SQL Assignment 1  
  
1. Explain different types of views. Demonstrate with suitable examples.  
  
In databases, a **view** is a virtual table based on the result-set of a SQL query. Views are used to simplify complex queries, provide security by limiting data access, and present data in a particular format. There are different types of views in SQL and relational databases, each with specific use cases.

|  |  |
| --- | --- |
| **Simple View** | Retrieves data from one table, used for basic filtering and simplification. |

|  |  |
| --- | --- |
| **Materialized View** | Physically stores query results, used to improve performance in read-heavy operations. |

|  |  |
| --- | --- |
| **Indexed View** | Physically stores data with indexing, improves query performance for frequently accessed data. |

|  |  |
| --- | --- |
| **Partitioned View** | Combines data from multiple tables for large datasets, helpful for partitioning and horizontal scaling. |

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

|  |  |  |
| --- | --- | --- |
| **Purpose** | Used to perform calculations or return a single value. Typically used in SELECT statements. | Used to perform a sequence of operations, may or may not return a value. |

|  |  |  |
| --- | --- | --- |
| **Return Value** | Always returns a value (scalar, table, or record). | Can return zero or more values, or no value at all. |

|  |  |  |
| --- | --- | --- |
| **Invocation** | Can be called within a SELECT statement. | Called using the EXEC or EXECUTE command. |

|  |  |  |
| --- | --- | --- |
| **Parameters** | Only input parameters are allowed. | Can have input, output, and input-output parameters. |

|  |  |  |
| --- | --- | --- |
| **Transaction Control** | Functions cannot perform transactions like BEGIN, COMMIT, or ROLLBACK. | Stored procedures can manage transactions. |

**Function Syntax:**CREATE FUNCTION GetEmployeeAge(@BirthDate DATE) RETURNS INT AS BEGIN RETURN DATEDIFF(YEAR, @BirthDate, GETDATE()); END;  
  
**Stored Procedure Syntax:**CREATE PROCEDURE AddNewEmployee @Name NVARCHAR(100), @DepartmentID INT, @Salary DECIMAL(10,2) AS BEGIN INSERT INTO Employees (Name, DepartmentID, Salary) VALUES (@Name, @DepartmentID, @Salary); -- Return a message SELECT 'New employee added successfully' AS ResultMessage; END;  
  
Understanding when to use a function versus a stored procedure depends on the task you need to accomplish. Use **functions** for returning a value or set of values in a query, and use **stored procedures** for more complex operations, such as data modifications or batch processing.  
  
3. What is an index in SQL? What are the different types of indexes in SQL?  
  
An **index** in SQL is a database object that improves the speed of data retrieval operations on a table at the cost of additional space and slower data modification operations (such as INSERT, UPDATE, and DELETE). Indexes allow the database engine to find and retrieve specific rows much faster than scanning the entire table.  
Types of index:  
**Clustered Index** , **Non-Clustered Index** , **Unique Index** , **Composite Index** , **Full-Text Index,** **Filtered Index** , **Bitmap Index** (Specific to certain databases) , **Spatial Index** (Used for geographical data)  
  
4. Showcase an example of exception handling in SQL stored procedure  
  
In SQL Server (or most relational databases), exception handling in a stored procedure is commonly done using TRY...CATCH blocks. This allows you to handle errors gracefully without stopping the execution of the whole batch.  
Here’s an example of how to implement exception handling in an SQL stored procedure:  
  
CREATE PROCEDURE InsertEmployee @EmployeeName NVARCHAR(100), @Age INT, @Department NVARCHAR(100) AS BEGIN   
-- Start of the TRY block   
BEGIN TRY   
-- Insert data into Employee table  
 INSERT INTO Employees (EmployeeName, Age, Department) VALUES (@EmployeeName, @Age, @Department);  
 -- If successful, return a success message  
 PRINT 'Employee inserted successfully.'; END TRY -- Start of the CATCH block BEGIN CATCH  
 -- In case of error, capture and log the error details  
 DECLARE @ErrorMessage NVARCHAR(4000); DECLARE @ErrorSeverity INT; DECLARE @ErrorState INT;   
-- Get the error details   
SELECT @ErrorMessage = ERROR\_MESSAGE(), @ErrorSeverity = ERROR\_SEVERITY(), @ErrorState = ERROR\_STATE();   
-- Log the error message (this could be inserted into an error log table)  
 PRINT 'Error occurred: ' + @ErrorMessage;  
 -- Optionally raise the error if you want it to propagate further   
RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState); END CATCH END

5.Create a SQL function to split strings into rows on a given character? Input String: Stephen;peter;berry;Olivier;caroline;  
  
CREATE FUNCTION dbo.SplitString  
( @InputString NVARCHAR(MAX),

@Delimiter CHAR(1))s

RETURNS @OutputTable TABLE

( Item NVARCHAR(100))

AS BEGIN DECLARE @Start INT, @End INT;  
 -- Initialize starting point  
SET @Start = 1;  
SET @End = CHARINDEX(@Delimiter, @InputString);

-- Loop through the string and extract substrings until no more delimiters are found

WHILE @End > 0 BEGIN  
 -- Insert each part of the string into the output table  
 INSERT INTO @OutputTable (Item)  
 VALUES (SUBSTRING(@InputString, @Start, @End - @Start));

-- Move to the next part  
SET @Start = @End + 1;  
SET @End = CHARINDEX(@Delimiter, @InputString, @Start);  
 END  
 -- Insert the last part after the final delimiter  
 INSERT INTO @OutputTable (Item)  
 VALUES (SUBSTRING(@InputString, @Start, LEN(@InputString) - @Start + 1));  
RETURN  
END;  
SELECT \* FROM dbo.SplitString('Stephen;peter;berry;Olivier;caroline', ';');  
  
6.What is a temporary and a variable table? Write suitable syntax to create temporary tables and variable tables.

Temporary tables in SQL are special tables that are created to store temporary data and automatically deleted when the session ends or the connection is closed. These tables are useful for intermediate data manipulation, staging, or storing data during complex operations.  
-- Local Temporary Table

CREATE TABLE #TempTableName (Column1 DataType,Column2 DataType,);

-- Global Temporary Table  
CREATE TABLE ##GlobalTempTableName (Column1 DataType,Column2 DataType ...);  
  
Table variables are similar to temporary tables but are declared as variables in SQL. They exist only for the duration of the batch or stored procedure where they are declared. They are often faster than temporary tables for smaller datasets and have more limited functionality.  
DECLARE @VariableTableName TABLE ( Column1 DataType, Column2 DataType, ... )