Explain different types of views. Demonstrate with suitable examples.

There are two types of views in the SQL Server, namely System Defined Views and User Defined Views.

System Defined Views

System-defined Views are predefined Views that already exist in the Master database of SQL Server . These are also used as template Views for all newly created databases. These system Views will be automatically attached to any user-defined database.

We have the following types of system defined views.

## 1)Information Schema View

In SQL Server we have twenty different schema views. These are used to display information in a database, like as tables and columns. This type of view starts with INFORMATION\_SCHEMA and after this view name.

## 2)Catalog View

Catalog Views were introduced with SQL Server 2005. These are used to show database self-describing information.

## Dynamic Management View

Dynamic Management Views were introduced in SQL Server 2005. These Views give the administrator information about the database about the current state of the SQL Server machine. These values help the administrator to analyze problems and tune the server for optimal performance. These are of two types

## i)Server-scoped Dynamic Management View

These are stored only in the Master database.

## ii)Database-scoped Dynamic Management View

These are stored in each database.

## User Defined Views

These types of views are defined by users. We have two types of user-defined views.

## Simple View

When we create a view on a single table, it is called a simple view.

## ii)Complex View

When we create a view on more than one table, it is called a complex view.

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

Both stored procedures and functions are database objects which contain a set of SQL statements to complete a task. In many ways, both are different from each other. In this article, we’re going to discuss the differences between stored procedures and functions.

1. The function must return a value but inStored Procedure it is optional. Even a procedure can return zero or n values.
2. Functions can have only input parameters for it whereas Procedures can have input or output parameters.
3. Functions can be called from Procedure whereas Procedures cannot be called from a Function.

Syntax of creating Procedure

CREATE [OR REPLACE] PROCEDURE procedure\_name (<Argument> {IN, OUT, IN OUT} <Datatype>,…)  IS Declaration section<variable, constant> ;

BEGIN Execution section EXCEPTION Exception section  END

Syntax of creating Functions

CREATE [OR REPLACE] FUNCTION function\_name [parameters]  RETURN return\_datatype; {IS, AS} Declaration\_section <variable,constant> ;

BEGIN   Execution\_section Return return\_variable;

EXCEPTION exception section   Return return\_variable;

END;

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

A SQL index is **used to retrieve data from a database very fast**. Indexing a table or view is, without a doubt, one of the best ways to improve the performance of queries and applications. A SQL index is a quick lookup table for finding records users need to search frequently.

**There are two types of Indexes in SQL Server:**

Clustered Index.

Non-Clustered Index.

Showcase an example of exception handling in SQL stored procedure.

BEGIN TRY

DECLARE @num INT, @msg varchar(200)

---- Divide by zero to generate Error

SET @num = 5/0

PRINT 'This will not execute'

END TRY

BEGIN CATCH

PRINT 'Error occured that is'

set @msg=(SELECT ERROR\_MESSAGE())

print @msg;

END CATCH

GO

https://dotnettrickscloud.blob.core.windows.net/img/sqlserver/dividebyzero1.png

BEGIN TRY

DECLARE @num INT

---- Divide by zero to generate Error

SET @num = 5/0

PRINT 'This will not execute'

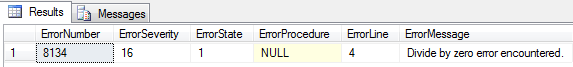
END TRY

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;

GO



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

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

|  |
| --- |
| Stephen |
| Peter |
| Berry |
| Oliver |
| Caroline |

Input\_String = " Stephen;peter;berry;Olivier;caroline "

Input\_String -split ";", 5

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

Temporary tables and table variables, both have their own pros and cons. We need to decide which one to use and when.

1) Table variable (@table) is created in the memory. Whereas, a Temporary table (#temp) is created in the tempdb database. However, if there is a memory pressure the pages belonging to a table variable may be pushed to tempdb.

2) Table variables cannot be involved in transactions, logging or locking. This makes @table faster then #temp. So table variable is faster then temporary table.

3) Temporary tables are allowed CREATE INDEXes whereas, Table variables aren’t allowed CREATE INDEX instead they can have index by using Primary Key or Unique Constraint.

4) Table variable can be passed as a parameter to functions and stored procedures while the same cannot be done with Temporary tables.

5) Temporary tables are visible in the created routine and also in the child routines. Whereas, Table variables are only visible in the created routine.

6) Temporary table allows Schema modifications unlike Table variables.

**Table Variable in SQL Server – Example**

Table variable is a very useful programming construct, like that of any other variable.

1. **DECLARE** @TStudent **TABLE**
2. (
3. RollNo **INT** IDENTITY(1,1),
4. StudentID **INT**,
5. **Name** **INT**
6. )
7. --Insert data to Table variable @TStudent
8. **INSERT** **INTO** @TStudent(StudentID,**Name**)
9. **SELECT** **DISTINCT** StudentID, **Name** **FROM** StudentMaster **ORDER** **BY** StudentID **ASC**
11. --Select data from Table variable @TStudent
12. **SELECT** \* **FROM** @TStudent
14. --Next batch
15. GO
16. **SELECT** \* **FROM** @TStudent --gives error
17. **DECLARE** @TStudent **TABLE**
18. (
19. RollNo **INT** IDENTITY(1,1),
20. StudentID **INT**,
21. **Name** **INT**
22. )
23. --Insert data to Table variable @TStudent
24. **INSERT** **INTO** @TStudent(StudentID,**Name**)
25. **SELECT** **DISTINCT** StudentID, **Name** **FROM** StudentMaster **ORDER** **BY** StudentID **ASC**
27. --Select data from Table variable @TStudent
28. **SELECT** \* **FROM** @TStudent
30. --Next batch
31. GO
32. **SELECT** \* **FROM** @TStudent --gives error

**Temporary Tables in SQL Server – Example**

In SQL Server, based on the scope and behavior, temporary tables are of two types,

1. **Local** **Temporary** Tables (#**temp**)
2. **Global** **Temporary** Tables (##**temp**)
4. **CREATE** **TABLE** #StudentTemp
5. (
6. StudentID **int**,
7. **Name** **varchar**(50),
8. Address **varchar**(150)
9. )
10. GO
11. **INSERT** **INTO** #StudentTemp **VALUES** ( 1, 'Dipendra','Pune');
12. GO
13. **SELECT** \* **FROM** #StudentTemp
14. **CREATE** **TABLE** #StudentTemp
15. (
16. StudentID **int**,
17. **Name** **varchar**(50),
18. Address **varchar**(150)
19. )
20. GO
21. **INSERT** **INTO** #StudentTemp **VALUES** ( 1, 'Dipendra','Pune');
22. GO
23. **SELECT** \* **FROM** #StudentTemp