1. Different Types of Views in SQL:

In SQL, views are virtual tables created using a SELECT statement. They provide a way to save and reuse complex queries and also act as a layer of abstraction.

Simple View: This is a view based on a single table. It can contain a few or all columns of the table.

CREATE VIEW SimpleView AS

SELECT FirstName, LastName FROM Employees;

Complex View: This is a view based on multiple tables, involving joins, aggregations, and other operations.

CREATE VIEW ComplexView AS

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

FROM Employees e JOIN Departments d ON e.DepartmentID = d.ID;

Indexed View: In some databases like SQL Server, you can create indexed views that physically store the result set of the view to improve query performance.

Materialized View: These are views where the result of the query is stored physically, providing performance benefits at the expense of potentially stale data. The data in such views can be refreshed periodically.

2. Function vs Stored Procedure:

Function:

Must return a value.

Can be used in a SELECT statement.

Doesn't support transactions.

Stored Procedure:

Might or might not return a value.

Cannot be used in a SELECT statement.

Supports transactions.

Syntax:

Function:

CREATE FUNCTION FunctionName(@parameter DataType)

RETURNS ReturnType

AS

```
BEGIN
  -- function body
  RETURN value
END;
```

Stored Procedure:

```
CREATE PROCEDURE ProcedureName @parameter DataType
AS
BEGIN
  -- procedure body
END;
```

3. Index in SQL:

An index is a database object used to improve the speed of data retrieval operations.

Types of Indexes:

Clustered Index: Defines the physical order of data in a table. Each table can have only one clustered index.

Non-Clustered Index: Does not determine the physical order of data but creates a logical order for data rows, acting as a pointer to data in the table.

Unique Index: Ensures data in the indexed column is unique.

Full-text Index: Used for full-text search operations.

4. Exception Handling in Stored Procedure:

Using SQL Server as an example:

```
CREATE PROCEDURE SampleProcedure
AS
BEGIN
  BEGIN TRY
    -- SQL Code here
  END TRY
  BEGIN CATCH
    SELECT ERROR_MESSAGE() as ErrorMessage;
```

```
  END CATCH
END;
```

5. SQL Function to Split Strings:

This is for SQL Server:

```
CREATE FUNCTION SplitString(@input NVARCHAR(MAX), @delimiter CHAR(1))
RETURNS @OutputTable TABLE (value NVARCHAR(MAX))
AS
BEGIN
  DECLARE @value NVARCHAR(MAX)
  WHILE CHARINDEX(@delimiter, @input) > 0
  BEGIN
    SELECT @value = LTRIM(RTRIM(SUBSTRING(@input, 1, CHARINDEX(@delimiter, @input)-1))),
        @input = LTRIM(RTRIM(SUBSTRING(@input, CHARINDEX(@delimiter, @input)+1, LEN(@input))))

    INSERT INTO @OutputTable (value) VALUES (@value)
  END
  INSERT INTO @OutputTable (value) VALUES (@input)
  RETURN
END;
```

6. Temporary vs Variable Tables:

Temporary Table: Created in the tempdb and lasts for the duration of the user session. It's prefixed by #

```
CREATE TABLE #TempTable (ID INT, Name NVARCHAR(50))
```

Variable Table: Stored in tempdb and exists only for the duration of the batch in which it's declared. It's declared with @.

```
DECLARE @VarTable TABLE (ID INT, Name NVARCHAR(50))
```

Temporary tables are suitable for larger datasets or multiple operations, while table variables are efficient for smaller datasets or quick operations.