SQL Assignment 4

# 1. Different Types of Views in SQL

A view in SQL is a virtual table that is based on the result of a query. It doesn’t store data physically, but it presents data from one or more tables.  
  
Types of Views:  
  
1. Simple View:  
 - A simple view is based on a single table.  
 - It can be used to select specific columns or rows.  
 - You cannot modify the data in a simple view if it’s based on multiple tables or aggregate functions.  
   
 Example:  
 CREATE VIEW simple\_view AS  
 SELECT first\_name, last\_name  
 FROM employees;  
  
2. Complex View:  
 - A complex view is based on multiple tables, often using joins or aggregate functions.  
 - It may involve grouping, ordering, or filtering.  
   
 Example:  
 CREATE VIEW complex\_view AS  
 SELECT orders.order\_id, customers.customer\_name, orders.order\_date  
 FROM orders  
 JOIN customers ON orders.customer\_id = customers.customer\_id;  
  
3. Materialized View:  
 - A materialized view stores the query result physically.  
 - It can be refreshed periodically to reflect changes in the underlying tables.  
   
 Example:  
 CREATE MATERIALIZED VIEW sales\_summary AS  
 SELECT product\_id, SUM(sales) AS total\_sales  
 FROM sales  
 GROUP BY product\_id;  
  
4. Inline View:  
 - An inline view is a temporary result set created in the FROM clause of a query.  
   
 Example:  
 SELECT employee\_name  
 FROM (SELECT employee\_name, salary FROM employees WHERE department\_id = 10) AS dept\_10\_employees;

# 2. Difference Between Function and Stored Procedure

- Function:  
 - A function returns a value (can be scalar or table).  
 - It is used for computations and operations.  
 - A function can be used in SQL queries (e.g., in SELECT, WHERE, or HAVING clauses).  
  
- Stored Procedure:  
 - A stored procedure may or may not return a value.  
 - It performs actions such as inserting, updating, or deleting data.  
 - A stored procedure cannot be used directly in SQL queries.  
  
Syntax for Creating Function:  
CREATE FUNCTION function\_name (parameter1 datatype, parameter2 datatype)  
RETURNS return\_type  
AS  
BEGIN  
 -- function logic  
 RETURN value;  
END;  
  
Syntax for Creating Stored Procedure:  
CREATE PROCEDURE procedure\_name (parameter1 datatype, parameter2 datatype)  
AS  
BEGIN  
 -- procedure logic  
 -- It can be any action like INSERT, UPDATE, DELETE  
END;

# 3. Index in SQL

An index in SQL is a database object used to speed up the retrieval of rows from a table.  
  
Types of Indexes:  
1. Single Column Index:  
 - An index created on a single column to speed up searches.  
 Example:  
 CREATE INDEX idx\_name ON employees (last\_name);  
  
2. Composite Index (Multi-column Index):  
 - An index created on multiple columns.  
 Example:  
 CREATE INDEX idx\_name ON employees (last\_name, first\_name);  
  
3. Unique Index:  
 - A unique index ensures that all values in the indexed column(s) are unique.  
 Example:  
 CREATE UNIQUE INDEX idx\_name ON employees (email);  
  
4. Full-Text Index:  
 - A special type of index designed for text searching.  
 Example:  
 CREATE FULLTEXT INDEX idx\_name ON articles (content);  
  
5. Clustered Index:  
 - A clustered index sorts and stores the data rows based on the index key.  
 - A table can have only one clustered index.  
 Example:  
 CREATE CLUSTERED INDEX idx\_name ON employees (last\_name);

# 4. Exception Handling in SQL Stored Procedure

SQL stored procedures allow exception handling to capture errors and take appropriate actions.  
  
Example of Exception Handling:  
CREATE PROCEDURE handle\_error\_example  
AS  
BEGIN  
 BEGIN TRY  
 -- Generate an error  
 SELECT 1 / 0; -- This will throw a divide-by-zero error  
 END TRY  
 BEGIN CATCH  
 -- Handle error  
 PRINT 'An error occurred: ' + ERROR\_MESSAGE();  
 END CATCH  
END;

# 5. SQL Function to Split Strings into Rows

To split a string into rows, you can use a recursive Common Table Expression (CTE) or a string-splitting function.  
  
Example:  
CREATE FUNCTION SplitString(@inputString VARCHAR(MAX), @delimiter CHAR(1))  
RETURNS @outputTable TABLE (Value VARCHAR(MAX))  
AS  
BEGIN  
 DECLARE @startIndex INT, @endIndex INT;  
 SET @startIndex = 1;  
 SET @endIndex = CHARINDEX(@delimiter, @inputString);  
  
 WHILE @endIndex > 0  
 BEGIN  
 INSERT INTO @outputTable (Value)  
 SELECT SUBSTRING(@inputString, @startIndex, @endIndex - @startIndex);  
  
 SET @startIndex = @endIndex + 1;  
 SET @endIndex = CHARINDEX(@delimiter, @inputString, @startIndex);  
 END  
  
 -- Insert last value after the last delimiter  
 INSERT INTO @outputTable (Value)  
 SELECT SUBSTRING(@inputString, @startIndex, LEN(@inputString) - @startIndex + 1);  
  
 RETURN;  
END;  
  
Example Usage:  
SELECT \* FROM dbo.SplitString('Stephen;peter;berry;Olivier;caroline;', ';');

# 6. Temporary and Variable Tables

- Temporary Table:  
 - Temporary tables exist only during the session in which they are created. They are dropped automatically when the session ends.  
  
Syntax:  
CREATE TABLE #temp\_table (id INT, name VARCHAR(50));  
  
Example:  
CREATE TABLE #temp (id INT, name VARCHAR(50));  
INSERT INTO #temp (id, name) VALUES (1, 'John'), (2, 'Jane');  
SELECT \* FROM #temp;  
DROP TABLE #temp;  
  
- Variable Table:  
 - A variable table is a table variable that is scoped to the batch, stored procedure, or function in which it is declared.  
  
Syntax:  
DECLARE @table\_variable TABLE (id INT, name VARCHAR(50));  
  
Example:  
DECLARE @temp TABLE (id INT, name VARCHAR(50));  
INSERT INTO @temp (id, name) VALUES (1, 'John'), (2, 'Jane');  
SELECT \* FROM @temp;