**SQL Assignment 4**

**Q1)** Explain different types of views. Demonstrate with suitable examples.

### System Defined Views

The System Defined Views are predefined views that already exist in the SQL Server database, such as Tempdb, Master, and temp. Each of the databases has its own properties and functions.

The template database for all User Defined views is from the Master database. It contains many predefined views that are templates for tables and other databases.

**Information Schema**

There are twenty different schema views in the SQL server. They are used to display the physical information of the database, such as tables, constraints, columns, and views. This view starts with INFORMATION\_SCHEMA and followed by the View Name. INFORMATION\_SCHEMA.CHECK\_CONSTRAINTS is used to receive information about any constraint available in the database.

**Dynamic Management View**

These were introduced in the SQL server in 2005. The administer can get information about the server state to diagnose problems, monitor the health of the server instance, and tune performance through these views. The Server-scoped Dynamic Management View is only stored in the Master database, whereas the Database-scoped Dynamic Management View is stored in each database.

**User Defined Views**

These are the types of views that are defined by the users. There are two types under User Defined views, Simple View and Complex View.

Q2) What is the difference between function and stored procedure? Write syntax for creating functions and stored procedures.

Following are the main differences between functions and procedures:

| **Functions** | **Procedures** |
| --- | --- |
| 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 procedure. |
| 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. |

|  |
| --- |
|  |
| syntax: |
|  | create function function\_name (@parameter1 datatype, @parameter2 datatype,..... @parameterN datatype) |
|  | returns return\_datatype |
|  | as |
|  | begin |
|  | -----function body---- |
|  | end |
|  |  |

SYNTAX: create procedure spname

as

begin

end

Q3) What is an index in SQL? What are the different types of indexes in SQL?

Indexes are **used to retrieve data from the database more quickly than otherwise**. The users cannot see the indexes, they are just used to speed up searches/queries. Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update).

CLUSTERED INDEX

Clustered Index stores and sort rows of data in a view or table depending on their central values. There may be an instance of having just one clustered index in each table, as it can empower the client to store data in a solitary request. Clustered index store data in an arranged way, and in this way, at whatever point data is contained in the table in an arranged manner implies it is orchestrated with a clustered index.

At the point when a table contains a clustering in SQL server, it is named a clustered table. A clustered index is liked to utilize when adjustment of gigantic information is needed in any data set. If the data put away in a table or data set are not organized in descending or ascending request, at that point, the data table is named as a heap.

NON CLUSTERED INDEX

Clustered Index stores and sort rows of data in a view or table depending on their central values. There may be an instance of having just one clustered index in each table, as it can empower the client to store data in a solitary request. Clustered index store data in an arranged way, and in this way, at whatever point data is contained in the table in an arranged manner implies it is orchestrated with a clustered index.

It represents a structure, which is isolated from data rows. This types of indexes in SQL server covers the non-clustered key values, and each worth pair has a pointer to the data row that comprises vital significance.

In the non-clustered index, the client can undoubtedly add non-key to the leaf level, as it sidesteps the current index key cut-off points and performs completely covered recorded questions.

Q4) Showcase an example of exception handling in SQL stored procedure.

The TRY CATCH construct allows you to gracefully handle exceptions in SQL Server. To use the TRY CATCH construct,

we should first place a group of Transact-SQL statements that could cause an exception in a BEGIN TRY...END TRY block

as follows:

BEGIN TRY

-- statements that may cause exceptions

END TRY

Then you use a BEGIN CATCH...END CATCH block immediately after the TRY block:

BEGIN CATCH

-- statements that handle exception

END CATCH

The following illustrates a complete TRY CATCH construct:

BEGIN TRY

-- statements that may cause exceptions

END TRY

BEGIN CATCH

-- statements that handle exception

END CATCH

If the statements between the TRY block complete without an error, the statements between the CATCH block will not

execute. However, if any statement inside the TRY block causes an exception, the control transfers to the statements

in the CATCH block.

--example of exception handling in SQL stored procedure:

use master

CREATE PROC usp\_divide(

@a decimal,

@b decimal,

@c decimal output

) AS

BEGIN

BEGIN TRY

SET @c = @a / @b;

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_SEVERITY() AS ErrorSeverity

,ERROR\_STATE() AS ErrorState

,ERROR\_PROCEDURE() AS ErrorProcedure

,ERROR\_LINE() AS ErrorLine

,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

END;

GO

--call the usp\_divide stored procedure to divide 10 by 2:

DECLARE @r decimal;

EXEC usp\_divide 10, 2, @r output;

PRINT @r;

--Because no exception occurred in the TRY block, the stored procedure completed at the TRY block.

--attempt to divide 20 by zero by calling the usp\_divide stored procedure:

DECLARE @r2 decimal;

EXEC usp\_divide 10, 0, @r2 output;

PRINT @r2;

Because of division by zero error which was caused by the formula, the control was passed to the statement

inside the CATCH block which returned the error’s detailed information

Q5) Create a SQL function to split strings into rows on a given character?

Input String: Stephen;peter;berry;Olivier;caroline;

|  |
| --- |
|  |
| use [AdventureWorks2019] |
|  | select value from string\_split('Stephen;peter;berry;Olivier;caroline;', |
|  |  |

Q6) What is a temporary and a variable table? Write suitable syntax to create temporary tables and variable tables.

|  |  |
| --- | --- |
| Temp Table | Table Variable |
| A Temp table is easy to create and back up data. | Table variable involves the effort when you usually create the normal tables. |
| Temp table result can be used by multiple users. | Table variable can be used by the current user only. |
| Temp table will be stored in the tempdb. It will make network traffic. When you have large data in the temp table then it has to work across the database. A Performance issue will exist. | Table variable will store in the physical memory for some of the data, then later when the size increases it will be moved to the tempdb. |
| Temp table can do all the DDL operations. It allows creating the indexes, dropping, altering, etc.. | Table variable won't allow doing the DDL operations. But the table variable allows us to create the clustered index only. |
| Temp table can be used for the current session or global. So that a multiple user session can utilize the results in the table. | Table variable can be used up to that program. (Stored procedure) |

|  |
| --- |
|  |
|  |  |
|  | The Syntax to create a Temporary |
|  | Table is given below: |
|  | To Create Temporary Table: |
|  | CREATE TABLE #EmpDetails (id INT, name VARCHAR(25)) |
|  | To Insert Values Into Temporary Table: |
|  | INSERT INTO #EmpDetails VALUES (01, 'Lalit'), (02, 'Atharva') |
|  | To Select Values from Temporary Table: |
|  | SELECT \* FROM #EmpDetails |
|  | Result: |
|  | id name |
|  | 1 Lalit |
|  | 2 Atharva |
|  | There are 2 types of Temporary Tables: Local Temporary Table, and Global Temporary Table. These are explained as |
|  | following below. |
|  |  |
|  | Local Temporary Table: |
|  | A Local Temp Table is available only for the session that has created it. It is automatically dropped (deleted) |
|  | when the connection that has created it, is closed. To create Local Temporary Table Single “#” is used as the |
|  | prefix of a table name. |
|  | Also, the user can drop this temporary table by using the “DROP TABLE #EmpDetails” query. There will be Random |
|  | Numbers are appended to the Name of Table Name. If the Temporary Table is created inside the stored procedure, |
|  | it get dropped automatically upon the completion of stored procedure execution. |
|  |  |
|  | Example: |
|  |  |
|  | CREATE PROCEDURE ProcTemp |
|  | AS |
|  | BEGIN |
|  | CREATE TABLE #EmpDetails |
|  | INSERT INTO #EmpDetails VALUES ( 01, 'Lalit'), ( 02, 'Atharva') |
|  | SELECT \* FROM #EmpDetails |
|  | END |
|  | EXECUTE ProcTemp |
|  |  |
|  | Global Temporary Table: |
|  | To create a Global Temporary Table, add the “##” symbol before the table name. |
|  | Example: |
|  |  |
|  | CREATE TABLE ##EmpDetails (id INT, name VARCHAR(25)) |
|  | Global Temporary Tables are visible to all connections and Dropped when the last connection referencing the table is |
|  | closed. Global Table Name must have an Unique Table Name. There will be no random Numbers suffixed at the end of the |
|  | Table Name. |