clause will be the last statement in any query.

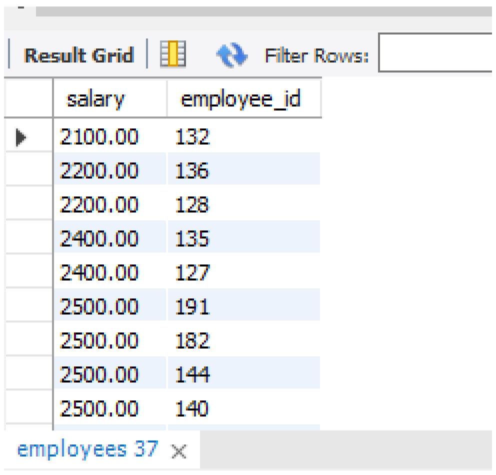
```
select first_name,salary+1000 as increment from employees order by increment;
```

**Sequence in which SQL statement retrieves the data(How the developer or DBA writes a SQL query)**

1. **Select**
2. **From**
3. **Where**
4. **Group by**
5. **Having**
6. **Order by**

---

```
select salary,employee_id from employees order by employee_id desc,salary
asc;
```



| | salary | employee_id |
|---|---|---|
| ▶ | 2100.00 | 132 |
| | 2200.00 | 136 |
| | 2200.00 | 128 |
| | 2400.00 | 135 |
| | 2400.00 | 127 |
| | 2500.00 | 191 |
| | 2500.00 | 182 |
| | 2500.00 | 144 |
| | 2500.00 | 140 |

Here, you have selected salary and employee_id. It will retrieve the data first by salary in ascending order and then the employee_id in descending order.
This will give the result sorted in ascending order of the salary.

## DDL Commands:

- **ALTER**: ALTER statement is used to add, delete, modify a column. Using ALTER you can rename a table, you can add a column a to the table, you can drop a column, increase width of a column, decrease width of a column, change data type of a column, copy a column, copy a table, copy a structure, rename a column, change position of column in the table structure.

Syntax - >
rename table table_name to new_table_name
This command will rename the table (Renaming a table is not very ideal)

To add a column to a table
alter table table_name add column_name datatype

To drop a column
alter table table_name drop column_name

To increase width of a column
alter table table_name modify column_name new_size

Modify is a clause that helps you to use the alter command.

To decrease width of a column
alter table table_name modify column_name new_size_to_decrease

**In mySQL, data will be truncated if you try to insert more than the decreased size.**

- **UPDATE**: UPDATE statement is used to modify the existing record.

Syntax - >
update table_name set column = value where condition

| UPDATE | ALTER |
|---|---|
| Update command is used to modify value stored in the specific column | Alter command is used to modify the structure of database object such as table_name, column_name |
| Update statement use **set** keyword to specify new value for one or more columns and use where clause also to specify the condition | The keywords like ADD, DROP, MODIFY are used to make the changes in the structure |
| | |

**If we don't write WHERE clause in update statement the whole column will be updated**

To change datatype of any column
alter table table_name modify column_name new_datatype
alter table employees modify age float;
desc employees;

You can copy a table
create table copy_table_name as select * from existing_table_name

The structure of copy_table_name is created on the basis of select statement, when select is executed the output of select statement will be inserted into the new table (copy_table_name)

To copy data from one table to another
```
insert into new_table_name select * from existing_table
```

The structure of the new table and the structure of the existing table should be the same.

- **DROP**: DROP command removes the entire database object such as table_index, view, from the database.
  This command cannot be undone once executed. All the data stored in this will be lost.
  Syntax - >
  ```
  drop table table_name
  ```

- **DELETE**: DELETE command is used to remove records from a table. This falls under data manipulation language. We can undo these statements by using **rollback** operations.
  Syntax - >
  ```
  delete from table_name where condition
  ```

- **TRUNCATE**: TRUNCATE command removes all the data from a table but not the table structure. It is similar to delete operation and faster than delete operation as it does not have to deal with **where**(searching of particular record) condition. This statement cannot be undone.

| DELETE | TRUNCATE |
|---|---|
| DELETE is DML command | TRUNCATE is DDL command |
| You can use **where** clause in **delete** | While in **truncate** there is no such keyword to use |
| **Delete** is slower than **truncate** | **Truncate** is faster than **delete** |
| After **delete** free space is retained by the table | After **truncate** free space is deallocated from the table |
| **Delete** statement can be undone using **rollback** | While **truncate** cannot be undone. Once deleted all the data is lost |

## Group By

Under aggregate function we saw,(avg,sum,count, min, max).
Select salary from employees where salary> avg(salary);

- ❖ We cannot use aggregate functions in **where** clause.

**Group By** clause is used to group 2 rows to have the same value in one or more columns. It typically works with aggregate functions(sum,count,min,max,avg).
The group by clause should be written after the **where** clause to specify which column should be grouped together. When you write a **select** statement, the column which you have grouped should be there in the **select** statement.

Rules for **Group by** clause:

- Besides group function or aggregate function whichever column is present in **select** clause that column_name has to be there in **group by** clause
- Whichever column is present in group by clause it may or may not be present in the select statement.

Eg. select max(salary) from table_name group by department_id
In this case department_id will also be brought to server RAM, sorting will be done department wise, sorting in salary will also be done but department_id will not be displayed

- There is no upper limit in group by clause, if you have a large number of columns in group by clause it will be slow because sorting will take time.

```
select job_id,department_id,sum(sal) from employee group by job_id,department_id;
select department_id,job_id,sum(sal) from employee group by job_id,department_id;
```

The position of the column in the **select** clause and the position of column in **group by** clause need not to be the same. The position of the column in the **select** clause will determine the position of the column in the output. The position of column in **group by** clause will determine sorting order,grouping order

NOTE: If you have 1 column in group by clause, this means 2D query.
If you have 2 columns in group by clause, this means a 3D query.
If you have 3 columns in a group by clause, this means a 4D query.
If you have multiple dimension query this means spatial query.

```
select department_id, count(*) number_of_employee
from employees group by department_id;

select year(hire_date) from employees;

select year(hire_date), count(*) number_of_employee
from employees group by year(hire_date);
```

## Having clause:

**Having** clause is used to filter the result of a query based on a condition that involves an aggregate function. It is used in combination with **group by** clause which groups the row based on one or more columns.
**Having** clause is applied to the group rowand filters out any group that does not satisfy the condition.
Syntax - >
select column_name_to_be_grouped, aggregate_function from table_name where condition group by column_to_be_grouped having condition

- **Where** clause is used to restrict the row
- **Having** clause works after all searching, sorting and conditioning done on any SQL statement
- It is recommended that only group functions should be used in having clause.
  A statement like:
  select department_id,sum(sal) from emp group by department_id having sal>17000
  This will give you error as sal is not a group function
  select department_id,sum(sal) from emp group by department_id having department_id = 110
  This SQL statement will work, but this is not an efficient way of using **having** clause.

- It is recommended that only group functions should be used in having clause

| Where | Having |
|---|---|
| Filters the row depending upon the condition | Filters on the group condition |
| Where clause is applicable without **group by** clause | Having clause does not work without **group by** clause |
| **Where** give you row restriction/row function | **Having** gives you column restriction/column function |

| | |
|---|---|
| **Where** clause is used before group by function | **Having** clause is used after group by function |
| **Where** clause is single row operation | **Having** clause is multiple row operation as it uses aggregate functions |

## Joins:

**Join** statement is used to combine data or rows from two or more tables based on a common field between them. Join is used to view columns of two or more tables.

- ➔ Join works from right to left.

### Types of joins:

1. **Equi join(Natural Join)** : This join is based on equality conditions. So matching rows of both the tables.

   ```
   select e.department_id, DEPARTMENT_NAME
   from employees e, departments d
   where e.DEPARTMENT_ID=d.DEPARTMENT_ID;
   ```

2. **Inequi join(Natural Join)** : This joins the table based on inequality conditions. It will show non-matching rows of both the tables. It is used in exception reports.

3. **Cartesian join**: This is a join without where clause. This is the fastest join. Every row of one table is combined with every row of another table.

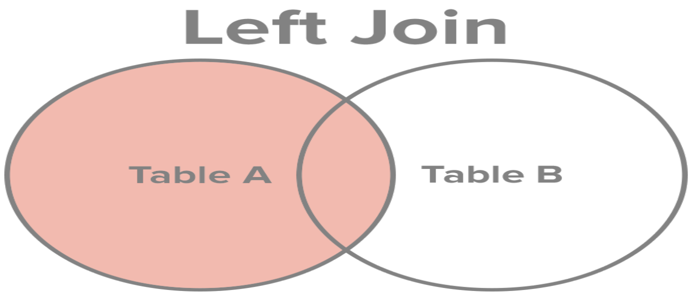
4. **Left join**: Left join returns all the rows from the left table and matching row from the right table. If there is no matching row in the right table, the result will contain NULL value for those columns.

| ROLL_NO | NAME | ADDRESS | PHONE | Age |
|---|---|---|---|---|
| 1 | HARSH | DELHI | XXXXXXXXXX | 18 |
| 2 | PRATIK | BIHAR | XXXXXXXXXX | 19 |
| 3 | RIYANKA | SILIGURI | XXXXXXXXXX | 20 |
| 4 | DEEP | RAMNAGAR | XXXXXXXXXX | 18 |
| 5 | SAPTARHI | KOLKATA | XXXXXXXXXX | 19 |
| 6 | DHANRAJ | BARABAJAR | XXXXXXXXXX | 20 |
| 7 | ROHIT | BALURGHAT | XXXXXXXXXX | 18 |
| 8 | NIRAJ | ALIPUR | XXXXXXXXXX | 19 |

| COURSE_ID | ROLL_NO |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 2 | 3 |
| 3 | 4 |
| 1 | 5 |
| 4 | 9 |
| 5 | 10 |
| 4 | 11 |

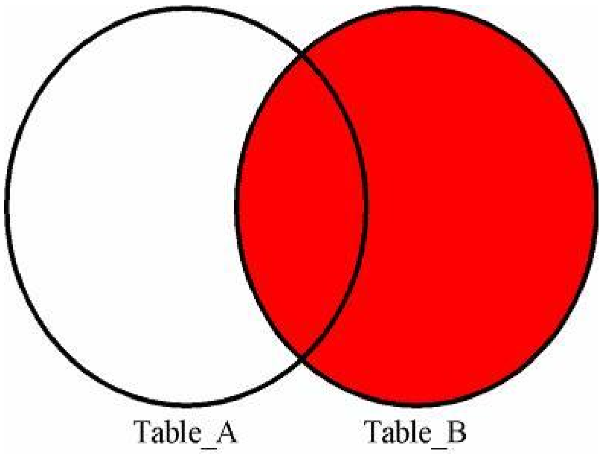
Select s.name, c.course_id from student s left join on course c c.roll_no=s.roll_no;

| NAME | COURSE_ID |
|---|---|
| HARSH | 1 |
| PRATIK | 2 |
| RIYANKA | 2 |
| DEEP | 3 |
| SAPTARHI | 1 |
| DHANRAJ | *NULL* |
| ROHIT | *NULL* |
| NIRAJ | *NULL* |



| NAME | COURSE_ID |
|---|---|
| HARSH | 1 |
| PRATIK | 2 |
| RIYANKA | 2 |
| DEEP | 3 |
| SAPTARHI | 1 |
| DHANRAJ | *NULL* |
| ROHIT | *NULL* |
| NIRAJ | *NULL* |

5. **Right join**: RIght join returns all the rows from the right table and matching rows from the left table. If there are no matching rows in the left table, the result will contain NULL values for those columns.



| ROLL_NO | NAME | ADDRESS | PHONE | Age |
|---|---|---|---|---|
| 1 | HARSH | DELHI | XXXXXXXXXX | 18 |
| 2 | PRATIK | BIHAR | XXXXXXXXXX | 19 |
| 3 | RIYANKA | SILIGURI | XXXXXXXXXX | 20 |
| 4 | DEEP | RAMNAGAR | XXXXXXXXXX | 18 |
| 5 | SAPTARHI | KOLKATA | XXXXXXXXXX | 19 |
| 6 | DHANRAJ | BARABAJAR | XXXXXXXXXX | 20 |
| 7 | ROHIT | BALURGHAT | XXXXXXXXXX | 18 |
| 8 | NIRAJ | ALIPUR | XXXXXXXXXX | 19 |

| COURSE_ID | ROLL_NO |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 2 | 3 |
| 3 | 4 |
| 1 | 5 |
| 4 | 9 |
| 5 | 10 |
| 4 | 11 |

Select s.name, c.course_id from student s left join on course c c.roll_no=s.roll_no;

| NAME | COURSE_ID |
|---|---|
| HARSH | 1 |
| PRATIK | 2 |
| RIYANKA | 2 |
| DEEP | 3 |
| SAPTARHI | 1 |
| *NULL* | 4 |
| *NULL* | 5 |
| *NULL* | 4 |

**1.** Write a query to find the addresses (location_id, street_address, city, state_province, country_name) of all the departments.

1. Department(depat_name, dept_id ), location(addrees)

Sub queries

    Date and time methods
    Transaction