13. Conditional Statements

IF ELSE Statement

```
IF BooleanExpression
BEGIN
      Statement block
END
```

The below function uses the IF ELSE statement to check an integer.

Function:

```
CREATE FUNCTION [dbo].[FnFindIntegerType] (@Value INT) RETURNS
VARCHAR(10)
AS
BEGIN
      DECLARE @Return VARCHAR(10);

      IF (@Value < 0)
                  BEGIN
                  SET @Return = 'Negative'
                  END
      ELSE IF (@Value = 0)
                  BEGIN
                SET @Return = 'Zero'
            END
      ELSE
            BEGIN
                  SET @Return = 'Positive'
            END
      RETURN @Return;
END
```

The function identifies 2 as a positive number.

```
SELECT DBO.FnFindIntegerType(2)
```

Output:

CASE Statement

The CASE statement is almost similar to IF ELSE statements. The case statement check conditions one after one. It returns from the statement when the condition is a success. It also returns the relevant result for the matching condition
Above IF ELSE statement can be easily converted into the CASE statement as shown below.

Function:

```
ALTER FUNCTION [dbo].[FnFindIntegerType] (@Value INT) RETURNS
VARCHAR(20)
AS
BEGIN
      RETURN
      (SELECT CASE
            WHEN (@Value < 0) THEN 'Negative'
            WHEN (@Value = 0) THEN 'Zero'
            WHEN (@Value > 0) THEN 'Positive'
            ELSE 'Cannot Identify'
      END)
END
```

Use the below command for execution.

```
SELECT [dbo].[FnFindIntegerType](3)
```

Result:
Below is another example of a CASE statement.

Function:

```sql
CREATE FUNCTION [dbo].[FnShowNumber] (@Value INT) RETURNS VARCHAR(20)
AS
BEGIN
	DECLARE @Number VARCHAR(20);
	RETURN
	(SELECT CASE
		WHEN (@Value = 1) THEN 'One'
		WHEN (@Value = 2) THEN 'Two'
		WHEN (@Value = 3) THEN 'Three'
		WHEN (@Value = 4) THEN 'Four'
		WHEN (@Value = 5) THEN 'Five'
		ELSE 'Cannot Identify'
	END)
END
```

```sql
CREATE FUNCTION [dbo].[FnShowDayByNumber] (@Value INT) RETURNS
VARCHAR(20)
AS
BEGIN
	DECLARE @Number VARCHAR(20);
	RETURN
	(SELECT CASE
		WHEN (@Value = 1) THEN 'Sunday'
		WHEN (@Value = 2) THEN 'Monday'
		WHEN (@Value = 3) THEN 'Tuesday'
		WHEN (@Value = 4) THEN 'Wednesday'
		WHEN (@Value = 5) THEN 'Thursday'
		WHEN (@Value = 6) THEN 'Friday'
		WHEN (@Value = 7) THEN 'Saturday'
		ELSE 'Cannot Identify'
	END)
END
```

```
CREATE FUNCTION [dbo].[FnShowNumberOfTheDay] (@Value VARCHAR(20))
RETURNS INT
AS
BEGIN
      DECLARE @Number VARCHAR(20);
      RETURN
      (SELECT CASE
            WHEN (@Value = 'Sunday')     THEN 1
            WHEN (@Value = 'Monday')     THEN 2
            WHEN (@Value = 'Tuesday')    THEN 3
            WHEN (@Value = 'Wednesday')  THEN 4
            WHEN (@Value = 'Thursday')   THEN 5
            WHEN (@Value = 'Friday')     THEN 6
            WHEN (@Value = 'Saturday')   THEN 7
            ELSE 0
      END)
END
```

Use the below command for execution.

```
SELECT dbo.FnShowNumber(3)
```

WHILE Loop

In a while loop, the query is repeating until it meets a certain condition.

Syntax:

```
WHILE BooleanExpression
BEGIN
      Statement block
END
```

The following function will calculate the total of integers from zero to a defined value using a while loop. For loops are not available in SQL.

Function:

```
ALTER FUNCTION [dbo].[FnCalculateTotal] (@Value INT) RETURNS INT
AS
BEGIN
            DECLARE @Value2 INT = 0;
            WHILE (@Value > 0)
            BEGIN
                  SET @Value2 += @Value;
                  SET @Value = @Value - 1;
            END
      RETURN @Value2
END
```

The above function can be used to find the integer total from 0 to 4. As shown below.
1 + 2 + 3 + 4 = 10

```
SELECT dbo.FnCalculateTotal(4);
```

Output:

```
ALTER FUNCTION [dbo].[FnCalculateSquareTotal] (@Value INT) RETURNS INT
AS
BEGIN
            DECLARE @Value2 INT = 0;
            WHILE (@Value > 0)
            BEGIN
                  SET @Value2 += @Value * @Value;
                  SET @Value = @Value - 1;
            END
      RETURN @Value2
END
```