SQL Interview Questions

1. Difference Between nvarchar and varchar?

Feature	VARCHAR	NVARCHAR
Full Form	Variable-length character data	National Variable-length character data
Storage	1 byte per character (ASCII only)	2 bytes per character (Unicode)
Character Support	Only **non-Unicode characters** (ASCII)	Supports **Unicode characters** (e.g., Chinese, Arabic)
Maximum Size	Up to 8000 characters in SQL Server	Up to 4000 characters (or MAX for large data)
Usage	When only **English or ASCII text** is stored	When **multilingual/unicode text** is stored
Performance	Slightly faster and uses less storage	Slightly slower and uses more storage
Prefix	No prefix	N prefix required for Unicode literals

---

2. What is Primary Key,Foreign Key,Unique Key and Composite Key?

Primary Key : A **Primary Key** is a column (or set of columns) that uniquely identifies each row in a table.

- It does not allow NULL values.
- Only one primary key can exist in a table.

Foreign Key : A **Foreign Key** creates a relationship between two tables by referencing the **Primary Key** of another table.

- It ensures **referential integrity** (child table cannot have a value that does not exist in the parent table).

Unique Key : A **Unique Key** ensures that all values in a column (or set of columns) are unique.

- Unlike Primary Key, it **allows one NULL value**.

- A table can have **multiple unique keys**.

Composite Key : A **Composite Key** is a primary key made up of **two or more columns**.

- It's used when a single column is not enough to uniquely identify rows

---

3. What is Constrains in SQL?

In SQL, **Constraints** are rules applied on columns of a table to **enforce data integrity, accuracy, and reliability**.

- They prevent invalid data from being entered into the database.
- Not Null -> Ensure a column cannot have null values.
- Unique -> Ensure all values in column are unique.
- Primary Key -> Not Null + Unique. And AutoIncrement Identity
- Foreign Key -> Maintains **referential integrity** by linking two tables.
- Check -> Ensure that value meet specific condition.
- Default -> Provide a default value if none is specified.

---

4. What is Normalization?

Normalization is the process of **organizing data in a database** to remove redundancy (duplicate data) and improve data integrity.

- It breaks large tables into smaller related tables and establishes relationships between them.

---

5. Explain 1NF , 2NF , 3NF , 4NF , 5NF , BCNF with Example.

Types of Normal Form:

1NF :

- Each column should have **atomic values** (no multiple values in one column).
- No repeating groups.

Before :-

```pgsql
StudentID | Name   | Subjects
----------|--------|-----------------
1         | John   | Math, Science
2         | Alice  | English, History
```

After :-

```pgsql
StudentID | Name   | Subject
----------|--------|---------
1         | John   | Math
1         | John   | Science
2         | Alice  | English
2         | Alice  | History
```

2NF :

- Table must be in **1NF**.
- No **partial dependency** (non-key column should depend on the full primary key, not part of it).

Before : -

```
(OrderID, ProductID) → Primary Key
OrderDate depends only on OrderID ❌ (Partial dependency)
```

After :

Fix → Split into two tables:

- Orders(OrderID, OrderDate)
- OrderDetails(OrderID, ProductID, Quantity)

3NF :

- Table must be in **2NF**.

- No **transitive dependency** (non-key column should not depend on another non-key column).

Before :

```
EmployeeID | Name    | DeptID | DeptName
```

❌ Problem → DeptName depends on DeptID (not directly on Primary Key).

After :

- Employees(EmployeeID, Name, DeptID)
- Departments(DeptID, DeptName)

BCNF : (Boyce-codd Normal Form)

- A stronger version of 3NF.
- **Rule:** For every functional dependency (X → Y), **X must be a super key**.
- Fixes anomalies that 3NF sometimes cannot.

Before :-

```
ProfessorID | Subject   | Department
---------------------------------
1           | Math      | Science
1           | Physics   | Science
```

- Functional dependency: Subject → Department (but Subject is not a key).
- This violates BCNF.

After :

Split into:

- Professors(ProfessorID, Subject)
- Subjects(Subject, Department)

4NF :

- Deals with **multi-valued dependencies**.
- **Rule:** A table should not have more than one **independent multi-valued dependency**.

Before :

StudentID	Hobby	Language
1	Cricket	English
1	Cricket	Hindi
1	Painting	English
1	Painting	Hindi

- Here, `Hobby` and `Language` are independent multi-valued attributes of `StudentID`.
- This causes redundancy.

After :

- Students_Hobbies(StudentID, Hobby)
- Students_Languages(StudentID, Language)

5NF :

- Deals with **join dependencies**.
- Ensures a table is decomposed into smaller tables in such a way that you can **reconstruct it using joins without redundancy**.
- Mainly used in **complex data models**.

Before :

Supplier	Product	Customer
S1	P1	C1
S1	P1	C2
S1	P2	C1

- A supplier can supply multiple products to multiple customers.
- Redundancy exists.

After :

- SupplierProduct(Supplier, Product)
- ProductCustomer(Product, Customer)
- SupplierCustomer(Supplier, Customer)

---

6. What is Temp Table in SQL?

A **Temporary Table** in SQL is a table that is created and stored in the **tempdb** database, and it exists only for the duration of a session or query.

- It is mainly used to **store intermediate results** or perform complex queries in steps without affecting the main database tables.

Type of Temp Tables :

Local Temporary Table : (#)

- Exists only for the current session/connection.
- Automatically dropped when the session ends.
- Name starts with #.

Example:-

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

INSERT INTO #TempEmployees VALUES (1, 'John'), (2, 'Alice');

SELECT * FROM #TempEmployees;  -- Works only in this session
```

Global Temporary Table : (##)

- Can be accessed by **multiple sessions/users**.
- Dropped automatically when the last session using it closes.
- Name starts with **##**.

Exmaple :-

```
CREATE TABLE ##GlobalTemp (
   DeptID INT,
   DeptName VARCHAR(50)
);

INSERT INTO ##GlobalTemp VALUES (1, 'HR'), (2, 'IT');

SELECT * FROM ##GlobalTemp;  -- Accessible by all sessions
```

---

7. What is View In SQL?

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

- It does not store data itself, but shows data stored in underlying tables.
- It's mainly used to **simplify complex queries, enhance security, and provide abstraction**.

Key points:

- A **view does not hold data** → it fetches data from base tables whenever queried.
- Can be used to **restrict access** → expose only specific columns/rows.
- Helps in **simplifying complex joins and aggregations**.
- Can be **updated**, but with some restrictions.

Example:

```
CREATE VIEW EmployeeSalaries AS
SELECT Name, Salary
FROM Employees;

-- Use the View
SELECT * FROM EmployeeSalaries;
```

---

8. What is With Keyword use in Query?

The WITH keyword in SQL is used to define a **Common Table Expression (CTE)**.

- A **CTE** is a temporary result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement.
- It makes queries **easier to read, maintain, and reuse**, especially when dealing with complex joins or recursive queries.

Example:

```
WITH CTE_Name AS (
   SELECT column1, column2
   FROM TableName
   WHERE condition
)
SELECT * FROM CTE_Name;
```

---

9. How many Type of Join in sql and Explain it.

A **JOIN** in SQL is used to combine rows from **two or more tables** based on a related column between them.

- **INNER JOIN**: only matching rows,
- **LEFT JOIN**: all left table rows + matched right rows,
- **RIGHT JOIN**: all right table rows + matched left rows,
- **FULL OUTER JOIN**: all rows from both tables.
- **CROSS JOIN** → Returns **Cartesian product** (all combinations of rows).
- **SELF JOIN** → Joining a table with itself (useful for hierarchical data).

---

10. What is Self Join give the example of it.

A **Self Join** is a join where a table is joined **with itself**.

- It is used when a table has a hierarchical or related data within the same table.
- You need to use table aliases to differentiate the “left” and “right” instances of the table.

SELECT e.Name AS Employee, m.Name AS Manager

FROM Employees e

LEFT JOIN Employees m ON e.ManagerID = m.EmpID;

Find Employee wise Manager.

---

11. What is the Group BY ?

The GROUP BY clause in SQL is used to group rows that have the same values in specified columns and allows aggregate calculations like COUNT, SUM, AVG, MIN, and MAX.

- It is usually used **with aggregate functions** such as COUNT(), SUM(), AVG(), MIN(), MAX().
- Helps in summarizing data for reports and analysis.

---

12. what is Aggregaton Function?

An Aggregation Function in SQL performs a calculation on a set of values and returns a single summarised value.

- These functions are mainly used to **analyze data** or generate reports.
- Often used with **GROUP BY** to summarize data per group.

---

13. Give the Flow of Query Execution?

- **FROM** → Identify tables and perform **joins**.
- **ON** → Apply **join conditions** (if any).
- **JOIN** → Combine tables based on join conditions.
- **WHERE** → Filter rows **before grouping**.
- **GROUP BY** → Group the filtered rows by specified columns.
- **HAVING** → Filter groups based on **aggregate conditions**.
- **SELECT** → Select columns and compute **expressions/aggregates**.
- **DISTINCT** → Remove duplicate rows (if used).
- **ORDER BY** → Sort the result set.
- **LIMIT / OFFSET** → Return specified number of rows.

14. Can We use Aggregate Function in Where Condition?

No, we cannot use aggregate functions like COUNT(), SUM(), AVG(), MAX(), or MIN() directly in the WHERE clause.

- WHERE filters rows **before aggregation**.
- Aggregate functions work **after rows are grouped**, so they are not available at the WHERE stage.
- **Correct way:** Use **HAVING** clause to filter aggregated results.

---

15. difference between Where and Having?

Feature	**WHERE**	**HAVING**
Purpose	Filters individual rows	Filters groups of rows (after aggregation)
Used With	Any table or view	Usually with GROUP BY
Aggregate Functions	Cannot use (SUM, COUNT, AVG)	Can use (SUM, COUNT, AVG)
Execution Order	Before grouping	After grouping
Example	WHERE Salary > 50000	HAVING COUNT(*) > 3

---

16. can we write Having Before Where?

No, we cannot write HAVING before WHERE.

- SQL executes queries in a **logical order**.
- **WHERE** filters rows **before grouping**.
- **HAVING** filters groups **after aggregation**.

---

17. What is Window Function? List of It and give the example of it.

A Window Function performs a calculation across a set of rows related to the current row, called a window, without collapsing the rows into a single output (unlike aggregation functions).

- Window functions are often used for ranking, running totals, moving averages, and analytics.
- They do not reduce the number of rows in the result set.

Examples of window functions :

ROW_NUMBER() : Assigns a unique sequential number to rows within a partition

```
SELECT EmpID, Name, DeptID, Salary,
ROW_NUMBER() OVER(PARTITION BY DeptID ORDER BY Salary DESC) AS RowNum
FROM Employees;
```

RANK() : Assigns rank with gaps for ties

```
SELECT EmpID, Name, DeptID, Salary,
    RANK() OVER(PARTITION BY DeptID ORDER BY Salary DESC) AS RankInDept
FROM Employees;
```

Running Total using SUM() :

```
SELECT EmpID, Name, DeptID, Salary,
    SUM(Salary) OVER(PARTITION BY DeptID ORDER BY EmpID) AS RunningTotal
FROM Employees;
```

- DENSE_RANK() : Assigns rank without gaps for ties
- NTILE(n) : Divides rows into n buckets
- Sum() : Cumulative sum over a window

---

18. What is Store Procedure in Sql and explain the use of it.

A **Stored Procedure** is a **precompiled collection of SQL statements** (like SELECT, INSERT, UPDATE, DELETE, etc.) stored in the database and executed as a single unit.

- It can accept **parameters**, return **values**, and include **business logic** (loops, conditions).
- Since it is stored in the database, it improves **performance**, **reusability**, and **security**.

---

19. what is the function in SQL and Explain the use of it.

A **Function** in SQL is a reusable database object that performs a calculation or operation and **returns a single value or a table**.

- Functions are used to **encapsulate reusable logic**.
- Unlike stored procedures, a function **must return a value**.
- Functions can be **system-defined** (built-in) or **user-defined**.
- It is use for Insert,Update or Delete Operations.

---

20. how many type of function in SQL.

System Functions (Built-in):

- Examples:
    - LEN() → returns string length
    - GETDATE() → returns current date
    - SUM(), AVG(), MIN(), MAX() → aggregate functions

User-Defined Functions (UDFs):

- **Scalar Function** → Returns a **single value**
- **Table-Valued Function (TVF)** → Returns a **table**

---

21. What is User Define Function in SQL.

A **User-Defined Function (UDF)** is a function created by the user in SQL Server (or other RDBMS) to perform a specific task.

Scalar Function

- Returns a **single value** (int, decimal, varchar, etc.)
- Can be used in the SELECT list, WHERE clause, etc.

Inline Table-Valued Function (Inline TVF)

- Returns a **table** using a single SELECT statement.
- Works like a view but can accept parameters.

Multi-Statement Table-Valued Function (Multi-TVF)

- Returns a **table**, but allows **multiple statements** inside.
- More flexible but slightly slower.

---

22. Can insert,update or Delete operation in the Function.

No, in SQL Server **User-Defined Functions (UDFs) cannot perform permanent changes** (INSERT, UPDATE, DELETE) to database tables.

- Functions are designed for **calculation and returning values only**, not for modifying data.

---

23. What is The Trigger? Give the Example of it.

A **Trigger** is a **special type of stored procedure** that is **automatically executed (fired)** in response to specific Event.

- It is used to **enforce business rules, maintain audit trails, or automatically perform actions** when data changes.

Example :

```
CREATE TRIGGER trg_PreventDeleteEmployee
ON Employees
INSTEAD OF DELETE
```

```
AS
BEGIN
   PRINT 'Deletion not allowed on Employees table!';
END;
```

---

24. What is the Cursor? Give the use and Example of it.

A **Cursor** in SQL is a **database object** that allows you to **retrieve data row by row** from a result set.

Example:

```
DECLARE @EmpName VARCHAR(100);

-- Step 1: Declare Cursor
DECLARE EmployeeCursor CURSOR FOR
SELECT EmpName FROM Employees;

-- Step 2: Open Cursor
OPEN EmployeeCursor;

-- Step 3: Fetch First Row
FETCH NEXT FROM EmployeeCursor INTO @EmpName;

-- Step 4: Loop through all rows
WHILE @@FETCH_STATUS = 0
BEGIN
   PRINT 'Employee: ' + @EmpName;

   FETCH NEXT FROM EmployeeCursor INTO @EmpName;  -- Next row
END;

-- Step 5: Close Cursor
CLOSE EmployeeCursor;

-- Step 6: Deallocate Cursor
DEALLOCATE EmployeeCursor;
```

---

25. What is SQL Injection ? how to prevent it.

SQL Injection is a vulnerability where attackers inject malicious SQL into input fields to gain unauthorized access to a database.

Prevent:

- Use parameterized queries / prepared statements

- Use stored procedures correctly
- Use ORMs

---

26. What Trasaction in SQL? Explain the use of it. and give example of it.

A **Transaction** in SQL is a **sequence of one or more SQL operations executed as a single unit of work**.

- Either **all operations succeed** (commit)
- Or **none take effect** (rollback)
- It follow's the ACID Property

Example:

```
BEGIN TRANSACTION;

-- Step 1: Deduct money from Account A
UPDATE Accounts
SET Balance = Balance - 500
WHERE AccountID = 101;

-- Step 2: Add money to Account B
UPDATE Accounts
SET Balance = Balance + 500
WHERE AccountID = 202;

-- If no error, commit
COMMIT;
-- If any error occurs, rollback
ROLLBACK;
```

---

27. What is ACID Property.

- **Atomicity** → All statements in the transaction succeed or none of them.
- **Consistency** → Data remains in a valid state before and after the transaction.
- **Isolation** → Transactions do not interfere with each other when running simultaneously.
- **Durability** → Once committed, the changes are permanent even if the system crashes.

---

28. List the SQL Components and Give Defination of it.

- DDL (Data Definition Language) : Used to **define or modify database structures** like tables, schemas, or indexes.
- DML (Data Manipulation Language) : Used to **manipulate data** within existing tables insert, update, delete, or retrieve.
- DQL (Data Query Language) : Used to **query data** from the database.

- DCL (Data Control Language) : Used to **control access and permissions** on the database objects.
- TCL (Transaction Control Language) : Used to **manage transactions**, ensuring ACID properties.

---

29. What is Roll Back in SQL? Explain it and provide Example of it.
ROLLBACK is a transaction control command in SQL that undoes all changes made by the current transaction and reverts the database to its previous stable state.
- It is used to **maintain data integrity** if something goes wrong during a transaction.
- Often used with **BEGIN TRANSACTION** and **COMMIT**.

Example:

```
BEGIN TRANSACTION;

-- Deduct money from Account A
UPDATE Accounts
SET Balance = Balance - 500
WHERE AccountID = 101;

-- Add money to Account B
UPDATE Accounts
SET Balance = Balance + 500
WHERE AccountID = 202;

-- Oops! Something went wrong, rollback
ROLLBACK;
```

---

30. What is Indexing? provide the type of it.
Indexing is a database optimization technique used to speed up data retrieval from a table without scanning the entire table.
- Cluster Index
- Non Cluster Index
- Unique Index
- Composite / Multi-Column Index
- Full-Text Index
- Filtered Index

---

31. What is Clustered and Non-Clustered Index.

Clustered Index

- Determines the **physical order of data rows** in the table.

- A table can have **only one clustered index**.
- **Faster for range queries** because data is stored in order.
- Usually created on **primary key**.

Example :

```
-- Create a clustered index on EmpID
CREATE CLUSTERED INDEX idx_EmpID
ON Employees(EmpID);
```

Non-Clustered Index

- Creates a **separate structure** (like a pointer) pointing to the actual rows.
- A table can have **multiple non-clustered indexes**.
- Does **not change physical order** of data.
- Best for queries that **search specific columns** frequently.

Example :

```
-- Create a non-clustered index on Name
CREATE NONCLUSTERED INDEX idx_Name
ON Employees(Name);
```

Feature	**Clustered Index**	**Non-Clustered Index**
Physical Order	Data rows sorted physically	Data rows not sorted physically
Number per Table	Only 1	Multiple allowed
Speed	Faster for range queries	Faster for selective lookups
Use Case	Primary key or range searches	Columns frequently searched
Storage	Part of table itself	Separate structure with pointers

---

32. What is Subquery? and How to Use of it. and give the example of it

A **Subquery** (or Inner Query / Nested Query) is a **query inside another query**.

- The subquery executes first, and its result is used by the **outer query**.
- Subqueries are useful when a value or set of values is needed for comparison or filtering.

Example: Second Highest Salary

```
SELECT MAX(Salary)
FROM Employees
WHERE Salary < (SELECT MAX(Salary) FROM Employees);
```

---

33. Difference Between IN,Exist and Any.

IN : Checks if a value matches **any value in a list or subquery result**.

Exist : Checks if the **subquery returns at least one row**.

Any : Compares a value to **any value returned by a subquery**.

Difference :

Difference Table (IN vs EXISTS vs ANY)

Feature	IN	EXISTS	ANY / SOME
Purpose	Check value in a list/subquery	Check if subquery returns rows	Compare value with subquery results
Returns	TRUE if value exists	TRUE if at least one row exists	TRUE if comparison matches any row
Correlated Subquery	Optional	Commonly correlated	Usually used with subquery
Performance	Fast for small lists	Better for large datasets	Depends on subquery and operator
Example	DeptID IN (1,2,3)	EXISTS (SELECT * FROM Dept ...)	Salary > ANY (SELECT Salary FROM ...)

---

34. Difference Between Delete,Truncate and Drop.

Feature	DELETE	TRUNCATE	DROP
Purpose	Remove **specific rows** from a table	Remove **all rows** from a table	Remove **entire table** (structure + data)
Condition / Filter	Can use **WHERE clause** to delete specific rows	Cannot use WHERE; deletes all rows	Not applicable
Transaction Log	Records **individual row deletions** in log	Minimal logging; faster	Removes table from database
Trigger Activation	Activates DELETE triggers	Does **not activate** triggers	Triggers are removed as table is dropped
Rollback	Can be **rolled back** if within a transaction	Can be rolled back in some DBs (e.g., SQL Server)	Cannot rollback in most DBs
Speed	Slower for large tables (row-by-row)	Faster; deallocates pages	Fast; table completely removed

Feature	DELETE	TRUNCATE	DROP
Effect on Structure	Table structure remains	Table structure remains	Table structure is deleted

Delete : Not Change the identity

Truncate : Change the identity to 1

---

35. What is the CTE.

A CTE (Common Table Expression) is a temporary named result set in SQL that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement.

- It exists **only during the execution of the query**.
- Helps make **complex queries readable and maintainable**.
- Often used for **recursive queries**.

---

36. What is the Pivot in SQL. and how to use of it.

A **Pivot** in SQL is used to **transform rows into columns**.

- It is helpful in creating **cross-tab reports**, summarizing data, and making the data easier to read.
- Pivoting is often combined with **aggregate functions** like SUM, COUNT, MAX, etc.

Example :

```
SELECT Product, [Jan], [Feb]

FROM

(

   SELECT Product, Month, Amount

   FROM Sales

) AS SourceTable

PIVOT

(

   SUM(Amount) FOR Month IN ([Jan], [Feb])

) AS PivotTable;
```

---

37. Difference between Union and Union All.

Feature	UNION	UNION ALL
Definition	Combines **two or more result sets** and **removes duplicate rows**	Combines **two or more result sets** and **keeps all rows including duplicates**
Duplicates	Automatically removes duplicates	Does not remove duplicates
Performance	Slower because it checks for duplicates	Faster because no duplicate check
Use Case	When **unique records** are needed	When **all records including duplicates** are needed
Sorting	May require extra processing	No extra processing for duplicates
Example	SELECT Name FROM Employees UNION SELECT Name FROM Contractors;	SELECT Name FROM Employees UNION ALL SELECT Name FROM Contractors;
