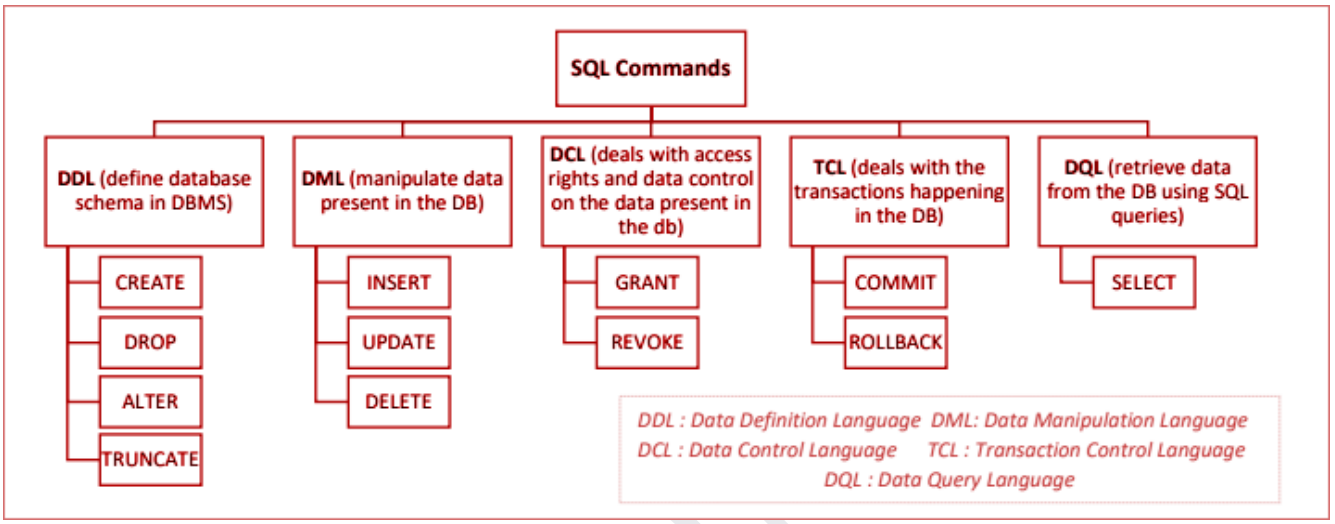
MySQL Statement

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.



Data Definition Language

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.
- Here are some commands that come under DDL:
-

– CREATE

It is used to create a table & database.

– DROP

It is used to delete both the structure and record stored in the table.

– ALTER

It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute/column or probably to add a new attribute/column.

– TRUNCATE

It is used to delete all the rows from the table and free the space containing the table.

– Rename

It is used to rename the table.

```
Create database revision;
Use revision;

Create table customers(
        Id int ,
Name varchar(20),
Email varchar(25),
City varchar(20)
);
```

Describe table : desc <table name>

Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.
- Here are some commands that come under DML:
-

– INSERT

- INSERT INTO is used to store data in the tables. The INSERT command creates a new row in the table to store data.
- The data is usually supplied by application programs that run on top of the database.
- Basic syntax:

INSERT INTO 'table_name' (column_1, column_2,...) VALUES (value_1,value_2,...);
ex.
insert into customers values(1, "vipul","vipul@gmail.com","Mumbai");

- The INSERT command can also be used to insert data into a table from another table.
- The basic syntax is as shown below.

INSERT INTO table_1 SELECT * FROM table_2;

– UPDATE

- UPDATE MySQL command is used to modify rows in a table. The update command can be used to update a single field or multiple fields at the same time.
- It can also be used to update a MySQL table with values from another table.
- Syntax

UPDATE 'table_name' SET 'column_name' = 'new_value' [WHERE condition];

– DELETE

- MySQL Delete command is used to delete rows that are no longer required from the database tables.
- It deletes the whole row from the table and returns count of deleted rows.
- Delete command comes in handy to delete temporary or obsolete data from your database.
- The Delete query in MySQL can delete more than one row from a table in a single query.
- This proves to be advantages when removing large numbers of rows from a database table.
- To delete a row in MySQL, the DELETE FROM statement is used:
- Syntax :

DELETE FROM 'table_name' [WHERE condition];

Data Control Language

- DCL commands are used to grant and take back authority from any database user.
- Here are some commands that come under DCL:

– Grant

Grant all privileges <db_name>.<db.table> to ‘user‘@’localhost’ identified by ‘password’;

Localhost - 127.0.0.1
All - select, insert, update , delete, alter , grant

– Revoke

Revoke all privileges <db_name>.<db.table> from ‘user‘@’localhost

Transaction Control Language

- TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.
- These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.
- Here are some commands that come under TCL:
  – COMMIT
  – ROLLBACK
  – SAVEPOINT
  - Start transaction

```
mysql> set autocommit = 0
    -> ;
Query OK, 0 rows affected (0.00 sec)

mysql> select @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            0 |
+--------------+
1 row in set (0.00 sec)
```

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into customers values(1, "vipul","vipul@gmail.com","Mumbai");
Query OK, 1 row affected (0.00 sec)

mysql> insert into customers(name,email,city) values( "Khetesh","khetesh@gmail.com","Mumbai");
Query OK, 1 row affected (0.00 sec)

mysql> select * from customers;
+------+---------+------------------+--------+
| Id   | Name    | Email            | City   |
+------+---------+------------------+--------+
|    1 | vipul   | vipul@gmail.com  | Mumbai |
| NULL | Khetesh | khetesh@gmail.com | Mumbai |
+------+---------+------------------+--------+
2 rows in set (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from customers;
Empty set (0.00 sec)
```

```
mysql> set autocommit = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into customers values(1, "vipul","vipul@gmail.com","Mumbai");
Query OK, 1 row affected (0.00 sec)

mysql> insert into customers(name,email,city) values( "Khetesh","khetesh@gmail.com","Mumbai");
Query OK, 1 row affected (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from customers;
+------+---------+-------------------+--------+
| Id   | Name    | Email             | City   |
+------+---------+-------------------+--------+
|    1 | vipul   | vipul@gmail.com   | Mumbai |
| NULL | Khetesh | khetesh@gmail.com | Mumbai |
+------+---------+-------------------+--------+
2 rows in set (0.00 sec)
```

```
mysql> set autocommit = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into customers(id,name,email,city) values(3,"Khetesh","khetesh@gmail.com","Mumbai"),(4,"Mohini","Mohini@gmail.com","Pun
e");
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> savepoint s1;
Query OK, 0 rows affected (0.00 sec)

mysql> update customers set id=10 where id=4;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from customers;
+------+---------+-------------------+--------+
| Id   | Name    | Email             | City   |
+------+---------+-------------------+--------+
|    1 | vipul   | vipul@gmail.com   | Mumbai |
| NULL | Khetesh | khetesh@gmail.com | Mumbai |
|    3 | Khetesh | khetesh@gmail.com | Mumbai |
|   10 | Mohini  | Mohini@gmail.com  | Pune   |
+------+---------+-------------------+--------+
4 rows in set (0.00 sec)

mysql> rollback to s1;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from customers;
+------+---------+-------------------+--------+
| Id   | Name    | Email             | City   |
+------+---------+-------------------+--------+
|    1 | vipul   | vipul@gmail.com   | Mumbai |
| NULL | Khetesh | khetesh@gmail.com | Mumbai |
|    3 | Khetesh | khetesh@gmail.com | Mumbai |
|    4 | Mohini  | Mohini@gmail.com  | Pune   |
+------+---------+-------------------+--------+
4 rows in set (0.00 sec)
```

```
mysql> select * from customers;
+------+---------+------------------+--------+
| Id   | Name    | Email            | City   |
+------+---------+------------------+--------+
|    1 | vipul   | vipul@gmail.com  | Mumbai |
| NULL | Khetesh | khetesh@gmail.com | Mumbai |
|    3 | Khetesh | khetesh@gmail.com | Mumbai |
|    4 | Mohini  | Mohini@gmail.com | Pune   |
+------+---------+------------------+--------+
4 rows in set (0.00 sec)

mysql> update customers set id=2 where id is null;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from customers;
+------+---------+------------------+--------+
| Id   | Name    | Email            | City   |
+------+---------+------------------+--------+
|    1 | vipul   | vipul@gmail.com  | Mumbai |
|    2 | Khetesh | khetesh@gmail.com | Mumbai |
|    3 | Khetesh | khetesh@gmail.com | Mumbai |
|    4 | Mohini  | Mohini@gmail.com | Pune   |
+------+---------+------------------+--------+
4 rows in set (0.00 sec)

mysql>
```

Data Query Language

- DQL is used to fetch the data from the database.
- It uses only one command:
  – SELECT
  Select * from <table_name>;
  Select col1, col2 from <table_name>;
  Select col1, col2 from <table_name> where {condition}

MySQL Data Types

- A Data Type specifies a particular type of data, like integer, floating points, Boolean, etc.
- It also identifies the possible values for that type, the operations that can be performed on that type, and the way the values of that type are stored.
- In MySQL, each database table has many columns and contains specific data types for each column.
- We can determine the data type in MySQL with the following characteristics:
  - The type of values (fixed or variable) it represents.
  - The storage space it takes is based on whether the values are a fixed-length or variable length.

MySQL Data Types: Numeric

- MySQL has all essential SQL numeric data types. These data types can include the exact numeric data types (For example, integer, decimal, numeric, etc.), as well as the approximate numeric data types (For example, float, real, and double precision).
- It also supports BIT datatype to store bit values. In MySQL, numeric data types are categories into two types, either signed or unsigned except for bit data type.

TINYINT

- It is a very small integer that can be signed or unsigned.
- If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. We can specify a width of up to 4 digits.
- It takes 1 byte for storage.

SMALLINT

- It is a small integer that can be signed or unsigned.
- If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535.
- We can specify a width of up to 5 digits. It requires 2 bytes for storage.

MEDIUMINT

- It is a medium-sized integer that can be signed or unsigned.

- If signed, the allowable range is from - 8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215.
- We can specify a width of up to 9 digits. It requires 3 bytes for storage.

INT

- It is a normal-sized integer that can be signed or unsigned.
- If signed, the allowable range is from - 2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295.
- We can specify a width of up to 11 digits. It requires 4 bytes for storage.

BIGINT

- It is a large integer that can be signed or unsigned.
- If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615.
- We can specify a width of up to 20 digits. It requires 8 bytes for storage.

FLOAT(m,d)

- It is a floating-point number that cannot be unsigned.
- You can define the display length (m) and the number of decimals (d).

DOUBLE(m,d)

- It is a double-precision floating-point number that cannot be unsigned. You can define the display length (m) and the number of decimals (d).

BIT(m)

- It is used for storing bit values into the table column. Here, M determines the number of bit per value that has a range of 1 to 64.

BOOL

- It is used only for the true and false condition. It considered numeric value 1 as true and 0 as false.

BOOLEAN

- It is Similar to the BOOL.

MySQL Data Types: Date and Time

YEAR[(2|4)]

- Year value as 2 digits or 4 digits. The default is 4 digits. It takes 1 byte for storage.

DATE

- Values range from '1000-01-01' to '9999-12-31'. Displayed as 'yyyy-mm-dd'. It takes 3 bytes for storage.

TIME

- Values range from '-838:59:59' to '838:59:59'. Displayed as 'HH:MM:SS'. It takes 3 bytes plus fractional seconds for storage.

DATETIME

- Values range from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Displayed as 'yyyy-mm-dd hh:mm:ss'.
- It takes 5 bytes plus fractional seconds for storage.

TIMESTAMP(m)

- Values range from '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' TC. Displayed as 'YYYY-MM-DD HH:MM:SS'.
- It takes 4 bytes plus fractional seconds for storage.

MySQL Data Types: String

- The string data type is used to hold plain text and binary data, for example, files, images, etc.
- MySQL can perform searching and comparison of string value based on the pattern matching such as LIKE operator

CHAR(size)

- It can have a maximum size of 255 characters.
- Here size is the number of characters to store. Fixed-length strings.

Char(4)

“ab__”

“Abcd”

“ABCDE” -> ABCD

VARCHAR(size)

- It can have a maximum size of 255 characters.
- Here size is the number of characters to store. Variable-length string.

varchar(4)

‘’Ab”
“Abcd”
ABCDE -> ABCDE

ADD Snnipets for varchar(4), char(4) with greater values.

TINYTEXT(size)

- It can have a maximum size of 255 characters.

• TEXT(size)

- Maximum size of 65,535 characters.

MEDIUMTEXT(size)

- It can have a maximum size of 16,777,215 characters.

LONGTEXT(size)

- It can have a maximum size of 4GB or 4,294,967,295 characters.

BINARY(size)

- It can have a maximum size of 255 characters.

ENUM

- It takes 1 or 2 bytes that depend on the number of enumeration values. An ENUM can have a maximum of 65,535 values.
- It is short for enumeration, which means that each column may have one of the specified possible values. It uses numeric indexes (1, 2, 3...) to represent string values.

Gender enum(“male”, “female”)

Creating a table

- MySQL allows us to create a table into the database by using the CREATE TABLE command. Following is a generic
- syntax for creating a MySQL table in the database.

```
CREATE TABLE [IF NOT EXISTS] table_name(
column_definition1,
column_definition2,

.....,
table_constraints
);
```

EX.

```
CREATE TABLE student(
id int NOT NULL AUTO_INCREMENT,
name varchar(45) NOT NULL,
age int NOT NULL,
PRIMARY KEY (id)
);
```

Primary Key

- MySQL primary key is a single or combination of the field, which is used to identify each record in a table uniquely.
- If the column contains primary key constraints, then it cannot be null or empty. A table may have duplicate columns, but it can contain only one primary key.
- It always contains unique value into a column.
- Following are the rules for the primary key:
    - The primary key column value must be unique.
    - Each table can contain only one primary key.
    - The primary key column cannot be null or empty.
    - MySQL does not allow us to insert a new row with the existing primary key.
    - It is recommended to use INT or BIGINT data type for the primary key olumn.

EX. - 1

```
Mysql> CREATE TABLE Login(
login_id INT AUTO_INCREMENT PRIMARY KEY,
username VARCHAR(40),
password VARCHAR(55),
email VARCHAR(55)
);
```

Ex - 2

```
mysql> CREATE TABLE Students (
Roll_No int,
Name varchar(45) NOT NULL,
Age int,
Primary Key(Roll_No) );
```

Foreign Key

- The foreign key is used to link one or more than one table together. It is also known as the referencing key.
- A foreign key matches the primary key field of another table. It means a foreign key field in one table refers to the primary key field of the other table.
- It identifies each row of another table uniquely that maintains the referential integrity in MySQL.
- A foreign key makes it possible to create a parent- child relationship with the tables.
- In this relationship, the parent table holds the initial column values, and column values of child table reference the parent column values.
- MySQL allows us to define a foreign key constraint on the child table.
- MySQL defines the foreign key in two ways:
    - Using CREATE TABLE Statement
    - Using ALTER TABLE Statement

EX

```
mysql> CREATE TABLE Students (
Roll_No int,
Name varchar(45) NOT NULL,
Age int,
Primary Key(Roll_No) );

mysql> CREATE TABLE address(
id int,
city_Name varchar(45) NOT NULL,
pincode bigint,
Roll_no int,
Primary Key(id)
CONSTRAINT fk_student FOREIGN KEY (Roll_No) REFERENCES Studnets(Roll_No)
);
```

NOT NULL is a field attribute, and it is used because we don't want this field to be NULL. If we try to create a record with a NULL value, then MySQL will raise an error.

UNIQUE: Ensures that all values in a column or a set of columns are distinct. No two rows can have the same value in a UNIQUE constrained column.

DEFAULT: Provides a default value for a column if no value is explicitly specified during an INSERT operation.

Check

- The check constraint is an integrity constraint that controls the value in a particular column.
- It ensures the inserted or updated value in a column must be matched with the given condition.

- In other words, it determines whether the value associated with the column is valid or not with the given condition.
- **EX :**
  CREATE TABLE vehicle (
vehicle_no VARCHAR(18) PRIMARY KEY,
model_name VARCHAR(45),
cost_price DECIMAL(10,2) NOT NULL CHECK (cost_price >= 0),
sell_price DECIMAL(10,2) NOT NULL CHECK (sell_price >= 0));
EX.
Insert into vehicle values (123, “tataNexon”, “10L” , “12L”); no error
Insert into vehicle values (123, “tataNexon”, -1 , “12L”); error

The field attribute AUTO_INCREMENT specifies MySQL to go ahead and add the next available number to the id field.

Select

The SELECT statement in MySQL is used to fetch data from one or more tables. We can retrieve records of all fields or specified fields that match specified criteria using this statement.
Syntax:

```
SELECT field_name1, field_name 2,... field_nameN
FROM table_name1, table_name2...
WHERE condition
GROUP BY field_name(s)
HAVING condition
ORDER BY field_name(s)
OFFSET M LIMIT N;
```

Where clause

- WHERE Clause in MySQL is a keyword used to specify the exact criteria of data or rows that will be affected by the specified SQL statement.
- The WHERE clause can be used with SQL statements like INSERT, UPDATE, SELECT, and DELETE to filter records and perform various operations on the data.
- Syntax
  SELECT * FROM tableName WHERE condition;

Where clause with AND
The WHERE condition in MySQL when used together with the AND logical operator, is only executed if ALL filter criteria specified are met.

Where clause with OR
The WHERE clause when used together with the OR operator, is only executed if any or the entire specified filter criteria is met.

Where clause with IN

- The WHERE in MySQL clause, when used together with the IN keyword only affects the rows whose values matches the list of values provided in the IN keyword.
- The MySQL IN statement helps to reduce number of OR clauses you may have to use. The following MySQL WHERE IN query gives rows where membership_number is either 1 , 2 or 3
- ex
- SELECT * FROM students WHERE roll_no IN (1,2,3);

Where clause with NOT IN

- The WHERE clause when used together with the NOT IN keyword DOES NOT affects the rows whose values matches the list of values provided in the NOT IN keyword.
- ex
- SELECT * FROM students WHERE roll_no IN (1,2,3);

Between

- The BETWEEN operator selects values within a given range.
- The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.
- syntax
- SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2;

Is NULL

MySQL IS NULL condition is used to check if there is a NULL value in the expression.
It is used with SELECT, INSERT, UPDATE and DELETE statement.
• Syntax:
– expression IS NULL

IS Not NULL
• Syntax:
– expression IS NOT NULL

Like

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
    - The percent sign (%) represents zero, one, or multiple characters
    - The underscore sign (_) represents one, single character
- The percent sign and the underscore can also be used in combinations!

Syntax:
SELECT column1, column2, … FROM table_name WHERE columnN LIKE pattern;

EX:
WHERE CustomerName LIKE 'a%' - Finds any values that start with "a"
WHERE CustomerName LIKE '%a' - Finds any values that end with "a"
WHERE CustomerName LIKE '%or%' - Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%' - Finds values that have "r" in the second position
WHERE CustomerName LIKE 'a_%' - Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE '__a__%'
WHERE ContactName LIKE 'a%o%a' - Finds any values that start with "a" and ends with "o"

Distinct

- The DISTINCT keyword that allows us to omit duplicates from our results.
- This is achieved by grouping similar values together .

Aggregate Functions

- MySQL's aggregate function is used to perform calculations on multiple values and return the result in a single value like the average of all values, the sum of all values, and maximum & minimum value among certain groups of values.
- We mostly use the aggregate functions with SELECT statements in the data query languages.

count()

- It returns the number of rows, including rows with NULL values in a group.

sum()

- It returns the total summed values (Non-NULL) in a set.

avg()

- It returns the average value of an expression.

Min()

- It returns the minimum (lowest) value in a set.

max( )

- It returns the maximum (highest) value in a set.

Order by

- MySQL ORDER BY is used in conjunction with the SELECT query to sort data in an orderly manner.
- The MySQL ORDER BY clause is used to sort the query result sets in either ascending or descending order.

SELECT statement... [WHERE condition | GROUP BY 'field_name(s)' HAVING condition]
ORDER BY 'field_name(s)' [ASC | DESC];

Example:
SELECT * FROM students ORDER BY marks DESC;

group by

- The GROUP BY clause is a SQL command that is used to group rows that have the same values.
- The GROUP BY clause is used in the SELECT statement. Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.
- That's what it does, summarizing data from the database.
- The queries that contain the GROUP BY clause are called grouped queries and only return a single row for every grouped item.

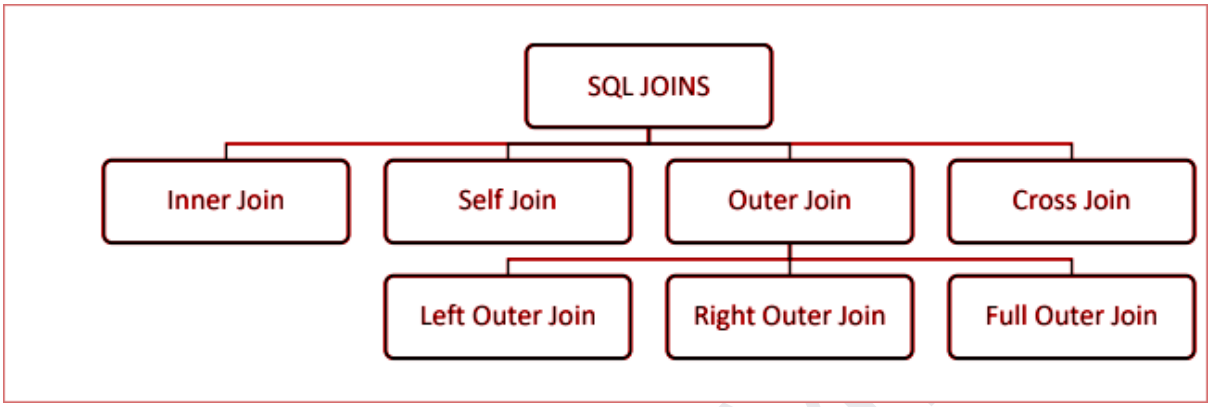
Syntax
SELECT statements... GROUP BY column_name1[,column_name2,...];

Having Clause

- MySQL HAVING Clause is used with GROUP BY clause. It always returns the rows where condition is TRUE.
- Syntax:

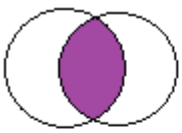
- SELECT expression1, expression2, ... expression_n, aggregate_function expression)
  FROM tables [WHERE conditions]
  GROUP BY expression1, expression2, ... expression_n
  HAVING condition;

Joins

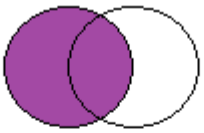


- In a relational database, data is often spread across multiple tables. When retrieving information from multiple tables, we use a database operation called join.
- A JOIN is used to combine rows from two or more tables based on a related column (usually a foreign key–primary key relationship).
- It allows you to query data that is spread across multiple tables.
- Without joins, data would be scattered across multiple tables.
- For example:
  - You have a student's table with student details.
  - And a courses table with course details.
  - You need a report showing *student name + course name*.
    Here we require JOINS

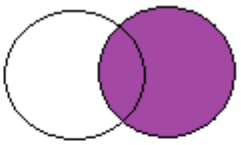
Syntax :
SELECT columns
FROM table1
JOIN table2
ON table1.common_column = table2.common_column;



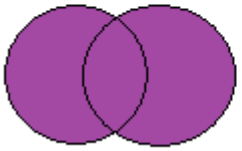
INNER JOIN



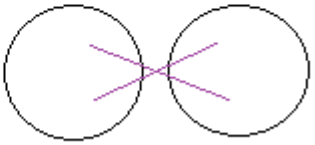
LEFT JOIN



RIGHT JOIN



FULL OUTER JOIN



CROSS JOIN

Inner Join in MySQL

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

Or

```
SELECT column_name(s)
FROM table1
JOIN table2
ON table1.column_name = table2.column_name;
```

Left Join in MySQL

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Right Join in MySQL

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Cross Join in MySQL

```
SELECT column_name(s)
FROM table1
CROSS JOIN table2;
```

Or

```
SELECT column_name(s)
FROM table1, table2;
```

SELF JOIN

```
SELECT a.column1, b.column2
  FROM table_name AS a
  INNER JOIN table_name AS b
  ON a.id = b.parent_id;
```

```
CREATE TABLE customers (
  customer_id INT PRIMARY KEY AUTO_INCREMENT,
  name        VARCHAR(80) NOT NULL,
  email       VARCHAR(120) UNIQUE,
  city        VARCHAR(80)
);

CREATE TABLE orders (
  order_id     INT PRIMARY KEY AUTO_INCREMENT,
  customer_id  INT NOT NULL,
  order_date   DATE NOT NULL,
  amount       DECIMAL(10,2) NOT NULL CHECK (amount >= 0),
  status       ENUM('NEW','PAID','CANCELLED') DEFAULT 'NEW',
  CONSTRAINT fk_orders_customer FOREIGN KEY (customer_id) REFERENCES
customers(customer_id)
);

INSERT INTO customers (name, email, city) VALUES
('Amit Sharma','amit@shop.com','Mumbai'),
('Bhavna Iyer','bhavna@shop.com','Pune'),
('Chetan Rao','chetan@shop.com','Delhi'),
('Divya Menon','divya@shop.com','Mumbai'),
('Esha Khan','esha@shop.com','Bengaluru');
```

```
INSERT INTO orders (customer_id, order_date, amount, status) VALUES
(1,'2025-09-01',1200.00,'PAID'),
(1,'2025-09-10', 750.00,'PAID'),
(2,'2025-09-11',  99.00,'NEW'),
(3,'2025-09-12', 560.00,'CANCELLED'),
(3,'2025-09-20',2100.00,'PAID'),
(4,'2025-09-25', 300.00,'PAID'),
(5,'2025-09-30', 890.00,'NEW'),
(2,'2025-10-01', 450.00,'PAID');

– IN Operator
SELECT customer_id, name, city
FROM customers
WHERE city IN ('Mumbai','Pune')
ORDER BY name ASC
LIMIT 3;

Select customer_id, name, city
From customers
Where city='Mumbai' and city='Pune'
Order by name asc
Limit 3;

-- In = OR
Select customer_id, name, city
From customers
Where city='Mumbai' or city='Pune'
Order by name asc
Limit 3;

Select customer_id, name, city
From customers
Where city='Mumbai' or city='Pune'
Order by name
Limit 3;
```

```
-- Between

select * from orders where order_date between "00-01-01" AND
"2024-01-01";

-- Like

Select * from orders where status like "c%";
Select * from orders where status like "p%";
Select * from orders where status like "%"; - all row
Select * from orders where status like "_A%"; - second letter as a

--
print the customer name who status paid
variation 1 _ select c.name , o.status from customer c inner join
orders o on c.customer_id = o.customer_id where o.status = "paid";

variation 2 _ select c.name , o.status from orders o inner join
customers c  on c.customer_id = o.customer_id where o.status =
"paid";

-- print latest order of customer

select c.name, max(o.order_date)
from customers c join orders o
on c.customer_id = o.customer_id
group by c.name;

select c.customer_id, c.name, max(o.order_date)
from customers c join orders o
on c.customer_id = o.customer_id
group by c.customer_id, c.name;

select c.customer_id, c.name, max(o.order_date)
```

```
from customers c join orders o
on c.customer_id = o.customer_id
group by c.customer_id, c.name having month(max(o.order_date))=10;

select c.customer_id, c.name, max(o.order_date)
from customers c join orders o
on c.customer_id = o.customer_id
group by c.customer_id, c.name having month(max(o.order_date))=9;
```