1. Laboratory session 1: Basics of T-SQL programming

Learning Objective:

At the end of this lesson, students will be able to:

- Declare variables
- Understand the usage of If…Else statements
- Understand the usage of WHILE Loop With CONTINUE and BREAK Keywords
- Understand the usage of CASE…WHEN expressions
- Use control structures to solve problems

1.1 Decalring variables and Parameters:

In SQL Server, a variable is typical known as a local variable is only available in the batch, stored procedure or code block in which it is defined. A local variable is defined using the Transact-SQL (T-SQL) DECLARE statement. The name of the local variable needs to start with the @ symbol as the first character of its name. A local variable can be declared as any system or user defined data type. Here is a typical declaration for an integer variable named @Count.

```
DECLARE @Count INT
```

More than one variable can be defined with a single DECLARE statement. To define multiple variables, with a single DECLARE statement, you separate each variable definition with a comma, like this:

```
DECLARE @Count INT, @X INT, @Y INT, @Z CHAR (10)
```

In the above declaration, 4 local variables with a single DECLARE statement. A local variable is initially assigned a NULL value. A value can be assigned to a local variable by using the SET or SELECT statement. On the SET command you specify the local variable and the value you wish to assign to the local variable. Here is an example of where I have defined my @Count variable and then initialize the variable to 1.

```
DECLARE @Count INT
SET @Count = 1
```

Here is an example of how to use the SELECT statement to initialize the value of a local variable with the value returned from a select that fetch the total number of rows in the employee table.

```
DECLARE @ROWCNT INT
SELECT @ROWCNT= (SELECT COUNT (*) FROM employee)
```

One of the uses of a variable is to programmatically control the records returned from a SELECT statement. You do this by using a variable in the WHERE clause. Here is an example that returns all the Customers records in the Sales database where the Customer's Country column is equal to 'Germany'

```sql
Declare @Country varchar(25)
set @Country = 'Germany'
select CompanyName from Sales.dbo.Customer where Country = @Country
```

1.2 IF ... ELSE statemens

T-SQL has the **IF** statement to help with allowing different code to be executed based on the results of a condition. The "IF" statement allows a T-SQL programmer to selectively execute a single line or block of code based upon a Boolean condition. There are two formats for the "IF" statement, both are shown below:

Format one: IF <condition> <then code to be executed when condition true>

Format two: IF <condition> <then code to be executed when condition true>   ELSE < else code to be executed when condition is false>

In both of these formats, the <condition> is a Boolean expression or series of Boolean expressions that evaluate to true or false. If the condition evaluates to true, then the "then code" is executed. For format two, if the condition is false, then the "else code" is executed. If there is a false condition when using format one, then the next line following the IF statement is executed, since no else condition exists. The code to be executed can be a single TSQL statement or a block of code. If a block of code is used then it will need to be enclosed in a BEGIN and END statement.

Let's review how "Format one" works. This first example will show how the IF statement would look to execute a single statement, if the condition is true. Here, the if statement tests whether a variable is set to a specific value. If the variable is set to a specific value, then, print out the appropriate message.

```sql
     Declare @x int
     set @x = 29
     if @x = 29 print 'The number is 29'
     if @x = 30 print 'The number is 30'
```

The above code prints out only the phrase "The number is 29", because the first IF statement evaluates to true. Since the second IF is false, the second print statement is not executed. Now the condition statement can also contain a SELECT statement. The SELECT statement will need to return value or set of values that can be tested. If a SELECT statement is used the statement needs to be enclosed in parentheses.

```
      if (select count(*) from Sales.dbo.Customer
              where fname like '[A-D]%') > 0
         print 'A-D Customers are Found '
```

The above examples only showed how to execute a single T-SQL statement if the condition is true. T-SQL allows you to execute a block of code as well. A code block is created by using a "BEGIN" statement before the first line of code in the code block, and an "END" statement after that last line of code in the code block. Here is any example that executes a code block when the IF statement condition evaluates to true.

```
if db_name() = 'master'
       begin
         Print 'We are in the Master Database'
         Print ''
         Print 'So whatever code you execute must be done carefully'
       End
else if db_name() = 'Sales'
       begin
         Print 'We are in the test Sales' database
              Print 'So Enjoy'
       End
```

Sometimes you want to not only execute some code when you have a true condition, but also want to execute a different set of T-SQL statements when you have a false condition. If this is your requirement then you will need to use the IF...ELSE construct, that I called format two above.

Example:

Suppose we want to determine whether to update or add a record to the employee table in the HR database. The decision is based on whether the employee already exists in the HR.dbo.Employee table. Here is the T-SQL code to perform this existence test for two different EmployeeId's.

```
      if exists(select * from HR.dbo.Employee
                  where CustomerId = 'Abebe123')
         Print 'Need to update Record 'Abebe123'
      else
         Print 'Need to add Employee Record 'Abebe123'
```

1.3 WHILE Loop With CONTINUE and BREAK Keywords

BREAK keyword will exit the stop the while loop and control is moved to next statement after the while loop. CONTINUE keyword skips all the statement after its execution and control is sent to first statement of while loop.

Syntax:

```
WHILE Boolean_expression
{ sql_statement | statement_block | BREAK | CONTINUE }
{sql_statement | statement_block}
```

BREAK**:** Causes an exit from the innermost WHILE loop. Any statements that appear after the END keyword, marking the end of the loop, are executed.

CONTINUE: Causes the WHILE loop to restart, ignoring any statements after the CONTINUE keyword.

Example 1:

```
WHILE (SELECT AVG(ListPrice) FROM Production.Product) < 300
    BEGIN
      UPDATE Production.Product
        SET ListPrice = ListPrice * 2
      SELECT MAX(ListPrice) FROM Production.Product
      IF (SELECT MAX(ListPrice) FROM Production.Product) > 500
        BREAK
      ELSE
        CONTINUE
    END
```

Example 2:

```
DECLARE @intFlag INT
SET @intFlag = 1
WHILE (@intFlag <=5)
BEGIN
PRINT @intFlag
SET @intFlag = @intFlag + 1
END
```

Example 3: Usage of WHILE Loop with BREAK keyword

```
    DECLARE @intFlag INT
    SET @intFlag = 1
    WHILE (@intFlag <=5)
    BEGIN
    PRINT @intFlag
    SET @intFlag = @intFlag + 1
    IF @intFlag = 4
    BREAK;
    END
GO
```

1.4.CASE... When expressions

Evaluates a list of conditions and returns one of multiple possible result expressions.

The CASE expression has two formats:

- The simple CASE expression compares an expression to a set of simple expressions to determine the result.
- The searched CASE expression evaluates a set of Boolean expressions to determine the result. Within a SELECT statement, the searched CASE expression allows for values to be replaced in the result set based on comparison values.

Syntax for The simple CASE expression:

```
CASE input_expression
    WHEN when_expression THEN result_expression [ ...n ]
    [ELSE else_result_expression]
END
```

Example:

```
USE HR;
GO
SELECT   FName, AcademicRank =
     CASE Qualification
       WHEN 'Bsc' THEN 'Graduate Assistant'
       WHEN 'Msc' THEN 'Lecturer'
       WHEN 'Phd' THEN 'Assistant Professor'
       ELSE 'Technical Assistant'
     END,
  salary
FROM employee
ORDER BY sex;
GO
```

Syntax for Searched CASE expression:

```
CASE
    WHEN Boolean_expression THEN result_expression [ ...n ]
    [ ELSE else_result_expression ]
END
```

Example:

```
USE HR;
GO
SELECT   FName, LName, AcademicRank =
     CASE
       WHEN 'Bsc' THEN 'Graduate Assistant'
       WHEN 'Msc' THEN 'Lecturer'
       WHEN 'Phd' THEN 'Assistant Professor'
       ELSE 'Technical Assistant'
     END     FROM employee  ORDER BY sex;
GO
```

1.5 Exercises

1. Use if …Else statement and write T-SQL program to convert Letter grades into their corresponding Number grades
2. Use while… loop and write T-SQL program to find the factorial of a number
3. Use CASE……When and classify the payment category of employees as follows:
    a. For salary >=8000………..High paid
    b. For salary >=4000………..Midium paid
    c. For salary <400…………...Low paid

2. Laboratory session 2: User defined functions

Learning objective

At the end of this lab session, students will be able to:

- Create and alter user defined functions
- Use functions to solve problems

User defined functions are functions which are developed by the users of the database system to supplement and extend the built-in functions.A user-defined function takes zero, or more, input parameters and returns either a scalar value or a table. Input parameters can be any data type except timestamp,cursor, or table. User defined functions may take the following forms:

a. Scalar user defined functions
b. Inline table valued functions
c. Multi statements table valued functions

2.1 Creating scalar user defined functions using SQL server:

Syntax:

```
CREATE FUNCTION [schema_name.]function_name ([@parameter_name1
parameter_data_type],[@parameter_name2 parameter_data_type], [@parameter_name3
parameter_data_type],…[parameter_nameN parameter_data_type]
)
RETURNS return_data_type
   [ WITH<function_option>]
   AS
   BEGIN
   function_body
      RETURN scalar_expression
END
```

Where

Schema_name = the name of the function owner

function_name = any valid sysname for the function

@ parameter_name1 …………**@ parameter_nameN**= the name of the parameter for the function

return_data_type = any scalar data type data type like int, Varchar,Nvarcahr, date, char,Ncahr, etc.

function_option = specification of whether to encrypt the function definition or not,schema bound or not, etc

Example: create a function that calculates the net pay of Instructos based on the tax dedcution logic shown in the next table below.

Assume we have a table named employee with the follwong attributes
id int primary key,fname varchar(20),sex char,salary float

Tax deduction logic in a given range

Salary	Tax rate (%)
0-150	0 (free from tax)
151-650	10
651-1400	15
1401-2350	20
2351-3500	25
3500-5000	30
>5000	35

```
create function netPay(@sal float)
returns float
as
begin
declare @np float
if @sal<=150
        set @np=@sal
else if @sal<650
        set @np=@sal-0.10*@sal
else if @sal<1400
        set @np=@sal-(50+0.15*(@sal-650))
else if(@sal<2350)
        set @np=@sal-(162.50 +0.2 *(@sal-1400))
else if(@sal<3500)
        set @np=@sal-(352.50 + 0.25 *(@sal-2350))
else if(@sal<5000)
        set @np=@sal-(640 + 0.30 *(@sal-3500))
else
        set @np=@sal-(1090 + 0.35 *(@sal-5000))
return @np
end
```

When referencing a scalar user defined function, specify the function owner and the function name in two-part syntax as shown next:

select dbo.netPay(Instructor.salary) from Instructor

Setting Permissions for User-defined Functions

- User must have CREATE FUNCTION permission to create, alter, and drop user defined functions
- Users other than the owner must be granted EXECUTE permission on a function before they can use it in a Transact-SQL statement

- If the function is being schema-bound, you must have REERENCE permission on tables, views, and functions referenced by the function. REFERENCE permissions can be granted through the GRANT statement to views and user defined functions as well as tables.

- If a CREATE TABLE or ALTER TABLE S statement references a user-defined function in a CHECK constraint, DEFAULT clause, or computed column, the table owner must also own the function.

2.2. Inline table valued functions

These type of functions are fucntions that return list of records/tables based on a logic defined using a single SQL statement.

Syntax:

```
CREATE FUNCTION [ schema_name. ] function_name
        ( [ { @parameter_name ] parameter_data_type
            [ ,...n ]
          ]
        )
        RETURNS TABLE
            [ WITH<function_option> [ ,...n ] ]
            [ AS ]
            RETURN [ ( ] select_stmt [ ) ]
        [ ; ]
```

Example: create an inline table valued function to list the name of all female Instructors

```
create function fn_FemaleInstructors()

returns table

as

return select FirstName,MidleNamefrom Instructor where sex='F'
```

To call the function, run the function in following manner:

```
select * from dbo. fn_FemaleInstructors()
```

2.3 Multi statements table valued functions

These type of functions are fucntions that return list of records/tables based on a logic defined using many SQL statements that perform complex operations. In the same way that you use a view, you can usea table-valued function in the FROM clause of a Transact-SQL statement.

When using a multi-statement table-valued function, consider the following facts:

- The BEGIN and END delimit the body of the function
- The RETURNS clause specifies **table** as the data type returned.
- The RETUENS clause defines a name for the table and defines the format of the table.

The scope of the return variable name is local to the function.

Syntax:

```
CREATE FUNCTION [schema_name.] function_name
( [@parameter_name parameter_data_type,
    [ ,...n ]
  ]
)
RETURNS @return_variable TABLE <table_type_definition>
    [ WITH<function_option> [ ,...n ] ]
     AS
   BEGIN
   function_body
       RETURN
  END
```

Example: create a multi-statement table-valued function that returns the name orFull names of employees, depending on the parameter provided.

Solution:

```
CREATE FUNCTION fn_Employees
(@length varchar(60))
RETURNS @fn_EmployeeTABLE(EmployeeIDint Primary Key Not Null,
EmployeeName varchar(16) Not Null)
AS
BEGIN
IF (@length =’ShortName’)
INSERT @fn_Employees SELECT EmployeeID, fNameFROM Employees
        ELSE IF(@length =’FullName’)
                INSERT @fn_Employees SELECT EmployeeID,fName + ‘ ’+MidName + ‘ ’  +
        GfatherName FROM Employees
RETURN
END
```

2.4. Altering Functions

You modify a user-defined function by using the ALTER FUNCTION statement

Syntax:

```
        ALTER FUNCTION function_name
        <New Function Content>
```

This example shows how to alter a function

```
ALTER FUNCTION dbo. fn_NetPay
<New Function Content here>
```

2.5. Dropping Functions

You can drop a user-defined function by using DROP FUNCTION statement.

Syntax: DROP FUNCTION function_name

Example: DROP FUNCTION dbo.fn_NetPay

2.6. Exercise

1. Create a function which gives you the maximum of two integers.
2. Suppose that we have two tables named as:
    a. Student(fname, studID, sex)
    b. Stud_Grade (studID, courseNo,CourseTitle, CrHr, Grade)
3. Based on the values that could be populated to these tables, create a function that returns the GPA of a student when you pass studID as an argument to the function.

The following table structures are taken from ABC publishing database**.** Refer these tables and write T-SQL statements to answer questions that follow.

Take the following assumptions:

- The attribute **"BookTitle"** in the book table can have values like C++, Database, Java, etc…
- The attribute **"City"** in the publisher table can have values like AA, Jimma,New York, London, etc…

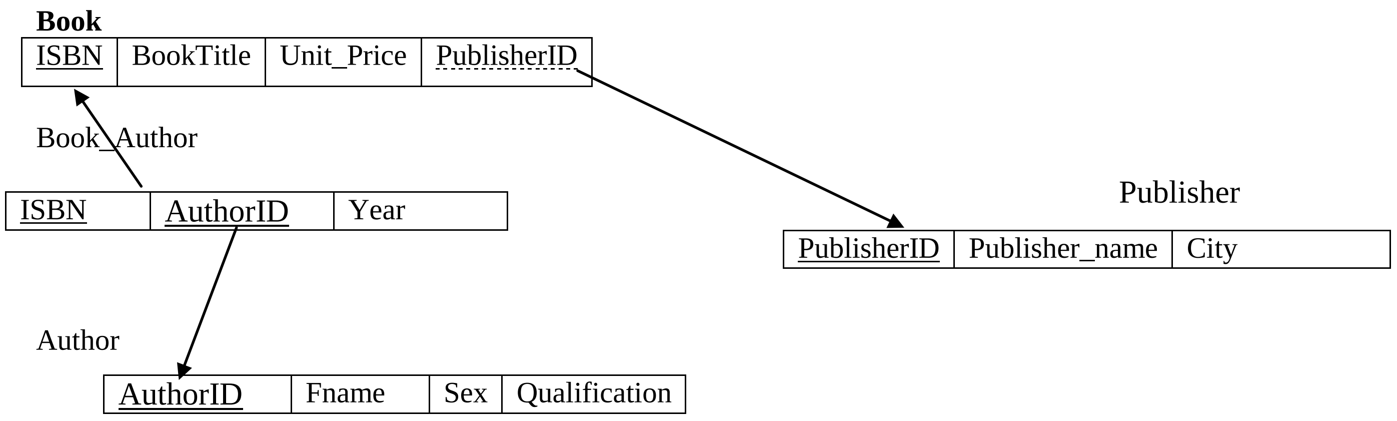

Table 2.1: Tables from ABC publishing database

1. Write a scalar function that returns the average price of Books published by male authors.
2. Write a scalar function that returns the total number of female Authors who published database books by **Jimma** publishers.
3. Write a scalar function that takes ‘Publisher_Name’ as an argument and return total number of authors who published books in the past ten years.
4. Create an inline table valued function that generate the list of all authors who published java books
5. Create a multi statement table valued function to find ID, name and sex of all authors who already published books. Call this function under a new scalar function that returns the total number of books published by female authors.

2.7 Creating function using Oracle SQL

Synax:

```
create [or replace] function [schema .] function
[( argumentdatatype
[, argument  datatype]...
)]
returndatatype
{ is | as } { pl/sql_function_body }};
```

If a function already exists, you may replace it via **create or replace function** command. Ifyou use the **or replace** clause, any EXECUTE grants previously made on the function will remainin place.

Example:

```
create or replace
functionNetPay(sal float) return
is
Np float;
begin
if (sal<=150) then
Np:=sal;
elsif (sal<=650) then
 NP:=sal-(sal-150)*.1;
Elsif(sal>=651) then
NP:=sal-(50+(sal-650)*.15)
else
 NP:=sal-(112.50+(sal-1400)*.15);
end if;
returnNp;
end;
```

Salary deduction in a given range

Salary	**Tax(%)**
0-150	0
151-650	10
651-1400	15
>=1401	20

To call the above oracle function, use the following code
Select NetPay(NetPay) from dual;
N.B. Dual is a dummy table with one element in it. It is useful because Oracle doesn't allow statements without specifying the From clause.

Exercise

Show oracle equivalent SQL codes for the functions we created Using SQL server codes.