MySQL®

Let's start learning..

Table of Contents

Note: Click on the page number given in front to go to any topic.

What is SQL?

- SQL (Structured Query Language) is a standard language for storing, manipulating and retrieving data in databases.
- SQL is not a database system, It is language which is used by database management system like (Oracle, MySQL, Microsoft SQL Server, MongoDB and more).
- It is also pronounced as See-Quell.
- SQL language is mainly designed for maintaining the data in relational database management systems (RDBMS).
- Big enterprises like Facebook, Instagram, and LinkedIn, use SQL for storing the data in the back-end.
- Programming languages like php, asp.net and python use sql for backend.

What is data?

- Data is a collection of information.
- Data can be name, age, height, phone number, picture, PDF, audio and more.

What is database?

- A database is an electronic storage which is created by DBMS software's that stores the organized collection of records. It can be accessed and manage by the user very easily. It allows us to organize data into tables, rows, columns, and indexes to find the relevant information very quickly.

What is table?

- Table is a collection of data which is organized in terms of rows and columns.

Employee_ID	Employee_Name	City	Mobile Number
1	AJAY	Bhopal	1111-2222-33
2	SUNIL	Jaipur	3311-2222-33
3	MANOJ	Udaipur	2211-2222-33
4	KOMAL	SURAT	2211-2222-33

What is field?

- A field is a data structure for a single piece of data. For example, in a table called student contact information, telephone number is a field in column.

Employee_ID	Employee_Name	City	Mobile Number
1	AJAY	Bhopal	1111-2222-33
2	SUNIL	Jaipur	3311-2222-33
3	MANOJ	Udaipur	2211-2222-33
4	KOMAL	SURAT	2211-2222-33

What is record?

- In relational databases, a record is a group of related data held within the same structure.

Employee_ID	Employee_Name	City	Mobile Number
1	AJAY	Bhopal	1111-2222-33
2	SUNIL	Jaipur	3311-2222-33
3	MANOJ	Udaipur	2211-2222-33
4	KOMAL	SURAT	2211-2222-33

How to work with SQL?

To working with SQL you can use any database management system but in this tutorial I am using XAMPP and MySQL Workbench. So first you have to download and install XAMPP and MySQL Workbench.

Download Link For XAMPP -
https://www.apachefriends.org/download.html

Download Link For MySQL Workbench-
https://www.mysql.com/products/workbench/

To connect to MySQL from the command line, follow these:

Syntax: c:\xampp\mysql\bin>mysql -u user -p password -h localhost IP

Example: c:\xampp\mysql\bin>mysql -u root -p 12345 -h 127.0.0.1

Create Database

- **Syntax :** CREATE DATABASE database_name;
- **Example :** CREATE DATABASE company;

Select Database

- **Syntax :** USE database_name;
- **Example :** USE company;

Show Database

- **Syntax :** SHOW DATABASES;

Drop Database

- **Syntax :** DROP DATABASE database_name;

Create Table Statement

- **Syntax:**

```
 CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

Example:

```
CREATE TABLE employeeInfo(
ID int NOT NULL AUTO_INCREMENT PRIMARY KEY,
Emp_name varchar(30) NOT NULL,
Address varchar(255) NOT NULL,
City varchar(50) NOT NULL,
Age int NOT NULL,
```

DOJ date **NOT NULL**,
Designation varchar(50) **NOT NULL**,
Salary decimal(15,2) **NOT NULL**,
Mobile varchar(10) **NOT NULL**);

Drop/Delete Table Statement

The **DROP TABLE** statement is used to remove a table for database.

Syntax: DROP TABLE table_name;

ALTER TABLE Statement

- This statement is used to add, delete, or modify columns in an existing table.
- This is also used to add and drop various constraints on an existing table.

Add Column

```
ALTER TABLE table_name
ADD column_name datatype;
```

Example:-

```
ALTER TABLE employeeInfo
ADD Email varchar(255);
```

DROP Column

```
ALTER TABLE table_name;
DROP column_name;
```

Example:-

```
ALTER TABLE employeeInfo;
DROP Email;
```

MYSQL INSERT INTO Statement

The **INSERT INTO** Statement is used to add new data in the table.

Syntax:

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);
```

Example: (Inserting record in table)

```
INSERT INTO employeeInfo (Emp_name,  Address,  City,  Age,  DOJ,
Designation,  Salary,  Mobile)
VALUES ('Sunil Kumar', '104, Street No. 13', 'Jaipur', 29,'2020-5-3',
'Manager',  42000, '9188822200');

INSERT INTO employeeInfo (Emp_name,  Address, City,  Age,  DOJ,
Designation,  Salary,  Mobile)
VALUES ('Manoj Singh', '72, Street No. 1', 'Jaipur', 30, '2021-2-1',
'Programmer', 43000, '9188822211');

INSERT INTO employeeInfo (Emp_name,  Address,  City,  Age,  DOJ,
Designation,  Salary,  Mobile)
VALUES ('Anil Kumar', 'House No.53, Street No.7', 'Udaipur', 32,'2021-3-
1', 'Programmer',  42000, '9133322211');
```

INSERT INTO employeeInfo (Emp_name, Address, City, Age, DOJ, Designation, Salary, Mobile)
VALUES ('Kamal', 'P76, Block No.5', 'Jaipur', 36, '2020-2-1', 'Sr. Programmer', 60000, '9188833311');

INSERT INTO employeeInfo (Emp_name, Address, City, Age, DOJ, Designation, Salary, Mobile)
VALUES ('Daanish', 'RK Nagar, New Bus Stand', 'Ajmer', 37, '2021-4-1', 'Sr. Programmer', 60000, '9188822255');

SQL SELECT Statement

The **SELECT** statement is used to selecting (showing) the data from a table.

<u>To selecting whole data from table</u>

Syntax: SELECT *FROM table_name;
Example: SELECT *FROM employeeInfo;

Syntax: SELECT column1, column2, columnN FROM table_name;
Example: SELECT Emp_name, Mobile from employeeInfo;

The SQL WHERE Clause

The **WHERE** clause is used to filter records from table.

Syntax:
SELECT column1, column2, columnN
FROM table_name

```
WHERE Designation = [Condition];
```

Example: 1

```
SELECT Emp_name, Mobile
FROM employeeInfo
WHERE Designation = 'Programmer';
```

Example: 2

```
SELECT Emp_name, Mobile, Designation, Salary
FROM employeeInfo
WHERE Salary>45000;
```

The SQL UPDATE

If you want to modify the existing records in a table then you can use the query.

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2....
WHERE [condition];
```

Example: 1

```
UPDATE employeeInfo
SET Address =
WHERE [condition];
```

Example: 2

```
UPDATE employeeInfo
SET Address = 'P79, Block No.5'
WHERE ID=4;
```

Example: 3

```
UPDATE employeeInfo
SET Salary = 45000
WHERE ID = 5;
```

The SQL DELETE

The SQL DELETE Query is used to delete the existing records from a table.

Syntax:

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

Example:

```
DELETE FROM employeeInfo
WHERE ID = 5;
```

If you want to delete all records from table then you can use

```
DELETE FROM TABLE_NAME;
```

The SQL AND OPERATOR

You can use this operator for checking multiple conditions.

Syntax:

```
SELECT column1, column2,....
FROM table_name
WHERE [condition1] AND [condition2]...AND [conditionN];
```

Example: 1

```
SELECT ID, Emp_name
FROM employeeInfo
WHERE Designation = 'Programmer' AND Age > 30;
```

Example: 2

```
SELECT ID, Emp_name
FROM employeeInfo
WHERE Designation = 'Programmer' AND City = 'Jaipur';
```

The SQL OR OPERATOR

Syntax:

```
SELECT column1, column2,....
FROM table_name
WHERE [condition1] OR [condition2]...OR [conditionN];
```

Example:

```
SELECT ID, Emp_name, City
FROM employeeInfo
WHERE City = 'Jaipur' OR Salary = 42000;
```

The SQL NOT OPERATOR

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Example:

```
SELECT ID, Emp_name, City
FROM employeeInfo
WHERE NOT City = 'Jaipur';
```

The SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

Example: 1

```
SELECT *FROM employeeInfo
WHERE City LIKE 'J%';
```

(Start With) you can also use (End With Like – %J)

Example: 2

```
SELECT *FROM employeeInfo
WHERE Mobile LIKE '__8%';
```

Example: 3

```
SELECT * FROM employeeInfo
WHERE Emp_name NOT LIKE  'm%';
```

The SQL IN Operator

Example: 1

SELECT * FROM employeeInfo

WHERE City IN ('Udaypur', 'Ajmer');

Example: 2

SELECT * FROM employeeInfo

WHERE City NOT IN ('Udaypur', 'Ajmer');

The SQL BETWEEN Operator

The BETWEEN operator selects values within a given range.

Example: 1

SELECT * FROM employeeInfo

WHERE Salary **BETWEEN** 30000 AND 50000;

Example: 2

SELECT * FROM employeeInfo

WHERE ID **BETWEEN** 3 AND 5;

Example: 3

SELECT * FROM employeeInfo

WHERE DOJ **BETWEEN** '2021-01-01' AND '2021-05-30';

The SQL SELECT TOP Clause

SELECT TOP Clause is used to show specify the number of records from table.

Note: Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause.

Syntax:

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE [condition];
```

Example: (Note: Only For MySQL)

```
SELECT * FROM employeeInfo
LIMIT 3;
```

The SQL ORDER BY Clause

The SQL **ORDER BY** clause is used to sort the column data in ascending or descending order.

Example: 1

```
SELECT * FROM employeeInfo
ORDER BY Emp_name, Salary;
```

Example: 2

```
SELECT * FROM employeeInfo
ORDER BY Emp_name DESC;
```

Working with SQL functions

COUNT() Example:

SELECT COUNT(Emp_name)

FROM employeeInfo;

SUM() Example:

SELECT SUM(Salary)

FROM employeeInfo;

AVG() Example:

SELECT AVG(Salary)

FROM employeeInfo;

MIN() Example:

SELECT Emp_name, MIN(Salary) AS LowestSalary

FROM employeeInfo;

MAX() Example:

SELECT Emp_name, MIN(Salary) AS HighestSalary

FROM employeeInfo;

SQL Joins

The SQL **Joins** clause is used to combine records from two or more tables.

Example:

Step 1 (Create database)

```
CREATE DATABASE companydb;
```

Step 2 (Select database)

```
USE  companydb;
```

Step 3 (Create table)

```
CREATE TABLE Customer
(
 CustomerID int primary key,
 CustomerName varchar(20),
 City varchar(20)
);
```

Step 4 (Insert records in table)

```
INSERT INTO Customer VALUES(101,'Ashish','Kota');
INSERT INTO Customer VALUES(102,'Ajay','Ajmer');
INSERT INTO Customer VALUES(103,'Jay','Bundi');
INSERT INTO Customer VALUES(104,'Aman','Jhalawar');
INSERT INTO Customer VALUES(105,'Chirag','Udaipur');
INSERT INTO Customer VALUES(106,'Deepak','Jodhpur');
INSERT INTO Customer VALUES(107,'Rohan','Jaipur');
INSERT INTO Customer VALUES(108,'Dinesh','Alwar');
INSERT INTO Customer VALUES(109,'Suresh','Kota');
INSERT INTO Customer VALUES(110,'Ankit','Jaipur');
```

Step 5 (Create IInd table)

```
CREATE TABLE Orders
(
OrderID int,
CustomerID int primary key,
OrderDate date
);
```

Step 6 (Insert records in this table)

```
INSERT INTO Orders VALUES (1,101,'2021-01-05');
INSERT INTO Orders VALUES (2,102,'2021-02-10');
INSERT INTO Orders VALUES (3,103,'2021-01-04');
INSERT INTO Orders VALUES (4,104,'2021-03-08');
INSERT INTO Orders VALUES (5,105,'2021-04-09');
INSERT INTO Orders VALUES (6,106,'2021-01-05');
INSERT INTO Orders VALUES (7,107,'2021-05-12');
INSERT INTO Orders VALUES (8,108,'2021-02-05');
INSERT INTO Orders VALUES (9,109,'2021-06-03');
INSERT INTO Orders VALUES (10,110,'2021-07-04');
```

Step 7 (Check both tables)

```
SELECT *FROM Customer;

SELECT * FROM Orders;
```

<u>INNER JOIN QUERY</u> 1

```
SELECT Orders.OrderID, Customer.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customer ON Orders.CustomerID=Customer.CustomerID;
```

<u>INNER JOIN QUERY</u> 2

```
SELECT Orders.OrderDate, Customer.City
FROM Orders
INNER JOIN Customer ON Orders.CustomerID=Customer.CustomerID;
```

Some other SQL Joins Like – LEFT JOIN, RIGHT JOIN, FULL JOIN and SELF JOIN, you can read from Google.

The SQL GROUP BY

The GROUP BY statement groups rows that have the same values into rows. We can also use these functions
(COUNT(), MAX(), MIN(), SUM(), AVG()) with group by statements.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s);
```

Example:

```
SELECT COUNT(Designation), SUM(Salary), Designation
From employeeInfo
GROUP BY Designation;
```

SQL INSERT INTO SELECT Statement

You can copies data from one table and inserts it into another table.

Syntax:

```
INSERT INTO New_Table
SELECT * FROM Old_Table
WHERE condition;
```

Example:

```
INSERT INTO employeeInfoBackup
SELECT * FROM employeeInfo;
```

The SQL TRUNCATE TABLE Command

If you want to delete complete data from an existing table then you can use this command. If you use DROP Table command it would remove complete table structure form the database.

Syntax:

```
TRUNCATE TABLE  table_name;
```

SQL Constraints

By using SQL constraints you can set specify rules for the data in a table.

The following constraints are commonly used in SQL:

NOT NULL CONSTRAINT- Ensures that a column value should not NULL.

Example of NOT NULL Constraints:

Step 1 Create new database

```
CREATE DATABASE constraintsExample;
```

Step 2 Select this database

```
USE constraintsExample;
```

Step 3 Create Table

```
CREATE TABLE notnullExample
(
EmpName varchar(30),
Age int
);
```

Step 4 Insert Records

```
INSERT INTO notnullExample (EmpName, Age) VALUES ('Raju', 26);
INSERT INTO notnullExample (Age) VALUES (26);
```

Step 5 View Table

```
Select * from notnullExample;
```

Step 6 Delete Table

```
DROP TABLE notnullExample;
```

Step 7 Create Table Again

```
CREATE TABLE notnullExample
(
EmpName varchar(30) NOT NULL,
Age int
);
```

Step 8 Insert Records

```
INSERT INTO notnullExample (EmpName, Age)  VALUES ('Raju', 26);
INSERT INTO notnullExample (Age)  VALUES (26);
```

Step 9 View Table

```
Select * from notnullExample;
```

DEFAULT CONSTRAINT - Set a default value for a column if no value is inserting.

Example of DEFAULT Constraints:

Step 1 Create Table

```
CREATE TABLE defaultExample
(
EmpName varchar(30) NOT NULL,
Age int,
Mobile varchar(50) default 'Mobile no. is not available'
);
```

Step 2 Insert Data

```
INSERT INTO defaultExample (EmpName, Age, Mobile)
VALUES ('Raju', 26, '001223665');
```

Step 3 Insert this Record

Note: - To checking DEFAULT constraint insert this record in the table then 'Mobile no. is not available' this default value will store in mobile number column because of DEFAULT Constraint.

```
INSERT INTO defaultExample (EmpName, Age)
VALUES ('Raju', 26);
```

Step 4 View Table

```
SELECT * FROM defaultExample;
```

CHECK CONSTRAINT - You can define a CHECK constraint on a column it will allow only certain values for this column. If condition is matched then value will store in column.

Step 1 Creat Table

```
CREATE TABLE checkExample
(
EmpName varchar(255),
Age int,
CHECK (Age>=18)
);
```

Step 2 Insert Record

```
INSERT INTO checkExample (EmpName, Age)  VALUES ('Raju', 26);
```

Step 3 Insert this Record

Note: - To checking CHECK constraint insert this record in the table then you will find error because in this record you are inserting age 17 which is less than 18.

```
INSERT INTO checkExample (EmpName, Age)  VALUES ('Raju', 17);
```

UNIQUE CONSTRAINT - The UNIQUE constraint ensures that all values in a column are different.

Example of UNIQUE CONSTRAINT

Step 1 Create Table

```
CREATE TABLE uniqueExample (
EmpName varchar(255) NOT NULL,
Mobile varchar(10) NOT NULL UNIQUE
);
```

Step 2 Insert Record

```
INSERT INTO uniqueExample (EmpName, Mobile)  VALUES ('Raju',
'0123456789');
```

Step 3 Insert this Record

Note: - To checking UNIQUE Constraint insert this record in table then you will find error because in this record mobile number should be different and you are again inserting same mobile number.

```
INSERT INTO uniqueExample (EmpName, Mobile)  VALUES ('Sunil',
'0123456789');
```

PRIMARY KEY CONSTRAINT - Primary keys column must contain UNIQUE values, and cannot contain NULL values.

SQL FOREIGN KEY CONSTRAINT -

Step 1 Create table

```
CREATE TABLE Department
(
Dept_Id int primary key,Dept_Name varchar(50)
);
```

Step 2 Insert Records

```
INSERT INTO Department VALUES (1,'Account');
INSERT INTO Department VALUES (2,'HR');
INSERT INTO Department VALUES (3,'IT');
```

Step 3 Create IInd table

```
CREATE TABLE Employee_Details
(
Emp_ID int primary key,
Emp_Name varchar(50) Not null,
Dept_Id int,
FOREIGN KEY (Dept_Id) REFERENCES Department(Dept_Id)
);
```

Step 4 Insert Records

```
INSERT INTO Employee_Details VALUES (1,'Raj',1);
INSERT INTO Employee_Details VALUES (2,'Rahul',2);
INSERT INTO Employee_Details VALUES (3,'Kunal',3);

SELECT * FROM Employee_Details;
```

Note: - To checking FOREIGN KEY Constraint insert this record in table then you will find error because Dept_Id will not match.

Step 5 Insert this Records

```
INSERT INTO Employee_Details VALUES (4, 'sohan', 4);
```

The End

LEC-9: SQL in 1-Video

1. **SQL**: Structured Query Language, used to access and manipulate data.
2. SQL used **CRUD** operations to communicate with DB.
    1. **CREATE** - execute INSERT statements to insert new tuple into the relation.
    2. **READ** - Read data already in the relations.
    3. **UPDATE** - Modify already inserted data in the relation.
    4. **DELETE** - Delete specific data point/tuple/row or multiple rows.
3. **SQL is not DB, is a query language.**
4. What is **RDBMS**? (Relational Database Management System)
    1. Software that enable us to implement designed relational model.
    2. e.g., MySQL, MS SQL, Oracle, IBM etc.
    3. Table/Relation is the simplest form of data storage object in R-DB.
    4. **MySQL** is open-source RDBMS, and it uses SQL for all CRUD operations
5. **MySQL** used client-server model, where client is CLI or frontend that used services provided by MySQL server.
6. **Difference between SQL and MySQL**
    1. SQL is Structured Query language used to perform CRUD operations in R-DB, while MySQL is a RDBMS used to store, manage and administrate DB (provided by itself) using SQL.

SQL DATA TYPES (Ref: https://www.w3schools.com/sql/sql_datatypes.asp)
1. In SQL DB, data is stored in the form of tables.
2. Data can be of different types, like INT, CHAR etc.

DATATYPE	Description
CHAR	string(0-255), string with size = (0, 255], e.g., CHAR(251)
VARCHAR	string(0-255)
TINYTEXT	String(0-255)
TEXT	string(0-65535)
BLOB	string(0-65535)
MEDIUMTEXT	string(0-16777215)
MEDIUMBLOB	string(0-16777215)
LONGTEXT	string(0-4294967295)
LONGBLOB	string(0-4294967295)
TINYINT	integer(-128 to 127)
SMALLINT	integer(-32768 to 32767)
MEDIUMINT	integer(-8388608 to 8388607)
INT	integer(-2147483648 to 2147483647)
BIGINT	integer (-9223372036854775808 to 9223372036854775807)
FLOAT	Decimal with precision to 23 digits
DOUBLE	Decimal with 24 to 53 digits

DATATYPE	Description
DECIMAL	Double stored as string
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYYMMDDHHMMSS
TIME	HH:MM:SS
ENUM	One of the preset values
SET	One or many of the preset values
BOOLEAN	0/1
BIT	e.g., BIT(n), n upto 64, store values in bits.

3. Size: TINY < SMALL < MEDIUM < INT < BIGINT.
4. **Variable length Data types** e.g., VARCHAR, are better to use as they occupy space equal to the actual data size.
5. Values can also be unsigned e.g., INT UNSIGNED.
6. **Types of SQL commands:**
    1. **DDL** (data definition language): defining relation schema.
        1. **CREATE**: create table, DB, view.
        2. **ALTER TABLE**: modification in table structure. e.g, change column datatype or add/remove columns.
        3. **DROP**: delete table, DB, view.
        4. **TRUNCATE**: remove all the tuples from the table.
        5. **RENAME**: rename DB name, table name, column name etc.
    2. **DRL/DQL** (data retrieval language / data query language): retrieve data from the tables.
        1. **SELECT**
    3. **DML** (data modification language): use to perform modifications in the DB
        1. **INSERT**: insert data into a relation
        2. **UPDATE**: update relation data.
        3. **DELETE**: delete row(s) from the relation.
    4. **DCL** (Data Control language): grant or revoke authorities from user.
        1. **GRANT**: access privileges to the DB
        2. **REVOKE**: revoke user access privileges.
    5. **TCL** (Transaction control language): to manage transactions done in the DB
        1. **START TRANSACTION**: begin a transaction
        2. **COMMIT**: apply all the changes and end transaction
        3. **ROLLBACK**: discard changes and end transaction
        4. **SAVEPOINT**: checkout within the group of transactions in which to rollback.

MANAGING DB (DDL)

1. **Creation of DB**
    1. **CREATE DATABASE IF NOT EXISTS db-name;**
    2. **USE** db-name; //need to execute to choose on which DB CREATE TABLE etc commands will be executed. //make switching between DBs possible.
    3. **DROP** DATABASE IF EXISTS db-name; //dropping database.
    4. **SHOW** DATABASES; //list all the DBs in the server.
    5. **SHOW** TABLES; //list tables in the selected DB.

DATA RETRIEVAL LANGUAGE (DRL)

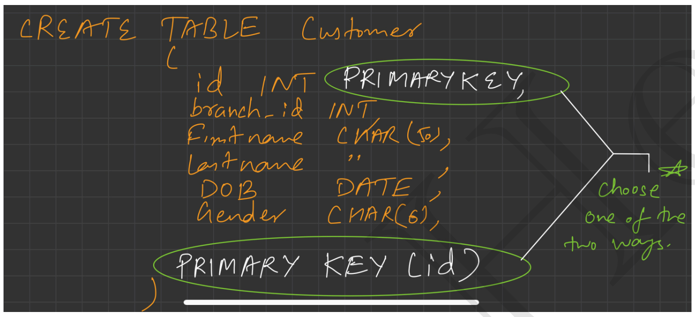
1. Syntax: SELECT <set of column names> FROM <table_name>;
2. Order of execution from RIGHT to LEFT.
3. Q. Can we use SELECT keyword without using FROM clause?
    1. Yes, using DUAL Tables.
    2. Dual tables are dummy tables created by MySQL, help users to do certain obvious actions without referring to user defined tables.
    3. e.g., SELECT 55 + 11;
       SELECT now();
       SELECT ucase(); etc.
4. **WHERE**
    1. Reduce rows based on given conditions.
    2. E.g., SELECT * FROM customer WHERE age > 18;
5. **BETWEEN**
    1. SELECT * FROM customer WHERE age between 0 **AND** 100;
    2. In the above e.g., 0 and 100 are inclusive.
6. **IN**
    1. Reduces **OR** conditions;
    2. e.g., SELECT * FROM officers WHERE officer_name IN ('Lakshay', 'Maharana Pratap', 'Deepika');
7. **AND/OR/NOT**
    1. AND: WHERE cond1 AND cond2
    2. OR: WHERE cond1 OR cond2
    3. NOT: WHERE col_name NOT IN (1,2,3,4);
8. **IS NULL**
    1. e.g., SELECT * FROM customer WHERE prime_status is NULL;
9. **Pattern Searching / Wildcard ('%', '_')**
    1. '%', any number of character from 0 to n. Similar to '*' asterisk in regex.
    2. '_', only one character.
    3. SELECT * FROM customer WHERE name **LIKE** '%p_';
10. **ORDER BY**
    1. Sorting the data retrieved using **WHERE** clause.
    2. ORDER BY <column-name> DESC;
    3. DESC = Descending and ASC = Ascending
    4. e.g., SELECT * FROM customer ORDER BY name DESC;
11. **GROUP BY**
    1. GROUP BY Clause is used to collect data from multiple records and group the result by one or more column. It is generally used in a SELECT statement.
    2. Groups into category based on column given.
    3. SELECT c1, c2, c3 FROM sample_table WHERE cond GROUP BY c1, c2, c3.
    4. All the column names mentioned after SELECT statement shall be repeated in GROUP BY, in order to successfully execute the query.
    5. Used with aggregation functions to perform various actions.
        1. COUNT()
        2. SUM()
        3. AVG()
        4. MIN()
        5. MAX()
12. **DISTINCT**
    1. Find distinct values in the table.
    2. SELECT DISTINCT(col_name) FROM table_name;
    3. GROUP BY can also be used for the same
        1. "Select col_name from table GROUP BY col_name;" same output as above DISTINCT query.

2. SQL is smart enough to realise that if you are using GROUP BY and not using any aggregation function, then you mean "DISTINCT".

13. **GROUP BY HAVING**
    1. Out of the categories made by GROUP BY, we would like to know only particular thing (cond).
    2. Similar to WHERE.
    3. Select COUNT(cust_id), country from customer GROUP BY country HAVING COUNT(cust_id) > 50;
    4. WHERE vs HAVING
        1. Both have same function of filtering the row base on certain conditions.
        2. WHERE clause is used to filter the rows from the table based on specified condition
        3. HAVING clause is used to filter the rows from the groups based on the specified condition.
        4. HAVING is used after GROUP BY while WHERE is used before GROUP BY clause.
        5. If you are using HAVING, GROUP BY is necessary.
        6. WHERE can be used with SELECT, UPDATE & DELETE keywords while GROUP BY used with SELECT.

CONSTRAINTS (DDL)

1. **Primary Key**



```
CREATE TABLE Customer
(
  id INT PRIMARYKEY,
  branch_id INT
  Firstname CHAR(50),
  Lastname  ''  ,
  DOB   DATE ,
  Gender  CHAR(6),
PRIMARY KEY (id)
)
```

Choose one of the two ways.

    1. PK is not null, unique and only one per table.

2. **Foreign Key**
    1. FK refers to PK of other table.
    2. Each relation can having any number of FK.
    3. CREATE TABLE ORDER (
       id INT PRIMARY KEY,
       delivery_date DATE,
       order_placed_date DATE,
       cust_id INT,
       FOREIGN KEY (cust_id) REFERENCES customer(id)
       );

3. **UNIQUE**
    1. Unique, can be null, table can have multiple unique attributes.
    2. CREATE TABLE customer (
       …
       email VARCHAR(1024) UNIQUE,
       …
       );

4. **CHECK**
    1. CREATE TABLE customer (
       …
       CONSTRAINT age_check CHECK (age > 12),
       …
       );
    2. "age_check", can also avoid this, MySQL generates name of constraint automatically.

5. **DEFAULT**
    1. Set default value of the column.
    2. CREATE TABLE account (
       …
       saving-rate DOUBLE NOT NULL DEFAULT 4.25,
       …
       );
6. An attribute can be **PK and FK both** in a table.
7. **ALTER OPERATIONS**
    1. Changes schema
    2. **ADD**
        1. **Add new column.**
        2. ALTER TABLE table_name ADD new_col_name datatype ADD new_col_name_2 datatype;
        3. e.g., ALTER TABLE customer ADD age INT NOT NULL;
    3. **MODIFY**
        1. **Change datatype of an attribute.**
        2. ALTER TABLE table-name MODIFY col-name col-datatype;
        3. E.g., VARCHAR TO CHAR
           ALTER TABLE customer MODIFY name CHAR(1024);
    4. **CHANGE COLUMN**
        1. **Rename column name.**
        2. ALTER TABLE table-name CHANGE COLUMN old-col-name new-col-name new-col-datatype;
        3. e.g., ALTER TABLE customer CHANGE COLUMN name customer-name VARCHAR(1024);
    5. **DROP COLUMN**
        1. **Drop a column completely.**
        2. ALTER TABLE table-name DROP COLUMN col-name;
        3. e.g., ALTER TABLE customer DROP COLUMN middle-name;
    6. **RENAME**
        1. **Rename table name itself.**
        2. ALTER TABLE table-name RENAME TO new-table-name;
        3. e.g., ALTER TABLE customer RENAME TO customer-details;

DATA MANIPULATION LANGUAGE (DML)
1. **INSERT**
    1. INSERT INTO table-name(col1, col2, col3) VALUES (v1, v2, v3), (val1, val2, val3);
2. **UPDATE**
    1. UPDATE table-name SET col1 = 1, col2 = 'abc' WHERE id = 1;
    2. Update multiple rows e.g.,
        1. UPDATE student SET standard = standard + 1;
    3. **ON UPDATE CASCADE**
        1. Can be added to the table while creating constraints. Suppose there is a situation where we have two tables such that primary key of one table is the foreign key for another table. if we update the primary key of the first table then using the ON UPDATE CASCADE foreign key of the second table automatically get updated.
3. **DELETE**
    1. DELETE FROM table-name WHERE id = 1;
    2. DELETE FROM table-name; //all rows will be deleted.
    3. **DELETE CASCADE** - (**to overcome DELETE constraint of Referential constraints**)
        1. What would happen to child entry if parent table's entry is deleted?
        2. CREATE TABLE ORDER (
           order_id int PRIMARY KEY,
           delivery_date DATE,
           cust_id INT,

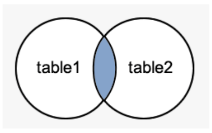
FOREIGN KEY(cust_id) REFERENCES customer(id) ON DELETE CASCADE
);

3. **ON DELETE NULL** - (**can FK have null values?**)
    1. CREATE TABLE ORDER (
    order_id int PRIMARY KEY,
    delivery_date DATE,
    cust_id INT,
    FOREIGN KEY(cust_id) REFERENCES customer(id) ON DELETE SET NULL
    );

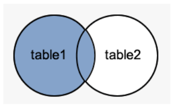
4. **REPLACE**
    1. Primarily used for already present tuple in a table.
    2. As UPDATE, using REPLACE with the help of WHERE clause in PK, then that row will be replaced.
    3. As INSERT, if there is no duplicate data new tuple will be inserted.
    4. REPLACE INTO student (id, class) VALUES(4, 3);
    5. REPLACE INTO table SET col1 = val1, col2 = val2;

JOINING TABLES

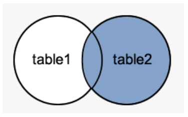
1. All **RDBMS** are relational in nature, we refer to other tables to get meaningful outcomes.
2. FK are used to do reference to other table.
3. **INNER JOIN**
    1. Returns a resultant table that has matching values from both the tables or all the tables.
    2. SELECT column-list FROM table1 INNER JOIN table2 ON condition1
    INNER JOIN table3 ON condition2
    …;



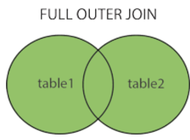
3. **Alias in MySQL (AS)**
    1. Aliases in MySQL is used to give a temporary name to a table or a column in a table for the purpose of a particular query. It works as a nickname for expressing the tables or column names. It makes the query short and neat.
    2. SELECT col_name AS alias_name FROM table_name;
    3. SELECT col_name1, col_name2,... FROM table_name AS alias_name;
4. **OUTER JOIN**
    1. **LEFT JOIN**
        1. This returns a resulting table that all the data from left table and the matched data from the right table.
        2. SELECT columns FROM table LEFT JOIN table2 ON Join_Condition;



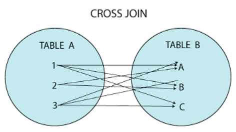
    2. **RIGHT JOIN**
        1. This returns a resulting table that all the data from right table and the matched data from the left table.
        2. SELECT columns FROM table RIGHT JOIN table2 ON join_cond;



    3. **FULL JOIN**
        1. This returns a resulting table that contains all data when there is a match on left or right table data.
        2. **Emulated** in MySQL using LEFT and RIGHT JOIN.
        3. LEFT JOIN UNION RIGHT JOIN.
        4. SELECT columns FROM table1 as t1 LEFT JOIN table2 as t2 ON t1.id = t2.id
        UNION
        SELECT columns FROM table1 as t1 RIGHT JOIN table2 as t2 ON t1.id = t2.id;
        5. UNION ALL, can also be used this will duplicate values as well while UNION gives unique values.



5. **CROSS JOIN**
    1. This returns all the cartesian products of the data present in both tables. Hence, all possible variations are reflected in the output.
    2. Used rarely in practical purpose.
    3. Table-1 has 10 rows and table-2 has 5, then resultant would have 50 rows.
    4. SELECT column-lists FROM table1 CROSS JOIN table2;



6. **SELF JOIN**

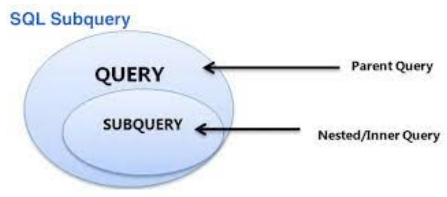
    1. It is used to get the output from a particular table when the same table is joined to itself.
    2. Used very less.
    3. Emulated using INNER JOIN.
    4. SELECT columns FROM table as t1 INNER JOIN table as t2 ON t1.id = t2.id;
7. **Join without using join keywords.**
    1. SELECT * FROM table1, table2 WHERE condition;
    2. e.g., SELECT artist_name, album_name, year_recordedFROM artist, albumWHERE artist.id = album.artist_id;

SET OPERATIONS
1. Used to combine multiple select statements.
2. Always gives distinct rows.

JOIN	SET Operations
Combines multiple tables based on matching condition.	Combination is resulting set from two or more SELECT statements.
Column wise combination.	Row wise combination.
Data types of two tables can be different.	Datatypes of corresponding columns from each table should be the same.
Can generate both distinct or duplicate rows.	Generate distinct rows.
The number of column(s) selected may or may not be the same from each table.	The number of column(s) selected must be the same from each table.
Combines results horizontally.	Combines results vertically.

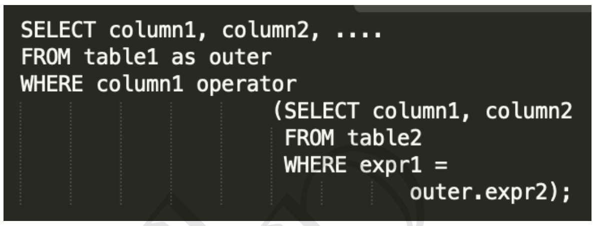
3. **UNION**
    1. Combines two or more SELECT statements.
    2. SELECT * FROM table1
       UNION
       SELECT * FROM table2;
    3. Number of column, order of column must be same for table1 and table2.
4. **INTERSECT**
    1. Returns common values of the tables.
    2. Emulated.
    3. SELECT DISTINCT column-list FROM table-1 INNER JOIN table-2 USING(join_cond);
    4. SELECT DISTINCT * FROM table1 INNER JOIN table2 ON USING(id);
5. **MINUS**
    1. This operator returns the distinct row from the first table that does not occur in the second table.
    2. Emulated.
    3. SELECT column_list FROM table1 LEFT JOIN table2 ON condition WHERE table2.column_name IS NULL;
    4. e.g., SELECT id FROM table-1 LEFT JOIN table-2 USING(id) WHERE table-2.id IS NULL;

SUB QUERIES
1. Outer query depends on inner query.
2. Alternative to joins.
3. Nested queries.
4. SELECT column_list (s) FROM table_name WHERE column_name OPERATOR (SELECT column_list (s) FROM table_name [WHERE]);
5. e.g., SELECT * FROM table1 WHERE col1 IN (SELECT col1 FROM table1);
6. Sub queries exist mainly in 3 clauses
    1. Inside a WHERE clause.



   2. Inside a FROM clause.
   3. Inside a SELECT clause.

7. **Subquery using FROM clause**
   1. SELECT MAX(rating) FROM (SELECT * FROM movie WHERE country = 'India') as temp;
8. **Subquery using SELECT**
   1. SELECT (SELECT column_list(s) FROM T_name WHERE condition), columnList(s) FROM T2_name WHERE condition;
9. **Derived Subquery**
   1. SELECT columnLists(s) FROM (SELECT columnLists(s) FROM table_name WHERE [condition]) as new_table_name;
10. **Co-related sub-queries**
    1. With a normal nested subquery, the inner SELECT query runs first and executes once, returning values to be used by the main query. A correlated subquery, however, executes once for each candidate row considered by the outer query. In other words, the inner query is driven by the outer query.



```
SELECT column1, column2, ....
FROM table1 as outer
WHERE column1 operator
                    (SELECT column1, column2
                     FROM table2
                     WHERE expr1 =
                               outer.expr2);
```

JOIN VS SUB-QUERIES

JOINS	SUBQUERIES
Faster	Slower
Joins maximise calculation burden on DBMS	Keeps responsibility of calculation on user.
Complex, difficult to understand and implement	Comparatively easy to understand and implement.
Choosing optimal join for optimal use case is difficult	Easy.

MySQL VIEWS

1. A view is a database object that has no values. Its contents are based on the base table. It contains rows and columns similar to the real table.
2. In MySQL, the View is a **virtual table** created by a query by joining one or more tables. It is operated similarly to the base table but does not contain any data of its own.
3. The View and table have one main difference that the views are definitions built on top of other tables (or views). If any changes occur in the underlying table, the same changes reflected in the View also.
4. CREATE VIEW view_name AS SELECT columns FROM tables [WHERE conditions];
5. ALTER VIEW view_name AS SELECT columns FROM table WHERE conditions;
6. DROP VIEW IF EXISTS view_name;
7. CREATE VIEW Trainer AS SELECT c.course_name, c.trainer, t.email FROM courses c, contact t WHERE c.id = t.id; (View using Join clause).

NOTE: We can also import/export table schema from files (.csv or json).