Lesson 7

**Implementing Stored procedure and Functions**

**Creating Batches**

A Batch is a group of statements submitted together to the SQL Server for execution. While executing batches, the SQL server compiles the statements of batch into a single executable unit called an execution plan. This helps in saving execution time.

Ex:

If you are having 10 statements to execute one by one requests. It takes time to finish the requests. Instead if we execute all the 10 in a batch. Then the execution process becomes faster as all the statements are sent to the server together.

**Guidelines to Create Batches**

1. you cannot combine statements, such as CREATE DEFAULT, CREATE FUNCTION,

CREATE PROCEDURE, CREATE RULE, CREATE TRIGGER and CREATE VIEW with the other statements while creating a batch.

2. You can use the execute statement in a batch when it is not the first statement of the batch. Otherwise the execute statements works implicity.

3. you cannot bind rules and defaults to columns and use them in the same batch.

4. You cannot define and use the CHECK constraint in the same batch.

5. You cannot drop objects and recreate them in the same batch.

6. You cannot alter a table by adding a column and then refer to the new columns in the batch earlier.

drop table party;

create table party (pid int, pname char(15), padd varchar(20), ploc char(25), sale money);

insert party values (001, 'rama', '100feet','cbe', 50000 );

select \* from Party;

**Note :** the GO keyword is used to send all the statements together to the SQL Server.

insert party values (002, 'ragu', '321gandhipuram','madurai', 10000 );

insert party values (003, 'sam', '123rspuram','chennai', 25000 );

select \* from Party;

go

Using Variables

While creating batches , We might save some values temporarily during the execution time.Ex: we might need to store some intermediate values of calculations.

Note : DECLARE ( you can declare a variable by using the DECLARE statement)

**EG** :

Declare @amt float

select @amt = max(sale) from party

print @amt

go

declare@amtint

set@amt=10900

print@amt

declare@amtint

select@amt=9000

print@amt

declare@amtint

select@amt=min(salary)fromemployees

if(@amt>70000)

begin

select\*fromEmployeeswheresalary=@amt

print'Amout is graeter than 70000'+Convert(varchar,@amt)

print@amt

end

else

begin

select\*fromEmployeeswheresalary=@amt

print@amt

print'Amount is not greater than 70000'

End

--concatinating string(varchar) and integer value while print

declare@amtint

set@amt=1233

print'Amount '+Convert(varchar(50),@amt)+' as Principal'

**Displaying User-Defined Messages**

declare @vin varchar(10)

select @vin='radha'

select @vin

o/p >>

no column name

Radha

declare@vinvarchar(10)

set@vin='Paul'

print@vin

go

declare @vin11 varchar(10)

select @vin11='shafeek'

print @vin11

o/p >>

Shafeek

**IF-ELSE STATEMENT**

drop table employ

create table employ (empid int, empname char(15),empadd varchar(20), empcity char(25))

insert employ values (001, 'rama', '123abc','coimbatore')

insert employ values (002, 'radha', '789cba', 'salem')

insert employ values (003, 'balaji', '321gandhipuram','madurai')

insert employ values (004, 'moideen', '123rspuram','salem')

insert employ values (005, 'alagiri', '369fort','chennai')

insert employ values (006, 'krishna', '456fort','madurai')

select \* from employ

**EX** :

if exists (select \* from employ where empcity='chennai')

print 'balaji'

else

print 'TryOnce'

o/p>>

balaji

**Eg:**

if exists (select \* from employ where empcity='Kovai')

print 'Success'

else

print 'TryOnce'

o/p>>

TryOnce

**Eg:**

if exists (select \* from employ where empcity='Chennai')

begin

print 'Successfully'

print 'Completed'

end

else

print 'TryOnce'

o/p>>

Successfully

Completed

**CASE STATEMENT**

Eg:

declare @vin varchar(10)

declare @vish varchar(10)

select @vish = 'O'

select @vin =

case @vish

when 'O' then 'ONE'

when 'T' then 'TWO'

when 'TH' then 'THREE'

else 'TryOnce'

end

print @vin

xO/P >>

ONE

declare@xint,@resvarchar(200)

select@x=min(salary)fromEMPLOYEE

select@res=

case@x

when21000then'Salary is'+convert(varchar,(@res=@x+500))

when23000then'Salary is 23000'

when25000then'Salary is 25000'

else'Data is not found'

end

print@res

**WHILE Statement**

**Eg** :

declare @vwhile int

select @vwhile=1

while @vwhile<=10

begin

print @vwhile

select @vwhile=@vwhile+1

end

declare@vcountint

select@vcount=11

while@vcount>(15/3)

begin

select@vcount=@vcount-1

if@vcount=4

break

if@vcount=6

continue

print@vcount

end

createprocedureretrive\_Empinfoas

begin

select\*fromEmployees

end

execretrive\_Empinfo

deletefromemployeeswhereEmpid=121

dropprocedureinsert\_empinfo

createprocedureinsert\_empinfoas

insertintoEmployees (Empid,Fname,LName,Salary)values(121,'gg','yy',3424)

execretrive\_Empinfo

execinsert\_empinfo

createprocedureinsert\_empinfo(@eidint,@fnvarchar(20),@Lnvarchar(20),@salmoney)as

insertintoEmployees (Empid,Fname,LName,Salary)values(@eid,@fn,@ln,@sal)

execinsert\_empinfo 123,Null,'bbb',34556

execretrive\_Empinfo

--pass default value for parameter

createprocedureupdate\_empinfo (@enamevarchar(20),@salint=100)as

begin

updateEmployeessetsalary=@salwhereFname=@ename

End

execretrive\_Empinfo

execupdate\_empinfo 'steven'

execupdate\_empinfo 'Neena',23000

alterprocedureRetrive\_empinfoas

selectEmpid,Fname,ManagerIdfromEmployees

Stored Procedure with Output Parameter:

alter procedure sp\_getAllEmployees (@rowcount int output)

as

begin

select \* from Employee

select @rowcount=@@rowcount

end

declare @rowcount int

exec sp\_getAllEmployees @rowcount output

select @rowcount 'Employees '

**Uisng TRY-CATCH**

/\*

begin try

insert employ values (003, 'balaji', '321gandhipuram','madurai')

insert employ values (w004, 'ramesh', '123rspuram','salem')

end try

begin catch

select 'there was an error!' + error\_message() as errormessage,

ERROR\_LINE() as errorline,

ERROR\_NUMBER() as errornumber,

ERROR\_PROCEDURE() as errorprocedure,

ERROR\_SEVERITY() as errorseverity,

ERROR\_STATE() as errorstate

end catch

go

\*/

VARIABLES OF SQL SERVER:(GLOBAL)

select 'The Name Of The Server: ' + @@version

select @@spid

select @@procid

select @@error

select @@rowcount

select @@connections

select @@trancount

select @@max\_connections

select @@total\_errors

**STORED PROCEDURE**

Batches are temporary in nature. To execute a batch more than once , you need to recreate SQL statements and submit them to the server.

Therefore, if you need to execute a batch multiple times, you can save it within a stored procedure.A Stored Procedure is a precompiled object stored in the database stored procedure can invoke the Data Definition Language(DDL) and Data Manipulation Language (DML) statements and can return the values.

if we need to assign values to the variables declared in the procedures at the run time, we can pass parameters while executing them.

we can also execute a procedure from another procedure.

Stored procedures involves these steps

**Create Stored Procedures**

drop table authors

create table authors(au\_id int, au\_name varchar(20))

insert authors values('101', 'Balaguruswamy')

select \* from authors

-

create procedure sha as

begin

select \* from authors

end

**Executing a Stored Procedure**

exec sha

**Altering a Stored procedure**

alter procedure sha

as

begin

select au\_name from authors

end

exec sha

**Dropping a Stored Procedure**

drop procedure sha

**Creating Parameterized Stored Procedure**

**PASSING PARAMETERS**

drop table emp

create table emp (empid int,empname char(20) )

insert emp values(001,'sha')

insert emp values(002,'ram')

insert emp values(003,'sam')

select \* from emp

Eg:

create procedure pass @p int as

begin

select \* from emp where empid=@p

end

exec pass 2

drop procedure pass

Eg:

alter procedure pass @p int=0

as

begin

if @p = 0

begin

print 'Give empid with this execute statement '

return

end

print 'List of Employees'

select \* from emp where empid=@p

end

execute pass 1

drop table authors

drop table authors

drop table emp

drop table dept

create table authors(au\_id int, aname varchar(20))

insert authors values('10001', 'Balaguruswamy')

select \* from authors

create table emp(eid int, ename varchar(20))

insert emp values('101', 'sha')

select \* from emp

create table dept(did int, dname varchar(20))

insert dept values('501', 'it')

select \* from dept

create procedure abc as

begin

select \* from authors

select \* from emp

select \* from dept

end

exec abc

alter procedure abc as

begin

select \* from emp

select \* from dept

end

--inserting values into the table using stored procedure

createprocedureinsert\_emp (@eidint,@enamevarchar(20))as

begin

insertintoempvalues(@eid,@ename)

end

execinsert\_emp 4,'Preethi'

-- update the values into the table using stored procedure

createprocedureupdate\_emp (@eidint,@enamevarchar(20))as

begin

updateempsetempname=@enamewhereempid=@eid

end

select\*fromemp

execupdate\_emp 4,'pranav'

-- delete the values into the table using stored procedure

createproceduredelete\_emp (@eidint,@enamevarchar(20))as

begin

deletefromempwhereempid=@eidorempname=@ename

end

select\*fromemp

execdelete\_emp 4,'ram'

**One stored procedure can call other stored procedure :**

Example :

droptableUserlogin

createtableUserLogin (logidintidentity(1,1),UserNamevarchar(20),Passwordvarchar(30))

insertintoUserLogin(Username,Password)values('System','Welcome')

select\*fromUserLogin

dropprocedureLoginId

createprocedureLoginId (@usrnamevarchar(20),@pwdvarchar(30))

as

begin

declare@lgidint

select@lgid=logidfromUserLoginwhereUserName=@usrnameandPassword=@pwd

return@lgid

End

select\*fromemprecord

createtableemprecord (Usernamevarchar(20),passwordvarchar(20),

Emailvarchar(40),

IsAdminbit,

EmpNamevarchar(30),

EmpAddressvarchar(200),

Contactnovarchar(10),

Companynamevarchar(max))

createprocedureinsert\_Emprecord

@usrnamevarchar(30),

@pwdvarchar(30),

@mailvarchar(30),

@adminbit,

@enamevarchar(30),

@eaddressvarchar(maX),

@novarchar(10),

@cnamevarchar(max)

as

begin

insertintoUserLogin (UserName,password)values(@usrname,@pwd)

declare@emploginidint

exec@emploginid=LoginId @usrname,@pwd

insertintoemprecordvalues(@emploginid,@pwd,@mail,@admin,@ename,@eaddress,@no,@cname)

end

execinsert\_Emprecord 123,'querty','gem@gmail.com',1,'geetha','chennai',

23424,Triumphsys

select\*fromemprecord

select\*fromUserLogin

altertable

**Difference between stored procedure and Function:**

* Procedure can return zero or n values whereas function can return one value which is mandatory.
* Procedures can have input/output parameters for it whereas functions can have only input parameters.
* Procedure allows select as well as DML statement in it whereas function allows only select statement in it.
* Functions can be called from procedure whereas procedures cannot be called from function.
* Exception can be handled by try-catch block in a procedure whereas try-catch block cannot be used in a function.
* We can go for transaction management in procedure whereas we can't go in function.
* Procedures can not be utilized in a select statement whereas function can be embedded in a select statement.
* UDF can be used in the SQL statements anywhere in the WHERE/HAVING/SELECT section where as Stored procedures cannot be.
* UDFs that return tables can be treated as another rowset. This can be used in JOINs with other tables.
* Inline UDF's can be though of as views that take parameters and can be used in JOINs and other Rowset operations.

**IMPLEMENTING FUNCTIONS**

Similar to the stored procedure, we can also create functions to store a set of T-SQL Statements permanently. These functions are also referred to as a user defined functions (UDF) .

It accepts parameters, perform an action, and returns the result of that action as a value. It can be either be a single scalar value or a result set.

drop table emp

create table emp (empid int,empname char(20), city varchar(30))

insert emp values(001,'sha','cbe');

insert emp values(002,'ram','salem');

insert emp values(003,'sam','trichy');

select \* from emp;

**simple User Defined Function**

create function ft2

(@sal1 float, @sal2 float)

returns float

as begin

return(@sal1+@sal2)

end

select dbo.ft2(200,300) as sumoftwovalues

**Scalar Functions**

The user-defined scalar function also returns a single value as a result of actions performed by the function. We return any datatype value from a function.

CREATE FUNCTION ft3

( @Val1 int, @Val2 int )

RETURNS int

AS

BEGIN

WHILE @Val1 <100

BEGIN

SET @Val1 =@Val1 +1

END

RETURN (@Val1+@Val2)

END

go

SELECT dbo.ft3(1,7) AS SumOfTwoValues

Example :

CREATE TABLE Employee

(

EmpID int PRIMARY KEY,

FirstName varchar(50) NULL,

LastName varchar(50) NULL,

Salary int NULL,

Address varchar(100) NULL,

)

Insert into Employee Values(1,'Mohan','Chauahn',22000,’Chennai’);

Insert into Employee Values(2,'Asif','Khan',15000,'Delhi');

Insert into Employee Values(3,'Bhuvnesh','Shakya',19000,’hyderabad’);

Insert into Employee Values(4,'Deepak','Kumar',19000,’Mumbai’)

**creating table valued Functions**

USE sha

go

CREATE FUNCTION ft4 **( @City varchar(30) )**

RETURNS table

AS

RETURN (select empname from emp where city =@city )

go

SELECT \* FROM ft4('salem')

--Types of Function

--Scalar Functions:

/\*

A scalar function accepts any number of parameters and returns one value.The term scalar differentiates a single,"flat" value from more complex structured values, such as arrays or result sets.

This pattern is much like that of traditional functions written in common programming language.

\*/

Another definition :

[Scalar Functions](javascript:void(0))

User-defined scalar functions return a single data value of the type defined in the RETURNS clause. For an inline scalar function, there is no function body; the scalar value is the result of a single statement.

For a multistatement scalar function, the function body, defined in a BEGIN...END block, contains a series of Transact-SQL statements that return the single value. The return type can be any data type except  **text**, **ntext**, **image**, **cursor**, and **timestamp**.

The following examples creates a multistatement scalar function. The function takes one input value, a ProductID, and returns a single data value, the aggregated quantity of the specified product in inventory.

Transact-SQL

USE AdventureWorks2008R2;

GO

IF OBJECT\_ID (N'dbo.ufnGetInventoryStock', N'FN') IS NOT NULL

DROP FUNCTION ufnGetInventoryStock;

GO

CREATE FUNCTION dbo.ufnGetInventoryStock(@ProductID int)

RETURNS int

AS

-- Returns the stock level for the product.

BEGIN

DECLARE @ret int;

SELECT @ret = SUM(p.Quantity)

FROM Production.ProductInventory p

WHERE p.ProductID = @ProductID

AND p.LocationID = '6';

IF (@ret IS NULL)

SET @ret = 0;

RETURN @ret;

END;

GO

The following example uses the ufnGetInventoryStock function to return the current inventory quantity for products that have a ProductModelID between 75 and 80.

Transact-SQL

USE AdventureWorks2008R2;

GO

SELECT ProductModelID, Name, dbo.ufnGetInventoryStock(ProductID)AS CurrentSupply

FROM Production.Product

WHERE ProductModelID BETWEEN 75 and 80;

GO

[Table-Valued Functions](javascript:void(0))

User-defined table-valued functions return a **table** data type.

For an inline table-valued function, there is no function body; the table is the result set of a single SELECT statement.

The following example creates an inline table-valued function.

The function takes one input parameter, a customer (store) ID, and returns the columns ProductID, Name, and the aggregate of year-to-date sales as YTD Total for each product sold to the store.

Transact-SQL

USE AdventureWorks2008R2;

GO

IF OBJECT\_ID (N'Sales.ufn\_SalesByStore', N'IF') IS NOT NULL

DROP FUNCTION Sales.ufn\_SalesByStore;

GO

CREATE FUNCTION Sales.ufn\_SalesByStore (@storeid int)

RETURNS TABLE

AS

RETURN

(

SELECT P.ProductID, P.Name, SUM(SD.LineTotal) AS 'Total'

FROM Production.Product AS P

JOIN Sales.SalesOrderDetail AS SD ON SD.ProductID = P.ProductID

JOIN Sales.SalesOrderHeader AS SH ON SH.SalesOrderID = SD.SalesOrderID

JOIN Sales.Customer AS C ON SH.CustomerID = C.CustomerID

WHERE C.StoreID = @storeid

GROUP BY P.ProductID, P.Name

);

GO

The following example invokes the function and specifies customer ID 602.

Transact-SQL

SELECT \* FROM Sales.ufn\_SalesByStore (602);

For a multistatement table-valued function, the function body, defined in a BEGIN...END block, contains a series of Transact-SQL statements that build and insert rows into the table that will be returned.

The following example creates a table-valued function. The function takes a single input parameter, an EmployeeID and returns a list of all the employees who report to the specified employee directly or indirectly. The function is then invoked specifying employee ID 109.

Transact-SQL

USE AdventureWorks2008R2;

GO

IF OBJECT\_ID (N'dbo.ufn\_FindReports', N'TF') IS NOT NULL

DROP FUNCTION dbo.ufn\_FindReports;

GO

CREATE FUNCTION dbo.ufn\_FindReports (@InEmpID INTEGER)

RETURNS @retFindReports TABLE

(

EmployeeID int primary key NOT NULL,

FirstName nvarchar(255) NOT NULL,

LastName nvarchar(255) NOT NULL,

JobTitle nvarchar(50) NOT NULL,

RecursionLevel int NOT NULL

)

--Returns a result set that lists all the employees who report to the

--specific employee directly or indirectly.\*/

AS

BEGIN

WITH EMP\_cte(EmployeeID, OrganizationNode, FirstName, LastName, JobTitle, RecursionLevel) -- CTE name and columns

AS (

SELECT e.BusinessEntityID, e.OrganizationNode, p.FirstName, p.LastName, e.JobTitle, 0 -- Get the initial list of Employees for Manager n

FROM HumanResources.Employee e

INNER JOIN Person.Person p

ON p.BusinessEntityID = e.BusinessEntityID

WHERE e.BusinessEntityID = @InEmpID

UNION ALL

SELECT e.BusinessEntityID, e.OrganizationNode, p.FirstName, p.LastName, e.JobTitle, RecursionLevel + 1 -- Join recursive member to anchor

FROM HumanResources.Employee e

INNER JOIN EMP\_cte

ON e.OrganizationNode.GetAncestor(1) = EMP\_cte.OrganizationNode

INNER JOIN Person.Person p

ON p.BusinessEntityID = e.BusinessEntityID

)

-- copy the required columns to the result of the function

INSERT @retFindReports

SELECT EmployeeID, FirstName, LastName, JobTitle, RecursionLevel

FROM EMP\_cte

RETURN

END;

GO

-- Example invocation

SELECT EmployeeID, FirstName, LastName, JobTitle, RecursionLevel

FROM dbo.ufn\_FindReports(1);

GO

--Inline Table-Valued Functions:

1)Thistypeoffunctionsreturnsaresultset,muchlikeaview.

2)thisfunctionscanacceptparameters.

--The inline function's syntax is quite simple.In the function definition,

3)thereturntypeissettoatable.

4)Areturnstatementisusedwithaselectqueryinparenthesis.

--Multi-Statement Table-Valued Functions:

Multi-StatementfunctionscanbeusedtodosomeveryuniquethingsoutsidethecontextofastandardSELECTstatement.

1)Thistypeoffunctionreturnsatable-typeresultset,butthetableisexplicitlyconstructedinscript.

2)Thiscanbeusedtoaccomplishoneoftwothings:

eithertoprocesssomeveryuniquebusinesslogicbyassemblingavirtualtableonthefly,ortoduplicatethefunctionalityofaninlinefunctioninamoreverboseandcompiledway.Insort,ifyouneedtoselectrecordsfromanexistingaresultset,useaninlinetable-valuedfunction

SystemDefinedFunction.ThesefunctionsaredefinedbySqlServerfordifferentpurpose.WehavetwotypesofsystemdefinedfunctioninSqlServer

--Scalar Function

Scalarfunctionsoperatesonasinglevalueandreturnsasinglevalue.

BelowisthelistofsomeusefulSqlServerScalarfunctions.

--System Scalar Function:

ScalarFunctionDescription

abs(-10.67)Thisreturnsabsolutenumberofthegivennumbermeans10.67.

rand(10)Thiswillgeneraterandomnumberof10characters.

round(17.56719,3)Thiswillroundoffthegivennumberto3placesofdecimalmeans17.567

upper('dotnet')Thiswillreturnsuppercaseofgivenstringmeans'DOTNET'

lower('DOTNET')Thiswillreturnslowercaseofgivenstringmeans'dotnet'

ltrim(' dotnet')Thiswillremovethespacesfromlefthandsideof'dotnet'string.

convert(int,15.56)Thiswillconvertthegivenfloatvaluetointegermeans15.

--Aggregate Function

Aggregatefunctionsoperatesonacollectionofvaluesandreturnsasinglevalue.BelowisthelistofsomeusefulSqlServerAggregatefunctions.

SystemAggregateFunction

AggregateFunctionDescription

max()Thisreturnsmaximumvaluefromacollectionofvalues.

min()Thisreturnsminimumvaluefromacollectionofvalues.

avg()Thisreturnsaverageofallvaluesinacollection.

count()Thisreturnsnoofcountsfromacollectionofvalues.

--User Defined Function

Thesefunctionsarecreatedbyuserinsystemdatabaseorinuserdefined

database.

Wethreetypesofuserdefinedfunctions.

ScalarFunction

Userdefinedscalarfunctionalsoreturnssinglevalueasaresultofactionsperformbyfunction.Wereturnanydatatypevaluefromfunction.

--Create a table

\*/

CREATETABLEEmployee

(

EmpIDintPRIMARYKEY,

FirstNamevarchar(50)NULL,

LastNamevarchar(50)NULL,

SalaryintNULL,

Addressvarchar(100)NULL,

)

--Insert Data

InsertintoEmployee(EmpID,FirstName,LastName,Salary,Address)

Values(1,'Mohan','Chauahn',22000,'Delhi');

InsertintoEmployee(EmpID,FirstName,LastName,Salary,Address)

Values(2,'Asif','Khan',15000,'Delhi');

InsertintoEmployee(EmpID,FirstName,LastName,Salary,Address)

Values(3,'Bhuvnesh','Shakya',19000,'Noida');

InsertintoEmployee(EmpID,FirstName,LastName,Salary,Address)

Values(4,'Deepak','Kumar',19000,'Noida');

--See created table

Select\*fromEmployee

--Create function to get emp full name

CreatefunctionfnGetEmpFullName

(

@FirstNamevarchar(50),

@LastNamevarchar(50)

)

returnsvarchar(101)

As

Beginreturn (Select@FirstName+' '+@LastName);

end

--Calling the above created function

Selectdbo.fnGetEmpFullName(FirstName,LastName)asName,SalaryfromEmployee

--Inline Table-Valued Function

Userdefinedinlinetable-valuedfunctionreturnsatablevariableasaresultofactionsperformbyfunction.Thevalueoftablevariableshould

bederivedfromasingleSELECTstatement.

--Create function to get employees

CreatefunctionfnGetEmployee()

returnsTable

As

return (Select\*fromEmployee)

--Now call the above created function

Select\*fromfnGetEmployee()

--Multi-Statement Table-Valued Function

Userdefinedmulti-statementtable-valuedfunctionreturnsatablevariableasaresultofactionsperformbyfunction.

Inthisatablevariablemustbeexplicitlydeclaredanddefinedwhose valuecanbederivedfromamultiplesqlstatements.

--Create function for EmpID,FirstName and Salary of Employee

CreatefunctionfnGetMulEmployee()

returns@EmpTable

(

EmpIDint,

FirstNamevarchar(50),

Salaryint

)

As

begin

Insert@EmpSelecte.EmpID,e.FirstName,e.SalaryfromEmployeee;

--Now update salary of first employee

update@EmpsetSalary=25000whereEmpID=1;

--It will update only in @Emp table not in Original Employee table

return

end

--Now call the above created function

Select\*fromfnGetMulEmployee()

/\*Now see the original table. This is not affected by above function

update command\*/

Select\*fromEmployee

CreatefunctionFUNINSERT()

returns@EmpTable

(EmpIDint,FirstNamevarchar(50),Salaryint

)

As

begin

Insert@EmpVALUES(102,'Gem',345435)

--Now update salary of first employee

--update @Emp set Salary=25000 where EmpID=1;

--It will update only in @Emp table not in Original Employee table

return

end

select\*fromfuninsert()

select\*from@emp

Difference Between stored Procedure and Functions:

**Basic Difference**

1. Function must return a value but in Stored Procedure it is optional( Procedure can return zero or n values).
2. Functions can have only input parameters for it whereas Procedures can have input/output parameters .
3. Function takes one input parameter it is mandatory but Stored Procedure may take o to n input parameters..
4. Functions can be called from Procedure whereas Procedures cannot be called from Function.

**Advance Difference**

1. Procedure allows SELECT as well as DML(INSERT/UPDATE/DELETE) statement in it whereas Function allows only SELECT statement in it.
2. Procedures cannot be utilized in a SELECT statement whereas Function can be embedded in a SELECT statement.
3. Stored Procedures cannot be used in the SQL statements anywhere in the WHERE/HAVING/SELECT section whereas Function can be.
4. Functions that return tables can be treated as another rowset. This can be used in JOINs with other tables.
5. Inline Function can be thought of as views that take parameters and can be used in JOINs and other Rowset operations.
6. Exception can be handled by try-catch block in a Procedure whereas try-catch block cannot be used in a Function.
7. We can go for Transaction Management in Procedure whereas we can't go in Function.