Created by: Vinay Kumar Panika

Optimization Techniques

91.	How to Optimize SQL Queries?
92.	What is Query Execution Plan?
93.	How to Improve Query Performance?
94.	What is Indexing?
95.	What is Table Partitioning?
96.	How to Avoid Deadlocks in SQL?
97.	What is the use of EXISTS in SQL?
98.	What is Query Optimization?
99.	What is the Difference Between Stored Procedure and Function in SQL?
100.	What is the difference between OLTP and OLAP in SQL?

Created by: Vinay Kumar Panika

Basic Level (Beginner)

1. What is SQL?

SQL (Structured Query Language) is a standard programming language used to interact with relational databases. It is used to store, retrieve, update, and delete data. SQL is also used to create and modify database structures such as tables, views, and indexes.

Example:

```
SELECT * FROM Employees;
```

This query retrieves all the records from the Employees table.

2. What is a Database?

A database is an organized collection of data that is stored and managed electronically. It allows users to efficiently store, retrieve, update, and manage data. Databases are used to handle large amounts of information in various applications such as websites, business systems, and applications.

Example:

A customer database in an e-commerce website may store customer details like name, email, contact number, and purchase history.

3. What are the types of SQL commands?

SQL commands are categorized into five types based on their functionality:

1. **DDL (Data Definition Language)** – Defines the structure of the database.
   - CREATE, ALTER, DROP, TRUNCATE
2. **DML (Data Manipulation Language)** – Manages data stored in the database.
   - SELECT, INSERT, UPDATE, DELETE
3. **DCL (Data Control Language)** – Controls access to the data.
   - GRANT, REVOKE
4. **TCL (Transaction Control Language)** – Manages transactions in the database.
   - COMMIT, ROLLBACK, SAVEPOINT
5. **DQL (Data Query Language)** – Retrieves data from the database.
   - SELECT

4. What is Primary Key?

A Primary Key is a column or a combination of columns in a table that uniquely identifies each row in that table. It does not allow NULL values and must always contain unique values.

Key Features:

- Uniquely identifies each record
- Cannot have duplicate values
- Cannot contain NULL values
- Only one primary key is allowed per table

Example:

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT
);
```

Here, EmployeeID is the primary key that uniquely identifies each employee.

5. What is Foreign Key?

A Foreign Key is a column or combination of columns in one table that refers to the Primary Key in another table. It is used to create a relationship between two tables and enforce referential integrity.

Key Features:

- Establishes a relationship between two tables
- Can contain duplicate values
- Can accept NULL values
- Helps maintain data consistency

Example:

```
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(50)
);

CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(50),
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

Here, DepartmentID in the Employees table is a foreign key that references the DepartmentID column in the Departments table.

6. What is UNIQUE Key?

A UNIQUE Key is a constraint that ensures all values in a column or combination of columns are distinct across all rows in the table. It prevents duplicate values but allows NULL values (only one NULL value in most databases).

Key Features:

- Ensures uniqueness of each record in the column
- Allows one NULL value (depending on the database)
- Multiple UNIQUE keys can be defined in a table
- Helps maintain data integrity

Example:

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Email VARCHAR(100) UNIQUE,
    Name VARCHAR(50)
);
```

Here, the Email column has a UNIQUE constraint, ensuring no two employees can have the same email address.

7. What is the Difference Between Primary Key and UNIQUE Key?

Primary Key	UNIQUE Key
Uniquely identifies each row in a table	Ensures all values in the column are unique
Does not allow NULL values	Allows one NULL value (in most databases)
Only one primary key is allowed per table	Multiple UNIQUE keys can be defined in a table
Automatically creates a unique clustered index	Creates a unique non-clustered index
Used to uniquely identify a record	Used to enforce uniqueness in a column without being a primary identifier

8. What is NOT NULL Constraint?

The NOT NULL constraint ensures that a column cannot have NULL values. It is used to enforce that every row must have a value in that column.

Key Features:

- Prevents insertion of NULL values
- Ensures mandatory fields have data
- Can be applied to one or more columns

Example:

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(50) NOT NULL,
    Email VARCHAR(100)
);
```

In this example, the Name column cannot have NULL values, while the Email column can accept NULL values.

9. What is Default Constraint?

The Default Constraint provides a default value for a column when no value is specified during the insertion of a new record.

Key Features:

- Automatically assigns a default value if no value is provided
- Helps avoid NULL values in specific columns
- Can be applied to any data type

Example:

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(50),
    Salary DECIMAL(10, 2) DEFAULT 5000
);
```

In this example, if no salary is provided while inserting a record, the Salary column will automatically be set to 5000.

10. What is the Difference Between DELETE, TRUNCATE, and DROP?

Command	Function	Can Rollback	Affects Structure	Speed
DELETE	Removes specific rows based on a condition using the WHERE clause	Yes (with COMMIT/ROLLBACK)	No	Slow (Row-by-row deletion)
TRUNCATE	Removes all rows from the table without a condition	No	No	Faster than DELETE
DROP	Deletes the entire table including data and structure	No	Yes (Removes table structure)	Fastest

Example:

```
DELETE FROM Employees WHERE EmployeeID = 101; -- Deletes specific row
TRUNCATE TABLE Employees; -- Deletes all rows
DROP TABLE Employees; -- Deletes table completely
```

11. What is the Difference Between WHERE and HAVING?

Clause	Purpose	Used With	Filter Type	Execution Order
WHERE	Filters rows before grouping	SELECT, UPDATE, DELETE	Row-level filter	Applied before GROUP BY
HAVING	Filters groups after grouping	SELECT with GROUP BY	Group-level filter	Applied after GROUP BY

Example:

```sql
-- WHERE Example
SELECT Name, Salary
FROM Employees
WHERE Salary > 5000;

-- HAVING Example
SELECT Department, AVG(Salary) AS AvgSalary
FROM Employees
GROUP BY Department
HAVING AVG(Salary) > 5000;
```

In the WHERE clause, filtering is applied before grouping, while HAVING filters the aggregated result.

12. What are Joins in SQL?

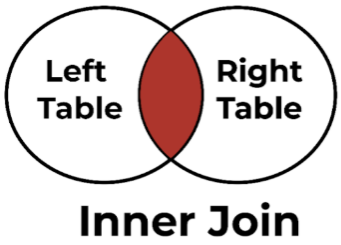
Joins in SQL are used to combine data from two or more tables based on a related column between them.

Types of Joins:

1. **INNER JOIN** – Returns only matching rows from both tables.
2. **LEFT JOIN** – Returns all rows from the left table and matching rows from the right table.
3. **RIGHT JOIN** – Returns all rows from the right table and matching rows from the left table.
4. **FULL JOIN** – Returns all rows from both tables (matching and non-matching).
5. **SELF JOIN** – Joins a table with itself.
6. **CROSS JOIN** – Returns the Cartesian product of both tables (all possible combinations).

13. What is INNER JOIN?

INNER JOIN is used to combine rows from two or more tables based on a matching condition between them. It returns only those records where the specified condition is true in both tables.



Key Features:

- Returns matching rows from both tables
- Ignores unmatched rows
- Most commonly used type of join

Syntax:

```
SELECT table1.column1, table2.column2
FROM table1
INNER JOIN table2 ON table1.common_column = table2.common_column;
```

Example:

```
SELECT Employees.Name, Departments.DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

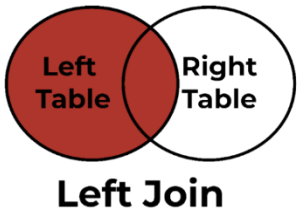
This query returns the employee names along with their department names where the DepartmentID is common in both tables.

14. What is LEFT JOIN?

LEFT JOIN is used to return all records from the left table and the matching records from the right table. If no match is found, the result will contain NULL values from the right table.

Key Features:

- Returns all rows from the left table
- Returns matching rows from the right table
- Displays NULL for non-matching rows from the right table



Left Join

Syntax:

```
SELECT table1.column1, table2.column2
FROM table1
LEFT JOIN table2 ON table1.common_column = table2.common_column;
```

Example:

```
SELECT Employees.Name, Departments.DepartmentName
FROM Employees
LEFT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

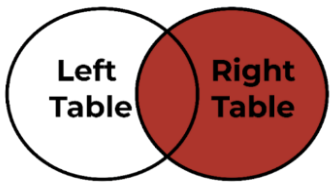
This query returns all employee names from the Employees table, and if the department is not assigned, the Department Name will be displayed as NULL.

15. What is RIGHT JOIN?

RIGHT JOIN is used to return all records from the right table and the matching records from the left table. If no match is found, the result will contain NULL values from the left table.

Key Features:

- Returns all rows from the right table
- Returns matching rows from the left table
- Displays NULL for non-matching rows from the left table



Right Join

Syntax:

```
SELECT table1.column1, table2.column2
FROM table1
RIGHT JOIN table2 ON table1.common_column = table2.common_column;
```

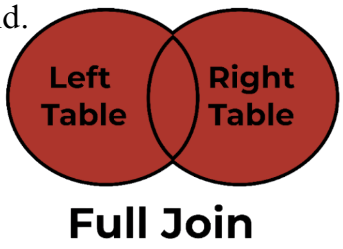
Example:

```
SELECT Employees.Name, Departments.DepartmentName
FROM Employees
RIGHT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

This query returns all department names from the Departments table, and if no employee is assigned to a department, the Employee Name will be displayed as NULL.

16. What is FULL JOIN?

FULL JOIN combines the results of both LEFT JOIN and RIGHT JOIN. It returns all records from both tables, with matching rows from both sides where available. If there is no match, the result will contain NULL values on the side where no match was found.



Full Join

Key Features:

- Returns all rows from both tables
- Displays NULL where there is no match
- Useful to find unmatched records in both tables

Syntax:

```
SELECT table1.column1, table2.column2
FROM table1
FULL JOIN table2 ON table1.common_column = table2.common_column;
```

Example:

```
SELECT Employees.Name, Departments.DepartmentName
FROM Employees
FULL JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

This query returns all employee names and department names, including those where there is no matching DepartmentID between the two tables.

17. What is Self Join?

Self Join is a type of join where a table is joined with itself to compare rows within the same table. It is used when a table contains a hierarchical relationship or when comparing values in the same table.

Key Features:

- Joins a table with itself
- Requires table aliases to differentiate table instances
- Used to compare rows within the same table

Syntax:

```
SELECT A.column1, B.column2
FROM table_name A
JOIN table_name B ON A.common_column = B.common_column;
```

Example:

```
SELECT E1.EmployeeName AS Employee, E2.EmployeeName AS Manager
FROM Employees E1
JOIN Employees E2 ON E1.ManagerID = E2.EmployeeID;
```

In this example, the Employees table joins with itself to show the employee's name along with their manager's name based on the ManagerID column.

18. What is Cross Join?

Cross Join returns the Cartesian product of two tables, meaning it combines every row from the first table with every row from the second table. It does not require any condition.

Key Features:

- Combines all rows from both tables
- Number of rows in the result = (Rows in Table 1) × (Rows in Table 2)
- Can produce large result sets if tables have many rows

Syntax:

```
SELECT table1.column1, table2.column2
FROM table1
CROSS JOIN table2;
```

Example:

```
SELECT Employees.Name, Departments.DepartmentName
FROM Employees
CROSS JOIN Departments;
```

This query returns all possible combinations of Employees and Departments, with every employee paired with every department.

19. What is Union and Union All?

UNION and UNION ALL are used to combine the result sets of two or more SELECT statements.

UNION	UNION ALL
Removes duplicate rows	Includes duplicate rows
Slower due to duplicate removal	Faster as no duplicate removal
Automatically sorts the result set	Does not sort the result set
Syntax: UNION	Syntax: UNION ALL

Syntax:

```
SELECT column1 FROM table1
UNION
SELECT column1 FROM table2;
```

Example (UNION):

```
SELECT Name FROM Employees
UNION
SELECT Name FROM Managers;
```

This query combines employee and manager names without duplicates.

Example (UNION ALL):

```
SELECT Name FROM Employees
UNION ALL
SELECT Name FROM Managers;
```

This query combines employee and manager names, including duplicates.

20. What is the difference between UNION and UNION ALL?

Criteria	UNION	UNION ALL
Duplicates	Removes duplicate rows	Includes all duplicate rows
Performance	Slower (due to duplicate removal)	Faster (no duplicate removal)
Sorting	Automatically sorts the result set	Does not sort the result set
Usage	Used when duplicate data is not required	Used when duplicate data needs to be preserved
Syntax	SELECT column FROM table1 UNION SELECT column FROM table2;	SELECT column FROM table1 UNION ALL SELECT column FROM table2;

21. What is Normalization?

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves dividing large tables into smaller related tables and defining relationships between them.

Key Features:

- Reduces data redundancy
- Improves data consistency
- Simplifies data maintenance
- Increases data integrity

Types of Normalization:

1. **1NF (First Normal Form)** – Eliminates duplicate columns and ensures each column contains atomic values.
2. **2NF (Second Normal Form)** – Ensures no partial dependency by making all non-key attributes fully dependent on the primary key.
3. **3NF (Third Normal Form)** – Removes transitive dependencies where non-key columns depend on other non-key columns.
4. **BCNF (Boyce-Codd Normal Form)** – Ensures that every determinant is a candidate key.

Example:

Unnormalized Table:

EmployeeID	Employee Name	Department	DepartmentLocation
101	Vinay	IT	Bangalore
102	Awadhesh	IT	Bangalore

Normalized Table (1NF & 2NF):

Employee Table:

EmployeeID	EmployeeName	DepartmentID
101	Vinay	1
102	Awadhesh	1

Department Table:

DepartmentID	DepartmentName	Location
1	IT	Bangalore

Normalization improves the efficiency and consistency of the database.

22. What is Denormalization?

Denormalization is the process of combining tables or adding redundant data into a database to improve read performance at the cost of data redundancy.
It is the opposite of Normalization, used when fast data retrieval is more important than maintaining data integrity.

Key Features:

- Improves data retrieval speed
- Increases data redundancy
- Reduces the number of joins required
- Used in data warehouses and reporting systems

Example:

Normalized Tables:
Employee Table:

EmployeeID	EmployeeName	DepartmentID
101	Vinay	1
102	Awadhesh	2

Department Table:

DepartmentID	DepartmentName
1	IT
2	HR

Denormalized Table:

EmployeeID	EmployeeName	DepartmentName
101	Vinay	IT
102	Awadhesh	HR

In this example, the DepartmentName column is directly stored in the Employee table, making data retrieval faster but increasing redundancy.

23. What is the Difference Between CHAR and VARCHAR?

Criteria	CHAR	VARCHAR
Full Form	Character	Variable Character
Storage	Fixed-length	Variable-length
Memory Usage	Always uses the specified length, even if fewer characters are stored	Uses only the space required for the actual data plus one or two bytes for length
Performance	Faster for fixed-size data	Slower for varying data sizes
Padding	Pads extra spaces to match the fixed length	Does not pad spaces
Use Case	When data length is consistent (like PIN codes or gender)	When data length varies (like names or addresses)

Example:

```
CREATE TABLE Employees (
    EmpID INT,
    Name CHAR(10),
    Address VARCHAR(50)
);
```

In this example, Name will always take 10 bytes, while Address will use only the necessary space based on the actual data length.

24. What is the difference between SQL and MySQL?

Criteria	SQL	MySQL
Definition	Structured Query Language used to manage and manipulate databases	Relational Database Management System (RDBMS) that uses SQL
Type	Language	Software
Usage	Used to write queries to interact with databases	Used to store, manage, and retrieve data
Developer	Standard language developed by ANSI	Developed by Oracle Corporation
Platform	Universal (used by many databases like MySQL, Oracle, SQL Server)	Specific to MySQL
Purpose	Query language to communicate with databases	Database system to store and manage data

Example:

- **SQL:**

```
SELECT * FROM Employees;
```

- **MySQL:**

MySQL stores the Employees table and processes the SQL query to retrieve data.

25. What is Auto Increment in SQL?

Auto Increment is a property in SQL that automatically generates a unique sequential number whenever a new row is inserted into a table. It is typically used to create unique identifiers like primary keys.

Key Features:

- Automatically generates unique numbers
- Commonly used with Primary Key columns
- Starts from a defined value (default is 1)
- Automatically increments by 1 for each new row

Syntax (MySQL):

```
CREATE TABLE Employees (
    EmployeeID INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(50),
    Salary DECIMAL(10, 2)
);
```

Example (Insert Records):

```
INSERT INTO Employees (Name, Salary) VALUES ('Vinay', 50000);
INSERT INTO Employees (Name, Salary) VALUES ('Raj', 40000);
```

Output:

EmployeeID	Name	Salary
1	Vinay	50000
2	Awadhesh	40000

The EmployeeID column automatically increments without user input.

Created by: Vinay Kumar Panika

Intermediate Level

26. What is Subquery?

A Subquery is a query nested inside another query in SQL.
It is used to fetch data that will be used by the main query as a condition to filter or manipulate the result.

Key Points:

- Also called Inner Query or Nested Query.
- Always executes before the main query.
- The result of the subquery is used by the Outer Query.
- Can be used with SELECT, INSERT, UPDATE, or DELETE statements.

Syntax:

```
SELECT column_name
FROM table_name
WHERE column_name = (SELECT column_name FROM another_table WHERE condition);
```

Types of Subqueries:

1. **Single Row Subquery:** Returns only one value.

Example:

```
SELECT name, salary
FROM employees
WHERE salary = (SELECT MAX(salary) FROM employees);
```

Explanation: It selects the employee with the highest salary.

2. Multiple Row Subquery: Returns multiple rows of data.

Example:

```
SELECT name
FROM employees
WHERE department_id IN (SELECT department_id FROM departments WHERE location = 'Bangalore');
```

Explanation: It selects employees working in the Bangalore location.

3. Correlated Subquery: Uses each row of the outer query to execute the subquery.

Example:

```
SELECT e.name, e.salary
FROM employees e
WHERE e.salary > (SELECT AVG(salary) FROM employees WHERE department_id = e.department_id);
```

Explanation: It selects employees whose salary is higher than the average salary of their own department.

27. What is Nested Query?

A Nested Query is a query written inside another query to retrieve data based on the result of the inner query.

It helps break down complex queries into smaller, more manageable parts.

Key Points:

- Also known as Inner Query or Subquery.
- The Inner Query executes first, and its result is passed to the Outer Query.
- Used in SELECT, INSERT, UPDATE, or DELETE statements.
- Can be used with WHERE, HAVING, and FROM clauses.

Syntax:

```
SELECT column_name
FROM table_name
WHERE column_name = (SELECT column_name FROM another_table WHERE condition);
```

Example:

Find employees who work in the 'Sales' department.

```
SELECT name
FROM employees
WHERE department_id = (SELECT department_id FROM departments WHERE department_name = 'Sales');
```

Explanation:

- The inner query finds the department_id for the 'Sales' department.
- The outer query selects employees based on that department_id.

28. What is Correlated Subquery?

A Correlated Subquery is a subquery that depends on the values from the outer query to execute.

It is executed repeatedly for each row of the outer query, making it slower compared to regular subqueries.

Key Points:

- The Inner Query uses columns from the Outer Query.
- Executes row by row for each result of the outer query.
- Cannot be executed independently without the outer query.
- Used for row-by-row comparisons.

Syntax:

```
SELECT column_name
FROM table_name t1
WHERE column_name = (SELECT column_name FROM table_name t2 WHERE t1.column = t2.column);
```

Example:

Find employees whose salary is higher than the average salary of their department.

```
SELECT e.name, e.salary
FROM employees e
WHERE e.salary > (SELECT AVG(salary) FROM employees WHERE department_id = e.department_id);
```

Explanation:

- The inner query calculates the average salary of each department.
- The outer query checks if the employee's salary is higher than their department's average salary.

29. What is GROUP BY in SQL?

The GROUP BY clause in SQL is used to group rows that have the same values into summary rows. It is typically used with aggregate functions to perform calculations on each group of data.

Key Points:

- Groups data based on one or more columns.
- It is used to summarize data.
- Always used after the WHERE clause and before the ORDER BY clause.
- Commonly used with aggregate functions like:
  - COUNT() – Counts the number of rows.
  - SUM() – Calculates the total sum.
  - AVG() – Calculates the average value.
  - MAX() – Returns the maximum value.
  - MIN() – Returns the minimum value.

Syntax:

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

Example:

Find the total sales amount for each product category.

```
SELECT category, SUM(sales_amount) AS Total_Sales
FROM sales
GROUP BY category;
```

Conclusion:

The GROUP BY clause is essential for data summarization and helps in analyzing data patterns efficiently.

30. What is the Difference Between GROUP BY and ORDER BY in SQL?

Both GROUP BY and ORDER BY are SQL clauses used to organize query results, but they serve different purposes.

GROUP BY	ORDER BY
Used to group rows based on the same values in one or more columns.	Used to sort the result set in ascending or descending order.
Always works with aggregate functions like COUNT(), SUM(), AVG(), etc.	Does not require aggregate functions.
Syntax: Comes before ORDER BY.	Syntax: Always comes after GROUP BY.
Groups the result into summary rows.	Sorts the entire result set.
Example: Group sales by product category.	Example: Sort sales by highest to lowest amount.

Example with GROUP BY:

Find the total sales for each product category.

```sql
SELECT category, SUM(sales_amount) AS Total_Sales
FROM sales
GROUP BY category;
```

Example with ORDER BY:

Sort products by price in descending order.

```sql
SELECT product_name, price
FROM products
ORDER BY price DESC;
```

Conclusion:

- GROUP BY is used to group data and perform aggregate calculations.
- ORDER BY is used to sort the final result.

31. What is the Use of LIMIT in SQL?

LIMIT is used to restrict the number of rows returned by a query.

Syntax:

```
SELECT column_name
FROM table_name
LIMIT number_of_rows;
```

Example:

Fetch top 3 highest salaries:

```
SELECT name, salary
FROM employees
ORDER BY salary DESC
LIMIT 3;
```

Conclusion:

- LIMIT helps to fetch limited data and is commonly used for Top N records or Pagination.

32. How to Find the Second Highest Salary in SQL?

Using Subquery:

```
SELECT MAX(salary)
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);
```

Explanation:

- The inner query gets the highest salary.
- The outer query finds the highest salary below the top salary.

33. How to find Duplicate Records in a table?

You can find Duplicate Records using the GROUP BY clause with the HAVING keyword.

Syntax:

```
SELECT column_name, COUNT(*)
FROM table_name
GROUP BY column_name
HAVING COUNT(*) > 1;
```

Explanation:

- GROUP BY groups the same names.
- HAVING COUNT(*) > 1 filters only duplicate names.

34. What is CTE (Common Table Expression)?

CTE (Common Table Expression) is a temporary result set that is defined within the execution of a single SQL statement.

Syntax:

```
WITH cte_name AS (
SELECT column_name
FROM table_name
WHERE condition
)
SELECT * FROM cte_name;
```

Example:

Find employees who work in the IT department and have a salary greater than 50000.

```
WITH IT_Employees AS (
SELECT name, department, salary
FROM employees
WHERE department = 'IT'
)
SELECT name, salary
FROM IT_Employees
WHERE salary > 50000;
```

Explanation:

- The CTE IT_Employees selects all employees from the IT department.
- The SELECT query fetches employees with salary above 50000.

35. What is Temporary Table in SQL?

A Temporary Table in SQL is used to store temporary data during the session.

Key Points:

- Automatically deleted when the session ends.
- Used to store intermediate results.
- Prefixed with # in SQL Server or TEMP in MySQL.

Syntax:

MySQL:

```
CREATE TEMPORARY TABLE temp_table (
id INT,
name VARCHAR(50)
);
```

SQL Server:

```
CREATE TABLE #temp_table (
id INT,
name VARCHAR(50)
);
```

Example:

Create a temporary table to store employees with salary above 50000.

```
CREATE TEMPORARY TABLE temp_emp AS
SELECT name, salary
FROM employees
WHERE salary > 50000;

SELECT * FROM temp_emp;
```

Explanation:

- The table holds filtered data temporarily.
- It is automatically deleted after the session ends.

36. What is Window Function in SQL?

A Window Function performs calculations across a set of table rows related to the current row without collapsing the result into a single value.

Key Points:

- Works with OVER() clause.
- Does not group rows like aggregate functions.
- Commonly used for ranking, running totals, and moving averages.

Syntax:

```
SELECT column_name,
       window_function() OVER (PARTITION BY column_name ORDER BY column_name)
FROM table_name;
```

Types of Window Functions:

1. ROW_NUMBER()

Assigns a unique sequential number to each row in a partition.

Syntax:

```
ROW_NUMBER() OVER (PARTITION BY column ORDER BY column)
```

2. RANK()

Assigns a rank to each row with the same values having the same rank, but skips ranks for duplicate values.

Syntax:

```
RANK() OVER (PARTITION BY column ORDER BY column)
```

3. DENSE_RANK()

Similar to RANK(), but does not skip ranks for duplicate values.

Syntax:

```
DENSE_RANK() OVER (PARTITION BY column ORDER BY column)
```

4. NTILE(n)

Divides the result set into n equal parts and assigns a group number to each row.

Syntax:

```
NTILE(n) OVER (PARTITION BY column ORDER BY column)
```

5. SUM()

Calculates the cumulative total of a column within a partition.

Syntax:

```
SUM(column) OVER (PARTITION BY column ORDER BY column)
```

6. AVG()

Calculates the average value of a column within a partition.

Syntax:

```
AVG(column) OVER (PARTITION BY column ORDER BY column)
```

7. MAX() & MIN()

Returns the maximum or minimum value in a partition.

Syntax:

```
MAX(column) OVER (PARTITION BY column)
MIN(column) OVER (PARTITION BY column)
```

8. LEAD()

Returns the next row's value in the result set.

Syntax:

```
LEAD(column, offset) OVER (PARTITION BY column ORDER BY column)
```

9. LAG()

Returns the previous row's value in the result set.

Syntax:

```
LAG(column, offset) OVER (PARTITION BY column ORDER BY column)
```

Summary Table:

Function	Purpose	Duplicates	Skips Numbers	Example Usage
ROW_NUMBER	Unique Rank	No	No	Employee Ranking
RANK	Rank with Skips	Yes	Yes	Competition Rank
DENSE_RANK	Rank without Skips	Yes	No	Class Rank
NTILE	Divide Rows	No	No	Quartiles
SUM	Running Total	No	No	Salary Analysis
AVG	Running Average	No	No	Salary Average
LEAD	Next Row Value	No	No	Price Trends
LAG	Previous Row Value	No	No	Sales Trends

Conclusion:

Window Functions help in performing complex calculations across result sets without grouping them into a single row, making them essential for ranking, running totals, and trend analysis.

37. What is the Difference Between ROW_NUMBER(), RANK(), and DENSE_RANK()?

These three Window Functions are used to assign numbers to rows based on their order.

Key Differences:

Function	Purpose	Duplicates	Skips Numbers	Example
ROW_NUMBER()	Assigns unique sequential numbers to each row	No	No	1, 2, 3, 4
RANK()	Assigns rank to each row with the same values having the same rank	Yes	Yes	1, 2, 2, 4
DENSE_RANK()	Assigns rank without skipping numbers for duplicate values	Yes	No	1, 2, 2, 3

38. What is CASE Statement in SQL?

The CASE Statement is used to apply conditional logic in SQL queries, similar to IF-ELSE statements.

Syntax:

```
SELECT column,
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ELSE result3
END AS alias_name
FROM table_name;
```

Example:

```
SELECT name, salary,
CASE
    WHEN salary > 6000 THEN 'High Salary'
    WHEN salary BETWEEN 4000 AND 6000 THEN 'Medium Salary'
    ELSE 'Low Salary'
END AS Salary_Category
FROM employees;
```

39. What is COALESCE in SQL?

COALESCE returns the first non-null value from a list of expressions.

Syntax:

```
COALESCE(expr1, expr2, ..., exprN)
```

Example:

```
SELECT name, COALESCE(email, phone, 'No Contact') AS Contact_Info
FROM students;
```

Key Points:

- Returns the first non-null value.
- Used to handle NULL values.
- Can accept multiple expressions.

40. What is NVL Function in SQL?

NVL function replaces NULL values with a specified value.

Syntax:

```
NVL(expr1, expr2)
```

Example:

```
SELECT name, NVL(email, 'No Email') AS Email
FROM students;
```

41. What is Indexing in SQL?

Indexing improves the speed of data retrieval from a table by creating a lookup structure.

Syntax:

```
CREATE INDEX index_name ON table_name(column_name);
```

Example:

```
CREATE INDEX idx_name ON students(name);
```

Key Points:

- Speeds up SELECT queries.
- Automatically maintained by the database.
- Can be created on one or more columns.
- Slows down INSERT, UPDATE, DELETE operations.

Created by: Vinay Kumar Panika

42. What is Clustered Index in SQL?

A Clustered Index sorts and stores the data physically in the table based on the indexed column.

Syntax:

```
CREATE CLUSTERED INDEX index_name ON table_name(column_name);
```

Example:

```
CREATE CLUSTERED INDEX idx_name ON students(id);
```

Key Points:

- Only one clustered index is allowed per table.
- Faster for data retrieval.
- Automatically created on Primary Key by default.
- Rearranges table rows physically.

43. What is Non-Clustered Index in SQL?

A Non-Clustered Index creates a separate structure from the table data, storing pointers to the actual rows.

Syntax:

```
CREATE INDEX index_name ON table_name(column_name);
```

Example:

```
CREATE INDEX idx_name ON students(name);
```

Key Points:

- Multiple Non-Clustered Indexes can be created on a table.
- Improves search performance.
- Does not affect the physical order of data.
- Stores pointers to the actual data.

44. Difference between Clustered and Non-Clustered Index

Clustered Index	Non-Clustered Index
Stores data physically sorted.	Stores pointers to data.
Only one per table.	Multiple indexes allowed.
Faster for data retrieval.	Slower than Clustered Index.
Automatically created on Primary Key.	Manually created on any column.
Affects physical order of table.	Does not affect physical order.

45. What is View in SQL?

A View is a virtual table based on the result of a SQL query.

Syntax:

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

Example:

```
CREATE VIEW high_salary AS
SELECT name, salary
FROM employees
WHERE salary > 50000;
```

Key Points:

- Does not store data physically.
- Simplifies complex queries.
- Provides data security.
- Can be used like a table in SELECT queries.

46. Difference between View and Table

View	Table
Virtual table.	Physical table.
Does not store data physically.	Stores data physically.
Based on SQL queries.	Contains raw data.
Provides data security by restricting access to certain columns.	No data restriction unless applied.
Automatically updates when base table data changes.	Needs manual updates.

47. What is Stored Procedure ?

A Stored Procedure is a group of predefined SQL statements stored in the database that can be executed multiple times.

Syntax:

```
CREATE PROCEDURE procedure_name
AS
BEGIN
    SQL_Statements;
END;
```

Example:

```
CREATE PROCEDURE GetEmployees
AS
BEGIN
    SELECT * FROM employees;
END;
```

Key Points:

- Improves code reusability.
- Increases performance.
- Supports input and output parameters.
- Provides security by hiding SQL code.

48. What is the difference between Function and Stored Procedure?

Function	Stored Procedure
Returns a single value or table.	May or may not return a value.
Can be used in SELECT statements.	Cannot be used in SELECT statements.
Allows only input parameters.	Allows input and output parameters.
Cannot modify database state.	Can modify database state (INSERT, UPDATE, DELETE).
Always returns a value.	Does not always return a value.

49. What is Trigger in SQL?

A Trigger is an automatic action executed when a specified event occurs in a table.

Syntax:

```
CREATE TRIGGER trigger_name
AFTER INSERT
ON table_name
FOR EACH ROW
BEGIN
    SQL_Statements;
END;
```

Example:

```
CREATE TRIGGER after_insert
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO logs(message) VALUES('New employee added');
END;
```

Key Points:

- Automatically executes on INSERT, UPDATE, or DELETE.
- Used for data validation and logging.
- Cannot be called manually.
- Improves data integrity.

50. What is Cursor in SQL?

A Cursor is a database object used to retrieve, manipulate, and navigate row-by-row through the result set.

Syntax:

```
DECLARE cursor_name CURSOR FOR
SELECT column_name FROM table_name;

OPEN cursor_name;

FETCH NEXT FROM cursor_name INTO variable;

CLOSE cursor_name;
DEALLOCATE cursor_name;
```

Example:

```
DECLARE emp_cursor CURSOR FOR
SELECT name FROM employees;

OPEN emp_cursor;

FETCH NEXT FROM emp_cursor INTO @emp_name;

WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT @emp_name;
    FETCH NEXT FROM emp_cursor INTO @emp_name;
END;

CLOSE emp_cursor;
DEALLOCATE emp_cursor;
```

Key Points:

- Used to process row-by-row results.
- Slower than set-based operations.
- Helps in complex data manipulation.
- Not recommended for large datasets.

Created by: Vinay Kumar Panika

Advanced Level

51. What is the ACID Property in SQL?

The ACID properties in SQL define the key principles to ensure that database transactions are processed reliably without affecting data integrity.

ACID Stands for:

Property	Description
A - Atomicity	Transaction should be all or nothing. If one part of the transaction fails, the entire transaction fails, and the database remains unchanged.
C - Consistency	The database must be in a consistent state before and after the transaction. It ensures that data remains correct and valid.
I - Isolation	Transactions should be executed independently, without interfering with each other.
D - Durability	Once a transaction is committed, the changes must be permanent in the database, even if the system crashes.

Example:

```
BEGIN TRANSACTION;
UPDATE Accounts SET Balance = Balance - 1000 WHERE AccountNo = 101; -- Debit Amount
UPDATE Accounts SET Balance = Balance + 1000 WHERE AccountNo = 102; -- Credit Amount
COMMIT;
```

Explanation:

1. **Atomicity:** If one of the two queries fails, both updates will be rolled back.
2. **Consistency:** The total amount in both accounts will remain the same.
3. **Isolation:** If another transaction is trying to access the same account, it will wait until this transaction completes.
4. **Durability:** After COMMIT, changes will be saved permanently even in case of a power failure.

Created by: Vinay Kumar Panika

52. What is a Transaction in SQL?

A Transaction in SQL is a group of SQL operations that are executed as a single unit to perform a specific task on the database.
It follows ACID Properties to maintain data integrity.

Key ACID Properties:

Property	Description
Atomicity	All operations must succeed or none will happen.
Consistency	The database must remain in a valid state before and after the transaction.
Isolation	Transactions execute independently without affecting each other.
Durability	Once committed, changes are permanent even after system failure.

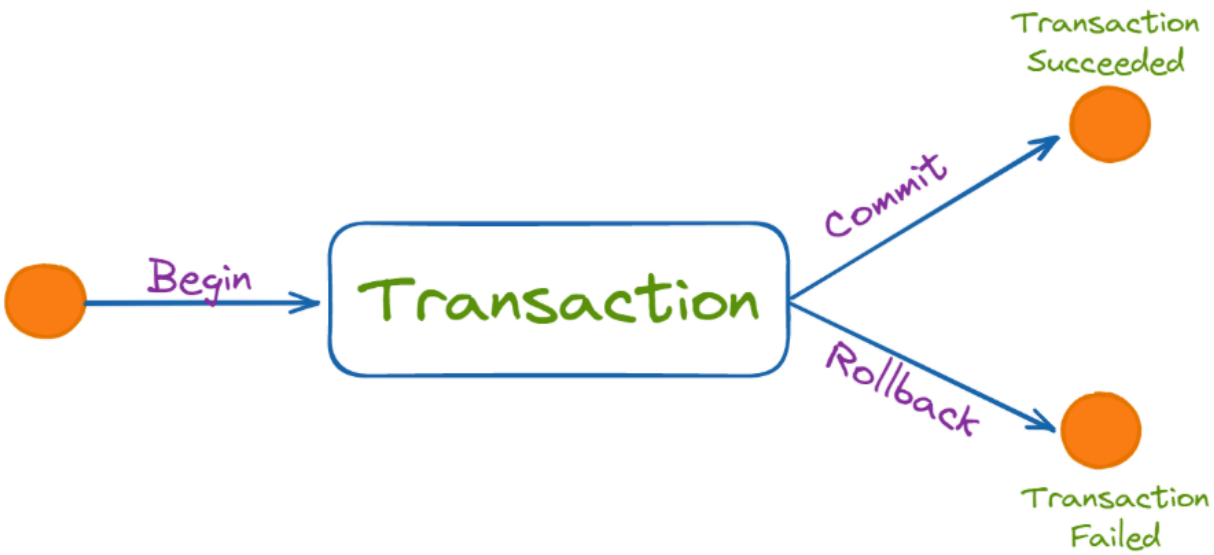
Transaction Commands:

Command	Description
BEGIN TRANSACTION	Starts a new transaction.
COMMIT	Saves the changes permanently.
ROLLBACK	Undo changes if any error occurs.
SAVEPOINT	Sets a point to roll back to partially.

Example:

```sql
BEGIN TRANSACTION;
UPDATE Employees SET Salary = Salary + 5000 WHERE ID = 101;
DELETE FROM Employees WHERE ID = 102;

-- If all queries are successful
COMMIT;
-- If any query fails
ROLLBACK;
```



Transactions in SQL

53. What is the difference between COMMIT and ROLLBACK?

COMMIT	ROLLBACK
Saves the changes made by the transaction permanently into the database.	Undo all changes made by the transaction.
Once executed, changes cannot be undone.	Restores the database to its previous state.
Syntax: COMMIT;	Syntax: ROLLBACK;
Used when all operations are successful.	Used when any error occurs during the transaction.
Improves data durability.	Helps to maintain data consistency.

Example:

```
BEGIN TRANSACTION;
UPDATE Employees SET Salary = 6000 WHERE ID = 101;

-- If no error occurs
COMMIT;

-- If error occurs
ROLLBACK;
```

54. What is Savepoint in SQL?

Savepoint in SQL is used to temporarily save a transaction at a specific point, allowing you to rollback only part of the transaction without affecting the entire transaction.

Key Points:

- Allows setting multiple points in a transaction.
- Helps in partial rollback.
- Improves error handling.
- Used with ROLLBACK.

Syntax:

```
BEGIN TRANSACTION;
UPDATE Employees SET Salary = 5000 WHERE ID = 101;
SAVEPOINT SP1; -- Savepoint 1
UPDATE Employees SET Salary = 7000 WHERE ID = 102;
ROLLBACK TO SP1; -- Rollback to Savepoint 1
COMMIT; -- Final Commit
```

Explanation:

- The first update will be saved.
- The second update will be rolled back.
- Remaining changes will be committed.

55. What is the difference between IN and EXISTS?

IN	EXISTS
Compares values from the main query with a list of values.	Checks if subquery returns any rows.
Works with static values or subqueries.	Only works with subqueries.
Slower with large datasets.	**Faster for large datasets.**
Returns all matching rows.	Stops checking after finding the first match.

56. What is the difference between DELETE and TRUNCATE?

DELETE	TRUNCATE
Removes specific rows based on a condition using the WHERE clause.	Removes all rows from the table without any condition.
Can be rolled back using ROLLBACK if inside a transaction.	Cannot be rolled back once executed.
Slower because it logs each row deletion.	Faster because it does not log individual row deletions.
Maintains table structure and identity column values.	Resets identity column values to the initial seed.

57. What is Index Fragmentation?

Index Fragmentation occurs when the logical order of index pages in the database does not match the physical order of data on disk, making data retrieval slower.

Types of Index Fragmentation:

1. **Internal Fragmentation** – Unused space inside index pages due to data deletion or updates.
2. **External Fragmentation** – Index pages are stored in non-sequential order, causing slower data access.

How to Check Index Fragmentation in MySQL:

```
SHOW INDEX FROM Employees;
```

How to Fix Index Fragmentation:

- Use OPTIMIZE TABLE to reorganize index pages.

```
OPTIMIZE TABLE Employees;
```

Conclusion:

Index fragmentation slows down query performance and should be fixed regularly to maintain database efficiency.

58. What is the difference between RANK() and DENSE_RANK()?

RANK()	DENSE_RANK()
Assigns a unique rank to each row, but skips the next rank if there are duplicate values.	Assigns a unique rank to each row without skipping ranks if there are duplicate values.
Gaps are created in ranking sequence.	No gaps in ranking sequence.
Slower in performance compared to DENSE_RANK().	Faster than RANK() because it doesn't skip ranks.

59. How to fetch common records from two tables?

To fetch common records from two tables, you can use the INNER JOIN clause in SQL.

Syntax:

```
SELECT table1.column, table2.column
FROM table1
INNER JOIN table2 ON table1.common_column = table2.common_column;
```

60. What is the difference between UNION and JOIN?

UNION	JOIN
Combines result sets vertically (rows) from two or more tables.	Combines result sets horizontally (columns) based on common columns.
Removes duplicates by default (UNION), or includes duplicates with UNION ALL.	Does not remove duplicates.
Tables should have the same number of columns and data types.	Tables can have different numbers of columns.
Used when tables have similar data.	Used when tables have related data through a common column.

61. What is Pivot Table in SQL?

A Pivot Table in SQL is used to transform row data into column data to provide a summary report of the dataset.
It is commonly used to perform data aggregation and present data in a more readable format.

Key Points:

- Converts rows into columns.
- Used to generate summary reports.
- Helps in data analysis.
- Commonly used with aggregate functions like SUM(), AVG(), COUNT(), etc.

Example: Pivot Table Query

```
SELECT Product,
SUM(CASE WHEN Month = 'Jan' THEN Sales ELSE 0 END) AS Jan_Sales,
SUM(CASE WHEN Month = 'Feb' THEN Sales ELSE 0 END) AS Feb_Sales
FROM Sales
GROUP BY Product;
```

Conclusion:

Pivot Tables help to summarize large datasets and present them in a structured format.

62. What is Case Sensitivity in SQL?

Case Sensitivity in SQL refers to whether the database treats uppercase and lowercase letters as different or same when performing queries.

Key Points:

- **Column Names:** Most databases are not case-sensitive (MySQL, SQL Server).
- **Table Names:** Case sensitivity depends on the database and operating system.
- **String Values:** By default, MySQL is case-insensitive for string comparisons.

Example in MySQL:

```
CREATE TABLE Employees (
    EmpID INT,
    Name VARCHAR(50)
);

INSERT INTO Employees VALUES (101, 'Vinay'), (102, 'VINAY');

SELECT * FROM Employees WHERE Name = 'vinay';
```

Output:

EmpID	Name
101	Vinay
102	VINAY

How to Make MySQL Case-Sensitive:

```
SELECT * FROM Employees WHERE BINARY Name = 'vinay';
```

This query will only return exact case-sensitive matches.

Conclusion:

- Case Insensitivity is default in MySQL for string comparisons.
- Use the BINARY keyword to perform case-sensitive searches.

63. How to find the Nth Highest Salary?

To find the Nth Highest Salary in SQL, you can use the LIMIT with OFFSET method or Subquery with ORDER BY.

Method 1: Using LIMIT with OFFSET (MySQL)

```
SELECT DISTINCT Salary
FROM Employees
ORDER BY Salary DESC
LIMIT 1 OFFSET N-1;
```

Example: Example (3rd Highest Salary):

```
SELECT DISTINCT Salary
FROM Employees
ORDER BY Salary DESC
LIMIT 1 OFFSET 2;
```

Explanation:

- **ORDER BY Salary DESC → Sorts the salaries in descending order.**
- **OFFSET 2 → Skips the top 2 salaries.**
- **LIMIT 1 → Selects the next salary as the 3rd highest.**

Method 2: Using Subquery with LIMIT

```
SELECT Name, Salary
FROM Employees
WHERE Salary = (SELECT DISTINCT Salary
                FROM Employees
                ORDER BY Salary DESC
                LIMIT 1 OFFSET 2);
```

Conclusion:

- Use LIMIT with OFFSET for faster results in MySQL.
- This method is commonly asked in interviews.
- Always use DISTINCT to remove duplicate salaries.

64. How to get First 3 Maximum Salaries?

To fetch the First 3 Maximum Salaries from a table, you can use the DISTINCT, ORDER BY, and LIMIT clauses.

Method : Using LIMIT (MySQL)

```
SELECT DISTINCT Salary
FROM Employees
ORDER BY Salary DESC
LIMIT 3;
```

Explanation:

- **DISTINCT →** Removes duplicate salaries.
- **ORDER BY Salary DESC →** Sorts salaries in descending order.
- **LIMIT 3 →** Fetches the top 3 salaries.

65. What is the difference between Drop, Delete, and Truncate?

Command	Purpose	Can Rollback	Speed	Structure Affected	Condition-Based
DROP	Deletes the entire table with its structure	No	Fastest	Yes (Removes table structure)	No
DELETE	Removes specific rows based on condition	Yes (if within Transaction)	Slow	No	Yes (With WHERE Clause)
TRUNCATE	Removes all rows from the table	No	Fast	No (Keeps structure)	No

Conclusion:

- Use DELETE for removing specific rows with conditions.
- Use TRUNCATE for removing all rows quickly without rollback.
- Use DROP to delete both data and table structure completely.

66. How to calculate Age from Date of Birth in SQL?

You can calculate the Age from the Date of Birth using the DATEDIFF() or YEAR() functions depending on the database.

Method 1: Using DATEDIFF() (MySQL)

```
SELECT Name,
FLOOR(DATEDIFF(CURDATE(), DOB) / 365) AS Age
FROM Employees;
```

Explanation:

- **CURDATE()** → Returns the current date.
- **DATEDIFF()** → Calculates the difference between the current date and date of birth in days.
- **FLOOR()** → Converts the result into whole years.

Method 2: Using YEAR() (MySQL)

```
SELECT Name,
YEAR(CURDATE()) - YEAR(DOB) AS Age
FROM Employees;
```

Explanation:

This method calculates the difference between the current year and the birth year.

Conclusion:

- Use DATEDIFF() for accurate age calculation.
- Use YEAR() for simple year-based age calculation.

67. What is Recursive Query in SQL?

A Recursive Query in SQL is a query that refers to itself to perform repetitive operations until a specific condition is met. It is commonly used to process hierarchical data such as employee-manager relationships or organizational structures.

Key Points:

- Used to handle hierarchical data.
- Implemented using Common Table Expressions (CTE).
- The recursion continues until the termination condition is satisfied.

Example:

Employee Table:

EmpID	Name	ManagerID
101	Vinay	NULL
102	Awadhesh	101
103	Nevendra	102

Recursive Query:

```
WITH EmployeeCTE AS (
    SELECT EmpID, Name, ManagerID FROM Employees WHERE ManagerID IS NULL -- Anchor Query
    UNION ALL
    SELECT E.EmpID, E.Name, E.ManagerID FROM Employees E
    JOIN EmployeeCTE C ON E.ManagerID = C.EmpID -- Recursive Query
)
SELECT * FROM EmployeeCTE;
```

Created by: Vinay Kumar Panika

68. What is the difference between Temporary Table and CTE?

Temporary Table	CTE (Common Table Expression)
Stores data physically in temporary memory.	Stores data logically without physical storage.
Needs to be explicitly created and dropped.	Automatically disappears after query execution.
Can be used multiple times within a session.	Can be used only once in the same query.
Supports Indexing and DDL operations.	Does not support Indexing or DDL operations.
Slower for small datasets.	Faster for small datasets.

Conclusion:

- Use Temporary Tables when data needs to be reused multiple times.
- Use CTE for short-term data manipulation and improved readability.

69. How to find Odd and Even records in SQL?

You can find Odd and Even records in SQL using the MOD() or ROW_NUMBER() functions.

Method 1: Using MOD() Function (MySQL)

```sql
SELECT * FROM Employees WHERE MOD(EmpID, 2) = 0; -- Even Records
SELECT * FROM Employees WHERE MOD(EmpID, 2) = 1; -- Odd Records
```

Explanation:

- MOD(EmpID, 2) → Returns the remainder when EmpID is divided by 2.
- If the remainder is 0, the record is Even.
- If the remainder is 1, the record is Odd.

Method 2: Using ROW_NUMBER() (SQL Server, PostgreSQL)

```sql
WITH RowCTE AS (
    SELECT Name, ROW_NUMBER() OVER (ORDER BY EmpID) AS RowNum FROM Employees
)
SELECT Name FROM RowCTE WHERE RowNum % 2 = 0; -- Even
SELECT Name FROM RowCTE WHERE RowNum % 2 = 1; -- Odd
```

Conclusion:

- Use MOD() for databases like MySQL.
- Use ROW_NUMBER() for databases that support Window Functions.

70. What is JSON in SQL?

JSON (JavaScript Object Notation) in SQL is used to store, retrieve, and manipulate data in a structured, text-based format within relational databases.

Key Points:

- Stores data in key-value pairs.
- Lightweight and easy to read.
- Commonly used for semi-structured data.
- Supported in MySQL, SQL Server, and PostgreSQL.

MySQL Example:

Create Table with JSON Column:

```
CREATE TABLE Employees (
    EmpID INT,
    Name VARCHAR(50),
    Details JSON
);
```

Insert JSON Data:

```
INSERT INTO Employees VALUES (101, 'Vinay', '{"City": "Bangalore", "Age": 25}');
```

Retrieve JSON Data:

```
SELECT Name, Details->>'$.City' AS City FROM Employees;
```

Conclusion:

JSON helps to handle semi-structured data within relational databases without the need for separate NoSQL databases.

71. What is XML in SQL?

XML (Extensible Markup Language) in SQL is used to store, retrieve, and manipulate structured data in a text-based format within relational databases.

Key Points:

- Stores data in hierarchical format using tags.
- Commonly used for data exchange between applications.
- Supported in SQL Server, MySQL, and Oracle.
- Helps to store semi-structured data.

MySQL Example:

Create Table with XML Data:

```
CREATE TABLE Employees (
    EmpID INT,
    Name VARCHAR(50),
    Details TEXT
);
```

Insert XML Data:

```
INSERT INTO Employees VALUES (101, 'Vinay', '<Employee><City>Bangalore</City><Age>25</Age></Employee>');
```

Retrieve XML Data:

```
SELECT Name, Details FROM Employees WHERE EmpID = 101;
```

Conclusion:

XML is used to store and transfer hierarchical data in relational databases, making it easier to exchange data between applications.

72. How to handle NULL values in SQL?

NULL represents missing or unknown data in SQL.

Methods to Handle NULL Values:

1. **IS NULL** – To check if a column contains NULL.

```
SELECT * FROM Employees WHERE Salary IS NULL;
```

2. IS NOT NULL – To check if a column does not contain NULL.

```
SELECT * FROM Employees WHERE Salary IS NOT NULL;
```

3. COALESCE() – Replaces NULL with a default value.

```
SELECT Name, COALESCE(Salary, 0) AS Salary FROM Employees;
```

4. IFNULL() (MySQL) – Replaces NULL with a specified value.

```
SELECT Name, IFNULL(Salary, 0) AS Salary FROM Employees;
```

5. NULLIF() – Returns NULL if two expressions are equal.

```
SELECT NULLIF(Salary, 5000) AS Result FROM Employees;
```

Conclusion:

Use COALESCE() or IFNULL() to replace NULL values and ensure data consistency in queries.

73. What is Dynamic SQL?

Dynamic SQL is a method of constructing and executing SQL statements at runtime instead of writing static queries.

Key Points:

- Allows flexible query creation based on user input or conditions.
- Used for complex queries with varying conditions.
- Helps in Parameterized Queries and stored procedures.
- Increases security risks if not handled properly.

Example (MySQL):

```
SET @query = 'SELECT * FROM Employees WHERE Department = "IT"';
PREPARE stmt FROM @query;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
```

Conclusion:

Dynamic SQL provides flexibility in query execution but should always be used with parameterized queries to prevent SQL injection attacks.

74. How to calculate Percentage in SQL?

You can calculate Percentage in SQL using arithmetic expressions and aggregate functions.

Syntax:

```
SELECT (PartValue * 100.0) / TotalValue AS Percentage
FROM TableName;
```

Example: Query to Calculate Percentage of Employees in Each Department:

```
SELECT Department,
(Employees * 100.0) / (SELECT SUM(Employees) FROM Employee) AS Percentage
FROM Employee;
```

75. How to find the Employees who earn more than their Manager?

To find employees who earn more than their manager, you need to join the Employee table with itself using Self Join.

Example Table:

Employee Table

EmpID	Name	Salary	ManagerID
101	Vinay	50000	NULL
102	Nevendra	40000	101
103	Awadhesh	60000	101
104	Rituraj	45000	102

Query:

```
SELECT E1.Name AS Employee, E1.Salary AS EmployeeSalary, E2.Name AS Manager, E2.Salary AS ManagerSalary
FROM Employees E1
JOIN Employees E2 ON E1.ManagerID = E2.EmpID
WHERE E1.Salary > E2.Salary;
```

Output:

Employee	EmployeeSalary	Manager	ManagerSalary
Awadhesh	60000	Vinay	50000

Conclusion: Use Self Join with a condition to compare employee salaries against their managers' salaries. This query helps in hierarchical data analysis.

Created by: Vinay Kumar Panika

Real-Time Scenarios

76. How to find Duplicate Emails in the Employee Table?

To find duplicate emails in SQL, you can use the GROUP BY clause with the HAVING condition.

Syntax:

```
SELECT Email, COUNT(Email) AS DuplicateCount
FROM Employees
GROUP BY Email
HAVING COUNT(Email) > 1;
```

Explanation:

- GROUP BY groups the records based on the Email column.
- COUNT(Email) counts how many times each email appears.
- HAVING COUNT(Email) > 1 filters only those emails that have more than one occurrence.

77. How to get the Highest Salary in each Department?

You can find the Highest Salary in Each Department using the GROUP BY clause with the MAX() aggregate function.

Syntax:

```
SELECT Department, MAX(Salary) AS HighestSalary
FROM Employees
GROUP BY Department;
```

78. How to find Employees joined in the last 3 months?

To find employees who joined in the last 3 months, you can use the DATE_ADD() or DATEDIFF() function along with the WHERE clause.

Method 1: Using DATEDIFF() (MySQL)

```
SELECT Name, JoiningDate
FROM Employees
WHERE DATEDIFF(CURDATE(), JoiningDate) <= 90;
```

Method 2: Using DATE_ADD() (MySQL)

```
SELECT Name, JoiningDate
FROM Employees
WHERE JoiningDate >= DATE_ADD(CURDATE(), INTERVAL -3 MONTH);
```

79. How to Display the First 5 Records in SQL?

You can display the First 5 Records using the LIMIT or TOP clause depending on the database.

Method 1: Using LIMIT (MySQL)

```
SELECT * FROM Employees
LIMIT 5;
```

Method 2: Using TOP (SQL Server)

```
SELECT TOP 5 * FROM Employees;
```

80. How to find the Number of Employees in each Department?

You can find the Number of Employees in each department using the COUNT() function with the GROUP BY clause.

Syntax:

```
SELECT Department, COUNT(EmpID) AS EmployeeCount
FROM Employees
GROUP BY Department;
```

81. How to find the Last 3 Records in SQL?

To fetch the Last 3 Records in SQL, you can use the ORDER BY clause along with LIMIT.

Method 1: Using ORDER BY with LIMIT (MySQL)

```
SELECT * FROM Employees
ORDER BY EmpID DESC
LIMIT 3;
```

Method 2: Using ROW_NUMBER() (SQL Server, PostgreSQL)

```
WITH CTE AS (
    SELECT *, ROW_NUMBER() OVER (ORDER BY EmpID DESC) AS RowNum
    FROM Employees
)
SELECT * FROM CTE WHERE RowNum <= 3;
```

82. How to find Employees without Managers?

To find employees without managers, you need to filter records where the ManagerID column is NULL or does not exist in the table.

Method 1: Using IS NULL (MySQL, SQL Server, Oracle)

```
SELECT Name, EmpID
FROM Employees
WHERE ManagerID IS NULL;
```

Method 2: Using LEFT JOIN (MySQL, SQL Server, PostgreSQL)

```
SELECT E1.Name AS Employee
FROM Employees E1
LEFT JOIN Employees E2 ON E1.ManagerID = E2.EmpID
WHERE E1.ManagerID IS NULL OR E2.EmpID IS NULL;
```

83. How to find the First Name starting with 'A'?

You can find employee names starting with the letter 'A' using the LIKE operator.

Syntax:

```
SELECT Name
FROM Employees
WHERE Name LIKE 'A%';
```

Explanation:

- **LIKE 'A%'** → Finds names that start with 'A'.
- % → Represents any number of characters after 'A'.

84. How to fetch Alternate Rows from a table?

You can fetch Alternate Rows using the MOD() or ROW_NUMBER() functions based on row position.

Method 1: Using MOD() (MySQL)

Fetch Even Rows:

```
SELECT *
FROM Employees
WHERE MOD(EmpID, 2) = 0;
```

Fetch Odd Rows:

```
SELECT *
FROM Employees
WHERE MOD(EmpID, 2) = 1;
```

Method 2: Using ROW_NUMBER() (SQL Server, PostgreSQL)

Fetch Odd Rows:

```
WITH RowCTE AS (
    SELECT *, ROW_NUMBER() OVER (ORDER BY EmpID) AS RowNum
    FROM Employees
)
SELECT * FROM RowCTE
WHERE RowNum % 2 = 1;
```

Fetch Even Rows:

```
WITH RowCTE AS (
    SELECT *, ROW_NUMBER() OVER (ORDER BY EmpID) AS RowNum
    FROM Employees
)
SELECT * FROM RowCTE
WHERE RowNum % 2 = 0;
```

85. How to swap two columns in SQL?

You can swap the values of two columns using the UPDATE statement with the TEMPORARY variable technique.

Syntax:

```
UPDATE TableName
SET Column1 = Column2,
    Column2 = Column1;
```

Correct Way Using Temporary Variable:

```
UPDATE Employees
SET FirstName = @temp := FirstName,
    FirstName = LastName,
    LastName = @temp;
```

86. How to display the Duplicate Records with their Count?

To display duplicate records along with their occurrence count, use the GROUP BY clause with the HAVING clause.

Syntax:

```
SELECT ColumnName, COUNT(ColumnName) AS Count
FROM TableName
GROUP BY ColumnName
HAVING COUNT(ColumnName) > 1;
```

87. How to find the Highest Salary without using MAX()?

You can find the Highest Salary without using the MAX() function by using the ORDER BY clause with the LIMIT or TOP keyword.

Method 1: Using ORDER BY with LIMIT (MySQL)

```
SELECT Salary
FROM Employees
ORDER BY Salary DESC
LIMIT 1;
```

Method 2: Using Subquery (MySQL, SQL Server)

```
SELECT Salary
FROM Employees E1
WHERE Salary >= ALL (SELECT Salary FROM Employees E2);
```

Method 3: Using NOT IN (MySQL, SQL Server)

```
SELECT Salary
FROM Employees
WHERE Salary NOT IN (SELECT Salary FROM Employees WHERE Salary < (SELECT Salary FROM Employees));
```

Conclusion:

- Use ORDER BY with LIMIT for simple queries.
- Use Subqueries or ALL for advanced scenarios.
- This method works in all RDBMS.

88. How to fetch common records from two tables without JOIN?

You can fetch common records from two tables without using JOIN by using the IN or INTERSECT operators.

Method 1: Using IN (MySQL, SQL Server)

```
SELECT Name
FROM Employees
WHERE Name IN (SELECT Name FROM Managers);
```

Method 2: Using INTERSECT (SQL Server, PostgreSQL)

```
SELECT Name FROM Employees
INTERSECT
SELECT Name FROM Managers;
```

89. How to delete Duplicate Records from a table?

You can delete duplicate records using CTE, ROW_NUMBER(), or GROUP BY methods.

Method 1: Using ROW_NUMBER() (SQL Server, PostgreSQL)

```
WITH CTE AS (
    SELECT EmpID, Name,
           ROW_NUMBER() OVER (PARTITION BY Name ORDER BY EmpID) AS RowNum
    FROM Employees
)
DELETE FROM Employees WHERE EmpID IN (
    SELECT EmpID FROM CTE WHERE RowNum > 1
);
```

Method 2: Using GROUP BY with MIN() (MySQL)

```
DELETE E1 FROM Employees E1
JOIN Employees E2
ON E1.Name = E2.Name AND E1.EmpID > E2.EmpID;
```

Method 3: Using DISTINCT INTO Temporary Table (MySQL)

```
CREATE TEMPORARY TABLE Temp AS
SELECT DISTINCT * FROM Employees;

TRUNCATE TABLE Employees;
INSERT INTO Employees SELECT * FROM Temp;
DROP TEMPORARY TABLE Temp;
```

Conclusion:

- Use ROW_NUMBER() for advanced databases.
- Use GROUP BY for simple queries.
- Always backup data before deleting duplicates.

90. How to find the Department with the highest Employee Count?

You can find the Department with the Highest Employee Count using the GROUP BY clause along with the ORDER BY and LIMIT clauses.

Method 1: Using GROUP BY with ORDER BY (MySQL)

```sql
SELECT Department, COUNT(EmpID) AS EmployeeCount
FROM Employees
GROUP BY Department
ORDER BY EmployeeCount DESC
LIMIT 1;
```

Method 2: Using Subquery (MySQL, SQL Server)

```sql
SELECT Department, EmployeeCount
FROM (
    SELECT Department, COUNT(EmpID) AS EmployeeCount
    FROM Employees
    GROUP BY Department
) AS DeptCount
ORDER BY EmployeeCount DESC
LIMIT 1;
```

Conclusion:

- Use GROUP BY with ORDER BY for a simple approach.
- Use Subqueries for better performance with large datasets.

Created by: Vinay Kumar Panika

Optimization Techniques

91. How to Optimize SQL Queries?

Optimizing SQL queries improves performance and execution speed while handling large datasets.

Best Practices to Optimize SQL Queries:

1. **Use Indexes:** Create indexes on columns used in WHERE, JOIN, and ORDER BY clauses.

```
CREATE INDEX idx_name ON Employees(Name);
```

2. Avoid SELECT: Select only required columns instead of using SELECT

```
SELECT Name, Salary FROM Employees;
```

3. Use Joins Efficiently: Use INNER JOIN instead of OUTER JOIN whenever possible.

4. **Use EXISTS Instead of IN:** EXISTS performs better with large datasets.

```
SELECT Name FROM Employees WHERE EXISTS (SELECT 1 FROM Managers WHERE Employees.EmpID = Managers.EmpID);
```

5. Use LIMIT or TOP: Fetch only required rows using LIMIT or TOP.

```
SELECT Name FROM Employees LIMIT 10;
```

6. Avoid Functions in WHERE Clause:

Instead of:

```
SELECT * FROM Employees WHERE UPPER(Name) = 'VINAY';
```

Use:

```
SELECT * FROM Employees WHERE Name = 'Vinay';
```

7. Optimize Subqueries: Use JOIN instead of subqueries whenever possible.

8. Partition Large Tables: Split large tables into smaller partitions.

9.Use CTEs and Temp Tables: Store temporary results for better performance.

10. Analyze Execution Plan: Use EXPLAIN or Query Plan to check how queries are executed.

```
EXPLAIN SELECT * FROM Employees;
```

92. What is Query Execution Plan?

A Query Execution Plan is a detailed roadmap used by the database engine to execute SQL queries efficiently.

Key Points:

- Shows how the database will retrieve data.
- Helps to analyze performance and optimize queries.
- Displays the order of execution for each query operation like joins, scans, and sorting.
- Provides information about indexes, table scans, and filtering methods.

How to View Execution Plan:

MySQL:

```
EXPLAIN SELECT Name, Salary FROM Employees WHERE Department = 'IT';
```

SQL Server:

```
SET SHOWPLAN_ALL ON;
SELECT Name, Salary FROM Employees WHERE Department = 'IT';
SET SHOWPLAN_ALL OFF;
```

Why Use Execution Plan?

- Identify performance bottlenecks.
- Check if the query is using Indexes or Table Scans.
- Understand how Joins and Filters are applied.

Created by: Vinay Kumar Panika

93. How to Improve Query Performance?

Improving Query Performance ensures faster data retrieval and better database efficiency, especially for large datasets.

Best Practices to Improve Query Performance:

1. **Use Indexes:** Create indexes on columns used in WHERE, JOIN, and ORDER BY clauses.

```
CREATE INDEX idx_name ON Employees(Name);
```

2. Avoid SELECT: Select only required columns instead of using SELECT

```
SELECT Name, Salary FROM Employees;
```

3. Use Joins Efficiently: Prefer INNER JOIN over OUTER JOIN when possible.

4. Use WHERE Instead of HAVING: Filter rows early using WHERE.

```
SELECT Name FROM Employees WHERE Salary > 5000;
```

5. Limit Results: Fetch only the required number of rows.

```
SELECT Name FROM Employees LIMIT 10;
```

6. Use EXISTS Instead of IN: EXISTS performs better with subqueries.

```
SELECT Name FROM Employees
WHERE EXISTS (SELECT 1 FROM Managers WHERE Employees.EmpID = Managers.EmpID);
```

7. Avoid Functions in WHERE Clause:

Instead of:

```
SELECT * FROM Employees WHERE UPPER(Name) = 'VINAY';
```

Use:

```
SELECT * FROM Employees WHERE Name = 'Vinay';
```

8. Partition Large Tables: Split large tables into smaller partitions.

9. Use Temporary Tables or CTEs: Store temporary results for better performance.

10. Analyze Execution Plans: Use EXPLAIN or SHOWPLAN to understand query execution.

94. What is Indexing?

Indexing in SQL is a technique used to improve the speed of data retrieval from a database by creating a lookup table for faster access.

Key Points:

- Works like a book index to find data quickly.
- Reduces the time required for SELECT queries.
- Automatically updated when data is inserted, updated, or deleted.
- Indexes are created on columns frequently used in WHERE, JOIN, and ORDER BY clauses.

Types of Indexes:

1. **Primary Index** – Automatically created on Primary Key columns.
2. **Unique Index** – Ensures that values in a column are unique.
3. **Clustered Index** – Sorts data rows physically based on key values (Only one per table).
4. **Non-Clustered Index** – Stores pointers to data rows (Multiple indexes allowed).

Conclusion: Indexing improves query performance by quickly locating data but may increase the time for INSERT, UPDATE, and DELETE operations.

95. What is Table Partitioning?

Table Partitioning is a technique used to divide large tables into smaller, more manageable pieces without changing the table structure.

Key Points:

- Improves query performance on large datasets.
- Simplifies data management.
- Helps in faster data retrieval.
- Each partition is stored separately.
- Data can be partitioned by range, list, hash, or composite methods.

Types of Partitioning:

1. **Range Partitioning** – Divides data based on value ranges.
2. **List Partitioning** – Divides data based on specific column values.
3. **Hash Partitioning** – Distributes data evenly using a hash function.
4. **Composite Partitioning** – Combination of Range and Hash partitioning.

Syntax (MySQL Range Partitioning):

```
CREATE TABLE Employees (
    EmpID INT,
    Name VARCHAR(50),
    JoiningDate DATE
)
PARTITION BY RANGE (YEAR(JoiningDate)) (
    PARTITION p1 VALUES LESS THAN (2022),
    PARTITION p2 VALUES LESS THAN (2023),
    PARTITION p3 VALUES LESS THAN (2024)
);
```

Example Query:

```
SELECT * FROM Employees
PARTITION (p2);
```

Conclusion: Table Partitioning improves performance and data management by splitting large datasets into smaller, more manageable sections.

96. How to Avoid Deadlocks in SQL?

A Deadlock occurs when two or more transactions block each other by holding locks on resources that the other transactions need.

Ways to Avoid Deadlocks:

1. **Access Tables in the Same Order:**
   - Always access tables in a consistent sequence across transactions.
2. **Minimize Lock Time:**
   - Keep transactions short and fast to reduce lock holding time.
3. **Use Lower Isolation Levels:**
   - Use READ COMMITTED instead of SERIALIZABLE isolation level if possible.
4. **Avoid User Interaction Inside Transactions:**
   - Do not wait for user input during a transaction.
5. **Use NOLOCK or Read Uncommitted:**
   - Allow non-blocking reads for read-only operations.

```
SELECT * FROM Employees WITH (NOLOCK);
```

6. **Proper Indexing:**
    - Use Indexes to minimize the number of rows locked.
7. **Break Large Transactions:**
    - Split large transactions into smaller batches.

Example:

Without Deadlock Prevention:

```
BEGIN TRANSACTION;
UPDATE Employees SET Salary = 60000 WHERE EmpID = 101;
UPDATE Departments SET Budget = 500000 WHERE DeptID = 1;
COMMIT;
```

With Deadlock Prevention:

```
BEGIN TRANSACTION;
UPDATE Departments SET Budget = 500000 WHERE DeptID = 1;
UPDATE Employees SET Salary = 60000 WHERE EmpID = 101;
COMMIT;
```

Conclusion:

Follow consistent table access patterns, minimize lock times, and use proper indexing to avoid deadlocks in SQL transactions.

97. What is the use of EXISTS in SQL?

EXISTS in SQL is used to check whether a subquery returns any rows. It returns:

- TRUE if the subquery returns at least one row.
- FALSE if the subquery returns no rows.

Syntax:

```
SELECT column_name
FROM table1
WHERE EXISTS (SELECT 1 FROM table2 WHERE table1.id = table2.id);
```

Conclusion: EXISTS is faster than IN for large datasets and is commonly used in subqueries to check data existence.

98. What is Query Optimization?

Query Optimization is the process of improving the efficiency and performance of SQL queries to retrieve data faster while using minimal system resources.

Key Points:

- Helps in reducing query execution time.
- Improves database performance.
- Minimizes CPU usage, memory usage, and I/O operations.
- Automatically performed by the Query Optimizer in most RDBMS.

Techniques for Query Optimization:

1. Use Indexes to speed up searches.
2. Avoid SELECT and select only necessary columns.
3. Use JOINs instead of subqueries when possible.
4. Use EXISTS instead of IN for large datasets.
5. Avoid Functions in WHERE clause.
6. Use LIMIT or TOP to fetch only required rows.
7. Analyze the Execution Plan using tools like EXPLAIN.

Conclusion:

Query Optimization improves execution speed, reduces resource usage, and enhances overall database performance.

99. What is the Difference Between Stored Procedure and Function in SQL?

Stored Procedure	Function
Can return multiple values.	Returns only one value (scalar or table).
Can perform DML operations like INSERT, UPDATE, DELETE.	Cannot perform DML operations.
Supports input and output parameters.	Only supports input parameters.
Can call Functions inside it.	Cannot call Stored Procedures inside it.
Used for business logic and complex operations.	Used for calculations and returning values.

100. What is Query Optimization?

OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
Used for day-to-day transactional operations.	Used for data analysis and reporting.
Focuses on data consistency and speed.	Focuses on data aggregation and analysis.
Stores detailed transactional data.	Stores historical and summarized data.
Supports INSERT, UPDATE, DELETE operations.	Supports SELECT operations with complex queries.
Small amounts of data processed per transaction.	Large amounts of data processed at once.
Example: Banking systems, E-commerce websites.	Example: Data Warehouses, Business Intelligence Tools.