**SQL Assignment 4**

1. Explain different types of views. Demonstrate with suitable examples.

A. n various contexts, "views" can refer to different things. In the realm of databases, "views" typically represent virtual tables that display data from one or more tables. Here are some common types of views along with examples:

Simple Views:

Simple views are based on a single table.

They can involve one or more columns from that table.

Example:

CREATE VIEW EmployeeView AS

SELECT EmpID, EmpName, Department

FROM Employees

WHERE Department = 'IT';

This view selects employee ID, name, and department from the Employees table for employees working in the IT department.

Complex Views:

Complex views involve multiple tables and possibly aggregate functions or joins.

Example:

CREATE VIEW SalesSummary AS

SELECT CustomerID, SUM(TotalAmount) AS TotalSales

FROM Orders

GROUP BY CustomerID;

This view summarizes total sales for each customer by aggregating sales amounts from the Orders table.

Indexed Views:

Indexed views are materialized views with an index.

They improve query performance by storing the result set of a query in the database, which can be indexed for faster retrieval.

Example:

CREATE VIEW SalesByProduct AS

SELECT ProductID, SUM(Quantity) AS TotalSold

FROM OrderDetails

GROUP BY ProductID;

CREATE UNIQUE CLUSTERED INDEX IX\_SalesByProduct\_ProductID

ON SalesByProduct(ProductID); This view calculates total quantities sold for each product and creates an index on the product ID for faster access.

Updatable Views:

Updatable views allow modification of the data through the view, which propagates changes to the underlying tables.

They have certain restrictions on which modifications are allowed.

Example:

CREATE VIEW HighValueProducts AS

SELECT ProductID, ProductName, UnitPrice

FROM Products

WHERE UnitPrice > 100;

CREATE VIEW HighValueProducts AS

SELECT ProductID, ProductName, UnitPrice

FROM Products

WHERE UnitPrice > 100;

1. This view selects products with a unit price higher than $100, and any updates to the selected products' details would reflect back in the Products table.
2. Partitioned Views:
3. Partitioned views divide data horizontally across multiple tables, typically for easier management or performance reasons.
4. Each partition may represent a subset of the data based on a certain criterion.
5. Example:
   * This view calculates total quantities sold for each product and creates an index on the product ID for faster access.
6. **Updatable Views**:
   * Updatable views allow modification of the data through the view, which propagates changes to the underlying tables.
   * They have certain restrictions on which modifications are allowed.
   * Example:

CREATE VIEW SalesPartition AS

SELECT \*

FROM Sales2019

UNION ALL

SELECT \*

FROM Sales2020;

This view combines sales data from two separate tables, Sales2019 and Sales2020, into a single virtual table for easier analysis.

These examples illustrate how views can be tailored to various requirements, from simple data selection to complex aggregations and partitioning, providing a flexible and efficient way to work with database information.

* + This view combines sales data from two separate tables, Sales2019 and Sales2020, into a single virtual table for easier analysis.

These examples illustrate how views can be tailored to various requirements, from simple data selection to complex aggregations and partitioning, providing a flexible and efficient way to work with database information.

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

A. In database management systems, functions and stored procedures are both units of reusable code, but they have some key differences in terms of their functionality and usage.

Functions:

Return Value: Functions return a single value. This value can be a scalar value or a table.

Usage: Functions are typically used to compute and return specific values based on input parameters.

Transaction Control: Functions do not support transaction control like COMMIT or ROLLBACK.

Scope: Functions can be used in SQL queries, WHERE clauses, SELECT lists, and other expressions.

Syntax for creating a function (in SQL Server):

CREATE FUNCTION function\_name

(

-- Parameters (if any)

@param1 datatype,

@param2 datatype

)

RETURNS return\_datatype

AS

BEGIN

-- Function body

-- SQL statements here

RETURN return\_value;

END;

CREATE FUNCTION GetTotalPrice

(

@quantity INT,

@unit\_price DECIMAL(10,2)

)

RETURNS DECIMAL(10,2)

AS

BEGIN

DECLARE @total\_price DECIMAL(10,2);

SET @total\_price = @quantity \* @unit\_price;

RETURN @total\_price;

END; Stored Procedures:

Return Value: Stored procedures do not necessarily return values. They can have output parameters, but they don't return a value like functions.

Usage: Stored procedures are used for executing a series of commands or queries. They can perform tasks like inserting, updating, deleting data, or performing complex operations.

Transaction Control: Stored procedures support transaction control commands like COMMIT or ROLLBACK.

Scope: Stored procedures can execute SQL queries, perform operations, and execute other stored procedures.

Syntax for creating a stored procedure (in SQL Server):

**Stored Procedures:**

1. **Return Value:** Stored procedures do not necessarily return values. They can have output parameters, but they don't return a value like functions.
2. **Usage:** Stored procedures are used for executing a series of commands or queries. They can perform tasks like inserting, updating, deleting data, or performing complex operations.
3. **Transaction Control:** Stored procedures support transaction control commands like COMMIT or ROLLBACK.
4. **Scope:** Stored procedures can execute SQL queries, perform operations, and execute other stored procedures.

**Syntax for creating a stored procedure (in SQL Server):**

CREATE PROCEDURE procedure\_name

(

-- Parameters (if any)

@param1 datatype,

@param2 datatype

)

AS

BEGIN

-- Procedure body

-- SQL statements here

END;

CREATE PROCEDURE UpdateProductPrice

(

@product\_id INT,

@new\_price DECIMAL(10,2)

)

AS

BEGIN

UPDATE Products

SET Price = @new\_price

WHERE ProductID = @product\_id;

END;

functions are primarily used for computations and return a value, while stored procedures are used for executing a series of commands or operations. Each has its own use cases and advantages depending on the requirements of the application.

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

A.

In SQL, an index is a database object that is created on one or more columns of a table to improve the speed of data retrieval operations. It works much like the index in a book, allowing the database to quickly locate specific rows within a table without having to scan through the entire table.

There are several types of indexes in SQL, including:

Single-Column Index: This type of index is created on a single column of a table. It speeds up queries that filter or sort data based on the indexed column.

Composite Index: Also known as a multi-column index, this type of index is created on multiple columns of a table. It is useful when queries involve filtering or sorting based on multiple columns.

Unique Index: A unique index ensures that all values in the indexed column(s) are unique, meaning no two rows can have the same value(s) in the indexed column(s). It enforces uniqueness and also provides fast access to data.

Clustered Index: In SQL Server, a clustered index determines the physical order of data in a table. Unlike other indexes, a table can have only one clustered index because the data rows themselves are stored in the order of the clustered index key. This can improve the performance of queries that use range predicates or retrieve ranges of data.

Non-Clustered Index: A non-clustered index does not affect the physical order of the table's data rows. Instead, it creates a separate structure containing the indexed column(s) and pointers to the actual rows. Non-clustered indexes are useful for improving the performance of queries that filter, sort, or join data.

Full-Text Index: This type of index is used for searching text data stored in character columns (typically large amounts of text) using full-text search capabilities. It allows for efficient searching of words and phrases within the text data.

Each type of index has its own advantages and use cases, and the choice of index type depends on the specific requirements of the database and the queries it needs to support.

Top of Form

1. Showcase an example of exception handling in SQL stored procedure.
2. Exception handling in SQL stored procedures is typically done using the **TRY...CATCH** block. Here's an example of how you can use exception handling in a SQL Server stored procedure:

CREATE PROCEDURE ExampleProcedure

AS

BEGIN

SET NOCOUNT ON;

BEGIN TRY

-- Your SQL statements that may cause an error

SELECT 1/0; -- This will cause a divide by zero error

END TRY

BEGIN CATCH

-- Error handling code

PRINT 'An error occurred: ' + ERROR\_MESSAGE();

-- Additional error handling logic can be added here, such as logging the error

END CATCH;

END;

In this example:

The TRY block contains the SQL statements that may cause an error.

If an error occurs within the TRY block, the control is transferred to the CATCH block.

Inside the CATCH block, you can handle the error as needed. The ERROR\_MESSAGE() function retrieves the error message text.

You can replace the SELECT 1/0; statement with any SQL operation that may cause an error. The CATCH block will handle the error and execute the specified error handling logic.

1. 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 |

A.You can create a SQL function to split strings into rows on a given character using the following steps. However, please note that the syntax may vary slightly depending on the SQL dialect you're using (e.g., MySQL, PostgreSQL, SQL Server, etc.). Here's a generic example:

CREATE FUNCTION dbo.SplitStringToRows (@inputString VARCHAR(MAX), @delimiter CHAR(1))

RETURNS @outputTable TABLE (SplitValue VARCHAR(MAX))

AS

BEGIN

DECLARE @startIndex INT = 1

DECLARE @endIndex INT

DECLARE @substring VARCHAR(MAX)

WHILE CHARINDEX(@delimiter, @inputString, @startIndex) > 0

BEGIN

SET @endIndex = CHARINDEX(@delimiter, @inputString, @startIndex)

SET @substring = SUBSTRING(@inputString, @startIndex, @endIndex - @startIndex)

INSERT INTO @outputTable (SplitValue) VALUES (@substring)

SET @startIndex = @endIndex + 1

END

SET @substring = SUBSTRING(@inputString, @startIndex, LEN(@inputString) - @startIndex + 1)

INSERT INTO @outputTable (SplitValue) VALUES (@substring)

RETURN

END

SELECT SplitValue

FROM dbo.SplitStringToRows('Stephen;peter;berry;Olivier;caroline;', ';')

SplitValue

-----------

Stephen

peter

berry

Olivier

caroline

sql

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

**A** **Temporary tables and table variables are both used in SQL to store temporary data within the scope of a session or a batch of statements. They are similar in purpose but have some differences in usage and behavior.**

**Temporary Tables:**

**Temporary tables are stored in the tempdb database.**

**They are visible only within the session that creates them, and they are automatically dropped when the session ends.**

**They can be accessed by any user with the necessary permissions.**

**Temporary tables can have indexes and statistics.**

**the syntax to create a temporary table:**

**CREATE TABLE #TempTableName (**

**Column1 DataType,**

**Column2 DataType,**

**...**

**)**

**CREATE TABLE #TempEmployees (**

**EmployeeID INT,**

**FirstName VARCHAR(50),**

**LastName VARCHAR(50)**

**)**

**Table Variables:**

**Table variables are similar to temporary tables but are declared and used within the batch or stored procedure in which they are defined.**

**They are automatically deallocated when the batch or stored procedure finishes execution.**

**They cannot have indexes or statistics.**

**They have limited scope, which is the batch or stored procedure where they are declared.**

**Here's the syntax to declare a table variable:**

**DECLARE @TableVariableName TABLE (**

**Column1 DataType,**

**Column2 DataType,**

**...**

**)**

**DECLARE @Employees TABLE (**

**EmployeeID INT,**

**FirstName VARCHAR(50),**

**LastName VARCHAR(50)**

**)**

temporary tables are stored in the tempdb database, have broader scope, and can have indexes and statistics. Table variables are scoped to the batch or stored procedure, cannot have indexes or statistics, and are deallocated automatically when the scope ends.

Top of Form