Edition 1

SQL Server

Simplified

Interview Questions And Answers

Vishal Garg

SQL SERVER

INTERVIEW QUESTION AND ANSWERS

Copyrights

SQL Server Simplified Interview Question and Answers

vishgeek@gmail.com

About the Book

SQL Server Simplified interview questions is designed to help readers learn the basic concepts of SQL Server.

This book covers all the concepts of SQL Server with the help of Interview question and Answers.

Contents

Copyrights
About the Book
Q1. What are Temporary tables?
Q2. What are Temp variables?
Q3. What is Table valued parameter (TVP)?
Q4. Difference between Temporary variables and Temporary tables?
Q5. What are Sub-Queries?
Q6. What is a Common table expression (CTE)?
Q7. What are Types of CTE?
Q8. Difference between CTE and Temp Table?
Q9. What is a Stored Procedure?
Q10. How to optimize performance of stored procedure?
Q11. Why Stored Procedures and not SQL queries?
Q12. What are User Defined Functions (UDF)?
Q13. What are Guidelines for creating a Function?
Q14. What are types of Function?
Q15. How function can return a table?
Q16. Difference between Function and Stored Procedure?
Q17. What are Views?
Q18. What are advantages Views?
Q19. What are disadvantages Views?
Q20. Can we update table in Views?
Q21. What are different types of Views?
Q22. Difference between VIEWS and FUNCTIONS?
Q23.Types of Joins?
Q24. What are Indexes?
Q25. What are Different ways to create Indexes?
Q26. When Indexes should be avoided?
Q27. What are benefits of Indexes?
Q28. What are Types of Indexes?
Q29. Difference between clustered and nonclustered Indexes?
Q30. Difference between primary key and clustered index?
Q31. Can we create clustered index on unique key?
Q32. What is Partition By?

Q33. Difference between GroupBy and Partition By?
Q34. What are RANK, DENSE_RANK and ROW_NUMBER functions?
Q35. Find 2nd Highest salary using DENSE_RANK ()?
Q36. Query to delete duplicate data in table?
Q37. Difference between Truncate and Delete?
Q38. How to do Transaction and Rollback?
Q39. What are Triggers?
Q40. What are constraints available in Sql server?
Q41. What is UNIQUE key Constraint?
Q42. What is PRIMARY key Constraint?
Q43. Can we create multiple primary key on a table?
Q44. What is FOREIGN key Constraint?
Q45. How to create a FOREIGN key Constraint?
Q46. What is difference between PRIMARY and FOREIGN key constraint?
Q47. Difference between varchar and nvarchar?
Q48. What is Group By Clause?
Q49. What is Having Clause?
Q50. Write some Group By Queries?
Q51. What is Union operator?
Q52. What is UnionAll operator?
Q53. What is COALESCE?
Q54. Use of Substring and Charindex?
Q55. What is STUFF() Function?
Q56. What is an IDENTITY column?
Q57. Difference between SEQUENCE and IDENTITY?
About the Author
More Books by this Author

Q1. What are Temporary tables?

Temporary tables get created in the TempDB and are automatically deleted, when they are no longer used.

There are 2 types of Temporary tables - **Local Temporary** tables and **Global Temporary** tables.

Local Temporary tables -

- **A local temporary table is available, only for the connection** that has created the table.
- To create a local Temporary table prefix the **table name with 1 pound (#) symbol**.
- **A local temporary table is automatically dropped**, when the connection that has created it, is closed.
- To explicitly drop temp table: - `DROP TABLE #tempEmployee`
- **If the temporary table, is created inside the stored procedure**, it gets dropped automatically upon the completion of stored procedure execution.

e.g.

```
create Procedure spCreateLocalTempTable
as
Begin
Create Table #tempEmployee(Id int, Name nvarchar(20))

Insert into #tempEmployee Values(1, 'Mike')
Insert into #tempEmployee Values(2, 'John')
Insert into #tempEmployee Values(3, 'Todd')

Select * from #tempEmployee
End
```

Note: Temporary table will be destroyed immediately after the completion of the stored procedure execution.

```
execute spCreateLocalTempTable; // Will give the result set

Select * from #tempEmployee
// Error: Invalid object name '#tempEmployee'.
```

Global Temporary Table:

- **Global temporary tables are visible** to all the connections and are only destroyed when the last connection referencing the table is closed.
- To create a Global Temporary Table, prefix the name of the table with 2 pound (##) symbols.
- **Multiple users, across multiple connections** can have local temporary tables with the same name, but, a global temporary table name has to be unique

e.g.

```
Create Procedure spCreateGlobalTempTable
as
Begin
Create Table ##globalEmployee(Id int, Name nvarchar(20))

Insert into ##globalEmployee Values(1, 'Mike')
Insert into ##globalEmployee Values(2, 'John')
Insert into ##globalEmployee Values(3, 'Todd')

Select * from ##globalEmployee
End

// Execute procedure
execute spCreateGlobalTempTable // Will give the result set
Select * from ##globalEmployee // Success

Note: unlike Local temp tables, we can use global temp tables in other stored procedures as well.
Create Procedure spCreateGlobalTempTable2
as
Begin
Select * from ##globalEmployee
End
execute spCreateGlobalTempTable2 ;

//Note: it will run successfully and will give the desired results
```

Q2. What are Temp variables?

- Temp Variables are also used for holding data temporarily just like a temp table.

- Temp Variables are created using a “DECLARE” statement and are assigned values using either a SET or SELECT command.
- This acts like a variable and exists for a particular batch of query execution.
- It gets dropped once it comes out of the batch.
- This is also created in the tempdb database but not the memory.
- This also allows you to create a primary key, identity at the time of Table variable declaration but not non-clustered index.
- We can implement all DML commands for Temp Variables.

e.g.

```
// Step1: Declare temp variable
Declare @My_vari TABLE
(
    ID int,
    Name Nvarchar(50),
    Salary Int ,
    City_Name Nvarchar(50)
)
//Step2: insert data
Insert Into @My_vari Values(1,'Mark',1000,'USA')
Insert Into @My_vari Values(2,'Steve',2000,'UK')
//Step3: Select data
select * from @My_vari
```

Q3. What is Table valued parameter (TVP)?
OR
How can we pass table as parameter to Stored Procedure?

- Table Valued Parameter allows a table (i.e. multiple rows of data) to be passed as a parameter to a stored procedure
- Table valued parameters must be passed as ***read-only*** to stored procedures, functions etc.
- You ***cannot perform DML*** operations like INSERT, UPDATE or DELETE on a table-valued parameter in the body of a function, stored procedure etc.

e.g.

Step 1 : Create User-defined Table Type

```
CREATE TYPE EmpTableType AS TABLE
(
    Id INT PRIMARY KEY,
    Name NVARCHAR(50),
    Gender NVARCHAR(10)
)
```

Step 2: Use the User-defined Table Type as a parameter in the stored procedure.

```
CREATE PROCEDURE spInsertEmployees
@EmpTableType EmpTableType READONLY
AS
BEGIN
    INSERT INTO Employees
    SELECT * FROM @EmpTableType
END
```

Step 3: Declare a **table variable**, insert the data and then pass the table variable as a parameter to the stored procedure.

```
DECLARE @EmployeeTableType EmpTableType

INSERT INTO @EmployeeTableType VALUES (1, 'Mark', 'Male')
INSERT INTO @EmployeeTableType VALUES (2, 'Mary', 'Female')
INSERT INTO @EmployeeTableType VALUES (3, 'John', 'Male')


EXECUTE spInsertEmployees @EmployeeTableType
```

Q4. Difference between Temporary variables and Temporary tables?

Temporary variables	Temporary tables
Scope: limited to the current batch and current Stored Procedure	**Scope**: scope of a Temp Table is wider than for Temp Variables. Local temporary tables are temporary tables that are available only to the session that created them and global temporary tables are temporary tables that are available to all sessions and all users.
Creation: Declared using Declare statement only	**Creation**: Temp Tables can be created using Create Table and Select Into commands.

Drop and Truncate Command: We cannot drop or truncate a Temp variable	**Drop and Truncate Command:** We can drop or truncate a Temp Tables.
Constraint: a Temp Variable doesn't support Foreign Keys.	**Constraint**: Temp Tables and Temp Variables both support unique key, primary key, check constraints, Not null and default constraints

Q5. What are Sub-Queries?

A sub-query is a query within a query. It is also called an inner query or a nested query. A sub-query is usually added in a where clause of the SQL statement.

e.g.

```
Select Name,Age, employeeID
From employee
Where employeeID in
(
    Select employeeID from salary where salary >=2000 --Sub Query
)
```

Whenever we refer the same data or join the same set of records using a sub-query, the code maintainability will be difficult.

A CTE makes improved readability and maintenance easier.

Q6. What is a Common table expression (CTE)?

A Common Table Expression, also called as CTE in short form, is a temporary named result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. The CTE can also be used in a View.

Syntax:

The CTE query starts with a “With” and is followed by the Expression Name

```
WITH expression_name [ ( column_name [,...n] ) ]
```

```
AS
( CTE_query_definition )

// To view a CTE
Select * from expression_name
```

Advantages of CTE

- CTE improves the code readability.
- CTE provides recursive programming.
- CTE makes code maintainability easier.

Q7. What are Types of CTE?

There are two types of CTEs: Recursive and Non-Recursive.

1. **Non-Recursive CTEs**

Non-Recursive CTEs are simple where the CTE doesn't use any recursion, or repeated processing in of a sub-routine.

e.g.

```
;with ROWCTE(ROWNO) as
    (
        SELECT ROW_NUMBER() OVER(ORDER BY name ASC) AS ROWNO
        FROM sys.databases
        WHERE database_id <= 10
     )

SELECT * FROM ROWCTE
```

2. **Recursive CTE**

Recursive CTEs are use repeated procedural loops aka recursion. The recursive query call themselves until the query satisfied the condition. In a recursive CTE we should provide a where condition to terminate the recursion.

```
With salaryCTE(EmployeeID)

AS
```

```
(Select employeeID from salary where salary >=1000)
, EmpDetailsCTE( Name, EmployeeID ,salary)
AS
(
Select Name,Age, employeeID
From employee Emp Join salaryCTE sa
on Emp. employeeID = sa. EmployeeID)
```

Q8. Difference between CTE and Temp Table?

CTE	Temp table
CTE stands for Common Table Expressions. It is a temporary result set and typically it may be a result of complex sub-query.	Temporary tables are created at run-time and you can do all the operations which you can do on a normal table. These tables are created inside the Tempdb database.
Scope: Unlike the temporary table, its life is limited to the current query. It is defined by using WITH statement.	**Scope** : Based on the scope and behavior temporary tables are of two types : **Local Temp Table:** Local temp tables are only available to the SQL Server session or connection (means single user) that created the tables. These are automatically deleted when the session that created the tables has been closed. The local temporary table name starts with a single hash ("#") sign. **Global Temp Table:** Global temp tables are available to

	all SQL Server sessions or connections (means all the user). These can be created by any SQL Server connection user and these are automatically deleted when all the SQL Server connections have been closed. The global temporary table name is stared with double hash ("##") sign.
Syntax : WITH cte (Column1, Column2, Column3) AS (SELECT Column1, Column2, Column3 FROM SomeTable) SELECT * FROM cte	Syntax : CREATE TABLE #tmpTable (Column1 int, Column2 varchar(50), Column3 varchar(150)) GO insert into #tmpTable values (1, '2','3'); GO Select * from #tmpTable
Persist only until the next query is run	Persist for the life of the current CONNECTION
Usage: This is used to store the result of a complex subquery for further use.	Usage: We required to hold data from further query.

Q9. What is a Stored Procedure?

Stored Procedure

A Stored Procedure is nothing more than prepared SQL code that you save so you can reuse the code over and over again. Instead of having to write the query each time you would save it as a Stored Procedure and then just call the Stored Procedure to execute the SQL code that you saved as part of the Stored Procedure.

Benefits of Stored Procedures

- *Precompiled execution*

  SQL Server compiles each Stored Procedure once and then reutilizes the execution plan. This results in tremendous performance boosts when Stored Procedures are called repeatedly.

- *Reduced client/server traffic*

  if network bandwidth is a concern in your environment then you'll be happy to learn that Stored Procedures can reduce long SQL queries to a single line that is transmitted over the wire.

- *Efficient reuse of code and programming abstraction*

  Stored Procedures can be used by multiple users and client programs. If you utilize them in a planned manner then you'll find the development cycle requires less time.

- *Enhanced security controls*

  you can grant users permission to execute a Stored Procedure independently of underlying table permissions.

Syntax:

```
// Creating  a Stored procedure

CREATE PROCEDURE procedure_name
AS
sql_statement
GO;

// Running a Stored procedure
EXEC procedure_name;
```

e.g.

```
 // Creating  a Stored procedure with multiple parameters
 CREATE PROCEDURE Proc_Employees @Name nvarchar(30), @EmpId nvarchar(10)
 AS
 SELECT * FROM Employees WHERE Name = @Name AND EmpId =  @EmpId
 GO;
```

```
// Running a Stored procedure with multiple parameters

EXEC Proc_Employees @Name = 'Mike', @EmpId = '100';
```

Q10. How to optimize performance of stored procedure?

1. **Use fully qualified procedure name**

A fully qualified object name is database.schema.objectname. When stored procedure is called as schemaname.procedurename, SQL Server can swiftly find the compiled plan instead of looking for procedure in other schemas when schemaname is not specified.

E.g. `SELECT EmpID, EmpName, EmpSalary FROM dbo.Employee`

2. **Use If Exists to check if record exists in table or not**

If Exists function is used to check if record exists in a table or not. If any records are found then it will return true, else it will return false. One benefit of using If Exists function is that if any match is found then it will stop execution and return true so it will not process the remaining records, so that will save time and improve performance.

E.g. `IF (EXISTS (SELECT 1 FROM db.Employees))`

3. **Specify column names instead of using * in SELECT statement**

4. **Create Proper Index**

Proper indexing will improve the speed of the operations in the database.

5. **Use Join query instead of sub-query**

Using JOIN is better for the performance than using subqueries or nested queries

6. **Set NOCOUNT ON statement at beginning of Store procedure** SET NOCOUNT ON will not return the how many rows are affected message. If we are not using SET NOCOUNT ON statement, then it will print a

message for how many rows are affected. It means if we do not use this then it will print an extra message and it affects performance.

Q11. Why Stored Procedures and not SQL queries?

- Performance: A stored procedure is cached in the server memory and its execution is much faster than SQL Query.
- Network Traffic : Stored procedures produce less network traffic then SQL queries because executing a stored procedure requires only the procedure name and parameters (if any) to be sent over the network. Executing dynamic SQL requires the complete query to be sent across the network, increasing network traffic, particularly if the query is very large.
- SQL Injection Attacks: Stored procedures are not vulnerable to SQL Injection attacks.
- Reusability of Cached Query Plans: Stored procedures improve database performance as they allow cached query plans to be reused.

Q12. What are User Defined Functions (UDF)?

User Defined Functions

Like functions in programming languages, SQL Server User Defined Functions are routines that accept parameters, perform an action such as a complex calculation, and returns the result of that action as a value. The return value can either be a single scalar value or a result set.

Benefits of User Defined Functions

- *They allow modular programming*
  you can create the function once, store it in the database, and call it any number of times in your program. User Defined Functions can be modified independently of the program source code.

- *They can reduce network traffic*

an operation that filters data based on some complex constraint that cannot be expressed in a single scalar expression can be expressed as a function. The function can then invoked in the WHERE clause to reduce the number or rows sent to the client.

e.g.:

```
CREATE FUNCTION  func_info (
                @city nvarchar(10)
)
RETURNS TABLE AS
RETURN
                SELECT *
                FROM dbo.persons
                WHERE city = @city;

//Executing a function
SELECT * from dbo.func_info('lon')
```

Q13. What are Guidelines for creating a Function?

SQL Server Functions are useful objects in SQL Server databases. A SQL Server function is a code snippet that can be executed on a SQL Server.

Here are some of the rules when creating functions in SQL Server.

1. A function must have a name and a function name can never start with a special character such as @, $, #, and so on.
2. ***Functions only work with select statements*.**
3. Functions can be used anywhere in SQL, like AVG, COUNT, SUM, MIN, DATE and so on with select statements.
4. Functions compile every time.
5. Functions *must return a value or result*.
6. Functions *only work with input parameters*.
7. Try and catch statements are not used in functions.

Q14. What are types of Function?

SQL Server supports two types of functions:

1. **User-Defined function**: User-defined functions are create by a user.
2. **System Defined Function**: System functions are built-in database functions.

User-Defined Functions

SQL Server supports two types of user-defined functions:

1. Table-Valued Functions
2. Scalar Valued Functions

1. Table-Valued Functions

In this type of function, we select a table data using a user-created function. A function is created using the Create function SQL command.

e.g.

```
Create function Fun_FuncName()
returns table
AS
return(select * from Employee  )
```

2. Scalar function

Now we are getting a table with two different data joined and displayed in a *single column data row*.

```
create function func_EmployeeInfo
(
    @EmpContact nchar(15),
    @EmpEmail nvarchar(50),
    @EmpCity varchar(30)
)
returns nvarchar(100)
AS
begin return(select @EmpContact+ ' ' +@EmpEmail + ' ' + @EmpCity)
end
```

Q15. How function can return a table?

A table-valued function is a user-defined function that returns data of a table type. The return type of a table-valued function is a table, therefore, you can use the table-valued function just like you would use a table.

E.g.

```
CREATE FUNCTION func_Employee(
    @empId INT
)
RETURNS TABLE
AS
RETURN
    SELECT
        Name,
        Address,
        Salary
    FROM
        Dev.Employee
    WHERE
        empId = @empId;

--Calling Function
SELECT
    Name,
    Salary
FROM
    func_Employee(20);
```

Note: We typically use table-valued functions as parameterized views.

Q16. Difference between Function and Stored Procedure?

Function	**Stored Procedure**
A function has a return type and returns a value.	A procedure does not have a return type. But it returns values using the OUT parameters.
You cannot use a function with Data Manipulation queries. ***Only Select queries*** *are allowed in functions.*	You can use ***DML queries*** such as insert, update, select etc… with procedures.
A function does not allow output parameters	A procedure allows both input and output parameters.

You cannot manage transactions inside a function.	You can manage transactions inside a SP.
You cannot call stored procedures from a function	You can call a function from a stored procedure.
You can call a function using a select statement.	You cannot call a procedure using select statements.

Q17. What are Views?

- The views are a compiled SQL query.
- Consider the Views as virtual tables.
- As a virtual table, the Views do not store any data physically by default.
- when we query a view it actually gets the data from the underlying database tables
- Views in SQL Server act as an interface between the Table(s) and the user.
- Views can also be used to perform DML operations like insert, update and delete data from a table

Syntax:

```
CREATE VIEW VIEW_NAME AS
SELECT column1, column2, column3.....
FROM table_name WHERE [condition];
```

```
--E.g.
CREATE VIEW DEPT_VIEW AS
SELECT EMPLOYEE.ID, EMPLOYEE.NAME, DEPARTMENT.DEPTNAME
FROM EMPLOYEE, DEPARTMENT
WHERE EMPLOYEE.ID = DEPARTMENT.ID;
SELECT * FROM DEPT_VIEW;
--Insert into View
INSERT INTO view_name(column1, column 2, column3,....) VALUES(value1, value2, value3,...);
--Delete from view
DELETE FROM view_name WHERE [condition];
--Drop view
DROP VIEW view_name;
```

Q18. What are advantages Views?

- **Security:** Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see, thus restricting the user's access to stored data
- **Query Simplicity**: A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view.
- **Views don’t take space:** Views are used to store your code, not complete tables. Each time you call a view, you’ll run the related query. Therefore, you don’t lose disk space on views

Q19. What are disadvantages Views?

- **Performance:** Views create the appearance of a table, but the DBMS must still translate queries against the view into queries against the underlying source tables. If the view is defined by a complex, multi-table query then simple queries on the views may take considerable time.
- **Database changes:** One of the major disadvantages of using view comes into the picture when we change the table structures frequently upon which the view is created. So when the table structures are changed, the view also needs to be changed.

Q20. Can we update table in Views?

The SQL **UPDATE VIEW** command can be used to modify the data of a view. An updatable view is one which allows performing a UPDATE command on itself without affecting any other table.

Following are conditions to update a View:

1. The view is defined based on one and only one table.

2. The view must include the PRIMARY KEY of the table based upon which the view has been created.
3. The view should not have any field made out of aggregate functions.
4. The view must not have any DISTINCT clause in its definition.
5. The view must not have any GROUP BY or HAVING clause in its definition.
6. The view must not have any SUBQUERIES in its definitions.
7. If the view you want to update is based upon another view, the later should be updatable.
8. Any of the selected output fields (of the view) must not use constants, strings or value expressions.

Issue: Though we would get error if we try to update view having multiple tables.

When we execute the view having multiple tables it gives us the error as **'View or function [view name] is not updatable because the modification affects multiple base tables. '**

Solution: We can overcome this issue by using [INSTEAD OF UPDATE] trigger.

Q21. What are different types of Views?

There are two types of views in SQL Server, they are

1. **Simple view or Updatable views:** The view that is created based on the columns of a ***single table***, then it is known as a simple view. We can ***perform all the DML operations*** on a simple view so that a simple view can also be called an updatable view or dynamic view.

Note: A view that is created based on a single table will also be considered as a complex view provided if the query contains any of the

following.
Distinct. Aggregate Function, Group By Clause, having Clause, calculated columns, and set operations.

2. **Complex view or non-updatable views:** When we create a view on more than 1 table then it is known as a complex view and on a complex view, we cannot perform DML operations so that a complex view is also called the non-updatable or static view.

Q22. Difference between VIEWS and FUNCTIONS?

VIEWS	FUNCTIONS
Views can be used to perform DML operations like insert, update and delete data from a table	You cannot use a function with Data Manipulation queries. Only Select queries are allowed in functions.
Views cannot accepts parameters	User-Defined Function can accept parameters
We cannot use output of Views in the SELECT clause	output of the User Defined Function can be directly used in the SELECT clause

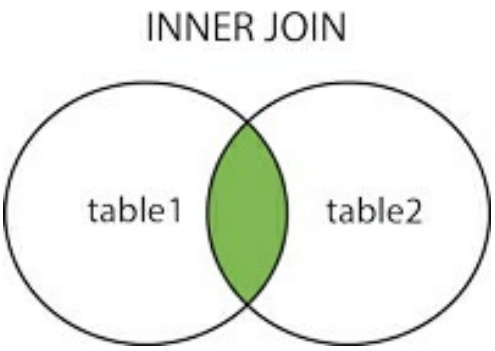
Q23.Types of Joins?

There are different types of joins used in SQL:

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

- **Inner Join**

The inner join is used to select all matching rows or columns in both tables

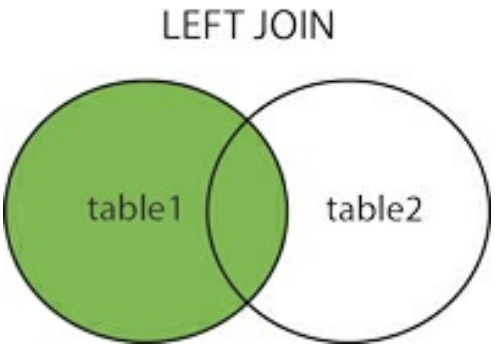


Syntax:

SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2*
ON *table1.column_name = table2.column_name*;

- **LEFT JOIN**

The Left Join is used to retrieve all records from the left table (table1) and the matched rows or columns from the right table (table2). If both tables do not contain any matched rows or columns, it returns the NULL.



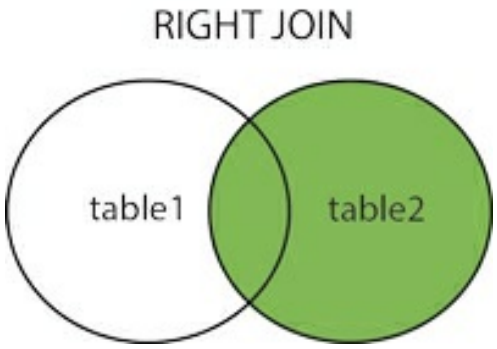
Syntax:

SELECT *column_name(s)*
FROM *table1*
LEFT JOIN *table2*
ON *table1.column_name = table2.column_name*;

- **RIGHT JOIN or RIGHT Outer JOIN:**

The Right Join is used to retrieve all records from the right table (table2) and

the matched rows or columns from the left table (table1). If both tables do not contain any matched rows or columns, it returns the NULL.

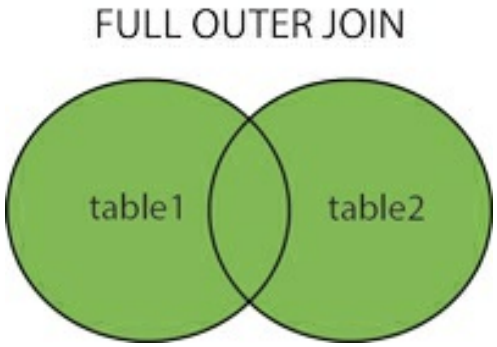


Syntax:

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

- **FULL JOIN or FULL Outer JOIN:**

It is a combination result set of both **LEFT JOIN** and **RIGHT JOIN**. The joined tables return all records from both the tables and if no matches are found in the table, it places NULL. It is also called a Full Outer Join.



Syntax:

```
SELECT column_name(s)
FROM table1
```

FULL OUTER JOIN *table2*
ON *table1.column_name = table2.column_name*
WHERE *condition*;

- **CROSS JOIN**

It is also known as **CARTESIAN JOIN**, which returns the Cartesian product of two or more joined tables. The Cross Join produces a table that merges each row from the first table with each second table row. It is not required to include any condition in CROSS JOIN.

Syntax:

SELECT * FROM *[TABLE1] CROSS JOIN [TABLE2]*

OR

SELECT * FROM *[TABLE1] , [TABLE2]*

- **Self Join**

A self join is a regular join, but the table is joined with itself.

Syntax:

SELECT *column_name(s)*
FROM *table1 T1, table1 T2*
WHERE *condition*;

Q24. What are Indexes?

Indexes are **special lookup tables** that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table.

An index helps to speed up **SELECT** queries and **WHERE** clauses, but it slows down data input, with the **UPDATE** and the **INSERT** statements.

Indexes can also be unique, like the **UNIQUE** constraint, in that the index prevents duplicate entries in the column or combination of columns on which there is an index.

Syntax:

CREATE INDEX index_name ON table_name;

Q25. What are Different ways to create Indexes?

We can have following type of indexes:

Single-Column Indexes

A single-column index is created based on only one table column. The basic syntax is as follows.

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

Unique Indexes

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows.

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

Composite Indexes

A composite index is an index on two or more columns of a table. Its basic syntax is as follows.

```
CREATE INDEX index_name
ON table_name (column1, column2);
```

Note: Should there be only one column used, a single-column index should be the choice. Should there be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

Implicit Indexes

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints.

Q26. When Indexes should be avoided?

- Indexes should not be used on small tables.
- Tables that have frequent, large batch updates or insert operations.
- Indexes should not be used on columns that contain a high number of NULL values.
- Columns that are frequently manipulated should not be indexed.

Q27. What are benefits of Indexes?

- Indexes are used to speed-up query process in SQL Server, resulting in high performance
- Without indexes, a DBMS has to go through all the records in the table in order to retrieve the desired results. This process is called ***table-scanning*** and is extremely slow.
- If you create indexes, the database goes to that index first and then retrieves the corresponding table records directly.

Q28. What are Types of Indexes?

There are two types of Indexes in SQL Server:

1. Clustered Index
2. Non-Clustered Index

Clustered Index

- A clustered index defines the order in which data is *physically stored in a table*.
- Table data can be sorted in only way, therefore, there can be *only one clustered index per table.*
- In SQL Server, the *primary key* constraint automatically creates a clustered index on that particular column.
- A B-Tree (computed) clustered index is the index that will arrange the rows physically in the memory in sorted order.
- An advantage of a clustered index is that searching for a range of values will be fast. A clustered index is internally maintained using a B-Tree data structure leaf node

e.g.

```
CREATE CLUSTERED INDEX index_custom
ON TableName(Column1 ASC, Column2 DESC)
```

Non-Clustered Indexes

- A non-clustered index doesn't sort the physical data inside the table.
- *A non-clustered index is stored at one place and table data is stored in another place*. This is similar to a textbook where the book content is located in one place and the index is located in another.
- This allows for *more than one non-clustered index per table*. You can create a maximum of *999* non-clustered indexes on a table
- The index contains column values on which the index is created and the address of the record that the column value belongs to.
- When a query is issued against a column on which the index is created, the database will first go to the index and look for the address of the corresponding row in the table. It will then go to that row address and fetch other column values. It is due to this additional step that *non-clustered indexes are slower than clustered indexes*.
- A non-clustered index is also maintained in a B-Tree data structure but leaf nodes of a B-Tree of non-clustered index contains the pointers to the pages that contain the table data and not the table data directly.

Syntax:

```
CREATE NONCLUSTERED INDEX index_custom
ON TableName(Column1 ASC)
```

Q29. Difference between clustered and nonclustered Indexes?

Clustered	Nonclustered
There can be only one clustered index per table.	You can create multiple non-clustered indexes on a single table.

Clustered indexes only sort tables. Therefore, they do not consume extra storage.	Non-clustered indexes are stored in a separate place from the actual table claiming more storage space.
Clustered indexes are faster than non-clustered indexes since they don’t involve any extra lookup step.	NonClustered indexes are slower as they have extra lookup step.

Q30. Difference between primary key and clustered index?

Primary key is unique identifier for record. It's responsible for unique value of this field. And **clustered index** is data structure that improves speed of data retrieval operations through an access of ordered records.

Q31. Can we create clustered index on unique key?

Yes, we can create Clustered index on unique key

E.g.

```
CREATE TABLE Employee (
      ID int  UNIQUE,
      LastName varchar(25) NOT NULL,
      FirstName varchar(25),
      Age int
);
CREATE clustered index in_empId on Employee (ID);
```

Sorting order and null values handling –

In this case all the data will be inserted in sorted order NULL values can also be inserted.

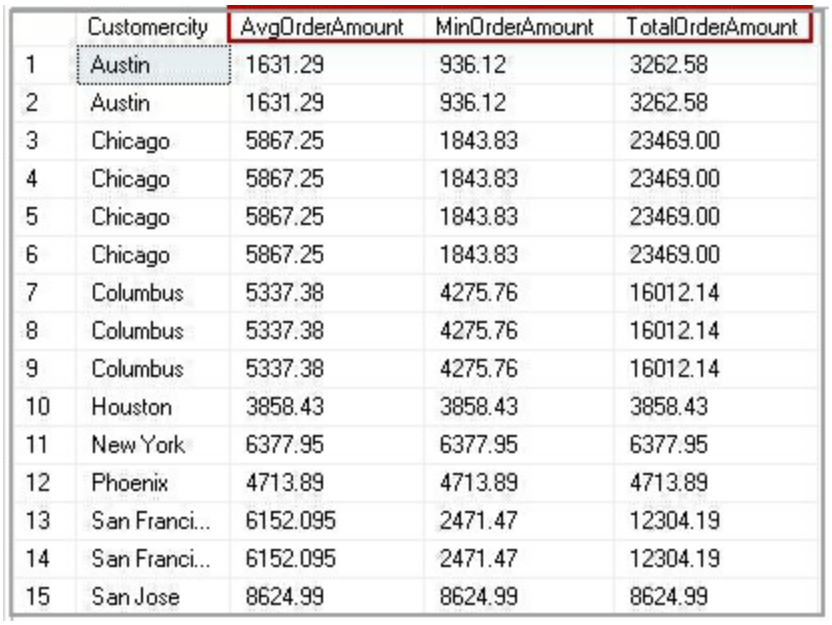
Q32. What is Partition By?

We can use the **SQL PARTITION BY** clause with the **OVER** clause to specify the column on which we need to perform aggregation.

e.g.

```
SELECT Customercity,
        AVG(Orderamount) OVER(PARTITION BY Customercity) AS AvgOrderAmount,
```

```
        MIN(OrderAmount) OVER(PARTITION BY Customercity) AS MinOrderAmount,
        SUM(Orderamount) OVER(PARTITION BY Customercity) TotalOrderAmount
FROM [dbo].[Orders];
```



	Customercity	AvgOrderAmount	MinOrderAmount	TotalOrderAmount
1	Austin	1631.29	936.12	3262.58
2	Austin	1631.29	936.12	3262.58
3	Chicago	5867.25	1843.83	23469.00
4	Chicago	5867.25	1843.83	23469.00
5	Chicago	5867.25	1843.83	23469.00
6	Chicago	5867.25	1843.83	23469.00
7	Columbus	5337.38	4275.76	16012.14
8	Columbus	5337.38	4275.76	16012.14
9	Columbus	5337.38	4275.76	16012.14
10	Houston	3858.43	3858.43	3858.43
11	New York	6377.95	6377.95	6377.95
12	Phoenix	4713.89	4713.89	4713.89
13	San Franci...	6152.095	2471.47	12304.19
14	San Franci...	6152.095	2471.47	12304.19
15	San Jose	8624.99	8624.99	8624.99

We get an error message if we try to add a column that is not a part of the GROUP BY clause.

We can add required columns in a select statement with the SQL PARTITION BY clause.

e.g.

```
SELECT Customercity,
        CustomerName,
        OrderAmount,
        AVG(Orderamount) OVER(PARTITION BY Customercity) AS AvgOrderAmount,
        MIN(OrderAmount) OVER(PARTITION BY Customercity) AS MinOrderAmount,
        SUM(Orderamount) OVER(PARTITION BY Customercity) TotalOrderAmount
FROM [dbo].[Orders];
```

	Customercity	CustomerName	OrderAmount	AvgOrderAmount	MinOrderAmount	TotalOrderAmount
1	Austin	Roland	936.12	1631.29	936.12	3262.58
2	Austin	Jorge	2326.46	1631.29	936.12	3262.58
3	Chicago	Marvin	7577.90	5867.25	1843.83	23469.00
4	Chicago	Alex	6847.66	5867.25	1843.83	23469.00
5	Chicago	Jerome	1843.83	5867.25	1843.83	23469.00
6	Chicago	Lawrence	7199.61	5867.25	1843.83	23469.00
7	Columbus	Salvador	4275.76	5337.38	4275.76	16012.14
8	Columbus	Aaliyah	5308.58	5337.38	4275.76	16012.14
9	Columbus	Gilbert	6427.80	5337.38	4275.76	16012.14
10	Houston	Ernest	3858.43	3858.43	3858.43	3858.43
11	New York	Ray	6377.95	6377.95	6377.95	6377.95
12	Phoenix	Edward	4713.89	4713.89	4713.89	4713.89
13	San Franci...	Aria	9832.72	6152.095	2471.47	12304.19
14	San Franci...	Stella	2471.47	6152.095	2471.47	12304.19
15	San Jose	Nicholas	8624.99	8624.99	8624.99	8624.99

PARTITION BY clause with ROW_NUMBER()

We can use the SQL PARTITION BY clause with ROW_NUMBER() function to have a row number of each row. We define the following parameters to use ROW_NUMBER with the SQL PARTITION BY clause.

- **PARTITION BY column** – In this example, we want to partition data on CustomerCity column
- **Order By:** In the ORDER BY column, we define a column or condition that defines row number. In this example, we want to sort data on the OrderAmount column

```
SELECT Customercity,
        CustomerName,
        ROW_NUMBER() OVER( PARTITION BY Customercity ORDER BY OrderAmount
DESC ) AS "Row Number",
        OrderAmount,
        COUNT(OrderID) OVER(PARTITION BY Customercity) AS CountOfOrders,
        AVG(Orderamount) OVER(PARTITION BY Customercity) AS AvgOrderAmount,
        MIN(OrderAmount) OVER(PARTITION BY Customercity) AS MinOrderAmount,
        SUM(Orderamount) OVER(PARTITION BY Customercity) TotalOrderAmount
FROM [dbo].[Orders];
```

	Customercity	CustomerName	OrderAmount	Row Number	CountOfOrders	AvgOrderAmount	MinOrderAmount	TotalOrderAmount
1	Austin	Jorge	2326.46	1	2	1631.29	936.12	3262.58
2	Austin	Roland	936.12	2	2	1631.29	936.12	3262.58
3	Chicago	Marvin	7577.90	1	4	5867.25	1843.83	23469.00
4	Chicago	Lawrence	7199.61	2	4	5867.25	1843.83	23469.00
5	Chicago	Alex	6847.66	3	4	5867.25	1843.83	23469.00
6	Chicago	Jerome	1843.83	4	4	5867.25	1843.83	23469.00
7	Columbus	Gilbert	6427.80	1	3	5337.38	4275.76	16012.14
8	Columbus	Aaliyah	5308.58	2	3	5337.38	4275.76	16012.14
9	Columbus	Salvador	4275.76	3	3	5337.38	4275.76	16012.14
10	Houston	Ernest	3858.43	1	1	3858.43	3858.43	3858.43
11	New York	Ray	6377.95	1	1	6377.95	6377.95	6377.95
12	Phoenix	Edward	4713.89	1	1	4713.89	4713.89	4713.89
13	San Francisco	Aria	9832.72	1	2	6152.095	2471.47	12304.19
14	San Francisco	Stella	2471.47	2	2	6152.095	2471.47	12304.19
15	San Jose	Nicholas	8624.99	1	1	8624.99	8624.99	8624.99

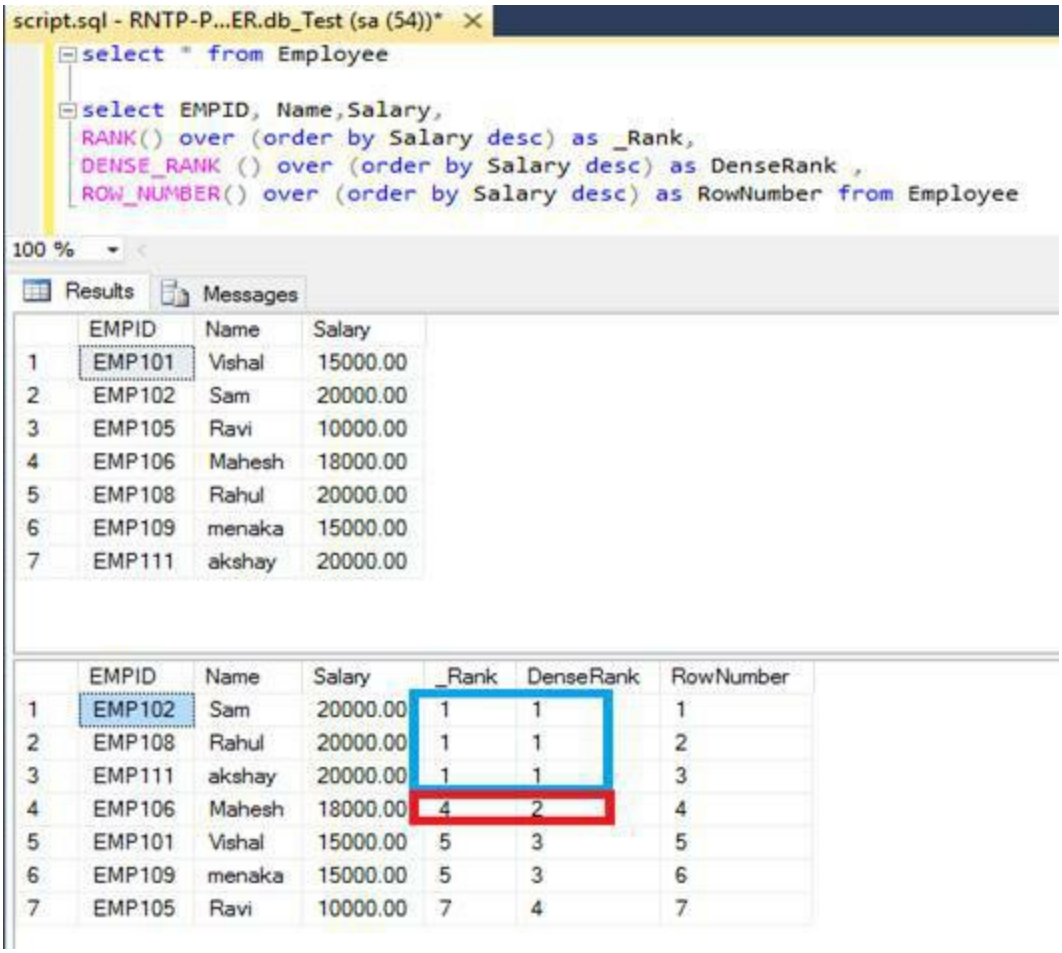
Q33. Difference between GroupBy and Partition By?

GroupBy	Partition By
We get a limited number of records using the Group By clause	We get all records in a table using the PARTITION BY clause.
It gives one row per group in result set.	It gives aggregated columns with each record in the specified table.

Q34. What are RANK, DENSE_RANK and ROW_NUMBER functions?

- **Increasing integer value**: The RANK, DENSE_RANK and ROW_NUMBER functions are used to get the *increasing integer value*, based on the ordering of rows by imposing ORDER BY clause in SELECT statement.
- **ORDER BY**: When we use RANK, DENSE_RANK or ROW_NUMBER functions, the ORDER BY clause is required and PARTITION BY clause is optional.
- **PARTITION BY**: When we use PARTITION BY clause, the selected data will get partitioned, and the integer value is reset to 1 when the partition changes.

e.g.



RANK (): - *next incremented rank*
In RANK function, the next row after the duplicate values (salary),marked in red color, will not give the integer value as next rank but instead of it, it skips those ranks and gives what is the
next incremented rank.

DENSE_RANK (): - *next rank in sequence/not skip any rank*
In DENSE_RANK function, *it will not skip any rank.* This means the next row after the duplicate value (salary) rows will have the *next rank in the sequence*.

Benefits of RANK () and DENSE_RANK ():
Using RANK or DENSE_RANK function, we can find nth highest salary.

Q35. Find 2nd Highest salary using DENSE_RANK ()?

```
with tempSal as
(
```

```
select *,DENSE_RANK() over (order by salary desc) as _rank from EmployeeTable
)
select top 1 salary from tempsal where _rank = 2;
```

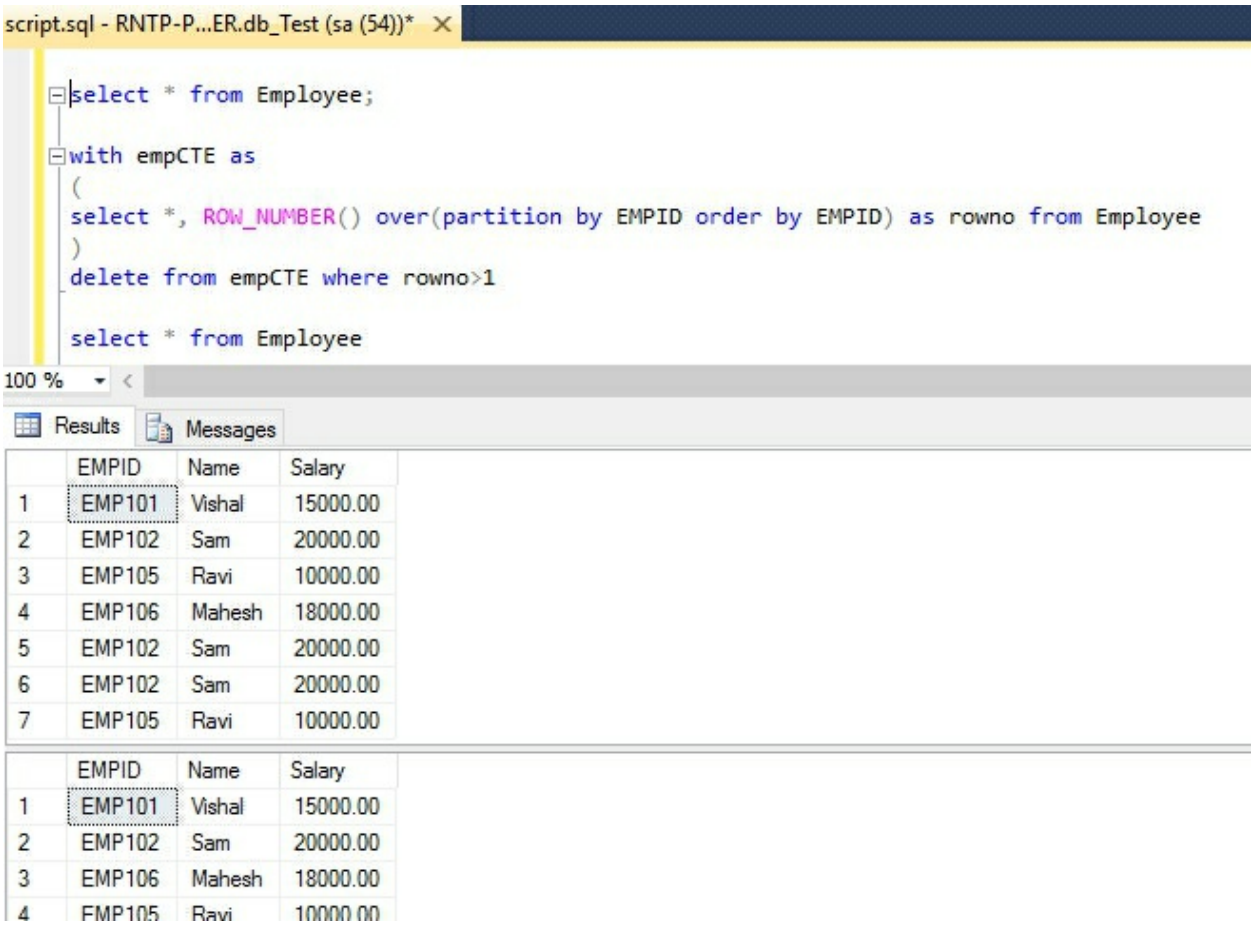
OR

```
SELECT TOP 1 Salary
FROM (
        SELECT DISTINCT TOP N Salary
        FROM Employee
        ORDER BY Salary DESC
        ) AS Emp
ORDER BY Salary
```

Q36. Query to delete duplicate data in table?

Using ROW_NUMBER function, we can delete duplicate data from table

```
with empCTE as
(
select *, ROW_NUMBER() over(partition by EMPID order by EMPID) as rowno from Employee
)
delete from empCTE where rowno>1
```



Q37. Difference between Truncate and Delete?

Delete	Truncate
The DELETE command is used to delete specified rows (one or more).	While this command is used to delete all the rows from a table.
It is a DML (Data Manipulation Language) command.	While it is a DDL (Data Definition Language) command.
There may be WHERE clause in DELETE command in order to filter the records.	While there may not be WHERE clause in TRUNCATE command.
DELETE command is slower than TRUNCATE command.	While TRUNCATE command is faster than DELETE command.
To use Delete you need DELETE permission on the table.	To use Truncate on a table we need at least ALTER permission on the table.

Syntax: DELETE FROM TableName WHERE condition;	Syntax: TRUNCATE TABLE TableName;
DELETE operations can be rolled back	TRUNCATE operations cannot be rolled back

Q38. How to do Transaction and Rollback?

Transaction

- Transactions in SQL Server are used to execute a set of SQL statements in a group. With transactions, either all the statements in a group execute or none of the statements execute.
- To start a transaction, the **BEGIN TRANSACTION** statement is used, followed by the set of queries that you want to execute inside the transaction. To mark the end of a transaction, the **COMMIT TRANSACTION** statement can be used.

Rollback

The rollback SQL statement is used to manually rollback transactions in MS SQL Server.

In the case where one of the queries in a group of queries executed by a transaction fails, all the previously executed queries are roll backed. Transactions in the SQL server are roll backed automatically. However, with the rollback SQL statement, you can manually rollback a transaction based on certain conditions.

E.g.

```
BEGIN TRANSACTION

   INSERT INTO Employee
   VALUES (1, 'Frank', 'USA', 1000)

   UPDATE Employee
   SET salary = '25 Hundred' WHERE id = 1

   DELETE from Employee
   WHERE id = 1
```

```
COMMIT TRANSACTION
```

Note: Since the queries are being executed inside a transaction, the failure of the second query will cause all the previously executed queries to rollback.

Manually rollback SQL transactions

Transactions automatically rollback themselves if one of the queries cannot be executed successfully. However, you may want to rollback a query based on certain conditions as well.

E.g.

```
DECLARE @EmpCount int

BEGIN TRANSACTION AddEmployee

   INSERT INTO Employee
   VALUES (1, 'Frank', 'USA', 1000)

   SELECT @EmpCount = COUNT(*) FROM Employee WHERE name = 'Frank'

   IF @EmpCount > 1
      BEGIN
         ROLLBACK TRANSACTION AddEmployee
         PRINT 'An employee with the same name already exists'
      END
   ELSE
      BEGIN
         COMMIT TRANSACTION AddEmployee
         PRINT 'New Employee added successfully'
      END
```

Q39. What are Triggers?

A trigger is a special kind of Stored Procedure that is automatically fired or executed when some event (insert, delete and update) occurs.

When to use a trigger

We use a trigger when we want some event to happen automatically on certain desirable scenarios.

Types of Triggers

We can create the following 3 types of triggers:

- Data Definition Language (DDL) triggers
- Data Manipulation Language (DML) triggers
- Logon triggers

DDL Triggers

In SQL Server we can create triggers on DDL statements (like CREATE, ALTER and DROP) and certain system-defined Stored Procedures that does DDL-like operations.

DML Triggers

In SQL Server we can create triggers on DML statements (like INSERT, UPDATE and DELETE) and Stored Procedures that do DML-like operations. DML Triggers are of two types.

- **After trigger (using FOR/AFTER CLAUSE)**

The After trigger (using the FOR/AFTER CLAUSE) fires ***after*** SQL Server finishes the ***execution*** of the action successfully that fired it.

- **Instead of Trigger (using INSTEAD OF CLAUSE)**

The Instead of Trigger (using the INSTEAD OF CLAUSE) fires ***before*** SQL Server starts the ***execution*** of the action that fired it.

Logon Triggers

- Logon triggers are a special type of triggers that fire when a LOGON event of SQL Server is raised. This event is raised when a user session is being established with SQL Server that is made after the authentication phase finishes, but before the user session is actually established.
- We can use these triggers to audit and control server sessions, such as to track login activity or limit the number of sessions for a specific login.

Syntax:

```
CREATE TRIGGER triggerName ON table
AFTER INSERT |After Delete |After Upadte
AS BEGIN
   INSERT INTO dbo.Employee.....
   END
```

Q40. What are constraints available in Sql server?

SQL Server supports six types of constraints for maintaining data integrity. They are as follows

1. Default Constraint
2. UNIQUE KEY constraint
3. NOT NULL constraint
4. CHECK KEY constraint
5. PRIMARY KEY constraint
6. FOREIGN KEY constraint.

Q41. What is UNIQUE key Constraint?

- To avoid duplicate values on columns we apply UNIQUE Constraint
- That means the UNIQUE constraint is used to ***avoid duplicate values but it accepts a single NULL value in that column.***
- A table can contain any number of UNIQUE constraints.
- We can apply the UNIQUE constraint on any data type column such as integer, character, money, etc.

```
CREATE TABLE Customer
(
      Id      INT UNIQUE,
      NAME    VARCHAR(30) UNIQUE,
      Emailid VARCHAR(100) UNIQUE
)
```

Q42. What is PRIMARY key Constraint?

- The **Primary Key** is the combination of **Unique** and **Not Null** Constraint
- It will not allow either **NULL** or **Duplicate** values
- A table should contain only 1 Primary Key which can be either on a single or multiple columns i.e. the composite primary key.

```
CREATE TABLE Customer
(
    Id      INT PRIMARY KEY,
    NAME    VARCHAR(30),
    Emailid VARCHAR(100)
)
```

Q43. Can we create multiple primary key on a table?

A table can have only one PRIMARY KEY either on one column or multiple columns. When multiple columns are defined as PRIMARY KEY, then, it is called ***COMPOSITE KEY***.

- PRIMARY KEY is a constraint in SQL which is used to identify each record uniquely in a table.
- By default, PRIMARY KEY is UNIQUE.
- PRIMARY KEY can't have null values.
- If we try to insert/update duplicate values for the PRIMARY KEY column, then, the query will be aborted.

Syntax (**Primary key**):

```
CREATE TABLE table_name
(
column_name1 data_type(size) PRIMARY KEY,
column_name2 data_type(size) NOT NULL,
column_name3 data_type(size),
etc…
)
```

Syntax (**Composite key**):

```
CREATE TABLE table_name
(
column_name1 data_type(size) NOT NULL,
```

```
column_name2 data_type(size) NOT NULL,
column_name3 data_type(size),
CONSTRAINT Constraint_name PRIMARY KEY (column_name1, column_name2)
etc…
);
```

Q44. What is FOREIGN key Constraint?

- In order to create a link between two tables, we must specify a Foreign Key in one table that references a column in another table.
- Foreign Key constraint is used for binding two tables with each other and then verify the existence of one table data in other tables.
- A foreign key in one TABLE points to a primary key or unique key in another table. The foreign key constraints are used to enforce referential integrity.

Q45. How to create a FOREIGN key Constraint?

- Two tables must have a common column for linking the tables.
- Common Column name need not be same but data type should be same
- The common column that is present under the parent table or master table is known as the reference key column and moreover, the reference key column should not contain any duplicate values. So we need to impose either UNIQUE or PRIMARY key constraint on that column.
- The common column which is present in the child or detailed table is known as the Foreign key column and we need to impose a Foreign key constraint on the column which refers to the reference key column of the master table.

Syntax (At column level):

```
CREATE TABLE Employee
(
    Empid  INT,
    Ename  VARCHAR(40),
    Job    VARCHAR(30),
```

```
    Salary MONEY,
    Deptno INT CONSTRAINT deptn0_fk REFERENCES Dept(Dno)
)
```

Syntax (At table level):

```
CREATE TABLE Employee
(
    Empid  INT,
    Ename  VARCHAR(40),
    Job    VARCHAR(30),
    Salary MONEY,
    Deptno INT,
    CONSTRAINT deptno_fk FOREIGN KEY (Deptno) REFERENCES Dept(Dno)
)
```

3 Rules should be kept in mind while creating foreign key constraint:

1. **Rule1:** Cannot insert a value into the foreign key column provided that value is not existing in the reference key column of the parent (master) table.
2. **Rule2:** Cannot update the reference key value of a parent table provided that the value has a corresponding child record in the child table without addressing what to do with the child records.
3. **Rule3:** Cannot delete a record from the parent table provided that records reference key value has child record in the child table without addressing what to do with the child record.

Q46. What is difference between PRIMARY and FOREIGN key constraint?

Primary key	Foreign key
uniquely identifies a record in the table	The Foreign Key is a field in a table that is a unique key in another table
Primary Key constraint neither accepts null values nor duplicate values	can accept both null values and duplicate values

By default Primary Key Constraint create a unique clustered index that will physically organize the data in the table.	By default, the foreign key does not create any index. If you need then you can create an index on the foreign key column manually.
We can create only one Primary Key on a table. Though you can create the primary key either on a single column or multiple columns.	We can create more than one Foreign key on a table

Q47. Difference between varchar and nvarchar?

varchar	Nvarchar
It is a variable-length data type (dynamic data type) and will store the character in a non-Unicode manner that means it will take 1 byte for 1 character.	It is a variable-length data type and will store the data type in the Unicode manner that means it will occupy 2bytes of memory per single character.
The maximum length of the varchar data type is from 1 to 8000 bytes	The maximum length of nvarchar data type is from up to 4000 bytes.

Q48. What is Group By Clause?

- The Group by Clause in SQL Server is used to divide the similar type of records or data as a group and then return.
- If we use group by clause in the query then we should use grouping/aggregate function such as count(), sum(), max(), min(), avg() functions.
- First Group By clause is used to divide similar types of data as a group and then an aggregate function is applied to each group to get the required results.
- When we use multiple columns in a group by clause first data in the table is divided based on the first column of the group by

clause and then each group is subdivided based on the second column of the group by clause and then the group function is applied on each inner group to get the result.

Syntax:

```
SELECT expression1, expression2, expression_n,
       aggregate_function (expression)
FROM tables
GROUP BY expression1, expression2, expression_n;
```

Note*: expression1, expression2, expression_n: The expressions that are not encapsulated within an aggregate function must be included in the GROUP BY clause.

Q49. What is Having Clause?

The **Having Clause is** used for filtering the data just like the where clause.

Syntax:

```
SELECT expression1, expression2, expression_n,
       aggregate_function (expression)
FROM tables
[WHERE conditions]
GROUP BY expression1, expression2, expression_n
HAVING having_condition;
```

Q50. Write some Group By Queries?

Query to get the number of employees working in each Gender per each department.

```
SELECT Department, Gender, EmployeeCount = COUNT(*)
FROM Employee
GROUP BY Department, Gender
ORDER BY Department
```

Query to find total salary in each department

```
SELECT Department, TotalSalary = SUM(Salary)
FROM Employee
GROUP BY Department
```

Find the highest salary in each department in the organization.

```
SELECT Department, MaxSalary = MAX(SALARY)
```

```
FROM Employee
GROUP BY Department
```

Get the number of employees working in each Gender per each department.

```
SELECT Department, Gender, EmployeeCount = COUNT(*)
FROM Employee
GROUP BY Department, Gender
ORDER BY Department
```

Query for retrieving total salaries by the city.

```
SELECT CITY, SUM(Salary) as TotalSalary
FROM Employee
GROUP BY CITY
```

Query for retrieving total salaries and the total number of employees by City, and by gender.

```
SELECT  City, Gender, SUM(Salary) as TotalSalary,
      COUNT(ID) as TotalEmployees
FROM Employee
GROUP BY  CITY, Gender
```

Q51. What is Union operator?

- The UNION operator is used to combine the result-set of two or more SELECT statements.
- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

Syntax:

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

Q52. What is UnionAll operator?

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

Syntax:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

Q53. What is COALESCE?

COALESCE is used to return first non-null expression within the arguments. This function is used to return a non-null from more than one column in the arguments.

e.g.

```
Select COALESCE(empno, empname, salary) from employee;
```

Q54. Use of Substring and Charindex?

The SUBSTR function is used to return specific portion of string in a given string. But, CHARINDEX function gives character position in a given specified string.

SUBSTRING(expression, starting position, length)

```
select SUBSTRING('Steve',1,3)
// output : Ste
```

CHARINDEX(expression varchar(1), expression varchar(1), start_location int)

```
select CHARINDEX('v', 'Steve',3)
//output: 4
```

Q55. What is STUFF() Function?

The STUFF() function deletes a part of a string and then inserts another part into the string, starting at a specified position.

Syntax:

```
STUFF(string, start, length, new_string)
```

string	Required. The string to be modified
start	Required. The position in string to start to delete some characters
l engtth	Required. The number of characters to delete from string
new_string	Required. The new string to insert into string at the start position

```
e.g.
SELECT STUFF('This is.', 8, 1, ' Test');
//Output : This is Test
```

Q56. What is an IDENTITY column?

IDENTITY column is used in table columns to make that column as Auto incremental number.

Q57. Difference between SEQUENCE and IDENTITY?

- **Sequence object** is similar to the Identity property, in the sense that it generates sequence of numeric values in an ascending order just like the identity property.
- Identity property is a table column property meaning it is tied to the table, whereas the sequence is a user-defined database object and is not tied to any specific table meaning its value can be shared by multiple tables.

e.g.

```
// IDENTITY
CREATE TABLE Employees
(
    Id INT PRIMARY KEY IDENTITY(1,1),
```

```
    Name NVARCHAR(50),
    Gender NVARCHAR(10)
)
// SEQUENCE
CREATE SEQUENCE [dbo].[SequenceObject]
AS INT
START WITH 1
INCREMENT BY 1

// Insert data
INSERT INTO Customers VALUES
  (NEXT VALUE for [dbo].[SequenceObject], 'Mike', 'Male')

INSERT INTO Users VALUES
  (NEXT VALUE for [dbo].[SequenceObject], 'Steve', 'Male')
```

About the Author

Vishal garg

Vishal Garg is a technical writer with a passion for writing technical books. He has passion for learning new technologies and share the knowledge with everyone. He is well versed in technologies like SQL Server, Azure, Devops, Angular, .Net core, Web API, C# etc. He also shares his knowledge with the community through book writings, blog writings, presentations etc.
He has written books on different technologies as well and got a positive reviews on that. He followed a very unique way to cover all major concepts.

With the help of various surveys and real time experience a question bank of a particular topic are compiled and logged in a book.

He is hoping that all readers will be benefited from this book and looking forward to put in more effort to produce quality books in future.

Note: If you like the book, please take some time to put in positive reviews on Amazon website. This feedback will encourage him to produce more quality books in future.

More Books by this Author

- .Net Core Simplified: Interview QA
- Angular Simplified: Learning made easy
- C# Interview Question and Answers: Simplified
- Azure Devops Interview Questions and Answers
- Angular 2021: Interview Questions and Answers
- C# Interview Questions and Answers : Edition:2021
- Web Api and Security: Interview Questions and Answers : Edition:2021