Table of Contents

No.	Database
	Select Database

No.	Tables
	Types of SQL Commands/subsets of SQL?
	Data Definition Language (DDL)
	Data Manipulation Language (DML)
	Data Control Language (DCL)
	Transaction Control Language (TCL)
	Create TABLE?
	ALTER Table?
	Change Datatype from alter cmd?
	DROP column in table?
	TRUNCATE table?
	RENAME column in table?
	RENAME table name?
	What is delete?
	Difference between Delete, Truncate & Drop?
	Difference b/w DROP and TRUNCATE statements?

No.	Keys
	Primary Key?
	Add primary Key?
	Delete primary Key?
	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? | | | What Is Union & Union All?### What is View | | | What is View? | | | What is Index? | | |

No.	**interview_Questions_answers**
	What are Constraints in SQL?

No.	**Sql Query questions**
	Check version of the sql?
	Current date?
	How to find **Nth** highest salary from a table?
	Top Nth Salery?
	How to Find Duplicate values in a Table?
	Replace a Column Values from 'male' to 'female' and 'female' to 'male'?

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

Common Questions

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;
```

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.
- The IN operator allows you to specify multiple values.

```
+----+----------+--------+
| 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   |
+----+----------+--------+
```

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;
```

User Management

Create User

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

Show all Users

```
select user from mysql.user;
+---------+
| User    |
+---------+
| root    |
| mohits4 |
+---------+
```

Show Current User

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

User Password Change

```
SET PASSWORD FOR 'mohits4'@'hostname' = PASSWORD('jtp12345');
OR
ALTER USER mohits4@hostname IDENTIFIED BY 'jtp123';
```

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**: It enables the user account to create databases and tables.
3. **DROP**: It enables the user account to drop databases and tables.
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;
```

```
-- 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';
```

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;
OR
SHOW SCHEMAS;
+---------------------+
| Database            |
+---------------------+
| employeesdb         |
| phpmyadmin          |
| profile_fastapi     |
| test                |
+---------------------+
```

Create Databse

```
CREATE DATABASE databasename;
```

Rename Database

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

Drop Database

```
DROP DATABASE databasename;
OR
DROP SCHEMA database_name;
```

Select Database

```
USE YourDatabaseName;
```

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);

-- 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      |
```

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?

- 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  |
```

Sql Query questions

Ques. Check version of the sql?

```
select version()
```

Current date?

```
select GETDATE();
```

Nth highest salary?

- Using the LIMIT Clause
    - 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 |
+----------+---------+

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

How to Find Duplicate values in a Table?

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

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;
```