Que

Friday, November 6, 2015 11:12 AM

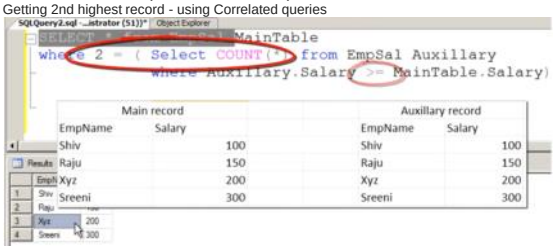
Standard sql questions
1) Difference between Inner join and outer join, what are the other kind of joins and where do we use them?
2) what is Union and union all, difference between them?
3) where do we use having and where clause and when to use what?
4) what is a sub query?
5) what is a co-related sub query? difference between 4 and 5 and when to use what?
6) what are derived tables?

Teradata specific questions
1) What is Spool space issue and how can we resolve it?
2) Difference between Set and multi set table?
3) What is Fload, mload, when to use what?
4) Did u use bteq scripts and wht is its functionality?
5) Why cant you load data into a multiset table with Fload ?
6) Any performance tuning experience?.

From <https://mail.google.com/mail/u/0/#inbox/150dd8d7ed0e9df4>

Can we have multiple Primary Indexes in a table? What is Skew ratio?	How is initial introduction? How frequent are meetings and new requirements? How long time given per requirement? How can I connect with you for any technical doubts? Can we stay longer in the office & work/learn? Will the DBA monitor?

CORRELATED QUERIES (QuestPond on Utube)
Getting 2nd highest record - using Correlated queries



Method 2, using **TOP and Subquery**

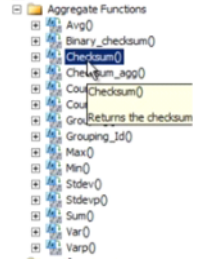
```
Select TOP 1 Salary From
 (Select DISTINCT TOP 2 Salary
 From Employees
 Order By Salary Desc)
 Result
 Order By Salary
```

Method 3, using **CTE** (WITH RESULT AS ())
DENSE_RANK() - gives a rank to every row, based on the value
ROW_NUMBER() will give simple ranking based on the line number, so in case of duplicate values the rank would be wrong

```
WITH RESULT AS
 (
     Select Salary, DENSE_RANK() over (Order by Salary DESC) as DENSERANK
     from Employees
 )
 Select Salary
 From RESULT
 Where RESULT.DENSERANK = 2
```

Union vs Union ALL
Union doesn't display duplicate values
Union ALL shows duplicates as well

Aggregate Functions - avg, binary_checksum, checksum_agg, count, count_big, grouping



An **aggregate value** can't be used in WHERE clause, use HAVING instead:

```
select CountryName, SUM(population)
        from tblCountry
        Group by CountryName
        having  SUM(population) > 1000
```

Normal Management -
1st Normal Form - break data in logical granular level for better data management and search query

1. First Normal Form (1NF) - A relation is said to be in first normal form if and only if all underlying domains contain atomic values only. After 1NF, we can still have redundant data.
2. Second Normal Form (2NF) - A relation is said to be in 2NF if and only if it is in 1NF and every non key attribute is fully dependent on the primary key. After 2NF, we can still have redundant data.
3. Third Normal Form (3NF) - A relation is said to be in 3NF, if and only if it is in 2NF and every non key attribute is non-transitively dependent on the primary key.

Scenarios Discussed

Date	Scenario
14 Nov	Rank()Over to vs GROUP BY & MIN/MAX CSUM - Cumulative by a Column Count()Over -

1. ~~Connecting to Teradata~~
2. A
3. ~~Logical/Conditional expressions~~
4. ~~Datatypes / CASTS~~
5. ~~Joins (Inner/Outer) - Self Joins~~
6. ~~Subqueries~~
7. ~~Derived tables~~
8. Creating Tables (structure by giving column names, data types)
9. ~~PI | CS~~
10. ~~Data manipulation - INSERT, UPDATE, DELETE~~
11. Aggregate functions
12. Case Statement
13. 0 if NULL
14. ~~Strings (Substring, Position)~~
15. Views
16. Macros
17. ~~OLAP~~
18. ~~Calendar Table~~
19. ~~Union / Minus operations~~

Select * from dbc.tables WHERE databasename='FINANCIAL'

React - FB for JS
Kafka - Twitter
Storm -
Hadoop - Google

Temp Tables vs CTE / Sub-Query
A temp table is a table (can have different scopes: local, global) that stores data in the TEMPDB (rather than the DB it was created in). A CTE is a derived table/inline view, that exists only for the scope of the query it is declared in -- using resources from the current DB

Explain DML, DDL, DCL and TCL statements with examples?
DML: DML stands for Data Manipulation Language. DML is used to retrieve, store, modify, delete, insert and update data in database.
Examples of DML statements: SELECT, UPDATE, INSERT, DELETE statements.

DDL: DDL stands for Data Definition Language. DDL is used to create and modify the structure of database objects.

Examples: CREATE, ALTER, DROP statements.

DCL: DCL stands for Data Control Language. DCL is used to create roles, grant and revoke permissions, establish referential integrity etc.
Examples: GRANT, REVOKE statements

TCL: TCL stands for Transactional Control Language. TCL is used to manage transactions within a database.
Examples: COMMIT, ROLLBACK statements

What is the difference between Drop, Delete and Truncate statements in SQL Server?
Drop, Delete and Truncate - All operations can be rolled back.

All the statements (**Delete, Truncate and Drop**) are logged operations, but the amount of information that is logged varies. **Delete** statement logs an entry in the transaction log for each deleted row, where as **Truncate Table** logs only the Page deallocations.**Hence, truncate is a little faster than Delete**.

You can have a where clause in Delete statement where as Truncate statement cannot have a where clause. Truncate will delete all the rows in a Table, but the structure of the table remains. Drop would delete all the rows including the structure of the Table.

Please refer to the screen shot below for the differences summary snapshot between Drop, Delete and Truncate statements in SQL Server.

Command	Table Structure Deleted	Table Rows Deleted
Drop	Yes	Yes
Truncate	No	Yes - All Recodrs
Delete	No	Yes - Selectively

What is Cascading referential integrity constraint?
Cascading referential integrity constraints allow you to define the actions Microsoft SQL Server should take when a user attempts to delete or update a key to which an existing foreign keys point.

You can instruct SQL Server to do the following:

1. No Action: This is the default behaviour. No Action specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, an error is raised and the DELETE or UPDATE is rolled back.

2. Cascade: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are also deleted or updated.

3. Set NULL: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those

2. Second Normal Form (2NF) - A relation is said to be in 2NF if and only if it is in 1NF and every non key attribute is fully dependent on the primary key. After 2NF, we can still have redundant data.
3. Third Normal Form (3NF) - A relation is said to be in 3NF, if and only if it is in 2NF and every non key attribute is non-transitively dependent on the primary key.

View
If I add a record in a view, will that update the table?

Org Hierarchy using SQL

Using Recursive CTE and joining Employee table with CTE, inside CTE, on employeeID and managerID

```
With EmployeeCTE AS
 (
     Select EmployeeID, EmployeeName, ManagerID
     From Employees
     Where EmployeeID = @ID

     UNION ALL

     Select Employees.EmployeeID, Employees.EmployeeName, Employees.ManagerID
     From Employees
     Join EmployeeCTE
     On Employees.EmployeeID = EmployeeCTE.ManagerID
)
```

Employee table:

EmployeeID	EmployeeName	ManagerID
1	John	5
2	Mark	8
3	Steve	8
4	Tom	3
5	Lara	8
6	Simon	2
7	David	4
8	Ben	NULL
9	Stacy	2
10	Sam	5

Delete Duplicate Rows

```
WITH EmployeesCTE AS
(
   SELECT *, ROW_NUMBER()OVER(PARTITION BY ID ORDER BY ID) AS RowNumber
   FROM Employees
)
DELETE FROM EmployeesCTE WHERE RowNumber > 1
```

ROW_NUMBER + PARTITION BY will divide by ID and then we can delete IDs > 1 (Deleting from CTE, deletes from base table, as long as there are no other tables joined)

ID	FirstName	LastName	Gender	Salary	RowNumber
1	Mark	Hastings	Male	60000	1
1	Mark	Hastings	Male	60000	2
1	Mark	Hastings	Male	60000	3
2	Mary	Lambeth	Female	30000	1
2	Mary	Lambeth	Female	30000	2
3	Ben	Hoskins	Male	70000	1
3	Ben	Hoskins	Male	70000	2
3	Ben	Hoskins	Male	70000	3

Method 2 (stackoverflow)

```
DELETE FROM MyTable
LEFT OUTER JOIN (
  SELECT MIN(RowId) as RowId, Col1, Col2, Col3
  FROM MyTable
  GROUP BY Col1, Col2, Col3
) as KeepRows ON
  MyTable.RowId = KeepRows.RowId
WHERE
  KeepRows.RowId IS NULL
```

Using DateDiff

```
Select *, DATEDIFF(DAY, HireDate, GETDATE()) as Diff
 from Employees
 where DATEDIFF(DAY, HireDate, GETDATE()) between 1 and 30
order by HireDate desc
```

Transform Rows into Columns

PIVOT / UNPIVOT

Country	City
USA	New York
USA	Houston
USA	Dallas
India	Hyderabad
India	Bangalore
India	New Delhi
UK	London
UK	Birmingham
UK	Manchester

	Country	City1	City2	City3	City4
1	India	Chennai	Hyderabad	Bangalore	New Delhi
2	UK	London	Birmingham	Manchester	NULL
3	USA	New York	Houston	Dallas	NULL

```
Select Country, City1, City2, City3
From
(
  Select Country, City,
    'City'+
      cast(row_number() over(partition by Country order by Country)
          as varchar(10)) ColumnSequence
  from Countries
) Temp
pivot
(
  max(City)
  for ColumnSequence in (City1, City2, City3)
) Piv
```

ISNUMERIC

```
SELECT City, PostalCode
FROM Person.Address
WHERE ISNUMERIC(PostalCode)<> 1;
```

JOINS

2. Cascade: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are also deleted or updated.

3. Set NULL: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are set to NULL.

4. Set Default: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are set to default values.

DIfference between primary key and unique key in SQL Server?

1. A table can have only one primary key. On the other hand a table can have more than one unique key.
2. Primary key column does not accept any null values, where as a unique key column accept one null value.

What are the 2 types of Temporary Tables in SQL Server?

1. Local Temporary Tables
2. Global Temporary Tables

What is the difference between Local and Global Temporary Tables?
Local Temporary Tables:

1. Prefixed with a single pound sign (#).
2. Local temporary tables are visible to that session of SQL Server which has created it.
3. Local temporary tables are automatically dropped, when the session that created the temporary tables is closed.

Global Temporary Tables:

1. Prefixed with two pound signs (##).
2. Global temporary tables are visible to all the SQL server sessions.
3. Global temporary tables are also automatically dropped, when the session that created the temporary tables is closed.

Can you create foreign key constraints on temporary tables?

No

Do you have to manually delete temporary tables?

No, temporary tables are automatically dropped, when the session that created the temporary tables is closed. But if you maintain a persistent connection or if connection pooling is enabled, then it is better to explicitly drop the temporary tables you have created.
However, It is generally considered a good coding practice to explicitly drop every temporary table you create.

In which database, the temporary tables get created?

TEMPDB database.

How can I check for the existence of a temporary table?

```
IF object_id('tempdb..##TEMPTABLE') IS NOT NULL
BEGIN
    DROP TABLE ##TEMPTABLE
    PRINT 'Temporary Table ##TEMPTABLE is deleted'
END
ELSE
BEGIN
    PRINT 'Temporary Table ##TEMPTABLE is not found,
           it may have been deleted already'
END
```

What is the use of an Index in SQL Server?
Relational databases like SQL Server use indexes to find data quickly when a query is processed. Creating the proper index can drastically increase the performance of an application.

What is a table scan? or **What is the impact of table scan on performance?**
When a SQL Server has no index to use for searching, the result is similar to the reader who looks at every page in a book to find a word. The SQL engine needs to visit every row in a table. In database terminology we call this behavior a table scan, or just scan. A full table scan of a very large table can adversely affect the performance. Creating proper indexes will allow the database to quickly narrow in on the rows to satisfy the query, and avoid scanning every row in the table.

What is the system stored procedure that can be used to list all the indexes that are created for a specific table?
sp_helpindex is the system stored procedure that can be used to list all the indexes that are created for a specific table.

For example, to list all the indexes on table **tblCustomers**, you can use the following command.
EXEC sp_helpindex tblCustomers

What is the purpose of query optimizer in SQL Server?
An important feature of SQL Server is a component known as the query optimizer. The query optimizer's job is to find the fastest and least resource intensive means of executing incoming queries. An important part of this job is selecting the best index or indexes to perform the task.

What is the first thing you will check for, if the query below is performing very slow?
SELECT * FROM tblProducts ORDER BY UnitPrice ASC

Check if there is an Index created on the UntiPrice column used in the ORDER BY clause. An index on the UnitPrice column can help the above query to find data very quickly.When we ask for a sorted data, the database will try to find an index and avoid sorting the results during execution of the query. We control sorting of a data by specifying a field, or fields, in an ORDER BY clause, with the sort order as ASC (ascending) or DESC (descending).

With no index, the database will scan the **tblProducts** table and sort the rows to process the query. However, if there is an index, it can provide the database with a presorted list of prices. The database can simply scan the index from the first entry to the last entry and retrieve the rows in sorted order.

The same index works equally well with the following query, simply by scanning the index in reverse.

SELECT * FROM tblProducts ORDER BY UnitPrice DESC

What is the significance of an Index on the column used in the GROUP BY clause?
Creating an Index on the column, that is used in the **GROUP BY** clause, can greatly improve the perofrmance. We use a **GROUP BY** clause to group records and aggregate values, for example, counting the number of products with the same UnitPrice. To

```
SELECT City, PostalCode
FROM Person.Address
WHERE ISNUMERIC(PostalCode)<> 1;
```

JOINS

INNER - intersection
LEFT / RIGHT / FULL / CROSS

JOIN 2 tables without primary/foreign key relation

It is possible as long as the data type of join columns match or are compatible, e.g. decimal and integer...

```
alter table Departments
add primary key (ID)

Alter table Employees
add foreign key(DepartmentId)
references Departments(ID)
```

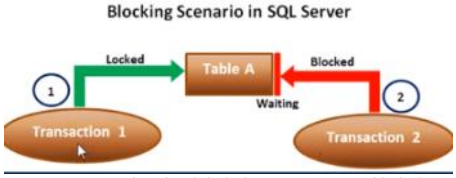
SELECT M* without using LIKE

```
SELECT * FROM Students WHERE CHARINDEX('M',Name) = 1
SELECT * FROM Students WHERE LEFT(Name, 1) = 'M'
SELECT * FROM Students WHERE SUBSTRING(Name, 1, 1) = 'M'
```

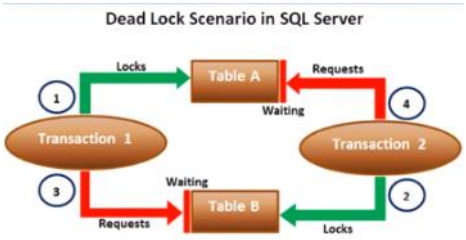
Composite Primary Key -

```
Alter table StudentCourses
Add Constraint PK_StudentCourses
Primary Key Clustered (CourseId, StudentId)
```

Blocking and Deadlocking -



Transaction 1 has already locked, so Transaction 2 is blocked
Till Transactio 1 is committed, with "Commit Transaction", Transaction 2 will remain blocked



In this case, SQL Server will cancel one of the transaction

```
SELECT * FROM tblProducts ORDER BY UnitPrice DESC
```

What is the significance of an Index on the column used in the GROUP BY clause?

Creating an Index on the column, that is used in the **GROUP BY** clause, can greatly improve the perofrmance. We use a **GROUP BY** clause to group records and aggregate values, for example, counting the number of products with the same UnitPrice. To process a query with a GROUP BY clause, the database will often sort the results on the columns included in the GROUP BY.

The following query counts the number of products at each price by grouping together records with the same UnitPrice value.
SELECT UnitPrice, Count(*) FROM tblProducts GROUP BY UnitPrice

The database can use the index (Index on **UNITPRICE** column) to retrieve the prices in order. Since matching prices appear in consecutive index entries, the database is able to count the number of products at each price quickly. Indexing a field used in a **GROUP BY** clause can often speed up a query.

What is the role of an Index in maintaining a Unique column in table?

Columns requiring unique values (such as primary key columns) must have a unique index applied. There are several methods available to create a unique index.
1. Marking a column as a primary key will automatically create a unique index on the column.
2. We can also create a unique index by checking the Create UNIQUE checkbox when creating the index graphically.
3. We can also create a unique index using SQL with the following command:
CREATE UNIQUE INDEX IDX_ProductName On Products (ProductName)

The above SQL command will not allow any duplicate values in the ProductName column, and an index is the best tool for the database to use to enforce this rule. Each time an application adds or modifies a row in the table, the database needs to search all existing records to ensure none of values in the new data duplicate existing values.

What are the disadvantages of an Index?
There are 2 disadvantages of an Index
1. Increased Disk Space
2. Insert, Update and Delete statements could be slow. In short, all DML statements could be slow.

Disk Space: Indexes are stored on the disk, and the amount of space required will depend on the size of the table, and the number and types of columns used in the index. Disk space is generally cheap enough to trade for application performance, particularly when a database serves a large number of users.

Insert, Update and Delete statements could be slow: Another downside to using an index is the performance implication on data modification statements. Any time a query modifies the data in a table (INSERT, UPDATE, or DELETE), the database needs to update all of the indexes where data has changed. Indexing can help the database during data modification statements by allowing the database to quickly locate the records to modify, however, providing too many indexes to update can actually hurt the performance of data modifications. This leads to a delicate balancing act when tuning the database for performance.

What are the 2 types of Indexes in SQL Server?
1. Clustered Index
2. Non Clustered Index

How many Clustered and Non Clustered Indexes can you have per table?
Clustered Index - Only one Clustered Index per table. A clustered index contains all of the data for a table in the index, sorted by the index key. Phone Book is an example for Clustered Index.
Non Clustered Index - You can have multiple Non Clustered Indexes per table. Index at the back of a book is an example for Non Clustered Index.

Which Index is faster, Clustered or Non Clustered Index?
Clustered Index is slightly faster than Non Clustered Index. This is because, when a Non Clustered Index is used there is an extra look up from the Non Clustered Index to the table, to fetch the actual rows.

When is it usually better to create a unique nonclustered index on the primary key column?
Sometimes it is better to use a unique nonclustered index on the primary key column, and place the clustered index on a column used by more queries. For example, if the majority of searches are for the price of a product instead of the primary key of a product, the clustered index could be more effective if used on the price field.

What is a Composite Index in SQL Server? or
What is the advantage of using a Composite Index in SQL Server? or
What is Covering Query?
A composite index is an index on two or more columns. Both clustered and nonclustered indexes can be composite indexes.
If all of the information for a query can be retrieved from an Index. A clustered index, if selected for use by the query optimizer, always covers a query, since it contains all of the data in a table.

By creating a composite indexes, we can have covering queries.

What is the difference between a Temporary Table and a Table Variable? Or
When do you use Table Variable over a Temporary Table and vice versa?

1. Table variable is created in the memory where as a temporary table is created in the TempDB. But, if there is a memory pressure, the pages belonging to a table variable may be pushed out to tempdb.

2. Table variables cannot be involved in transactions, logging or locking. This makes table variable faster than a temporary table.

3. You can pass table variable as parameter to functions and stored procedures, where as you cannot do the same with temporary table.

4. A temporary table can have indexes, whereas a table variable can only have a primary index. If speed is an issue Table variables can be faster, but if there are a lot of records, or there is a need to search the temporary table based on a clustered index, then a Temporary Table would be better. If you have less than 100 rows generally use a table variable. Otherwise use a temporary table. This is because SQL Server won't create statistics on table variables.

Let us understand the use of COALESCE with the help of an example
In this example, the **Candidate** table is shown to include three columns with information about a Candidate:
1. Candidate_id
2. PrimaryEmail
3. SecondaryEmail

COALESCE in the SELECT statement below, selects the PrimaryEmail if it is not null. If the PrimaryEmail is null then SecondaryEmail will be selected. If both PrimaryEmail and

SecondaryEmail is present then only PrimaryEmail is selected. So, COALESCE returns the first nonnull column among the list of columns passed. If both PrimaryEmail and SecondaryEmail is NULL, COALESCE returns NULL.

```
CREATE TABLE Candidate
(
   Candidate_id      tinyint    identity,
   PrimaryEmail   nvarchar(50)    NULL,
   SecondaryEmail    nvarchar(50)    NULL,
)

INSERT Candidate VALUES('abc@gmail.com',NULL)
INSERT Candidate VALUES('xyz@gmail.com', NULL)
INSERT Candidate VALUES('def@gmail.com', NULL)
INSERT Candidate VALUES(NULL,'123@gmail.com')
INSERT Candidate VALUES(NULL, '456@gmail.com')
INSERT Candidate VALUES(NULL, '789@gmail.com')
INSERT Candidate VALUES('xox@gmail.com', '789@gmail.com')

SELECT COALESCE(PrimaryEmail,SecondaryEmail) AS Email
FROM Candidate
```

COALESCE can also be used in joins as shown in the example below. If the Candidate table has a non null value in the Email column, then the value is selected. If the Email column is null in the Candidate Table then, CompanyEmail from CandidateCompany Table is selected.

```
CREATE TABLE Candidate
(
   CandidateId tinyint identity primary key,
   FullName       nvarchar(50)    NULL,
   Email          nvarchar(50)    NULL
)

INSERT Candidate VALUES('abc','abc@gmail.com')
INSERT Candidate VALUES('xyz', NULL)
INSERT Candidate VALUES('123', NULL)

CREATE TABLE CandidateCompany
(
   CompanyId      tinyint identity,
   CandidateId    tinyint REFERENCES Candidate(CandidateId),
   CompanyEmail   nvarchar(50)    NULL
)

INSERT CandidateCompany VALUES(1,'Company1@gmail.com')
INSERT CandidateCompany VALUES(2,'Company2@gmail.com')
INSERT CandidateCompany VALUES(3,'Company3@gmail.com')

SELECT Cand.FullName,COALESCE(Cand.Email,Comp.CompanyEmail) AS Email
FROM Candidate Cand INNER JOIN  CandidateCompany Comp
ON Cand.CandidateId = Comp.CandidateId
```

What is a Trigger in SQL Server?
A Trigger is a database object that is attached to a table. In many aspects it is similar to a stored procedure. As a matter of fact, triggers are often referred to as a "special kind of stored procedure." The main difference between a trigger and a stored procedure is that the former is attached to a table and is only fired when an INSERT, UPDATE or DELETE occurs.

What are the two types of Triggers in SQL Server?
1. After Triggers : Fired after Insert, Update and Delete operations on a table.
2. Instead of Triggers: Fired instead of Insert, Update and Delete operations on a table.

What are the special tables used by Triggers in SQL Server?
Triggers make use of two special tables called **inserted** and **deleted**. The **inserted** table contains the data referenced in an INSERT before it is actually committed to the database. The **deleted** table contains the data in the underlying table referenced in a DELETE before it is actually removed from the database. When an UPDATE is issued both tables are used. More specifically, the new data referenced in the UPDATE statement is contained in **inserted** table and the data that is being updated is contained in **deleted** table.

Give a real time example for triggers usage?
It is recomended to avoid triggers in a real time environment. There is one scenario I can think of why you may want to use triggers in a real time environment. Let us use an example to understand this.

I have 2 tables, **tblPerson** and **tblGender** as shown below. **GenderId** is the foriegn key in **tblPerson** table.

	Id	Name	GenderId
1	1	Mike	1
2	2	Pam	2
3	3	Sam	1
4	4	Sara	2
5	5	John	1

tblPerson

	Id	Gender
1	1	Male
2	2	Female

tblGender

Now create a view based on the above 2 tables as shown below.

```
Create View vWPersons
as
      Select P.Id,Name, Gender
      From tblPerson P
      Inner Join tblGender G
ON P.GenderId = G.Id
```

Select * from **vWPersons** will give us the result as shown below.

	Id	Name	Gender
1	1	Mike	Male
2	2	Pam	Female
3	3	Sam	Male
4	4	Sara	Female
5	5	John	Male

Now update the view the following query. This will change the Gender Text

to **Female** in **tblGender** table for Id = 1. This is not what we have expected.

Update tblGender Set Gender='Female' where Id=1

The base tables are updated incorrectly. So, Select * from **vWPersons** will give us the result as shown below.

	Id	Name	Gender
1	1	Mike	Female
2	2	Pam	Female
3	3	Sam	Female
4	4	Sara	Female
5	5	John	Female

To update the base tables correctly, you can create an **INSTEAD OF** trigger on the view as shown below.

```
Create Trigger trInstedOfUpdate
On vWPersons INSTEAD OF UPDATE
As
Begin
  Declare @PersonId int
  Declare @GenderId int
  Declare @GenderText varchar(50)

  Select @GenderText=Gender, @PersonId=Id from Inserted
  Select @GenderId=Id From tblGender Where Gender=@GenderText
  Update tblPerson Set GenderId=@GenderId where Id=@PersonId
End
```

Now run the query below which will update the underlying base tables correctly.
Update vWPersons Set Gender='Female' Where Id=1

Select * from vWPersons, will show the correct result set as shown below. The **INSTEAD OF** trigger has correctly updated the GenderId in tblPerson table.

	Id	Name	Gender
1	1	Mike	Female
2	2	Pam	Female
3	3	Sam	Male
4	4	Sara	Female
5	5	John	Male

So, **Instead of triggers** can be used to facilitate **updating Views that are based on multiple base tables**.

STORED PROCEDURES / USER DEFINED FUNCTIONS
1. Stored Procedure support deffered name resolution where as **functions** do not support deffered name resolution.

2. User Defined Function can be used in a select statement where as you cannot use a **stored procedure** in a select statement.

3. UDF's cannot return Image, Text where as a **StoredProcedure** can return any datatype.

4. In general **User Defined Functions** are used for computations where as **Stored Procedures** are used for performing business logic.

5. UDF should return a value where as **Stored Procedure** need not.

6. User Defined Functions accept lesser number of input parameters than **Stored Procedures. UDF** can have upto 1023 input parameters where as a **Stored Procedure** can have upto 21000 input parameters.

7. Temporary Tables can not be used in a **UDF** where as a **StoredProcedure** can use Temporary Tables.

8. UDF can not Execute Dynamic SQL where as a **Stored Procedure** can execute Dynamic SQL.

9. User Defined Function does not support error handling where as **Stored Procedure** support error handling. RAISEERROR or @@ERROR are not allowed in **UDF**s.

What is a View in SQL Server?
You can think of a **view** either as a **compiled sql query** or a **virtual** table. As a view represents a virtual table, it does not physically store any data. When you query a view, you actually retrieve the data from the underlying base tables.

What are the advantages of using views? Or **When do you usually use views?**
1. Views can be used to implement row level and column level security.

1) Combines complex tables and can have required columns from them.
2) Index can be created on views (indexed Views) for faster data retrieval.
3) View can be encrypted..make sure to store the source to decrypt later.
4) Views can be granted to the user for access instead of providing access of related tables used in it.
5) Parametrized views can be created using Inline UDF.
6) View can be materialized when indexed.

Example 1: Consider the **tblEmployee** table below. I don't want some of the users to have access to the salary column, but they should still be able to access ID, NAME and DEPT columns. If I grant access to the table, the users will be able to see all the columns. So, to achieve this, I can create a view as shown in **Listing 1** below. Now, grant access to the view and not the table. So using views we can provide **column level security**.

tblEmployee

ID	Name	Dept	Salary
1	Mike	HR	1000
2	Pam	IT	2000
3	Sam	Payroll	500
4	Todd	IT	1200
5	Jane	HR	800

Listing 1

```
Create View     vWEmployee
As
Select          ID, Name, Dept
From            tblEmployee
```

Example 2: Let us say, we have a few users who should be able to access only **IT employee** details and not any other dept. To do this, I can create a view as shown in **Listing 2** below. Now, grant access only to the view and not the table. So using views we can provide

row level security as well.

Listing 2

```
Create View    vWITEmployees
As
Select         ID, Name, Dept
From           tblEmployee
Where          Dept = 'IT'
```

2. Simplify the database schema to the users. You can create a view based on multiple tables which join columns from all these multiple tables so that they look like a single table.

3. Views can be used to present **aggregated** and **summarized** data.

Example 1: Consider the **tblEmployee** table above. I want to aggregate the data as shown in the image below. To do this I can create a view as shown in **Listing 3**. Now, you can simply issue a select query against the view rather than writing a complex query every time you want to retrieve the aggregated data.

Dept	Total
HR	2
IT	2
Payroll	1

Listing 3

```
Select      Dept, Count(*) As Total
From        tblEmployee
Group By    Dept
```

Can you create a view based on other views?
Yes, you can create a view **based on other views**. Usually we create views based on tables, but it also possible to create views based on views.

Can you update views?
Yes, views can be updated. However, updating a view that is based on multiple tables, may not update the underlying tables **correctly**. To correctly update a view that is based on multiple tables you can make use **INSTEAD OF triggers** in SQL Server. Click here for a real time example, that we have already discussed in SQL Server Interview Questions on triggers article.

What are indexed views? Or **What are materialized views?**
A view is a virtual table, it does not contain any physical data. A view is nothing more than compiled SQL query. Every time, we issue a select query against a view, we actually get the data from the underlying base tables and not from the view, as the view itself does not contain any data.

When you create an index on a view, the data gets physically stored in the view. So, when we issue a select query against an indexed view, the data is retrieved from the index without having to go to the underlying table, which will make the select statement to work slightly faster. However, the disadvantage is, INSERT, UPDATE and DELETE operations will become a little slow, because every time you insert or delete a row from the underlying table, the view index needs to be updated. Inshort, DML operations will have negative impact on performance.

Oracle refers to indexed views as materialized views.

Only the views created with schema binding, can have an Index. Simply adding WITH SCHEMABINDING to the end of the CREATE VIEW statement will accomplish this. However, the effect is that any changes to the underlying tables which will impact the view are not allowed. Since the indexed view is stored physically, any schema changes would impact the schema of the stored results set. Therefore, SQL Server requires that schema binding be used to prevent the view's schema (and therefore the underlying tables) from changing.

The first index for a view must be a UNIQUE CLUSTERED INDEX, after which, it's possible to create non-clustered indexes against the view.

Indexed Views are heavily used in data warehouses and reporting databases that are not highly transactional.

What are the limitations of a View?
1. You cannot pass parameters to a view.
2. Rules and Defaults cannot be associated with views.
3. The ORDER BY clause is invalid in views unless TOP or FOR XML is also specified.
4. Views cannot be based on temporary tables.

SQL Server can receive a huge performance boost by aggregating data with a view and applying an index. The first index to apply to a view is a clustered index. A clustered index will materialize the view. In other words, the index will persist the data in a view to disk. An indexed view can dramatically decrease the amount of time needed to aggregate data as the index already has the sorted, grouped, and calculated results. The drawbacks to watch for include increased disk space, and more overhead on INSERT, UPDATE, and DELETE statements on the view's base tables, because the database might need to update the index.

Index Scan:
Index Scan scans each and every record in the index. **Table Scan** is where the table is processed row by row from beginning to end. If the index is a clustered index then an **index scan** is really a **table scan**. Since a scan touches every row in the table whether or not it qualifies, the cost is proportional to the total number of rows in the table. **Hence, a scan is an efficient strategy only if the table is small.**

Index Seek:
Since a seek only touches rows that qualify and pages that contain these qualifying rows, the cost is proportional to the number of qualifying rows and pages rather than to the total number of rows in the table.

Example
I have an **employee** table as shown in the diagram below. EmployeeId is the primary key. We have a clustered index on the employeeId column.

EmployeeId	FirstName	LastName	Gender	Salary
1	Ben	B	M	10000
2	David	D	M	8000
3	Sara	S	F	7000
4	Tim	T	M	4000

Query 1 : **Select * from Employee where FirstName='Ben'**
Query 2 : **Select * from Employee where EmployeeId=2**

Query 1 will do an **Index scan** (Table Scan) to retrieve the record as there is no Index on the FirstName column.
Query 2 will do an **Index seek** to retrieve the record as there is an Index on the EmployeeId column.

Delete from tables involved in a SQL Join

This question is not asked that often in an interview, but you may have to write a query to delete from tables that are involved in a SQL join in your day to day job routine as a SQL Developer.

Let me explain you the question first. Consider the **Employees** and **Departments** tables below.
I want you to write a query using joins which will give me the **list of all HR Department employees**. The output should be as shown below.

To do this we will write a query using joins as shown below.

```
Select E.Name as [Employee Name],D.Name as [Department Name]
from Employees E
Inner Join Departments D
On E.DeptId = D.Id
Where D.Name = 'HR'
```

Now I want you to write a query, that **deletes all the HR department employees**. To do this we have to join **Employees** and **Departments** tables and usually we will be tempted to write the query as shown below, but this query will not execute and will give a syntax error.

Incorrect SQL Delete Query to delete from multiple tables invloved in a SQL join

```
Delete From Employees E
Inner Join Departments D
On E.DeptId = D.Id
Where D.Name = 'HR'
```

The query below shows the correct syntax for issuing a SQL delete that involves a SQL join.

```
Delete E From (Employees E
Inner Join Departments D
On E.DeptId = D.Id
and D.Name = 'HR')
```

The following syntax is also valid and has the same effect.

```
Delete Employees From
(
Employees join Departments
on Employees.DeptId = Departments.Id
)
where Departments.Name = 'HR'
```

Advantages of stored procedures

This is a very common sql server interview question. There are several advantages of using stored procedures over adhoc queries, as listed below.

1. **Better Performance** : Stored Procedures are compiled and their execution plan is cached and used again, when the same SP is executed again. Although adhoc queries also create and reuse plan, the plan is reused only when the query is textual match and the datatypes are matching with the previous call. Any change in the datatype or you have an extra space in the query then, a new plan is created.

2. **Better Security** : Applications making use of dynamically built adhoc sql queries are highly
susceptible to sql injection attacks, where as Stored Procedures can avoid SQL injection attacks completely.

3. **Reduced Network Traffic**: Stored procedures can reduce network traffic to a very great extent when compared with adhoc sql queries. With stored procedures, you only need to send the name of the procedure between client and server. Imagine the amount of network bandwith that can be saved especially if the stored procedure contains 1000 to 2000 lines of SQL.

4. **Better Maintainance and Reusability**: Stored procedures can be used any where in the application. It is easier to maintain a stored procedure that is used on several pages as themodfifcations just need to be changed at one place where the stored procedure is defined. On the other hand, maintaining an adhoc sql query that's used on several pages is tedious and error prone, as we have to make modifications on each and every page.

What are the different ways to replace NULL values in SQL Server

This interview question is not that common. My friend faced this interview question, when he attended an interview in London. My friend said we can use **COALESCE()** in SQL Server. Then the interviewer said, that's very good answer, but do you know of any other way?

Apart from using **COALESCE()**, there are 2 other ways to replace NULL values in SQL Server. Let's understand this with an example.

I have a Table tblEmployee, as shown in the diagram below. Some of the Employees does not have gender. All those employees who does not have Gender, must have a replacement value of **'No Gender'** in your query result. Let's explore all the 3 possible options we have.

Id	Name	Gender	Department
1	Don	Male	IT
2	Sam	NULL	HR
3	Ram	Male	IT
4	Pam	NULL	NULL
5	Steve	NULL	NULL

Option 1 : Replace NULL values in SQL Server using **ISNULL()** function.

```
Select Name, ISNULL(Gender,'No Gender') as Gender
From tblEmployee
```

Option 2 : Replace NULL values in SQL Server using **CASE**.

```
Select Name, Case  When Gender IS NULL Then 'No Gender' Else Gender End as Gender
From tblEmployee
```

Option 3 : Replace NULL values in SQL Server using **COALESCE()** function.

```
Select Name, Coalesce(Gender, 'No Gender') as Gender
From tblEmployee
```

SQL Server interview questions on string manipulation functions

The following 2 SQL Server Interview questions were asked when I attended an interview for SQL Server Developer role.

Can you list a few useful string manipulation functions in SQL Server?
LEN(), SUBSTRING(), CHARINDEX(), LEFT(), RIGHT() etc.

Then he asked me, Can you give me one example of where you have used these functions in your experience?

The following is one simple real time example, where we can use LEN(), CHARINDEX() and SUBSTRING() functions. Let us assume we have table as shown below.

Email	Name
don@aaa.com	Don
ram@bbb.com	Ram
sam@aaa.com	Sam
tom@ccc.com	Tom

I want you to write a query to find out total number of emails, by domain. The result of the query should be as shown below.

Email	Count
aaa.com	2
bbb.com	1
ccc.com	1

We can use LEN(), CHARINDEX() and SUBSTRING() functions to produce the desired results. Please refer to the query below.

```
Select SUBSTRING(Email,CHARINDEX('@',Email)+
1,(LEN(Email) - CHARINDEX('@',Email))) as EmailDomain, Count(*) as Total
From TableName
Group By SUBSTRING(Email,CHARINDEX('@',Email)+1,(LEN(Email) - CHARINDEX('@',Email)))
Order by Count(*) Desc
```

Write a Stored Procedure that takes column name as a parameter and returns the result sorted by the column that is passed

Let's understand the requirement better with an example. I have an Employee table as shown below.

	Id	Name	Gender	Salary	City
1	1	Tom	Male	55000	London
2	2	Mary	Female	40000	New York
3	3	John	Male	45000	Paris
4	4	Todd	Male	30000	Sydney
5	5	Sara	Female	35000	Delhi

I want a stored procedure that returns employee data sorted by a column, that the user is going to pass into the stored procedure as a parameter. There are 2 ways of doing this.

Option 1: Use Case Statement as shown below:

```
Create Proc spGetEmployeesSorted
@SortCoumn nvarchar(10)
as
Begin

Select [Id],[Name],[Gender],[Salary],[City]
From   [Employee]
Order by Case When @SortCoumn = 'Id' Then Id End,
            Case When @SortCoumn = 'Name' Then Name End,
            Case When @SortCoumn = 'Gender' Then Gender End,
            Case When @SortCoumn = 'Salary' Then Salary End,
            Case When @SortCoumn = 'City' Then City End
End
```

Option 2: Use Dynamic SQL as shown below:

```
Create Proc spGetEmployeesSortedUsingDynamicSQL
@SortCoumn nvarchar(10)
as
Begin

Declare @DynamicQuery nvarchar(100)
Set @DynamicQuery = 'select [Id],[Name],[Gender],[Salary],[City] from [Employee] order by ' +
@SortCoumn
Execute(@DynamicQuery)

End
```

What is deferred name resolution in SQL Server?

Let me explain deferred name resolution with an example. Consider the stored procedure shown below.

```
Create procedure spGetCustomers
as
Begin
Select * from Customers1
End
```

Customers1 table does not exist. When you execute the above SQL code, the stored procedure **spGetCustomers** will be successfully created without errors. But when you try to **call or execute** the stored procedure using **Execute spGetCustomers**, you will get a run time error stating Invalid object name 'Customers1'.

So, at the time of creating stored procedures, only the syntax of the sql code is checked. The objects used in the stored procedure are not checked for their existence. Only when we try to run the procedure, the existence of the objects is checked. So, the process of postponing, the checking of physical existence of the objects until runtime, is called as deffered name resolution in SQL server.

Functions in sql server does not support **deferred name resolution**. If you try to create an inline table valued function as shown below, we get an error stating Invalid object name 'Customers1' at the time of creation of the function itself.

```
Create function fnGetCustomers()
returns table
as
return Select * from Customers1
```