**SQL Assignment 4**

**1. Different Types of Views in SQL**

A **view** in SQL is a virtual table created by a query. It does not store data physically but acts as a saved query that can be reused. Views are of the following types:

**a) Simple View**

* Created from a single table.
* Does not contain functions or grouping of data.
* Allows DML operations (INSERT, UPDATE, DELETE).

**Example:**

CREATE VIEW EmployeeView AS

SELECT EmployeeID, FirstName, LastName

FROM Employees;

**b) Complex View**

* Created from multiple tables.
* May contain functions, grouping, or joins.
* Does not always allow DML operations.

**Example:**

CREATE VIEW DepartmentEmployeeView AS

SELECT d.DepartmentName, e.FirstName, e.LastName

FROM Departments d

JOIN Employees e ON d.DepartmentID = e.DepartmentID;

**c) Materialized View**

* Stores the result of the query physically.
* Used for improving performance by precomputing expensive operations.
* Needs to be refreshed when the underlying data changes.

**Example:**

CREATE MATERIALIZED VIEW SalesSummary AS

SELECT ProductID, SUM(Quantity) AS TotalQuantity

FROM Sales

GROUP BY ProductID;

**2. Difference Between Function and Stored Procedure**

| **Aspect** | **Function** | **Stored Procedure** |
| --- | --- | --- |
| **Return Value** | Must return a single value or a table. | Can return zero, one, or multiple values. |
| **Usage** | Can be used in SELECT statements. | Cannot be used in SELECT statements. |
| **DML Operations** | Cannot perform DML operations (e.g., INSERT). | Can perform DML operations. |
| \*\*Transaction Management | Cannot manage transactions. | Can manage transactions. |

**Syntax for Creating a Function**

CREATE FUNCTION GetEmployeeName (@EmployeeID INT)

RETURNS VARCHAR(100)

AS

BEGIN

RETURN (SELECT FirstName + ' ' + LastName FROM Employees WHERE EmployeeID = @EmployeeID);

END;

**Syntax for Creating a Stored Procedure**

CREATE PROCEDURE GetEmployeeDetails (@EmployeeID INT)

AS

BEGIN

SELECT \* FROM Employees WHERE EmployeeID = @EmployeeID;

END;

**3. Index in SQL**

An **index** is a database object used to speed up data retrieval operations. It works like a book index, allowing the database to find data without scanning the entire table.

**Types of Indexes**

1. **Clustered Index**: Determines the physical order of data in a table. Only one clustered index per table.

CREATE CLUSTERED INDEX IX\_EmployeeID ON Employees (EmployeeID);

1. **Non-Clustered Index**: Does not alter the physical order but creates a separate structure to speed up queries.

CREATE NONCLUSTERED INDEX IX\_LastName ON Employees (LastName);

1. **Unique Index**: Ensures that all values in the indexed column are unique.

CREATE UNIQUE INDEX IX\_Email ON Employees (Email);

1. **Composite Index**: Created on multiple columns

CREATE INDEX IX\_NameDepartment ON Employees (FirstName, DepartmentID);

**4. Exception Handling in SQL Stored Procedure**

Exception handling in SQL is done using TRY...CATCH blocks.

**Example**

CREATE PROCEDURE DivideNumbers (@Num1 INT, @Num2 INT)

AS

BEGIN

BEGIN TRY

SELECT @Num1 / @Num2 AS Result;

END TRY

BEGIN CATCH

SELECT

ERROR\_MESSAGE() AS ErrorMessage,

ERROR\_NUMBER() AS ErrorNumber,

ERROR\_STATE() AS ErrorState;

END CATCH

END;

**5. SQL Function to Split Strings into Rows**

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

**Function:**

CREATE FUNCTION SplitString (@InputString VARCHAR(MAX), @Delimiter CHAR(1))

RETURNS @OutputTable TABLE (Value VARCHAR(100))

AS

BEGIN

DECLARE @Value VARCHAR(100);

WHILE CHARINDEX(@Delimiter, @InputString) > 0

BEGIN

SET @Value = SUBSTRING(@InputString, 1, CHARINDEX(@Delimiter, @InputString) - 1);

INSERT INTO @OutputTable (Value) VALUES (@Value);

SET @InputString = SUBSTRING(@InputString, CHARINDEX(@Delimiter, @InputString) + 1, LEN(@InputString));

END

INSERT INTO @OutputTable (Value) VALUES (@InputString);

RETURN;

END;

**Usage:**

SELECT \* FROM SplitString('Stephen;peter;berry;Olivier;caroline;', ';');

**6. Temporary and Variable Tables**

**Temporary Table**

* A temporary table is stored in the tempdb database and is automatically deleted when the session ends.
* Syntax:

CREATE TABLE #TempTable (

ID INT,

Name VARCHAR(100)

);

**Variable Table**

* A table variable is stored in memory and is scoped to the batch, stored procedure, or function.
* Syntax:

DECLARE @VarTable TABLE (

ID INT,

Name VARCHAR(100)

);