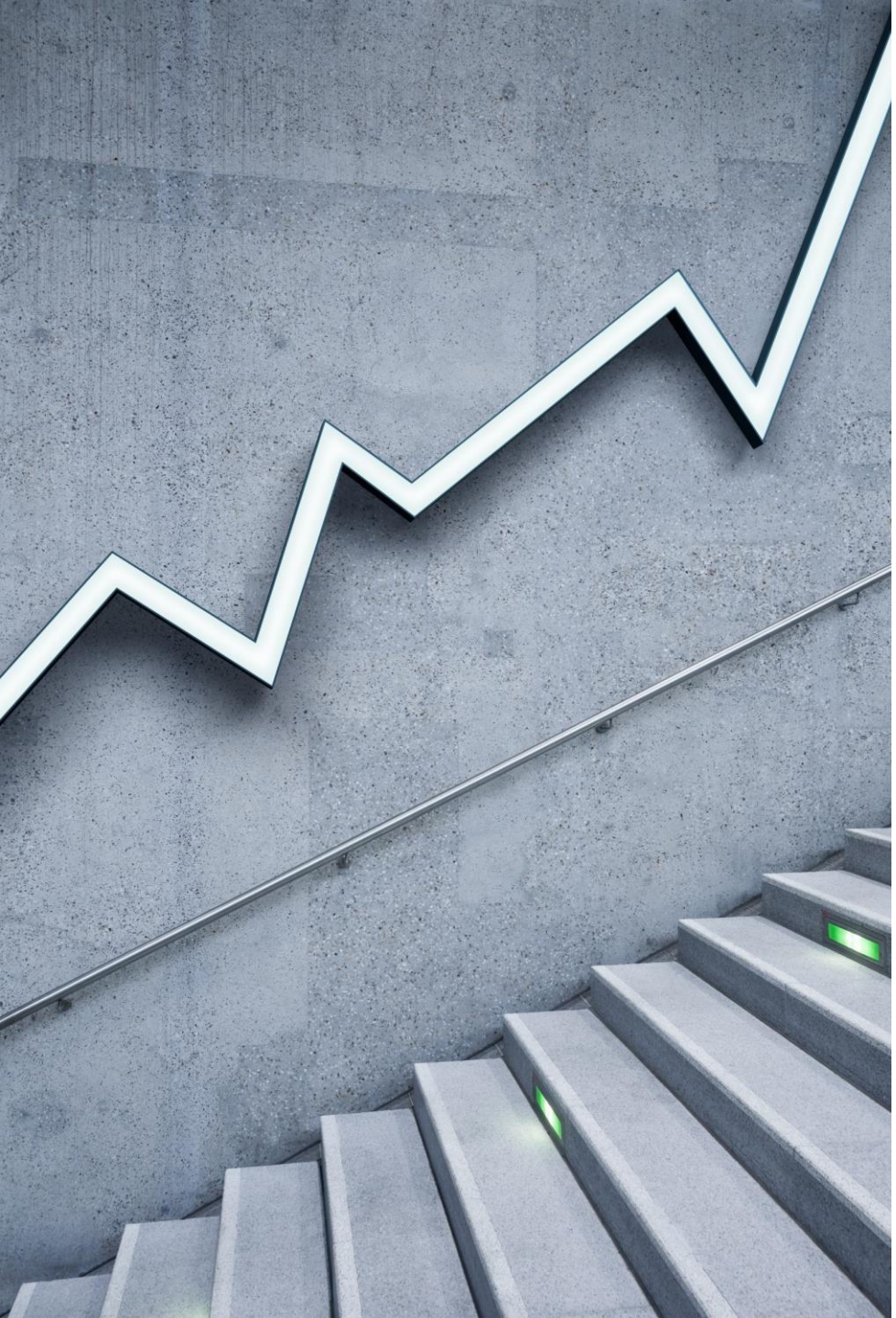

Stored Procedures

.NET

SQL Server creates an execution plan for a Stored Procedure and stores it in the cache. In following executions, the plan is reused so that the stored procedure can execute fast with reliable performance.

HTTPS://WWW.SQLSERVERTUTORIAL.NET/SQL-SERVER-STORED-PROCEDURES/

Function vs. Stored Procedure

https://www.c-sharpcorner.com/UploadFile/996353/difference-between-stored-procedure-and-user-defined-function/

Differences between Stored Procedure and User Defined Function in SQL Server

User Defined Function	Stored Procedure
Function must return a value.	Stored Procedure may or not return values.
Will allow only Select statements, it will not allow us to use DML statements.	Can have select statements as well as DML statements such as insert, update, delete and so on
It will allow only input parameters, doesn't support output parameters.	It can have both input and output parameters.
It will not allow us to use try-catch blocks.	For exception handling we can use try catch blocks.
Transactions are not allowed within functions.	Can use transactions within Stored Procedures.
We can use only table variables, it will not allow using temporary tables.	Can use both table variables as well as temporary table in it.
Stored Procedures can't be called from a function.	Stored Procedures can call functions.
Functions can be called from a select statement.	Procedures can't be called from Select/Where/Having and so on statements. Execute/Exec statement can be used to call/execute Stored Procedure.
A UDF can be used in join clause as a result set.	Procedures can't be used in Join clause

Stored Procedures – Overview

https://www.w3schools.com/sql/sql_stored_procedures.asp
https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver15

A ***Stored Procedure*** is a SQL query that you have saved as a ***Function*** so it can be reused. The execution plan is saved so the query runs faster than if it were written new each time it was used.

A commonly used SQL query can be saved as a ***Stored Procedure***, and then called when needed. ***Stored procedures*** can have parameters and act based on the parameter values passed.

SSMS has some built-in ***Stored Procedures***.

1. Use keywords 'CREATE PROCEDURE'
2. Give the procedure a name
3. Use the keyword 'AS'
4. Write the query

```
CREATE PROCEDURE
procedure_name
AS
sql_statement
```

Stored Procedures – Overview

https://www.w3schools.com/sql/sql_stored_procedures.asp
https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver15

Stored Procedures:

- Accept INPUT parameters
- Can return multiple values in the form of OUTPUT parameters.
- Contain programming statements that perform operations on the database, include calling other procedures.
- Return a status value to a calling program to indicate success, failure, and the reason for failure.
- Allow easy reuse and code maintenance
- Improve performance
- Reduce network traffic.

Stored Procedures – Types

https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver15#types-of-stored-procedures

---

There are four types of Stored Procedures

Type	Description
User-Defined	Created in a user-defined database or in any system database except the Resource database. Can be developed in SQL or as a reference to a method.
Temporary	Stored in **tempdb**. There are two types: ***local*** and ***global***. ***Local*** temp procedures have # as the first character of their names, are visible only to the current user connection, and are deleted when the connection is closed. ***Global*** temporary procedures have ## as the first two characters of their names, are visible to any user after they are created, and are deleted at the end of the last session using the procedure.
System	Included with SQL Server. They are physically stored in the internal, hidden Resource database and logically appear in the ***sys*** schema of every system- and user-defined database.
Extended User-Defined	Extended procedures enable creating external routines in a programming language such as C. These procedures are DLLs that an instance of SQL Server can dynamically load and run.

Stored Procedure – Create, Execute, Delete

https://www.sqlservertutorial.net/sql-server-stored-procedures/basic-sql-server-stored-procedures/
https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/create-a-stored-procedure?view=sql-server-ver15

This ***Stored Procedure*** is called GetAllCustomerNames.

It will produce a table with two columns, CustomerId and FirstName, from the Customers table, where the addressId (FK) is 1.

Execute with EXEC.

Drop with DROP PROCEDURE.

```
--Simple Stored Procedure
CREATE PROCEDURE GetAllCustomerNames
AS
SELECT CustomerId, FirstName
FROM Customers
WHERE addressID = 1;

EXEC GetAllCustomerNames;
DROP PROCEDURE GetAllCustomerNames;
```

SQL – Scalar Variables

Sometimes we want to store intermediate values.

- We can split queries into several parts.
- SQL Server supports ***scalar*** variables and ***table-valued*** variables.
- They only exist for the duration of that "batch" of commands

```
DECLARE @maxid INT;
SELECT @maxid = MAX(TypeId) FROM Poke.Type;
SET @maxid = (SELECT MAX(TypeId) FROM Poke.Type); -- another way to set variable
INSERT INTO Poke.Type (TypeId, Name) VALUES (@maxid + 1, 'Earth');
```

Declaring Scalar Variables

https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/specify-parameters?view=sql-server-ver15#passing-values-into-parameters

To declare a variable in SQL:

- Declare the variable by using the DECLARE keyword.
- Follow DECLARE with @[variableName] [datatype].
- Use the Keyword SET.
- Use @[variableName] = [value]
- The variable can be passes as an argument into a ***Procedure***.

```
DECLARE @ProductID int,
        @CheckDate datetime;
SET @ProductID = 819;
SET @CheckDate = '20050225';
EXEC dbo.uspGetProductID @ProductID, @CheckDate;
```

Stored Procedure with Parameters

https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/specify-parameters?view=sql-server-ver15

The parameter values supplied with a ***Procedure*** call must be constants or a variable. A ***Function*** name cannot be used as a parameter value.

When creating a ***Procedure*** and declaring a parameter name, the parameter name must begin with a single @ and must be unique in the scope of the procedure.

Parameters must also be defined with a data type when they are declared in a CREATE PROCEDURE statement.

```
--Stored Procedure with variables
CREATE PROCEDURE GetCustomerShe
    @LastName nvarchar(50),
    @FirstName nvarchar(50)
AS
    SELECT FirstName, LastName, Remarks
    FROM Customers
    WHERE FirstName = @FirstName
    AND
    LastName = @LastName
    AND
    Remarks LIKE 'S_e%';

EXEC GetCustomerShe
    @LastName = 'Moore',
    @FirstName = 'Maya';
DROP PROCEDURE GetCustomerShe;
```

Table-Valued Parameter

https://docs.microsoft.com/en-us/sql/relational-databases/tables/use-table-valued-parameters-database-engine?view=sql-server-ver15

Table-valued parameters are declared by using user-defined table types. You can use ***table-valued parameters*** to send multiple rows of data to a SQL statement or a routine, such as a ***Stored Procedure*** or ***Function***, without creating a temporary table or many parameters.

A ***Table-valued Parameter*** is scoped to the ***Stored Procedure*** or ***Function***, exactly like other parameters.

Table-valued Parameters must be passed as input READONLY parameters to routines. You cannot perform DML operations such as UPDATE, DELETE, or INSERT on a ***Table-valued Parameter*** in the body of a routine.

```
/* Create a table type. */
CREATE TYPE LocationTableType AS TABLE
    ( LocationName VARCHAR(50), CostRate INT );
/*Create a procedure to receive data for table-valued parameter.*/
CREATE PROCEDURE dbo.usp_InsertProductionLocation
  @TVP LocationTableType READONLY
    AS
    SET NOCOUNT ON
    INSERT INTO AdventureWorks2012.Production.Location
      ( Name, CostRate, Availability, ModifiedDate )
    SELECT *, 0, GETDATE()
    FROM @TVP;
/* Declare a variable that references the type. */
DECLARE @LocationTVP AS LocationTableType;
/* Add data to the table variable. */
INSERT INTO @LocationTVP (LocationName, CostRate)
  SELECT Name, 0.00
  FROM AdventureWorks2012.Person.StateProvince;

/* Pass the table variable data to a stored procedure. */
EXEC usp_InsertProductionLocation @LocationTVP;
```

Stored Procedure – OUT Parameter

https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/specify-parameters?view=sql-server-ver15#specifying-parameter-direction

The direction of a parameter is either INPUT (default) or OUTPUT, meaning the procedure returns a value to the calling program.

To specify an OUTPUT parameter, the OUTPUT keyword must be specified in the definition of the parameter in the CREATE PROCEDURE statement.

The procedure returns the current value of the OUTPUT parameter to the calling program when the procedure exits.

The calling program must also use the OUTPUT keyword when executing the procedure to save the parameter's value in a variable that can be used in the calling program.

```
CREATE PROCEDURE HowManyOrdersByCustomer (
   @CustomerNumber INT, --customerID
   @OrderCount INT OUTPUT
) AS
BEGIN
   SELECT
      CustomerID AS CustomersID
   FROM Orders
   WHERE CustomerID = @CustomerNumber;
-- @@ROWCOUNT is a system variable that returns
-- the number of rows read by the previous statement.
   SELECT @OrderCount = @@ROWCOUNT;
END;

DECLARE @HowMany INT; -- declare the OUTPUT variable
-- call the method with the 2 parameters
EXEC HowManyOrdersByCustomer 1, @HowMany OUTPUT;
SELECT @HowMany AS 'Ten times the total';
```

Procedure with Try/Catch

```
GO
CREATE OR ALTER PROCEDURE Poke.UpdateAllDateModified(@param INT, @rowschanged INT OUTPUT)
AS
BEGIN
    BEGIN TRY
    IF (NOT EXISTS (SELECT * FROM Poke.Pokemon))
    BEGIN
        RAISERROR ('No data found in table', 15, 1);
    END
    SET @rowschanged = (SELECT COUNT(*) FROM Poke.Pokemon);
    UPDATE Poke.Pokemon SET DateModified = GETDATE();
    END TRY
    BEGIN CATCH
        PRINT 'Error'
    END CATCH
END
GO
```

This procedure updates all the DateModified values to the current time and returns the number of rows modified.

```
DECLARE @result INT;
EXECUTE Poke.UpdateAllDateModified 123, @result OUTPUT;
SELECT @result;
```