Table of Contents

No.	Tables
	Transaction Control Language (TCL)

	Create TABLE?
	show Tables
	See the table structure
	Alter/Rename table name
	Delete/Drop table

	Alter
	ADD a column in the table
	Add column after particular field
	Add column in first
	Add multiple columns in the table

	RENAME column in table?
	UPDATE
	DELETE
	Change Datatype from alter cmd?
	DROP column in table?
	TRUNCATE table?
	RENAME table name?
	Difference between Delete, Truncate & Drop?
	Difference b/w DROP and TRUNCATE statements?

No.	SQL Comments?
	Difference between **CHAR** and **VARCHAR** data types?
	Difference between In and Between Operator in SQL?
	IN and NOT IN Operator
	BETWEEN and NOT BETWEEN Operator?
	Difference between WHERE and HAVING in SQL?
	Ques. Difference between Group By And Order By?

No.	SQL Comments?
	What is Aggregate function?(sum,avg,max,min,count)
	SQL Operators and Clauses
	LIKE
	INNER JOIN
	OUTER JOIN
	IF()
	IFNULL
	NULLIF
	IS NULL and IS NOT NULL
	Round()
	BETWEEN()
	AND
	OR
	Case
	GROUP BY
	Having
	Limit
	ORDER BY
	SELECT DISTINCT
	With()
	WHERE
	Wildcard Characters/Like Query
	what is Aliases?
	What Is Union & Union All
	What is Intersect?
	What is MINUS?

No.	Keys
	Primary Key?
	Add primary Key?
	Delete primary Key?

No.	Keys
	Unique Key?
	ALTER unique key?
	Drop unique key?
	Foreign Key?
	Foreign Key Add/ALTER?
	DROP Foreign Key?
	Composite Key?
	Difference between Primary Key & Unique Key?
	Difference between Primary Key & Foreign Key?

No.	Joins
	What Is Joins?
	self join
	INNER JOIN
	Left JOIN/LEFT OUTER JOIN
	Right JOIN
	Outer join
	CROSS Join
	Full Join/FULL OUTER JOIN
	Difference between Delete, Truncate & Drop?

No.	Index
	What is Index?
	Types of Indexes
	Unique Indexes
	Show Index
	Alter/Modify an Index
	Drop Index
	Unique Indexes
	Cluster Index
	Non cluster index
	difference between cluster and non cluster index?

No.	View
	What is View?
	Create view
	Show view
	Alter view
	Deleted view

	What is ACID property/SQL TRANSACTIONS?

No.	interview_Questions_answers
	What are Constraints in SQL?

No.	Sql Query questions
	Demo data for execute the query
	Check version of the sql?
	Current date?
	How to copy a table in another table?
	How to copy structure of a table but not data?
	Duplicate table through another table, with structure and data?
	How to find **Nth** highest salary from a table?
	Top Nth Salery?
	Find the Highest Salary of Each Department?
	How to Find Duplicate values in a Table?
	Delete Duplicate Records?
	Check max_salary is not exceed the upper limit of 25000
	Replace a Column Values from 'male' to 'female' and 'female' to 'male'?
	Update remaing days startdate - enddate
	Count phone number

https://www.w3resource.com/sql-exercises/joins-hr/sql-joins-hr-exercise-11.php

Sql

Ques. What is database?

- A database is an organized collection of data, stored and retrieved digitally from a remote or local computer system. Databases can be vast and complex, and such databases are developed using fixed design and modeling approaches.
- Database is nothing but an organized form of data for easy access, storing, retrieval and managing of data.
- This is also known as structured form of data which can be accessed in many ways.

Ques. What is Sql?

- SQL is stands for **structure query language**.
- It is a database language **used** for database **creation**, **deletion**, **fetching** rows and modifying rows etc.
- It is a kind of ANSI standard language, used with all database.

Ques. What Is DBMS?

- A database management system is program that control creation, maintenance and use of a database.
- DBMS can be termed as File Manager that manages data in a database rather than saving it in file systems.

What is RDBMS?

- RDBMS stands for Relational Database Management System. RDBMS store the data into the collection of tables, which is related by common fields between the columns of the table. It also provides relational operators to manipulate the data stored into the tables.

Ques. Difference between DBMS & RDBMS?

DBMS	RDBMS
DBMS applications store data as file	RDBMS applications store data in a tabular form
Normalization is not present in DBMS	Normalization is present in RDBMS
DBMS does not support distributed data hnbase	RDBMS support distributed database

Mysql User Management

Create User

```
CREATE USER username@hostname IDENTIFIED BY 'password';
CREATE USER username IDENTIFIED BY 'password'; -- The hostname is optional
```

Show all Users

```
SELECT user, host FROM mysql.user;
+----------+-----------+
| user     | host      |
+----------+-----------+
| root     | localhost |
| newuser  | localhost |
| admin    | %         |
+----------+-----------+
```

Show Current User

```
SELECT USER();
Select current_user();
```

User Password Change

```
SET PASSWORD FOR 'mohits4'@'hostname' = PASSWORD('jtp12345');  -- older versions
-- new version
ALTER USER mohits4@hostname IDENTIFIED BY 'jtp123';

-- iF current user logged-in:
SET PASSWORD = 'new_password';
-- After password change:
FLUSH PRIVILEGES;
```

Drop User

```
DROP USER mohits4@localhost;
--can also be used to remove more than one user accounts at once.
DROP USER john@localhost, peter@localhost;
```

Grant Privileges to the MySQL New User

1. **ALL PRIVILEGES**: It permits all privileges to a new user account.
2. **CREATE**: Allows the user to create new databases, tables, indexes, views, or stored procedures.
3. **DROP**: Allows the user to delete (drop) existing databases, tables, views, or other objects.
4. **DELETE**: It enables the user account to delete rows from a specific table.
5. **INSERT**: It enables the user account to insert rows into a specific table.
6. **SELECT**: It enables the user account to read a database.
7. **UPDATE**: It enables the user account to update table rows.

- [ ] Note:- Sometimes, you want to flush all the privileges of a user account for changes occurs immediately

```
FLUSH PRIVILEGES;
```

```
-- 1. The first asterisk (*) refers to all databases
-- 2. The second asterisk (*) refers to all tables
GRANT CREATE, SELECT ON database_name.* TO 'username'@'host';

-- If you want to give all privileges to a newly created user, execute the
following command.
GRANT ALL PRIVILEGES ON * . * TO username@hostname;

-- If you want to give specific privileges to a newly created user, execute the
following command.
GRANT CREATE, SELECT, INSERT ON * . * TO username@hostname;
```

Show Privileges

```
SHOW GRANTS for mohits4;
SHOW GRANTS FOR 'local_user'@'localhost';
-- if user loged in
SHOW GRANTS;
```

REVOKE Privileges

```
REVOKE ALL PRIVILEGES ON *.* FROM 'mohits4'@'hostname';
-- If you want to remove specific privileges to a newly created user
REVOKE SELECT ON *.* FROM 'mohits4'@'hostname';
-- Revoke **all privileges** on a specific **database**:
REVOKE ALL PRIVILEGES ON database_name.* FROM 'mohits4'@'hostname';
-- Revoke **SELECT privilege** on a **specific** **table**:
REVOKE SELECT ON database_name.table_name FROM 'mohits4'@'hostname';
```

Database

Show Database

```
SHOW DATABASES/SCHEMAS;
SHOW DATABASES [LIKE 'pattern' | WHERE expr];
SHOW DATABASES LIKE 'test%';
SELECT * FROM sys.databases;
EXEC sp_databases;
+---------------------+
| Database            |
+---------------------+
| employeesdb         |
| profile_fastapi     |
+---------------------+
```

Create Databse

```
CREATE DATABASE databasename;
```

Rename Database

```
RENAME DATABASE old_database_name TO new_database_name
(OR)
ALTER DATABASE old_datbase MODIFY NAME = new_database
```

Drop Database

```
DROP DATABASE/SCHEMA database_name;

-- DROP DATABASE IF EXISTS Statement
DROP DATABASE IF EXISTS DatabaseName;

-- Deleting Multiple Databases
DROP DATABASE testDB3, testDB4;
```

Select Database

```
USE YourDatabaseName;
```

Storage Engine/Table Type

What is storage engine in mysql?

- In MySQL, a storage engine (also called a table type) is the component that handles how data is stored, retrieved, and managed in the database.
- Key points about MySQL storage engines:
    - Each storage engine has its own way of storing data and managing indexes.
    - We can choose a storage engine for each table depending on your needs.

```
CREATE TABLE users (
  id INT PRIMARY KEY,
  name VARCHAR(100)
) ENGINE=InnoDB;

-- If you don’t specify an engine, MySQL uses the default storage engine (usually
InnoDB).
```

Types of mysql engine

1. **InnoDB (Default)**

- **Transaction-safe (ACID compliant):** InnoDB supports transactions, ensuring data integrity with Atomicity, Consistency, Isolation, and Durability properties.
- **Supports foreign keys:** This ensures that related data in different tables is properly linked.
- **Row-level locking:** Instead of locking the entire table, InnoDB locks only the rows that are changing, allowing multiple users to work more efficiently and improving system performance.(पूरी टेबल को लॉक करने की बजाय, InnoDB सिर्फ उन पंक्तियों (rows) को लॉक करता है जिनमें बदलाव हो रहा होता है, जिससे एक साथ कई यूजर बेहतर तरीके से काम कर सकते हैं और सिस्टम की परफॉर्मेंस बढ़ती है।)
- **Crash Recovery:** InnoDB uses a "write-ahead log" so that data is preserved and not lost or corrupted after a crash.(InnoDB एक “write-ahead log” का इस्तेमाल करता है ताकि किसी भी क्रैश के बाद डेटा सुरक्षित रहे और कोई नुकसान या खराबी न हो।)

2. **MyISAM**

- **No transaction support:** ❌ MyISAM does not support transactions, so it can’t guarantee ACID properties like InnoDB.
- **Table-level locking:** When a query modifies data, the entire table is locked, which can reduce performance in multi-user environments.
- **No support for foreign keys:** MyISAM does not enforce referential integrity or foreign key constraints between tables.
- Crash Recovery: ❌ Limited
- Use Case: Fast read-heavy operations (e.g., reporting, static content)

3. **MEMORY**

- **Stores data in RAM:** MEMORY engine keeps all data in the server's memory (RAM), making data access extremely fast.
- **Data is temporary:** Since data is stored in memory, it is lost when the MySQL server restarts or crashes.
- Best for temporary or cache tables: Ideal for temporary data storage, like caching or quick lookups during a session.
- Supports only hash indexes: By default, MEMORY tables use hash indexes for very fast lookups, but you can also use B-tree indexes.
- No support for transactions or foreign keys: MEMORY engine does not support transactions or foreign key constraints.
- Limited table size: The size of MEMORY tables is limited by the amount of available RAM.

4. **CSV**

- Stores data in CSV files: Tables are stored as plain text files with comma-separated values, making them easy to read and edit outside MySQL.
- No indexing support: CSV tables do not support indexes, so searches can be slow on large datasets.
- No transaction support: It does not support transactions or ACID properties.
- Good for data exchange: Useful for importing/exporting data between MySQL and other applications that use CSV format.
- No support for foreign keys: Does not enforce foreign key constraints or referential integrity.
- Data is stored as plain text: Easy to open and manipulate with text editors or spreadsheet software.

5. **ARCHIVE**

- Designed for storing large volumes of historical data: ARCHIVE is optimized for storing data that is rarely updated or deleted.
- Supports only INSERT and SELECT operations: You can insert and read data, but you cannot update or delete rows.
- High compression: Data is stored in a compressed format to save disk space.
- No indexes (except AUTO_INCREMENT): ARCHIVE tables do not support indexes, so searching can be slower (only AUTO_INCREMENT on primary key is allowed).
- No transaction or foreign key support: It does not support transactions, rollbacks, or foreign keys.
- Ideal for logging and auditing: Great for storing logs, archived records, or audit trails where data integrity and compact storage are more important than speed of access.

6. **BLACKHOLE**

- Accepts data but doesn't store it: All INSERT operations are accepted, but the data is discarded—nothing is saved.
- Returns empty results on SELECT: Since no data is stored, any SELECT query will return an empty result.
- Useful for replication: Often used on master servers in replication setups to send data to slaves without storing it locally.
- No indexes or data storage: BLACKHOLE tables do not support indexing or actual data storage.
- Supports triggers: Triggers (like AFTER INSERT) still work, which can be useful for logging or monitoring actions.
- No transaction or foreign key support: Transactions and referential integrity are not supported.

7. **MERGE**

- Combines multiple MyISAM tables into one virtual table: A MERGE table acts as a single table that maps to multiple underlying MyISAM tables with the same structure.
- Used for managing large datasets: Useful when you want to split large tables into smaller parts (like by date) but still query them together.
- Improves performance in some cases: Allows you to search across multiple tables as if they were one, which can be optimized for specific queries.
- Supports only MyISAM tables: All underlying tables must be MyISAM and have exactly the same structure.
- Inherits MyISAM limitations: No support for transactions or foreign keys, and uses table-level locking.
- Supports SELECT, DELETE, and INSERT (no UPDATE): You can read, delete, and insert data, but cannot use UPDATE queries directly on a MERGE table.

8. **FEDERATED**

- Accesses remote MySQL tables: FEDERATED allows you to create a table in your local MySQL server that connects to a table on a remote MySQL server.
- No data stored locally: Data is not stored on the local server—the table acts as a pointer to the remote table.
- Useful for distributed systems: Ideal when you need to work with data across different MySQL servers without replicating it.
- Supports only SELECT, INSERT, UPDATE, DELETE: You can perform basic queries, but not advanced operations like indexing locally.
- No transaction or foreign key support: Transactions and foreign keys are not supported with FEDERATED tables.
- Needs manual setup: You must provide connection info (host, user, password, etc.) when creating a FEDERATED table.

SQL Comments?

- There are typically two main types of SQL comments:

1. **Single-line Comments:** Started with two dashes (--)

```
-- This is a single-line comment
SELECT * FROM employees; -- Retrieve all employee records
```

2. **Multi-line Comments:** Started with /* and ended with */

```
/* This is a
   multi-line comment
   explaining the query */
SELECT id, name, email FROM employees;
```

Tables

Types of SQL Commands/subsets of SQL?

- DDL (Data Definition Language):
    - **CREATE:** Creates a new table or database.
    - **ALTER:** Modifies an existing database object.
    - **DROP:** Deletes an entire table, database, or other objects.
    - **TRUNCATE:** Removes all records from a table, deleting the space allocated for the records.
- DML (Data Manipulation Language):
    - **SELECT:** Retrieves data from the database.
    - **INSERT:** Adds new data to a table.
    - **UPDATE:** Modifies existing data within a table.
    - **DELETE:** Removes data from a table.
- DCL (Data Control Language):
    - **GRANT:** Gives users access privileges to the database.
    - **REVOKE:** Removes access privileges given with the GRANT command.
- TCL (Transaction Control Language):
    - **COMMIT:** Saves all changes made in the current transaction.
    - **ROLLBACK:** Restores the database to the last committed state.
    - **SAVEPOINT:** Sets a savepoint within a transaction.
    - **SET TRANSACTION:** Places a name on a transaction.

Tables commands

Create table

```
-- CREATE TABLE IF NOT EXISTS table_name (  (it used for if table already exist)
CREATE TABLE table_name(
    id int NOT NULL AUTO_INCREMENT,
    name varchar(45) NOT NULL,
    PRIMARY KEY (id)
);
```

Show tables

```
SHOW TABLES;
+-----------------------+
| Tables_in_employee123 |
+-----------------------+
| employee_table        |
+-----------------------+
```

See the table structure

```
DESCRIBE employee_table;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int(11)     | NO   | PRI | NULL    | auto_increment |
| name       | varchar(45) | NO   |     | NULL    |                |
| occupation | varchar(35) | NO   |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
```

Alter/Rename table name

```
RENAME old_table_name To new_table_name; -- (OR)
RENAME TABLE old_table_name TO new_table_name;
ALTER TABLE old_table_name RENAME TO new_table_name; -- using alter
```

Delete/Drop table

```
DROP TABLE table_name;
```

Alter

- ALTER TABLE lets you add columns to a table in a database.

```
ALTER TABLE table_name ADD column_name datatype;
(OR)
ALTER TABLE table_name
ADD new_column_name datatype
[ FIRST | AFTER column_name ];
```

ADD a column in the table

```
ALTER TABLE employee_table ADD cus_age varchar(40) NOT NULL;

DESCRIBE employee_table;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int(11)     | NO   | PRI | NULL    | auto_increment |
| name       | varchar(45) | NO   |     | NULL    |                |
| occupation | varchar(35) | NO   |     | NULL    |                |
| age        | int(11)     | NO   |     | NULL    |                |
| cus_age    | varchar(40) | NO   |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
```

Add column after particular field

```
ALTER TABLE employee_table ADD after_occupation varchar(40) NOT NULL AFTER
occupation;

+------------------+-------------+------+-----+---------+----------------+
| Field            | Type        | Null | Key | Default | Extra          |
+------------------+-------------+------+-----+---------+----------------+
| id               | int(11)     | NO   | PRI | NULL    | auto_increment |
| name             | varchar(45) | NO   |     | NULL    |                |
| occupation       | varchar(35) | NO   |     | NULL    |                |
| after_occupation | varchar(40) | NO   |     | NULL    |                |
+------------------+-------------+------+-----+---------+----------------+
```

Add column in first

```
ALTER TABLE employee_table ADD COLUMN unique_id INT(11) NOT NULL FIRST;

+-----------------+-------------+------+-----+---------+----------------+
| Field           | Type        | Null | Key | Default | Extra          |
```

```
+------------------+-------------+------+-----+---------+----------------+
| unique_id        | int(11)     | NO   |     | NULL    |                |
| id               | int(11)     | NO   | PRI | NULL    | auto_increment |
| name             | varchar(45) | NO   |     | NULL    |                |
| occupation       | varchar(35) | NO   |     | NULL    |                |
| after_occupation | varchar(40) | NO   |     | NULL    |                |
| age              | int(11)     | NO   |     | NULL    |                |
| cus_age          | varchar(40) | NO   |     | NULL    |                |
+------------------+-------------+------+-----+---------+----------------+
```

Add multiple columns in the table

```
ALTER TABLE employee_table
    ADD COLUMN unique_id1 INT(11) NOT NULL FIRST,
    ADD COLUMN unique_id2 INT(11) NOT NULL FIRST;

+------------------+-------------+------+-----+---------+----------------+
| Field            | Type        | Null | Key | Default | Extra          |
+------------------+-------------+------+-----+---------+----------------+
| unique_id2       | int(11)     | NO   |     | NULL    |                |
| unique_id1       | int(11)     | NO   |     | NULL    |                |
| unique_id        | int(11)     | NO   |     | NULL    |                |
| id               | int(11)     | NO   | PRI | NULL    | auto_increment |
| name             | varchar(45) | NO   |     | NULL    |                |
| occupation       | varchar(35) | NO   |     | NULL    |                |
| after_occupation | varchar(40) | NO   |     | NULL    |                |
| age              | int(11)     | NO   |     | NULL    |                |
| cus_age          | varchar(40) | NO   |     | NULL    |                |
+------------------+-------------+------+-----+---------+----------------+

-- add enum column
ALTER TABLE table_name ADD field_name enum('0','1') NOT NULL DEFAULT '0' after
password;
```

Change Datatype from alter cmd

```
ALTER TABLE employee_table MODIFY unique_id4 VARCHAR(40) NULL;
OR
ALTER TABLE employee_table MODIFY COLUMN unique_id4 VARCHAR(40) NULL;
```

DROP column in table

```
ALTER TABLE employee_table DROP COLUMN unique_id4;
```

↑ Back to top

SELECT

- SELECT statements are used to fetch data from a database. Every query will begin with SELECT.

```
SELECT * FROM table_name;   -- fetch Full data
SELECT column_name FROM table_name;     -- fetch particular column
```

Insert table

```
INSERT INTO table_name
(column1,column2,column3,...)
VALUES
('value1','value2','value3',...);
```

UPDATE

- UPDATE statements allow you to edit rows in a table.

```
UPDATE table_name
SET some_column = some_value
WHERE some_column = some_value;
--
Update customer set name="mohit" where id =1;
```

DELETE

- DELETE is a DML(Data Manipulation Language) command and is used when we specify the row (tuple) that we want to remove or delete from the table or relation. The DELETE command can contain a WHERE clause.

```
DELETE FROM table_name
WHERE some_column = some_value;
```

- But if you do not specify the WHERE condition it will remove **all the rows** from the table.

```
DELETE FROM table_name;
```

Truncate

- A truncate SQL statement is used to remove all rows (complete data) from a table.

- TRUNCATE is a DDL(Data Definition Language) command and is used to delete all the rows or tuples from a table. Unlike the DELETE command, the TRUNCATE command does not contain a WHERE clause.

```
TRUNCATE TABLE table_name;
```

Ques. Difference between Delete, Truncate & Drop?

Delete	Truncate	Drop
Delete is a DML command	Truncate is DDL command	
We can use where clause in delete command	We cannot use where clause with truncate	
Delete statement is used to delete a row from a table	Truncate statement is used to remove all the row from a table	Remove table and data
You can rollback data after using delete statement	It is not possible to rollback after using TRUNCATE statement.	Can’t rollback
Delete is slower	Truncate is faster	

Ques. Difference b/w DROP and TRUNCATE statements?

- **Drop Table:-**
  - Table structure will be dropped
  - Relationship will be dropped
  - Integrity constraints will be dropped
  - Access privileges will also be dropped
- **TRUNCATE Table:-**
  - On the other hand when we TRUNCATE a table, the table structure remains the same, so you will not face any of the above problems.

Ques. Difference between CHAR and VARCHAR data types?

- Both of these data types are used for characters.
- **CHAR data** type is used to store **fixed-length** character strings. When VARCHAR data type is used to store **variable-length** character strings.
- char occupies all the space and if space is remaining, then it fill all the blank space with "space". But in case of varchar, It takes only the required length & release remaining.

```
Char -> 10      | R | A | M | space | space | sapce | space | space | space |
space |
Varchar -> 10   | R | A | M |   |   |   |   |   |   |   |
| R | A | M |
```

- varchar is better than Char in term of space.
- char perform is better than varchar.
- Char max 256 characters, varchar 65535 characters.

Ques. Difference between In and Between Operator in SQL?

- BETWEEN operator is used to **select a range of data between two values** while The IN operator allows you to **specify multiple values**.
- The BETWEEN operator selects a range of data between two values. The values can be numbers, text,etc.

```
+----+----------+--------+
| ID | NAME     | mark   |
+----+----------+--------+
|  1 | Ramesh   |   89   |
|  2 | Khilan   |   81   |
|  3 | kaushik  |   73   |
|  3 | kaushik  |   67   |
|  4 | Chaitali |   52   |
+----+----------+--------+

-- between
SELECT * FROM emp WHERE marks BETWEEN 50 AND 80
+----+----------+--------+
| ID | NAME     | mark   |
+----+----------+--------+
|  3 | kaushik  |   73   |
|  3 | kaushik  |   67   |
|  4 | Chaitali |   52   |
+----+----------+--------+

-- In
SELECT * FROM emp WHERE marks IN (89,73)
+----+----------+--------+
| ID | NAME     | mark   |
+----+----------+--------+
|  1 | Ramesh   |   89   |
|  3 | kaushik  |   73   |
+----+----------+--------+
```

Ques. IN and NOT IN Operator?

IN Operator

- The IN operator is a shorthand for multiple OR conditions, It reduces the use of multiple OR conditions in SELECT, INSERT, UPDATE, and DELETE queries.
- The IN operator is used to retrieves results when the specified value matches any value in a set of values or is returned by a subquery.
- This operator allows us to specify multiple values along with the WHERE clause.

```
select * from customers
+----+----------+-----+-----------+-----------+
| cust_id | cust_name | city      | occupation|
+---------+-----------+-----------+-----------+
|  1      | Peter     | Londen    | Business  |
|  2      | Joseph    | Texas     | Doctor    |
|  3      | Mark      | New Delhi | Engineer  |
|  4      | Michael   | New York  | Scientist |
|  5      | Alexander | Maxico    | Student   |
+---------+-----------+-----------+-----------+
mysql> SELECT * FROM customer WHERE occupation IN ('Doctor', 'Scientist',
'Engineer');

+----+----------+-----+-----------+-----------+
| cust_id | cust_name | city      | occupation|
+---------+-----------+-----------+-----------+
|  2      | Joseph    | Texas     | Doctor    |
|  3      | Mark      | New Delhi | Engineer  |
|  4      | Michael   | New York  | Scientist |
+---------+-----------+-----------+-----------+
```

NOT IN Operator

- selects all customers that are located in "Texas", or "New York":

```
SELECT * FROM Customers WHERE city NOT IN ('Texas', 'New York');

Output:-
+----+----------+-----+-----------+-----------+
| cust_id | cust_name | city      | occupation|
+---------+-----------+-----------+-----------+
|  1      | Peter     | Londen    | Business  |
|  3      | Mark      | New Delhi | Engineer  |
|  5      | Alexander | Maxico    | Student   |
+---------+-----------+-----------+-----------+
```

Ques. BETWEEN and NOT BETWEEN Operator?

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.

```
SELECT column_name(s) FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

NOT BETWEEN

- To display the products outside the range of the previous example, use NOT BETWEEN:

```
SELECT column_name(s) FROM table_name
WHERE column_name NOT BETWEEN value1 AND value2;
```

Ques. Difference between Group By And Order By?

```
| id  | product  | category  | quantity | price |
| --- | -------- | --------- | -------- | ----- |
| 1   | Apple    | Fruit     | 10       | 1.00  |
| 2   | Banana   | Fruit     | 5        | 0.50  |
| 3   | Carrot   | Vegetable | 7        | 0.30  |
| 4   | Apple    | Fruit     | 3        | 1.00  |
| 5   | Broccoli | Vegetable | 4        | 0.80  |
```

Order By

- ORDER BY clause is used to **sort the data** returned by a query in **ascending** or **descending** order.

```
SELECT product, category, quantity, price FROM sales ORDER BY price DESC;
| product  | category  | quantity | price |
| -------- | --------- | -------- | ----- |
| Apple    | Fruit     | 10       | 1.00  |
| Apple    | Fruit     | 3        | 1.00  |
| Broccoli | Vegetable | 4        | 0.80  |
| Banana   | Fruit     | 5        | 0.50  |
| Carrot   | Vegetable | 7        | 0.30  |
```

Group By

- Group by statement is used to group the rows that have the same value.
- It is used with aggregate functions for example AVG(), COUNT(), SUM()etc.

```
SELECT category, SUM(quantity) AS total_quantity
FROM sales GROUP BY category;
| category  | total\_quantity |              |
| --------- | --------------- | ------------ |
| Fruit     | 18              | (10 + 5 + 3) |
| Vegetable | 11              | (7 + 4)      |
```

```
-- You Can Use Both Together
SELECT category, SUM(quantity) AS total_quantity
FROM sales
GROUP BY category
ORDER BY total_quantity DESC;

| category  | total\_quantity |
| --------- | --------------- |
| Fruit     | 18              |
| Vegetable | 11              |
```

Difference between WHERE and HAVING in SQL?

Where

- WHERE Clause is used to **filter** the records from the table or used while joining more than one table.
- Cannot be used with aggregate functions (like SUM(), COUNT(), AVG(), etc.).

```
SELECT * FROM emp WHERE salary > 50000;
```

HAVING

- HAVING Clause is used to filter the records from the groups based on the given condition in the HAVING Clause.
- It is applied after the grouping and aggregation of data.

```
SELECT department, COUNT(*) AS num_empl FROM employees
GROUP BY department
HAVING COUNT(*) > 10;
```

Having	Where
Having ke sath GROUP BY use hota hai	
Having post filter hai(data fatch hone ke baad filter lagta hai)	where pre filter hai(isme pahle filter lagta hai phir fatch data hota hai)
having can be used only with select command	can be used with select update delete
HAVING is used for column operations.	WHERE is used for row operations
having ke aggrigate function sath kar sakte hai	where ke sath aggrigate function use nahi kar sakte

What is Aggregate function?

```
-- Sum() :- The SUM() function returns the total sum of a numeric column.
SELECT SUM(column_name) FROM table_name;

-- AVG():- The AVG() function returns the average value of a numeric column.
SELECT AVG(column_name) FROM table_name;

-- MAX() :- The MAX() function returns the largest value of the selected column.
SELECT MAX(column_name) FROM table_name;

-- Min():- The MIN() function returns the smallest value of the selected column.
SELECT MIN(column_name) FROM table_name;

-- count():- The COUNT() function returns the number of rows that matches a
specified criterion.
SELECT COUNT(column_name) FROM table_name;
```

SQL Operators and Clauses

Truncate

- A truncate SQL statement is used to **remove all rows** (complete data) from a table.
- TRUNCATE is a **DDL**(Data Definition Language) command and is used to delete all the rows or tuples from a table. Unlike the DELETE command, the TRUNCATE command does not contain a WHERE clause.

```
TRUNCATE TABLE table_name;
```

LIKE

- LIKE is a special operator used with the WHERE clause to search for a specific pattern in a column.

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern;
```

INNER JOIN

- An inner join will combine rows from different tables if the join condition is true.

```
SELECT column_name(s)
FROM table_1
JOIN table_2
  ON table_1.column_name = table_2.column_name;
```

OUTER JOIN

- An outer join will combine rows from different tables even if the join condition is not met. Every row in the left table is returned in the result set, and if the join condition is not met, then NULL values are used to fill in the columns from the right table.

```
SELECT column_name(s)
FROM table_1
LEFT JOIN table_2
  ON table_1.column_name = table_2.column_name;
```

IF()

```
SELECT IF(200>350,'YES','NO'); -- Output:- NO
SELECT IF(251 = 251,' Correct','Wrong');  -- Output:- Correct
```

```
--
SELECT salary, IF(salary>3000,"Mature","Immature") As Result FROM employee;
+---------+----------+
| salary  | Result   |
+---------+----------+
| 2957.00 | Immature |
| 3100.00 | Mature   |
+---------+----------+
```

```
SELECT IF(STRCMP('Rinky Ponting','Yuvraj Singh')=0, 'Correct', 'Wrong');
```

IFNULL()

- If the first expression is not NULL, it will return the first expression, which is 'Hello' value.

```
SELECT IFNULL("Hello", "javaTpoint");   # Output:- Hello
```

```
SELECT IFNULL(NULL,5);  # Output:- 5
```

```
+-----------+-----------+-----------+
| emp_name  | cellphone | homephone |
+-----------+-----------+-----------+
| mohit     | 2531452   |  NULL     |
| Krishna   | NULL      | 345634634 |
| shivani   | 551113    | NULL      |
| akshra    | NULL      | 6574576   |
| abhinav   | NULL      | 674576457 |
+-----------+-----------+-----------+
SELECT emp_name, IFNULL(cellphone, homephone) phone  FROM  student_contact;
+-----------+-----------+
| emp_name  | phone     |
+-----------+-----------+
| mohit     | 2531452   |
| Krishna   | 345634634 |
| shivani   | 551113    |
| akshra    | 6574576   |
| abhinav   | 674576457 |
+-----------+-----------+
```

NULLIF()

- The NULLIF function accepts two expressions, and if the first expression is equal to the second expression, it returns the NULL. Otherwise, it returns the first expression.

```
SELECT NULLIF("javaTpoint", "javaTpoint");  # Output:- NULL
```

```
SELECT NULLIF("Hello", "404");  # Output:- Hello
```

```
+---------+---------------+-------------+--------+---------------+
| cust_id | cust_name     | occupation  | income | qualification |
+---------+---------------+-------------+--------+---------------+
|       1 | John Miller   | Developer   | 20000  | Btech         |
|       2 | Mark Robert   | Engineer    | 40000  | Btech         |
|       3 | Reyan Watson  | Scientist   | 60000  | MSc           |
|       4 | Shane Trump   | Businessman | 10000  | MBA           |
|       5 | Adam Obama    | Manager     | 80000  | MBA           |
|       6 | Rincky Ponting | Cricketer  | 200000 | Btech         |
+---------+---------------+-------------+--------+---------------+


SELECT cust_name, occupation, qualification, NULLIF (qualification,"Btech") result
FROM customer;
Output:-
+---------------+-------------+---------------+--------+
| cust_name     | occupation  | qualification | result |
+---------------+-------------+---------------+--------+
| John Miller   | Developer   | Btech         | NULL   |
| Mark Robert   | Engineer    | Btech         | NULL   |
| Reyan Watson  | Scientist   | MSc           | MSc    |
| Shane Trump   | Businessman | MBA           | MBA    |
| Adam Obama    | Manager     | MBA           | MBA    |
| Rincky Ponting | Cricketer  | Btech         | NULL   |
+---------------+-------------+---------------+--------+
```

IS NULL and IS NOT NULL

- IS NULL and IS NOT NULL are operators used with the WHERE clause to test for empty values.

```
SELECT column_name(s)
FROM table_name
WHERE column_name IS NULL;
```

ROUND()

- The ROUND() function is used to round a numeric value to a specified number of decimal places.*
- syntex:- syntex:- ROUND(number, decimal_places)

```
SELECT ROUND(123.4567, 2);  -- Returns 123.46
SELECT ROUND(123.4567, 0);  -- Returns 123
SELECT ROUND(123.4567, -1); -- Returns 120 (rounds to the nearest 10)

-- example:-
SELECT ROUND(salary, 2) AS rounded_salary FROM employees;
```

BETWEEN()

- The BETWEEN operator is used to filter the result set within a certain range. The values can be numbers, text or dates.

```
SELECT column_name(s) FROM table_name
WHERE column_name BETWEEN value_1 AND value_2;
```

AND

- AND is an operator that **combines two conditions**. **Both conditions must be true** for the row to be included in the result set.
- The MySQL AND Condition (also called the AND Operator) is used to test two or more conditions in a SELECT, INSERT, UPDATE, or DELETE statement.

```
SELECT column_name(s)
FROM table_name
WHERE column_1 = value_1
  AND column_2 = value_2;

SELECT * FROM contacts
WHERE state = 'California'
AND contact_id > 3000;
```

OR

- OR is an operator that filters the result set to only include rows where either condition is true.

```
SELECT column_name
FROM table_name
WHERE column_name = value_1
   OR column_name = value_2;
```

Case

- CASE statements are used to create different outputs (usually in the SELECT statement). It is SQL's way of handling if-then logic.

```
SELECT column_name,
  CASE
    WHEN condition THEN 'Result_1'
    WHEN condition THEN 'Result_2'
    ELSE 'Result_3'
  END
FROM table_name;
```

GROUP BY

- GROUP BY is a clause in SQL that is only used with aggregate functions. It is used in collaboration with the SELECT statement to arrange identical data into groups.

```
SELECT column_name, COUNT(*)
FROM table_name
GROUP BY column_name;
```

HAVING

- HAVING was added to SQL because the WHERE keyword could not be used with aggregate functions.

```
SELECT column_name, COUNT(*)
FROM table_name
GROUP BY column_name
HAVING COUNT(*) > value;
```

LIMIT

- LIMIT is a clause that lets you specify the maximum number of rows the result set will have.

```
SELECT column_name(s)
FROM table_name
LIMIT number;
```

ORDER BY

- ORDER BY is a clause that indicates you want to sort the result set by a particular column either alphabetically or numerically.

```
SELECT column_name
FROM table_name
ORDER BY column_name ASC | DESC;
```

SELECT DISTINCT

- SELECT DISTINCT specifies that the statement is going to be a query that returns unique values in the specified column(s).

```
SELECT DISTINCT column_name
FROM table_name;
```

With

- WITH clause lets you store the result of a query in a temporary table using an alias. You can also define multiple temporary tables using a comma and with one instance of the WITH keyword.
- The WITH clause is also known as common table expression (CTE) and subquery factoring.

```
WITH temporary_name AS (
   SELECT *
   FROM table_name)
SELECT *
FROM temporary_name
WHERE column_name operator value;
```

WHERE

- WHERE is a clause that indicates you want to filter the result set to include only rows where the following condition is true.

```
SELECT column_name(s)
FROM table_name
```

```
WHERE column_name operator value;
```

Keys

Primary Key?

- A PRIMARY KEY is a column or combination of columns that uniquely identifies each record in a database table.
- A Primary Key column cannot have Null values.
- A table can have only one primary key per table.
- When multiple fields are used as a primary key, they are called a composite key.

```
-- Create Primary Key
CREATE TABLE Students (
    student_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(100)
);
-- (OR)
CREATE TABLE Students (
    student_id INT AUTO_INCREMENT,
    name VARCHAR(50),
    email VARCHAR(100),
    PRIMARY KEY (student_id)
);

-- Create Primary Key with multiple column
CREATE TABLE Order_Items (
    order_id INT,
    product_id INT,
    quantity INT,
    PRIMARY KEY (order_id, product_id)  -- Multiple columns as primary key
);
```

Add primary Key

```
-- if primary key doesn't exists in the created table
ALTER TABLE table_name ADD PRIMARY KEY (Id)

-- For multiple column
ALTER table Employee ADD constraints PK_Employee PRIMARY KEY (column_name1,
column_name2);
-- (OR)
ALTER TABLE table_name ADD PRIMARY KEY (column1, column2);

-- Adding Primary Key with Auto-Increment
ALTER TABLE table_name MODIFY column_name INT AUTO_INCREMENT PRIMARY KEY;
```

Delete primary Key

```
ALTER TABLE table_name DROP PRIMARY KEY;

-- For multiple column
ALTER TABLE Employee DROP CONSTRAINT PK_Employee;
-- (OR)
ALTER TABLE table_name DROP PRIMARY KEY;
```

Ques. What Is Unique Key?

- A Unique Key is a constraint that ensures all values in a column or a combination of columns are unique across all records in a table.
- The Unique and Primary Key constraints both provide a guarantee for a column or set of columns.
- A Primary Key consist automatically has a unique constraint define on it.
    - Defining unique key

```
-- Single Column Unique Key
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    email VARCHAR(100) UNIQUE
);

-- Multiple Column Unique Key
CREATE TABLE Employees (
    id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    UNIQUE (first_name, last_name)
);
```

ALTER unique key

```
-- Single Column
ALTER table Employee ADD UNIQUE(column name);
-- (OR)
ALTER TABLE table_name ADD CONSTRAINT constraint_name UNIQUE (column_name);

-- Multiple Columns
ALTER TABLE table_name ADD CONSTRAINT constraint_name UNIQUE (column1, column2);
```

Drop unique key

```
ALTER TABLE Employee DROP CONSTRAINT Employee_ID;
-- (OR)
ALTER TABLE table_name DROP INDEX constraint_name;
```

Ques. What Is Foreign Key?

- A foreign key is a key used to link two tables together. This is something called a reference key.
- A column or set of columns in a table that references the PRIMARY KEY of another table.
- Foreign key is a column or a combination of columns whose values match a primary key in a different table.
- The relationship between two tables matches the primary key in one of the tables with a foreign key in the second table.

```
-- create Customers table
CREATE TABLE Customers (
  id INTEGER PRIMARY KEY,
  name VARCHAR(100),
  age INTEGER
);

-- create Products table
CREATE TABLE Products (
    customer_id INTEGER ,
    name VARCHAR(100),
    FOREIGN KEY (customer_id)
    REFERENCES Customers(id)
);

-- Here, the customer_id column in the Products table references the id column in
the Customers table.
```

```
-- One-to-One:- Each record in one table connects to single record in another
CREATE TABLE Employee (
    emp_id INT PRIMARY KEY,
    passport_number INT,
    FOREIGN KEY (passport_number)
    REFERENCES Passport(passport_number)
);

-- One-to-Many:- One record in parent table can relate to multiple records in
child table
CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    FOREIGN KEY (customer_id)
    REFERENCES Customers(customer_id)
);

-- Many-to-Many:- Multiple records in both tables can relate to each other
CREATE TABLE StudentCourses (
    student_id INT,
    course_id INT,
    PRIMARY KEY (student_id, course_id),
```

```
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

Adding Foreign Key to Existing Table?

```
-- if primary key doesn't exists in the created table
ALTER TABLE Employee ADD FOREIGN KEY (department_id) REFERENCES
Department(department_id);
(OR)
ALTER TABLE `bookings` ADD CONSTRAINT `advance_bookings_user_id_foreign` FOREIGN
KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE

-- For multiple column
ALTER TABLE Employee ADD CONSTRAINT FK_dept_id FOREIGN KEY (department_id)
REFERENCES Department(department_id);
```

DROP a Foreign Key from the table

```
-- For single column/multiple column
ALTER TABLE Employee DROP FOREIGN KEY FK_dept_id;
```

Ques. What is Composite Key?

- Composite key is combination of two or more columns that can uniquely identify each row in the table.
- composite key is also a primary key, but the difference is that it is made by the combination of more than one column to identify the particular row in the table.
- A composite key cannot be null.

```
CREATE TABLE student
(rollNumber INT,
name VARCHAR(30),
class VARCHAR(30),
section VARCHAR(1),
mobile VARCHAR(10),
PRIMARY KEY (rollNumber, mobile));
```

Types of Composite Keys

1. Composite Primary Key

```
CREATE TABLE StudentCourses (
    student_id INT,
    course_id INT,
    semester VARCHAR(10),
    PRIMARY KEY (student_id, course_id, semester)
);
```

2. Composite Unique Key

```
CREATE TABLE Employees (
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department VARCHAR(50),
    UNIQUE (first_name, last_name, department)
);
```

Ques. Difference between Primary Key & Unique Key?

Primary Key	Unique Key
A table can have only one primary key	A table can have more than one unique key
It does not allow null values	Allows null values
Primary key Can be made foreign key into another table	In SQL server, unique key Can be made foreign key into another table
By default it adds a clustered index	By default it adds a unique non-clustered index
Primary key support auto increment value.	Unique constraint does not support auto increment value.

Ques. Difference between Primary Key & Foreign Key?

Primary Key	Foreign Key
A table can have only one primary key	A table can have more than one foreign key
Primary key uniquely identified a record in the table	Foreign key is a field in the table that is primary key in another table
It does not allow null values	Allows null values
Duplicate not allowed	Duplicate allowed
Primary key support auto increment value	Foreign key do not automatically create an index.

Joins

Ques. What Is Joins?

- A JOIN is a method used to combine rows from two or more tables based on a related column between them.
- MySQL JOINS are used with SELECT statement.

Many types of MySQL joins:

1. Self Join
2. Inner Join
3. Left JOIN
4. Right JOIN
5. Full Join
6. Outer Join
7. Cross Join

Self Join

- A self join connects a table to itself. Used when you want to compare rows within the same table.

```
Customers table:
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+

-- Example
SELECT c1.NAME AS Customer1, c2.NAME AS Customer2, c1.SALARY FROM Customers c1
JOIN Customers c2 ON c1.SALARY = c2.SALARY AND c1.ID < c2.ID;
-- Output:
+------------+------------+----------+
| Customer1  | Customer2  | SALARY   |
+------------+------------+----------+
| Ramesh     | kaushik    | 2000.00  |
+------------+------------+----------+
```

INNER JOIN

- The MySQL Inner Join is used to returns only those results from the tables that **match** the specified condition and hides other rows and columns.
- Inner join: Inner join return rows when there is at least one match of rows between the tables.

```
Customers table:
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
Order table:
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+

SELECT customers.name, customers.age, customers.salary, order.date
FROM customers INNER JOIN order
ON customers.id = order.CUSTOMER_ID;
+----------+-----+---------+---------------------+
| NAME     | AGE | SALARY  |   DATE              |
+----------+-----+---------+---------------------+
| Khilan   |  25 | 1500.00 | 2009-11-20 00:00:00 |
| Chaitali |  25 | 6500.00 | 2008-05-20 00:00:00 |
+----------+-----+---------+---------------------+

-- Example 2
Order table:
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+

-- Output:-
SQL> SELECT ID, NAME, AMOUNT, DATE   FROM CUSTOMERS
INNER JOIN ORDERS
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
|  3 | kaushik  |  23 |   3000 |
|  3 | kaushik  |  23 |   1500 |
|  2 | Khilan   |  25 |   1560 |
|  4 | Chaitali |  25 |   2060 |
+----+----------+-----+--------+
```

Left JOIN/LEFT OUTER JOIN

- The LEFT JOIN keyword returns all records from the left table (the table listed first), and the matching records (if any) from the right table (table2).

```
CUSTOMERS Table
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+

Orders Table
+-----+---------------------+-------------+--------+
| OID | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+

SQL> SELECT  ID, NAME, AMOUNT, DATE FROM CUSTOMERS LEFT JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

Result:-
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
|  1 | Ramesh   |   NULL | NULL                |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|  5 | Hardik   |   NULL | NULL                |
|  6 | Komal    |   NULL | NULL                |
|  7 | Muffy    |   NULL | NULL                |
+----+----------+--------+---------------------+
```

Right JOIN

- The RIGHT JOIN keyword returns **all records** from the **right table** (table2), and the **matching records** (if any) from the **left table** (table1).

```
-- Customers Table
+----+----------+
| ID | NAME     |
+----+----------+
|  1 | Ramesh   |
|  2 | Khilan   |
|  3 | Kaushik  |

-- Orders Table
+----------+------------+
| OrderID  | CustomerID |
+----------+------------+
|   101    |     2      |
|   102    |     3      |
|   103    |     4      |
+----------+------------+

-- RIGHT JOIN Result
SELECT Customers.NAME, Orders.OrderID FROM Customers RIGHT JOIN Orders
ON Customers.ID = Orders.CustomerID;
+----------+------------+
| NAME     | OrderID    |
+----------+------------+
| Khilan   |   101      |
| Kaushik  |   102      |
| NULL     |   103      |
+----------+------------+
```

Outer join

- Returns all rows from both tables, Matching and non-matching rows.
- NULL values where no match exists

```
SELECT column_list
FROM table1
FULL OUTER JOIN table2 ON table1.column = table2.column;
```

```
-- Example query
employees:

| employee_id | employee_name | department_id |
| ----------- | ------------- | ------------- |
| 1           | John Smith    | 101           |
| 2           | Mary Johnson  | 102           |
| 3           | Sam Brown     | 103           |

departments:

| department_id | department_name |
| ------------- | --------------- |
| 101           | HR              |
| 102           | Finance         |
| 104           | Marketing       |

SELECT employees.employee_id, employees.employee_name, departments.department_name
FROM employees
FULL OUTER JOIN departments ON employees.department_id =
departments.department_id;

| employee_id | employee_name | department_name |
| ----------- | ------------- | --------------- |
| 1           | John Smith    | HR              |
| 2           | Mary Johnson  | Finance         |
| 3           | Sam Brown     | NULL            |
| NULL        | NULL          | Marketing       |
```

CROSS Join

- The CROSS JOIN keyword returns all records from both tables (table1 and table2).
- If you have a Products table with 3 products and a Colors table with 2 colors, a CROSS JOIN would return a result set with 6 combinations (3 * 2):

```
-- Products table
| ProductID | ProductName |
| --------- | ----------- |
| 1         | T-Shirt     |
| 2         | Jeans       |

-- Colors table
| ColorID | Color |
| ------- | ----- |
| 1       | Red   |
| 2       | Blue  |

-- Output:-
SELECT Products.ProductName, Colors.Color
FROM Products
CROSS JOIN Colors;

| ProductName | Color |
| ----------- | ----- |
| T-Shirt     | Red   |
| T-Shirt     | Blue  |
| Jeans       | Red   |
| Jeans       | Blue  |
```

Full Join/FULL OUTER JOIN

- A FULL JOIN (also called a FULL OUTER JOIN) returns all rows when there is a match in either left (table1) or right (table2) table. It returns all records from both tables, and the result set will have NULL values for columns where there is no match.

Key Points:

- Rows from both tables are included even if there is no match.
- If there is no match, the columns from the table that doesn't have a match will contain NULL.
- This join is useful when you want to retrieve all records, whether or not there's a match in both tables.

```
-- Employees Table:
| EmployeeID | EmployeeName | DepartmentID |
| ---------- | ------------ | ------------ |
| 1          | John         | 10           |
| 2          | Jane         | 20           |
| 3          | Mike         | 30           |
| 4          | Sara         | NULL         |

-- Departments Table:
| DepartmentID | DepartmentName |
| ------------ | -------------- |
| 10           | HR             |
| 20           | IT             |
| 30           | Sales          |
| 40           | Marketing      |

-- Output:-
SELECT Employees.EmployeeID, Employees.EmployeeName, Employees.DepartmentID,
Departments.DepartmentName
FROM Employees
FULL JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID;
| EmployeeID | EmployeeName | DepartmentID | DepartmentName |
| ---------- | ------------ | ------------ | -------------- |
| 1          | John         | 10           | HR             |
| 2          | Jane         | 20           | IT             |
| 3          | Mike         | 30           | Sales          |
| 4          | Sara         | NULL         | NULL           |
| NULL       | NULL         | 40           | Marketing      |
```

Index

What is Index?

- An index is used to enhance the performance of SQL Queries. It allows the database to find data quickly and efficiently by using Row ID, avoiding full table scans.
- Indexes can be created on one or more columns of a table.
- Index allows the database application to find data fast, without reading the whole table.
- An index can be created in a table to find data more quickly and efficiently.
- **How It Works:-** Behind the scenes, an index is usually implemented as a B-tree or similar structure.

Types of Indexes?

1. **Single-column index:-** on one column

```
CREATE INDEX idx_product_id ON Sales (product_id);
```

2. **Composite/Multi-column indexes:** on multiple columns

```
CREATE INDEX idx_dept_salary ON employees(department, salary);
```

3. Unique Index:-

```
CREATE UNIQUE INDEX idx_email_unique ON employees(email);
```

4. Full-Text Index :- Designed for efficient text search in large text fields (e.g., for searching keywords in articles, product descriptions, etc.)

```
CREATE FULLTEXT INDEX idx_description ON products(description);
```

Show Index

```
show index from table_name
```

Unique Indexes

- A Unique Index is a database index that ensures the uniqueness of values in one or more columns of a database table.

- This index ensures that no two rows in the Employees table have the same employee_id(Colum_name), which maintains data integrity and prevents duplicate entries.
- The SQL Unique Index ensures that no two rows in the indexed columns of a table have the same values (no duplicate values allowed).

```
* Behind the scenes, when a new record is inserted, the database would:
  * Check the Index: It looks up the email in the index (which could be a B-tree
or hash table) to find if that email already exists.
    * The database will search the index, and it will quickly identify whether the
email you're trying to insert is already in the table.
  * Check for Duplicates:
    * For example, if we try to insert a new row:
    * INSERT INTO users (Name, Email) VALUES ('David', 'alice@example.com');
    * The database will search the unique index for the email alice@example.com.
Since this email already exists in the index, the insert will fail with an error
like:
    "Duplicate entry 'alice@example.com' for key 'idx_unique_email'".
```

```
-- Single column unique index
CREATE UNIQUE INDEX idx_email ON users(email);

-- Composite unique index
CREATE UNIQUE INDEX idx_name_age ON employees(last_name, first_name);
```

Alter/Modify an Index

```
-- Rename Index name
ALTER TABLE table_name
RENAME INDEX old_index_name TO new_index_name;

-- Modify Index name
ALTER TABLE table_name DROP INDEX existing_index,
ADD INDEX new_index (column1, column2);

-- Rebuild Index
ALTER INDEX index_name
ON table_name REBUILD;

-- Reorganize Index
ALTER INDEX index_name
ON table_name REORGANIZE;

-- Disable Index
ALTER INDEX index_name
ON table_name DISABLE;

-- Change Index Properties
```

```
ALTER INDEX index_name
ON table_name
SET (
    STATISTICS_NORECOMPUTE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON
);
```

Drop Index

```
Drop index index_name on table_name
(OR) DROP INDEX table_name.index_name;

-- DROP INDEX with IF EXISTS
DROP INDEX IF EXISTS index_name
ON table_name;


ALTER TABLE table_name DROP CONSTRAINT constraint_name;
```

Cluster Index

- A clustered index is an index where the data in the table is physically stored in the order of the indexed column. In other words, the rows of the table are sorted on disk based on the values in the indexed column. A table can have only one clustered index because the data can only be sorted in one way.
- **Example:** If you create a clustered index on the EmployeeID column, the table rows will be stored on disk in ascending order of EmployeeID. The clustered index itself dictates the physical arrangement of the data.
- jab kisi table par hum primary key lagate hai to wo cluster index ban jata hai.

Non cluster index

- A non-clustered index is an index where the data in the table is stored independently of the index. It creates a separate structure that contains the indexed column(s) and pointers to the actual rows in the table. A table can have multiple non-clustered indexes, allowing for efficient lookups on different columns without changing the physical order of the data.
- **Example:** If you create a non-clustered index on the Department column, the index will store the department values along with pointers to the corresponding rows in the table, but the table data itself remains in its original order.
- table mai jis column par select command sabse jyda chalte hai to us column par hum non cluster index bna lete hai.

Ques. What is the difference between cluster and non cluster index?

Cluster index	Non cluster index

Cluster index	Non cluster index
Data rows are stored in the order of the index.	Data rows are not sorted in any particular order.
Only one clustered index per table.	Multiple non-clustered indexes can exist per table.
Created by default for the primary key.	Must be explicitly created (e.g., for specific queries).

View

What is View?

- A view is a **virtual table** based on the result set of a SELECT query.
- It does not store data itself but provides a way to look at data from one or more tables in a structured and reusable manner.
- It **behaves** like a **table** but **doesn't store** the **data physically**.

Create view

```
Create view view_name As
Select column1, column2
From  table_name
Where [condition];
```

Show view

```
SELECT * FROM view_name;
```

Alter view

- CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Deleted view

- A view is **deleted** with the **DROP VIEW statement**.

```
DROP VIEW view_name
DROP VIEW IF EXISTS view_name;
```

Ques. What is ACID property/SQL TRANSACTIONS?

- A transaction in SQL is a **sequence** of one or more SQL operations that are executed as a single unit.
- The goal of a transaction is to ensure that either all operations succeed or none of them do, maintaining the consistency of the database. Think of it like: "Do everything, or do nothing."
- (Transaction एक तरह का ब्लॉक है जिसमें कई SQL statements (जैसे INSERT, UPDATE, DELETE) एक साथ execute होते हैं। इसका मतलब है: या तो सारे काम पूरे होंगे, या कोई भी नहीं होगा।)

ACID Properties

1. **A - Atomicity:-** If all operations in a transaction succeed, the changes are saved to the database; if any operation fails, the entire transaction is rolled back.
2. **C - Consistency:-** Ensures that after a transaction, the data in the database is still correct and follows all rules.
3. **I - Isolation:-** Makes sure that when many transactions run at the same time, they don't interfere with each other.(Ensures that concurrent transactions do not affect each other. The final result should be the same as if transactions were run sequentially.)
4. **D - Durability:-** Once a transaction is committed, it is permanently saved in the database—even in case of system failure or crashes.

```
BEGIN TRANSACTION;

UPDATE accounts SET balance = balance - 100 WHERE account_id = 1;
UPDATE accounts SET balance = balance + 100 WHERE account_id = 2;

COMMIT;

-- If any of those updates fail (e.g., account not found), you can use:
ROLLBACK;
```

Wildcard Characters/Like Query

Ques. Wildcard Characters?

- Wildcard characters are used with the **LIKE operator**. The LIKE operator is **used** in a **WHERE** clause to search for a specified pattern in a column.

Symbol	Description
%	Represents zero or more characters
_	Represents a single character

Some Example

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%_%'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

```
SELECT * FROM Customers WHERE City LIKE 'ber%';
```

Aliases?

- AS is a keyword in SQL that allows you to rename a column or table using an alias.
- Aliases are used to give a table, or a column in a table, a temporary name.
- An alias is created with the **AS** keyword.

Alias Column Syntax:-

```
SELECT column_name AS alias_name FROM table_name;

-- Basic Column Alias
SELECT first_name AS name, last_name AS surname FROM employees;

-- Using quotes for aliases with spaces
SELECT
    first_name AS 'Employee First Name',
    last_name AS 'Employee Last Name',
    salary AS 'Monthly Compensation'
FROM
    employees;

-- Without AS keyword
SELECT first_name name, last_name surname FROM employees;

-- Simple Table Alias
SELECT e.first_name, e.last_name, d.department_name
FROM employees e JOIN departments d
ON e.department_id = d.department_id;

-- Calculation with Alias
SELECT
    product_name,
    price * quantity AS total_value,
    (price * quantity) * 1.1 AS total_with_tax
FROM products;

-- Concatenation Alias
SELECT
    CONCAT(first_name, ' ', last_name) AS full_name,
    email AS contact_email
FROM customers;

-- Subquery with Alias
SELECT
    (SELECT AVG(salary) FROM employees) AS avg_salary,
    (SELECT MAX(salary) FROM employees) AS max_salary;
```

```sql
-- Aggregate Function Aliases
SELECT
    department_id,
    AVG(salary) AS average_salary,
    COUNT(*) AS employee_count,
    MAX(salary) AS highest_salary
FROM employees
GROUP BY department_id;

-- Conditional Aliases
SELECT
    first_name,
    last_name,
    CASE
        WHEN salary < 50000 THEN 'Junior'
        WHEN salary BETWEEN 50000 AND 100000 THEN 'Mid-Level'
        ELSE 'Senior'
    END AS salary_category
FROM employees;
```

Ques. What Is Union & Union All?

- Both UNION and UNION ALL Operator combine rows from result sets into a single result set.

UNION

- The union operator **combines** the results of two or more select statements by **removing duplicate rows**.
- The columns and the data types must be the same in select statements.

```
+----+----------+
| ID | NAME     |
+----+----------+
|  1 | Ramesh   |
|  2 | Khilan   |
|  3 | kaushik  |
+----+----------+

+----+----------+
| ID | NAME     |
+----+----------+
|  3 | kaushik  |
|  4 | Mohit    |
|  5 | abhay    |
+----+----------+

Select Column1, Column2, Column3 from Table A
UNION
Select Column1, Column2, Column3 from Table B

+----+----------+
| ID | NAME     |
+----+----------+
|  1 | Ramesh   |
|  2 | Khilan   |
|  3 | kaushik  |
|  4 | Mohit    |
|  5 | abhay    |
+----+----------+
```

UNION ALL

- The UNION operator selects only distinct values by default. To **allow duplicate values**, use UNION ALL

```
Select Column1, Column2, Column3 from Table A
UNION ALL
Select Column1, Column2, Column3 from Table B

+----+----------+
| ID | NAME     |
+----+----------+
|  1 | Ramesh   |
|  2 | Khilan   |
|  3 | kaushik  |
|  3 | kaushik  |
|  4 | Mohit    |
|  5 | abhay    |
+----+----------+
```

Ques. Difference between Union & Union All?

Union	Union All
Union removes duplicate rows.	Union All does not remove the duplicate rows.
Union uses a distinct sort	Union All does not use a distinct sort
Union can’t work with a column that has a text data type.	Union All can work with all data type column.

What is Intersect?

- The INTERSECT statement will return only those rows that are **identical/common** to both of the SELECT statements from two or more tables

```
-- Employees                          -- Managers
| EmpID | EmpName | Department |   | EmpID | EmpName | Department |
| ----- | ------- | ---------- || ----- | ------- | ---------- |
| 1     | Alice   | HR         |     | 2     | Bob     | IT         |
| 2     | Bob     | IT         |     | 4     | David   | IT         |
| 3     | Charlie | Finance    |     | 6     | Frank   | Sales      |
| 4     | David   | IT         |     | 7     | Grace   | Marketing  |
| 5     | Eve     | Marketing  |

-- Output
SELECT EmpID, EmpName, Department FROM Employees
INTERSECT
SELECT EmpID, EmpName, Department FROM Managers;
| EmpID | EmpName | Department |
| ----- | ------- | ---------- |
| 2     | Bob     | IT         |
| 4     | David   | IT         |
```

What is MINUS?

- MINUS operator will return only those rows which are **unique** in only first SELECT query and not those rows which are common to both first and second SELECT queries.

```
-- Employees                          -- Managers
| EmpID | EmpName | Department |   | EmpID | EmpName | Department |
| ----- | ------- | ---------- || ----- | ------- | ---------- |
| 1     | Alice   | HR         |     | 2     | Bob     | IT         |
| 2     | Bob     | IT         |     | 4     | David   | IT         |
| 3     | Charlie | Finance    |     | 6     | Frank   | Sales      |
| 4     | David   | IT         |     | 7     | Grace   | Marketing  |
| 5     | Eve     | Marketing  |

-- output:-
SELECT EmpID, EmpName, Department FROM Employees
MINUS
SELECT EmpID, EmpName, Department FROM Managers;
| EmpID | EmpName | Department |
| ----- | ------- | ---------- |
| 1     | Alice   | HR         |
| 3     | Charlie | Finance    |
| 5     | Eve     | Marketing  |
```

Ques. Optimizing SQL Queries for Faster Performance?

- Select Only Needed Columns:- Select id, name
- Filter Early with WHERE

Demo data for execute the query

```
CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50),
    job_name VARCHAR(50),
    manager_id INT,
    hire_date DATE,
    salary DECIMAL(10, 2),
    commission DECIMAL(10, 2),
    dep_id INT
);

INSERT INTO employee (emp_id, emp_name, job_name, manager_id, hire_date, salary,
commission, dep_id)
VALUES
    (68319, 'KAYLING', 'PRESIDENT', NULL, '1991-11-18', 6000.00, NULL, 1001),
    (66928, 'BLAZE', 'MANAGER', 68319, '1991-05-01', 2750.00, NULL, 3001),
    (67832, 'CLARE', 'MANAGER', 68319, '1991-06-09', 2550.00, NULL, 1001),
    (65646, 'JONAS', 'MANAGER', 68319, '1991-04-02', 2957.00, NULL, 2001),
    (67858, 'SCARLET', 'ANALYST', 65646, '1997-04-19', 3100.00, NULL, 2001),
    (69062, 'FRANK', 'ANALYST', 65646, '1991-12-03', 3100.00, NULL, 2001),
    (63679, 'SANDRINE', 'CLERK', 69062, '1990-12-18', 900.00, NULL, 2001),
    (64989, 'ADELYN', 'SALESMAN', 66928, '1991-02-20', 1700.00, 400.00, 3001),
    (65271, 'WADE', 'SALESMAN', 66928, '1991-02-22', 1350.00, 600.00, 3001),
    (66564, 'MADDEN', 'SALESMAN', 66928, '1991-09-28', 1350.00, 1500.00, 3001),
    (68454, 'TUCKER', 'SALESMAN', 66928, '1991-09-08', 1600.00, 0.00, 3001),
    (68736, 'ADNRES', 'CLERK', 67858, '1997-05-23', 1200.00, NULL, 2001),
    (69000, 'JULIUS', 'CLERK', 66928, '1991-12-03', 1050.00, NULL, 3001),
    (69324, 'MARKER', 'CLERK', 67832, '1992-01-23', 1400.00, NULL, 1001);


+--------+----------+-----------+------------+------------+---------+------------
+--------+
| emp_id | emp_name | job_name  | manager_id | hire_date  | salary  | commission |
dep_id |
+--------+----------+-----------+------------+------------+---------+------------
+--------+
|  63679 | SANDRINE | CLERK     |      69062 | 1990-12-18 |  900.00 |       NULL |
2001 |
|  64989 | ADELYN   | SALESMAN  |      66928 | 1991-02-20 | 1700.00 |     400.00 |
3001 |
|  65271 | WADE     | SALESMAN  |      66928 | 1991-02-22 | 1350.00 |     600.00 |
3001 |
|  65646 | JONAS    | MANAGER   |      68319 | 1991-04-02 | 2957.00 |       NULL |
2001 |
|  66564 | MADDEN   | SALESMAN  |      66928 | 1991-09-28 | 1350.00 |    1500.00 |
3001 |
|  66928 | BLAZE    | MANAGER   |      68319 | 1991-05-01 | 2750.00 |       NULL |
3001 |
|  67832 | CLARE    | MANAGER   |      68319 | 1991-06-09 | 2550.00 |       NULL |
1001 |
```

```
|  67858 | SCARLET  | ANALYST   |      65646 | 1997-04-19 | 3100.00 |       NULL |
2001 |
|  68319 | KAYLING  | PRESIDENT |       NULL | 1991-11-18 | 6000.00 |       NULL |
1001 |
|  68454 | TUCKER   | SALESMAN  |      66928 | 1991-09-08 | 1600.00 |       0.00 |
3001 |
|  68736 | ADNRES   | CLERK     |      67858 | 1997-05-23 | 1200.00 |       NULL |
2001 |
|  69000 | JULIUS   | CLERK     |      66928 | 1991-12-03 | 1050.00 |       NULL |
3001 |
|  69062 | FRANK    | ANALYST   |      65646 | 1991-12-03 | 3100.00 |       NULL |
2001 |
|  69324 | MARKER   | CLERK     |      67832 | 1992-01-23 | 1400.00 |       NULL |
1001 |
+--------+----------+-----------+------------+------------+---------+------------
+--------+
```

Sql Query questions

Ques. Check version of the sql?

```
select version()
```

Current date?

```
select GETDATE();
```

Ques. How to copy a table in another table?

```
CREATE TABLE EMP1 AS (SELECT * FROM EMP); //constraint will not copied.
```

Ques. How to copy structure of a table but not data?

```
CREATE TABLE STD AS (SELECT * FROM EMP WHERE EMPNO=-1);
```

Creaet a table through another table/Duplicate table through another table.

```
CREATE TABLE IF NOT EXISTS new_table_name LIKE exsting_table_name;
```

Duplicate table through another table, with structure and data?

```
CREATE TABLE IF NOT EXISTS new_table_name AS SELECT * FROM exsting_table_name;
```

Nth highest salary?

- The limit clause has two components, the **First component** is to skip a number of rows from the top and the **second component** is to display the number of rows we want.

```
-- Syntex:- Using Limit
Select DISTINCT Salary from table_name order by Salary DESC limit n-1,1;
(OR) SELECT DISTINCT salary FROM employees ORDER BY salary DESC LIMIT 1 OFFSET N-
1;
+----------+---------+
| emp_name | salary  |
+----------+---------+
| JONAS    | 2957.00 |
+----------+---------+
-- Example:- 4th Highest salary using limit
Select DISTINCT emp_name, salary from Employee order by salary DESC limit 3,1;

-- Using Subquery:- 3rd higest salery
SELECT MAX(salary) AS ThirdHighestSalary FROM Employee WHERE salary < (SELECT
MAX(salary) FROM Employee WHERE salary < (SELECT MAX(salary) FROM Employee));
+-------------+
| MAX(salary) |
+-------------+
|     2957.00 |
+-------------+
```

Top N Salary?

```
-- Using Limit
SELECT salary FROM employee ORDER BY salary DESC LIMIT 4
+----------+---------+
| emp_name | salary  |
+----------+---------+
| KAYLING  | 6000.00 |
| FRANK    | 3100.00 |
| SCARLET  | 3100.00 |
| JONAS    | 2957.00 |
| BLAZE    | 2750.00 |
+----------+---------+
-- using subquery
SELECT name, salary FROM employees
WHERE salary IN ( SELECT DISTINCT salary FROM employees ORDER BY salary DESC LIMIT
5)
ORDER BY salary DESC;

-- In Oracle
SELECT SAL FROM(SELECT DISTINCT SAL FROM EMP WHERE SAL IS NOT NULL  ORDER BY SAL
DESC)WHERE ROWNUM <6;
```

Find the Highest Salary of Each Department?

```
-- Employee table:                          -- Department table:
+----+-------+--------+--------------+      +----+-------+
| id | name  | salary | departmentId |      | id | name  |
+----+-------+--------+--------------+      +----+-------+
| 1  | Joe   | 85000  | 1            |      | 1  | IT    |
| 2  | Henry | 80000  | 2            |      | 2  | Sales |
| 3  | Sam   | 60000  | 2            |      +----+-------+
| 4  | Max   | 90000  | 1            |
| 5  | Janet | 69000  | 1            |
| 6  | Randy | 85000  | 1            |
| 7  | Will  | 70000  | 1            |
| 8  | Mohit | 90000  | 1            |
+----+-------+--------+--------------+

-- for single table
SELECT dep_id, max(salary) FROM `employee` GROUP BY dep_id;
+--------+-------------+
| dep_id | max(salary) |
+--------+-------------+
|   1001 |     6000.00 |
|   2001 |     3100.00 |
|   2244 |     6000.00 |
|   3001 |     2750.00 |
+--------+-------------+

-- for two table
select max(employee.salary) AS salery, department.name from employee JOIN
department WHERE employee.departmentId = department.id GROUP BY department.name;
+--------+-------+
| salery | name  |
+--------+-------+
| 90000  | IT    |
| 80000  | Sales |
+--------+-------+
```

- Find the Highest Salary of Each Department with name

```
SELECT dep_id, emp_name, salary FROM employee WHERE (dep_id,salary) IN (SELECT
dep_id, MAX(salary) FROM employee GROUP BY dep_id);
+--------+----------+---------+
| dep_id | emp_name | salary  |
+--------+----------+---------+
|   2244 | Mohit    | 6000.00 |
|   3001 | BLAZE    | 2750.00 |
|   2001 | SCARLET  | 3100.00 |
|   1001 | KAYLING  | 6000.00 |
|   2001 | FRANK    | 3100.00 |
+--------+----------+---------+

-- Find the Highest Salary of Each Department with name from two table
SELECT department.name AS dep_name, employee.name AS emp_name, employee.salary AS
salary
FROM employee JOIN department
ON employee.departmentId = department.id
JOIN(
SELECT departmentId, max(salary) AS max_salary
from employee GROUP BY departmentId
) AS dept_max ON employee.departmentId = dept_max.departmentId AND employee.salary
= dept_max.max_salary

+----------+----------+--------+
| dep_name | emp_name | salary |
+----------+----------+--------+
| Sales    | Henry    | 80000  |
| IT       | Max      | 90000  |
| IT       | Mohit    | 90000  |
+----------+----------+--------+

-- Find the Highest Salary of Each Department with name Using SubQuery
SELECT department.name AS dep_name, employee.name AS emp_name, employee.salary AS
salary
FROM employee
JOIN department ON employee.departmentId = department.id
WHERE (employee.departmentId, employee.salary) IN (
    SELECT departmentId, MAX(salary)
    FROM employee
    GROUP BY departmentId
);

+----------+----------+--------+
| dep_name | emp_name | salary |
+----------+----------+--------+
| Sales    | Henry    | 80000  |
| IT       | Max      | 90000  |
| IT       | Mohit    | 90000  |
+----------+----------+--------+
```

How to Find Duplicate values in a Table?

```
select col1, Email, count(Email) as total_email from Person group by col1, Email
HAVING COUNT(Email) > 1;
+---------+-------------+
| Email   | total_email |
+ ------- + ----------- +
| a@b.com | 2           |
| c@d.com | 5           |
+---
```

Delete Duplicate Records?

```
DELETE t1
FROM my_table t1
INNER JOIN my_table t2
  ON t1.col1 = t2.col1
 AND t1.col2 = t2.col2 -- optinal this line if check multipal column
 AND t1.id > t2.id;

-- using subquery
DELETE FROM my_table
WHERE id IN (
  SELECT t1.id
  FROM my_table t1
  JOIN my_table t2
    ON t1.col1 = t2.col1
   AND t1.col2 = t2.col2
   AND t1.id > t2.id
);
```

Highest salary of the table

```
select emp_name, salary from employee where salary = (select max(salary) from
employee);
+----------+---------+
| emp_name | salary  |
+----------+---------+
| Mohit    | 6000.00 |
| KAYLING  | 6000.00 |
+----------+---------+
```

How many employees under the manager

```
 emp_id | emp_name | job_name  | manager_id | hire_date  | salary  | commission |
dep_id
--------+----------+-----------+------------+------------+---------+------------+-
-------
  68319 | KAYLING  | PRESIDENT |            | 1991-11-18 | 6000.00 |            |
1001
  66928 | BLAZE    | MANAGER   |      68319 | 1991-05-01 | 2750.00 |            |
3001
  67832 | CLARE    | MANAGER   |      68319 | 1991-06-09 | 2550.00 |            |
1001
  65646 | JONAS    | MANAGER   |      68319 | 1991-04-02 | 2957.00 |            |
2001
  67858 | SCARLET  | ANALYST   |      65646 | 1997-04-19 | 3100.00 |            |
2001
  69062 | FRANK    | ANALYST   |      65646 | 1991-12-03 | 3100.00 |            |
2001
  63679 | SANDRINE | CLERK     |      69062 | 1990-12-18 |  900.00 |            |
2001
  64989 | ADELYN   | SALESMAN  |      66928 | 1991-02-20 | 1700.00 |     400.00 |
3001
  65271 | WADE     | SALESMAN  |      66928 | 1991-02-22 | 1350.00 |     600.00 |
3001
  66564 | MADDEN   | SALESMAN  |      66928 | 1991-09-28 | 1350.00 |    1500.00 |
3001
  68454 | TUCKER   | SALESMAN  |      66928 | 1991-09-08 | 1600.00 |       0.00 |
3001
  68736 | ADNRES   | CLERK     |      67858 | 1997-05-23 | 1200.00 |            |
2001
  69000 | JULIUS   | CLERK     |      66928 | 1991-12-03 | 1050.00 |            |
3001
  69324 | MARKER   | CLERK     |      67832 | 1992-01-23 | 1400.00 |            |
1001
```

How many employees under the manager

```
SELECT w.manager_id,
       count(*)
FROM employees w,
     employees m
WHERE w.manager_id = m.emp_id
GROUP BY w.manager_id
ORDER BY w.manager_id ASC;

Output:-
  manager_id | count
------------+-------
       65646 |     2
       66928 |     5
       67832 |     1
       67858 |     1
```

```
       68319 |     3
       69062 |     1
```

and count higest emp under the manager.

```
SELECT m.emp_name,
       count(*)
FROM employees w,
     employees m
WHERE w.manager_id = m.emp_id
GROUP BY m.emp_name
HAVING count(*) =
  (SELECT MAX (mycount)
   FROM
     (SELECT COUNT(*) mycount
      FROM employees
      GROUP BY manager_id) a);

Output:-
 emp_name | count
----------+-------
 BLAZE    |     5
```

Create a table and check max_salary is not exceed the upper limit of 25000

```
CREATE TABLE IF NOT EXISTS jobs(
    JOB_ID varchar(10) NOT NULL,
    JOB_TITLE varchar(35) NOT NULL,
    MIN_SALARY decimal(6,0),
    MAX_SALARY decimal(6,0),
    CHECK(MAX_SALARY<=25000)
);
```

Replace a Column Values from 'male' to 'female' and 'female' to 'male'

```
UPDATE empdata
SET GENDER = CASE
    WHEN GENDER='male' THEN 'female'
    WHEN GENDER='female' THEN 'male'
    END;
(OR)
UPDATE EMPDATA
SET gender = CASE
    gender WHEN 'male' THEN 'female'
            WHEN 'female' THEN 'male'
```

```
    ELSE gender
END;
```

Update remaing days startdate - enddate

```
UPDATE advance_bookings
SET remaining_days = DATEDIFF('2025-01-31', CURDATE()), end_date = '2025-01-31'
WHERE id = 4529;

-- End date dynamic
UPDATE advance_bookings
SET remaining_days = DATEDIFF(end_date, CURDATE()), end_date = end_date
WHERE id = 4529;
```

Count phone number

```
SELECT phone, count(phone) as total_phone FROM `users` WHERE role_id = 4 group by
phone having count(phone) > 1;
```