Stored Procedures in SQL

A stored procedure is a set of SQL statements that are precompiled and stored in the database. Stored procedures are used to encapsulate a sequence of SQL statements that can be executed as a single unit. They can accept input parameters and return output parameters or result sets.

Benefits of Stored Procedures:

- Improved Performance: Stored procedures are precompiled, which means they execute faster than individual SQL statements.
- Code Reusability: Stored procedures can be called from multiple applications, reducing code duplication.
- Data Security: Stored procedures can be granted different permissions than individual users, allowing better control over data access.
- Code Modularity: Complex operations can be broken down into multiple stored procedures, improving code organization and maintainability.

Types of Stored Procedures

1. Simple Stored Procedures

These are basic stored procedures that perform a specific task, such as inserting, updating, or deleting data from a table.

Example (SQL Server):

```
sql
CREATE PROCEDURE InsertEmployee
@FirstName VARCHAR(50),
@LastName VARCHAR(50),
@Email VARCHAR(50)
AS
BEGIN
  INSERT INTO Employees (FirstName, LastName, Email)
  VALUES (@FirstName, @LastName, @Email)
END
```

2. Stored Procedures with Output Parameters

These stored procedures can return values to the calling application using output parameters.

Example (SQL Server):

```
CREATE PROCEDURE GetEmployeeCount
@EmployeeCount INT OUTPUT
AS
BEGIN
  SELECT @EmployeeCount = COUNT() FROM Employees
END
```

3. Stored Procedures with Result Sets

These stored procedures can return a result set to the calling application, similar to a SELECT query.

Example (SQL Server):

```
CREATE PROCEDURE GetEmployeesByDepartment
@DepartmentId INT
AS
BEGIN
  SELECT FirstName, LastName, Email
  FROM Employees
  WHERE DepartmentId = @DepartmentId
END
```

4. Nested Stored Procedures

These stored procedures can call other stored procedures within their execution, allowing for modular code and better code organization.

Example (SQL Server):

```
CREATE PROCEDURE ProcessOrder
@OrderId INT
AS
BEGIN
  Validate order
 EXEC ValidateOrder @OrderId

  Process payment
 EXEC ProcessPayment @OrderId

  Ship order
 EXEC ShipOrder @OrderId
END
```


Stored procedures can also be classified based on their execution scope, such as system stored procedures (builtin procedures provided by the database system) and userdefined stored procedures (created by developers).

Stored procedures can be broadly classified into two types based on their origin:

1. User-Defined Stored Procedures

These are custom stored procedures created by developers or database administrators to encapsulate specific business logic or operations within the database. User-defined stored procedures allow for code reusability, modularity, and improved performance by pre-compiling the SQL statements.

Example (SQL Server):

```
CREATE PROCEDURE InsertEmployee
@FirstName VARCHAR(50),
@LastName VARCHAR(50),
```

```
  @Email VARCHAR(50)
  AS
  BEGIN
    INSERT INTO Employees (FirstName, LastName, Email)
    VALUES (@FirstName, @LastName, @Email)
  END
```

2. System Stored Procedures

These are pre-defined stored procedures provided by the database management system (DBMS) itself. System stored procedures are typically used for administrative tasks, such as managing databases, tables, users, and permissions. They are built into the DBMS and cannot be modified by users.

Example (SQL Server):

```
EXEC sp_help 'Employees' -- Get information about the 'Employees' table
```

In this example, sp_help is a system stored procedure in SQL Server that provides information about a specified table or other database object.

While user-defined stored procedures are created by developers to encapsulate custom business logic, system stored procedures are provided by the DBMS vendor and are mainly used for administrative and maintenance tasks within the database system.

Both types of stored procedures offer benefits in terms of code reusability, performance optimization, and security, but they serve different purposes and have different scopes of access and modification.

When creating stored procedures, it's important to follow best practices, such as using proper error handling, avoiding SQL injection vulnerabilities, and documenting the procedure's purpose, inputs, and outputs.

ENTRI
elevate