CMPE 2400 Databases

T-SQL Programming

What is T-SQL?

- T-SQL is Microsoft’s version of SQL
- An extension of standard SQL which includes programming features
  - Local variables
  - Conditional structures
  - Looping structures
  - User-defined procedures/functions

Decisions

- T-SQL provides for decision-based logic through:
  - if…else
  - case

Relational Operators

- **a = b** (not a == b): The operator for equality comparison is a single equal sign instead of a double equal sign as in C#.
- There are two ways to indicate a not equal to comparison:
  - <> (ANSI Standard)
  - != (T-SQL)
- Other comparisons:
  - > (ANSI)
  - < (ANSI)
  - !> (T-SQL)
  - !< (T-SQL)

IF … ELSE IF … ELSE

- In SQL, the syntax for an *if … else* statement varies slightly from C#.
  - There is no need to include the parenthesis around the conditional statement(s).
  - When more than one statement is to be executed depending on a condition, the keywords **begin** and **end** take the place of curly braces ({ }) to define the subservient scope.
  - It should be noted that though if … else if … else ladders are supported in the latest versions of SQL Server, this has not always been the case.

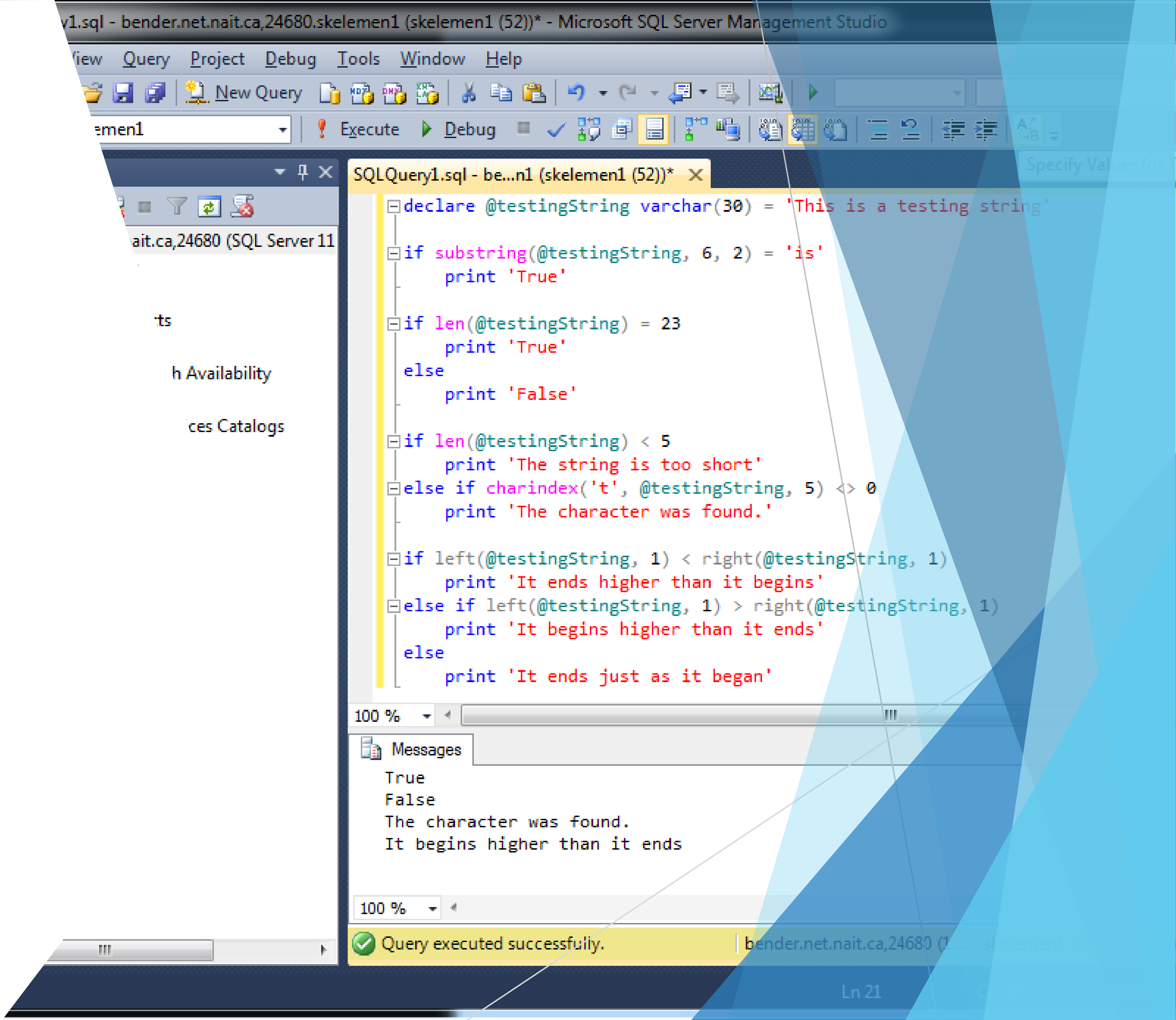
IF … ELSE IF … ELSE

- Syntax with single statements

```
if conditionalExpression
      singleStatement
[ else if   conditionalExpression
      singleStatement  ]
[ else
      singleStatement  ]
```

IF … ELSE IF … ELSE

- Examples with single statements



v1.sql - bender.net.nait.ca,24680.skelemen1 (skelemen1 (52))* - Microsoft SQL Server Management Studio
View Query Project Debug Tools Window Help
New Query
Execute Debug
SQLQuery1.sql - be...n1 (skelemen1 (52))*

```
declare @testingString varchar(30) = 'This is a testing string'

if substring(@testingString, 6, 2) = 'is'
    print 'True'

if len(@testingString) = 23
    print 'True'
else
    print 'False'

if len(@testingString) < 5
    print 'The string is too short'
else if charindex('t', @testingString, 5) <> 0
    print 'The character was found.'

if left(@testingString, 1) < right(@testingString, 1)
    print 'It ends higher than it begins'
else if left(@testingString, 1) > right(@testingString, 1)
    print 'It begins higher than it ends'
else
    print 'It ends just as it began'
```

```
True
False
The character was found.
It begins higher than it ends
```

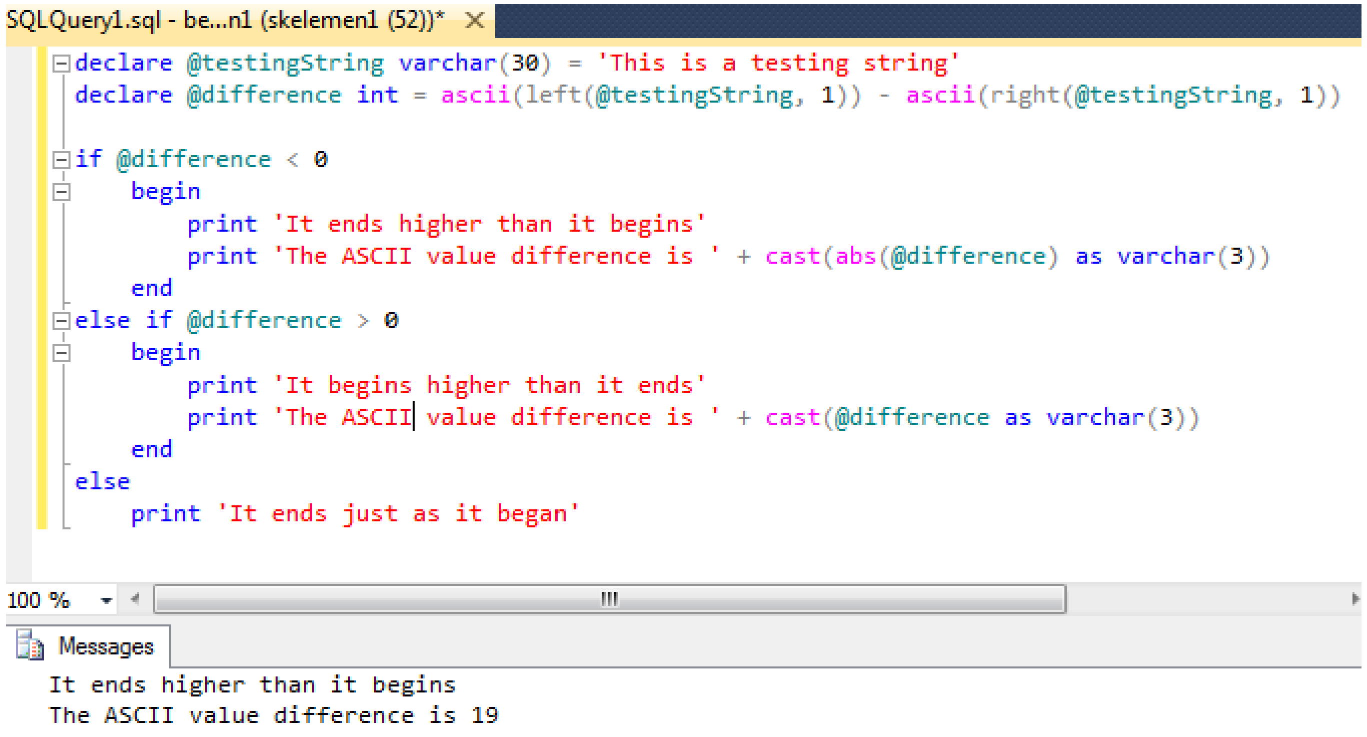
IF … ELSE IF … ELSE

- Syntax with multiple statements

```
if conditionalExpression
  begin
     multipleStatements
  end
[    else if    conditionalExpression
      begin
         multipleStatements
      end                    ]
[    else
      begin
         multipleStatements
      end                    ]
```

IF … ELSE IF … ELSE

- Example with multiple statement block



```sql
declare @testingString varchar(30) = 'This is a testing string'
declare @difference int = ascii(left(@testingString, 1)) - ascii(right(@testingString, 1))

if @difference < 0
    begin
        print 'It ends higher than it begins'
        print 'The ASCII value difference is ' + cast(abs(@difference) as varchar(3))
    end
else if @difference > 0
    begin
        print 'It begins higher than it ends'
        print 'The ASCII value difference is ' + cast(@difference as varchar(3))
    end
else
    print 'It ends just as it began'
```

```
It ends higher than it begins
The ASCII value difference is 19
```

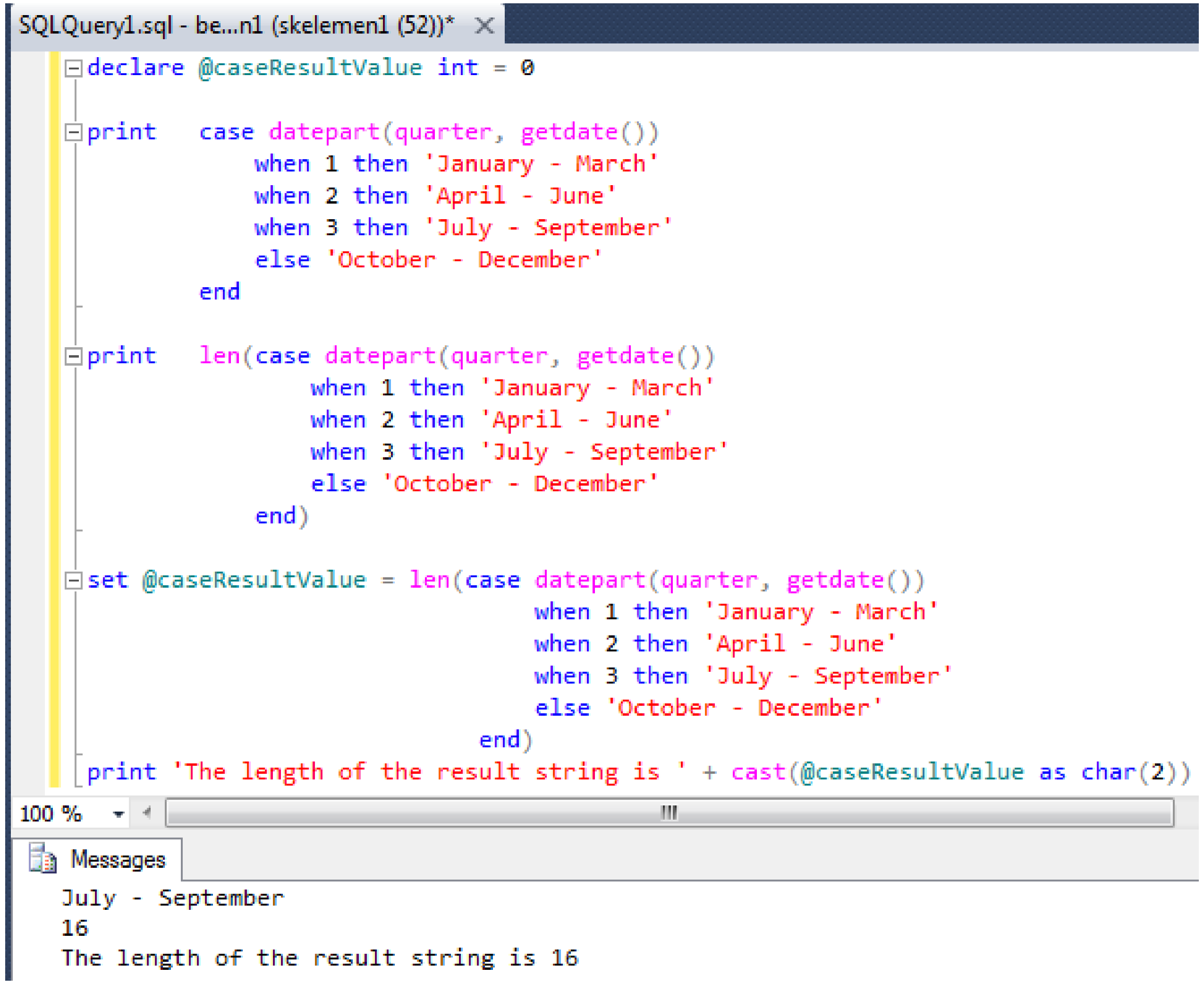
CASE Statement

- The SQL equivalent to C# switches.
  - Only a single result is ever returned
  - Execution flow cannot be controlled
  - Treat case like a function and use it wherever the return value may be used
- There are two general formats.
  - The *simple* form is most like a C# switch in that you compare different values to a starting value, and return a predefined result value
  - The *searched* form is more like an if … else ladder in that the result value is determined by evaluating boolean expressions

Simple CASE Syntax

```
case        startingValue
      when  comparisonValue then resultValue
      [  when  comparisonValue then resultValue  ]
      [  when  comparisonValue then resultValue  ]
      …
      [  else resultValue  ]
end
```

Simple CASE Examples



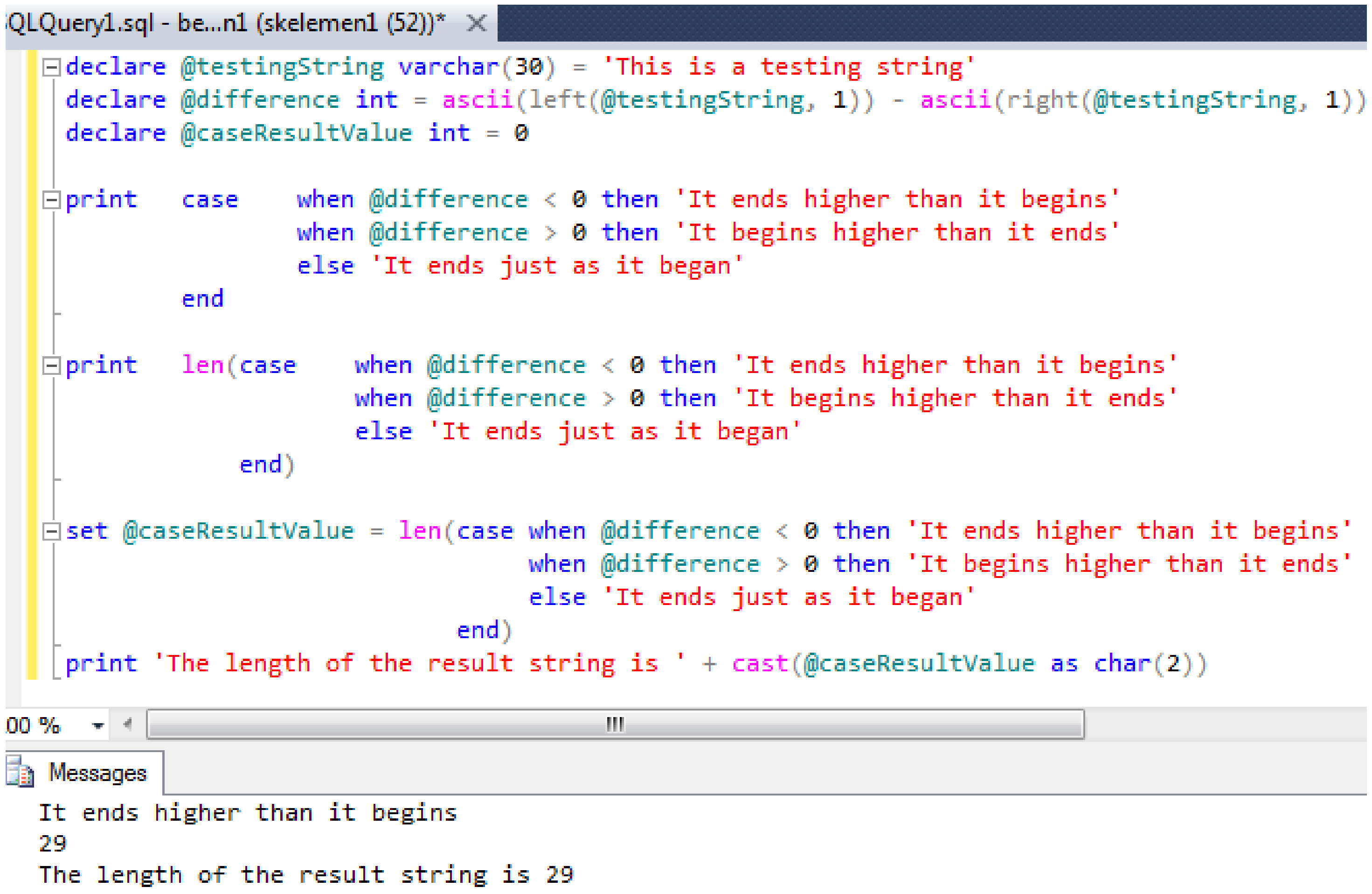
```sql
declare @caseResultValue int = 0

print   case datepart(quarter, getdate())
            when 1 then 'January - March'
            when 2 then 'April - June'
            when 3 then 'July - September'
            else 'October - December'
        end

print   len(case datepart(quarter, getdate())
                when 1 then 'January - March'
                when 2 then 'April - June'
                when 3 then 'July - September'
                else 'October - December'
            end)

set @caseResultValue = len(case datepart(quarter, getdate())
                                when 1 then 'January - March'
                                when 2 then 'April - June'
                                when 3 then 'July - September'
                                else 'October - December'
                            end)
print 'The length of the result string is ' + cast(@caseResultValue as char(2))
```

```
July - September
16
The length of the result string is 16
```

Searched CASE Syntax

```
case
    when  booleanExpression then resultValue
    [  when booleanExpression then resultValue  ]
    [  when booleanExpression then resultValue  ]
    …
    [  else resultValue  ]
end
```

Searched CASE Examples



QLQuery1.sql - be...n1 (skelemen1 (52))*

```
declare @testingString varchar(30) = 'This is a testing string'
declare @difference int = ascii(left(@testingString, 1)) - ascii(right(@testingString, 1))
declare @caseResultValue int = 0

print   case    when @difference < 0 then 'It ends higher than it begins'
                when @difference > 0 then 'It begins higher than it ends'
                else 'It ends just as it began'
        end

print   len(case    when @difference < 0 then 'It ends higher than it begins'
                    when @difference > 0 then 'It begins higher than it ends'
                    else 'It ends just as it began'
            end)

set @caseResultValue = len(case when @difference < 0 then 'It ends higher than it begins'
                                when @difference > 0 then 'It begins higher than it ends'
                                else 'It ends just as it began'
                           end)
print 'The length of the result string is ' + cast(@caseResultValue as char(2))
```

00 %

Messages

```
It ends higher than it begins
29
The length of the result string is 29
```

Logical Operators

- Here is the list of familiar operators

AND	*true* if the comparisons on both sides are *true*. Equivalent to (&&) in C#.
OR	*true* if either of the comparisons is *true*. Equivalent to (||) in C#.
NOT	Inverts the logic of any other logical operator. Equivalent to (!) In C#.

Logical Operators

- These operators are new, but quite useful.

  - BETWEEN

    true if the comparison value is between the specified range. For example, if @value = 45, @value between 1 and 100 will return *true*.

    Basically this operator is a nicer way of writing @value >= 1 and @value <= 100.

  - IN

    true if the comparison value is equivalent to one value in the comma separated list. For example, if @value = 45, @value in (1, 45, 90) will return *true*.

    Basically this operator is a nicer way of writing

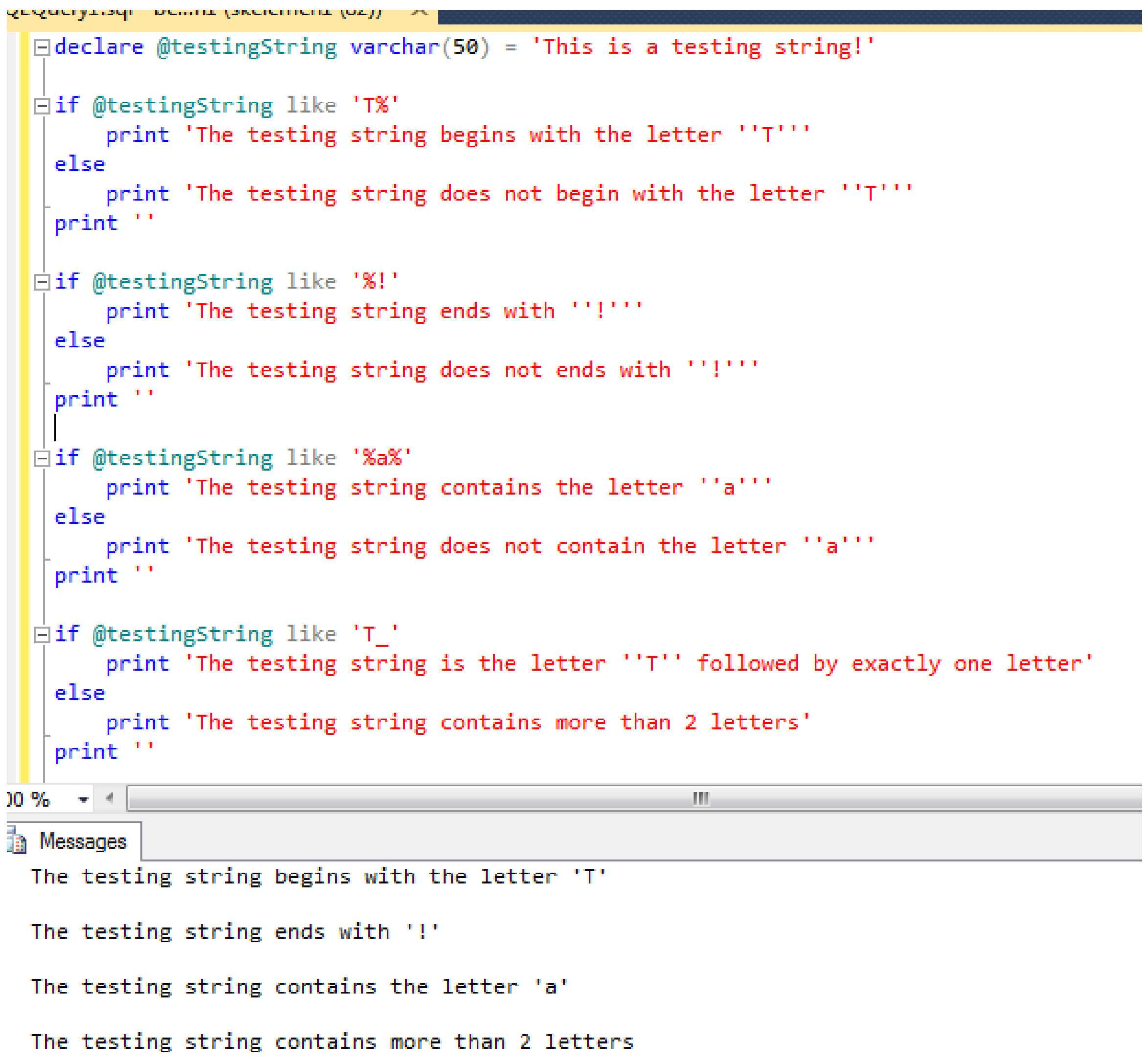
    @value = 1 or @value = 45 or @value = 90.

Logical Operators

- These operators are new, but quite useful

LIKE	*true* if the comparison value matches a specified pattern. This operator is used for comparison with strings, and makes use of a set of wildcard characters.
%	any number of characters (including 0)
_	any single character
[ ]	specify a list of acceptable characters
[-]	specify a range of characters
[^ -]	range of unacceptable characters

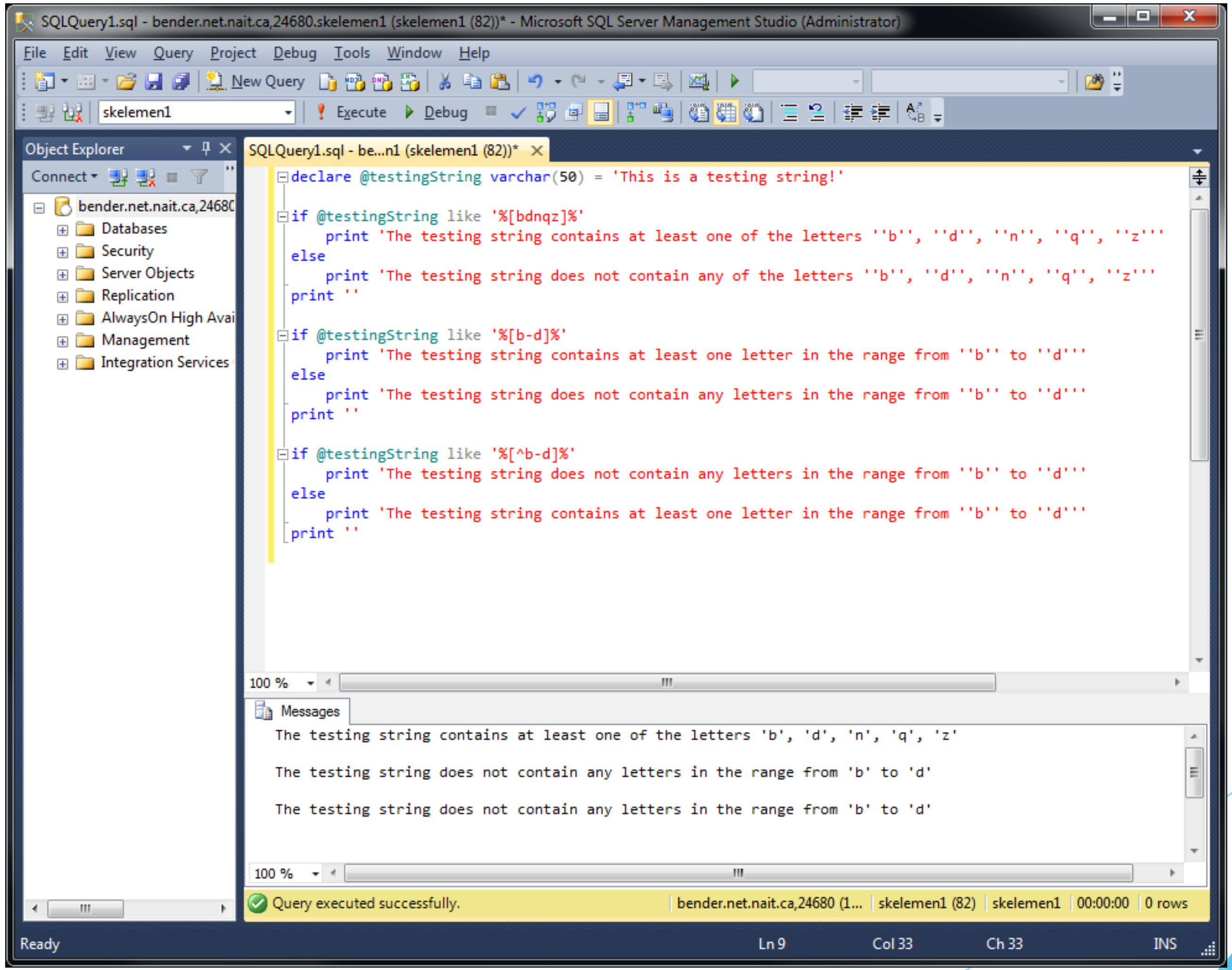
LIKE Example Comparisons



```
declare @testingString varchar(50) = 'This is a testing string!'

if @testingString like 'T%'
    print 'The testing string begins with the letter ''T'''
else
    print 'The testing string does not begin with the letter ''T'''
print ''

if @testingString like '%!'
    print 'The testing string ends with ''!'''
else
    print 'The testing string does not ends with ''!'''
print ''

if @testingString like '%a%'
    print 'The testing string contains the letter ''a'''
else
    print 'The testing string does not contain the letter ''a'''
print ''

if @testingString like 'T_'
    print 'The testing string is the letter ''T'' followed by exactly one letter'
else
    print 'The testing string contains more than 2 letters'
print ''
```

```
The testing string begins with the letter 'T'

The testing string ends with '!'

The testing string contains the letter 'a'

The testing string contains more than 2 letters
```

LIKE Example Comparisons


SQLQuery1.sql - bender.net.nait.ca,24680.skelemen1 (skelemen1 (82))* - Microsoft SQL Server Management Studio (Administrator)
File Edit View Query Project Debug Tools Window Help
New Query
skelemen1
Execute
Debug
Object Explorer
Connect
bender.net.nait.ca,24680
Databases
Security
Server Objects
Replication
AlwaysOn High Avai
Management
Integration Services
SQLQuery1.sql - be...n1 (skelemen1 (82))*
100 %
Messages
Query executed successfully.
bender.net.nait.ca,24680 (1...
skelemen1 (82)
skelemen1
00:00:00
0 rows
Ready
Ln 9
Col 33
Ch 33
INS

```sql
declare @testingString varchar(50) = 'This is a testing string!'

if @testingString like '%[bdnqz]%'
    print 'The testing string contains at least one of the letters ''b'', ''d'', ''n'', ''q'', ''z'''
else
    print 'The testing string does not contain any of the letters ''b'', ''d'', ''n'', ''q'', ''z'''
print ''

if @testingString like '%[b-d]%'
    print 'The testing string contains at least one letter in the range from ''b'' to ''d'''
else
    print 'The testing string does not contain any letters in the range from ''b'' to ''d'''
print ''

if @testingString like '%[^b-d]%'
    print 'The testing string does not contain any letters in the range from ''b'' to ''d'''
else
    print 'The testing string contains at least one letter in the range from ''b'' to ''d'''
print ''
```

```
The testing string contains at least one of the letters 'b', 'd', 'n', 'q', 'z'

The testing string does not contain any letters in the range from 'b' to 'd'

The testing string does not contain any letters in the range from 'b' to 'd'
```

Selecting values into variables

- For programming purposes, we generally want to store results of select statements into variables rather than get them displayed onto the console
- To store a selected column value into a variable (provided only one value is returned) we use:

  select @variable=column

- Note that we have to declare the variable first
- E.g.

```
use NorthwindTraders
declare @price int
select @unitprice=unitprice
from products
where productid=10

print 'price=' + cast(@unitprice as varchar)
```

Exercise 1

- For this exercise, we are going to use your version of Northwind
- Create a **MyProducts** table having the following columns:
  - ProductId int primary key
  - Product name varchar (30) not null
  - Unitprice smallmoney not null
  - UnitsInStock int
  - CategoryName varchar(30)
- Insert the following records into the file (the values are for the columns in the order they appear above):
  1. 'Pepsi-Can' 1.25 12 'Beverages'
  2. 'Canada Dry-Can' 1.50 9 'Beverages'
  3. 'Cauliflower' 2.50 29 'Vegetables'
  4. 'Green Beans' 3.75 18 'Vegetables'
  5. 'Gillette Shaving Cream' 4.25 35 'Cosmetics'
  6. 'Salomon Fillet' 12.25 45 'Seafood'
  7. 'Haddock Fillet' 14.25 54 'Seafood'

(cont'd on next slide)

Exercise1 –Cont’d

- Write a segment of code that performs the following:
  - Declare a variable called **@productid** of type **int** and set it to 1
  - Display a message according to the table below:

Amount In Stock	Message
< 10	Stock Very Low- Order immediately
10-19	Stock not very comfortable – Need to order next week
20-39	Stock-Comfortable- Order in 3 weeks time
>39	This item doesn’t seem to be selling enough. Need to consider a promotion

Exercise 1- contd

- Write the code using if..else, then make a copy of the code and modify it to use search case syntax. Finally make a third copy and modify it to use the simple case syntax
- Modify each of the 3 segments of code, by setting the productid to the different product ids in the table and verify that it's giving the correct results in each case

WHILE Loop Syntax

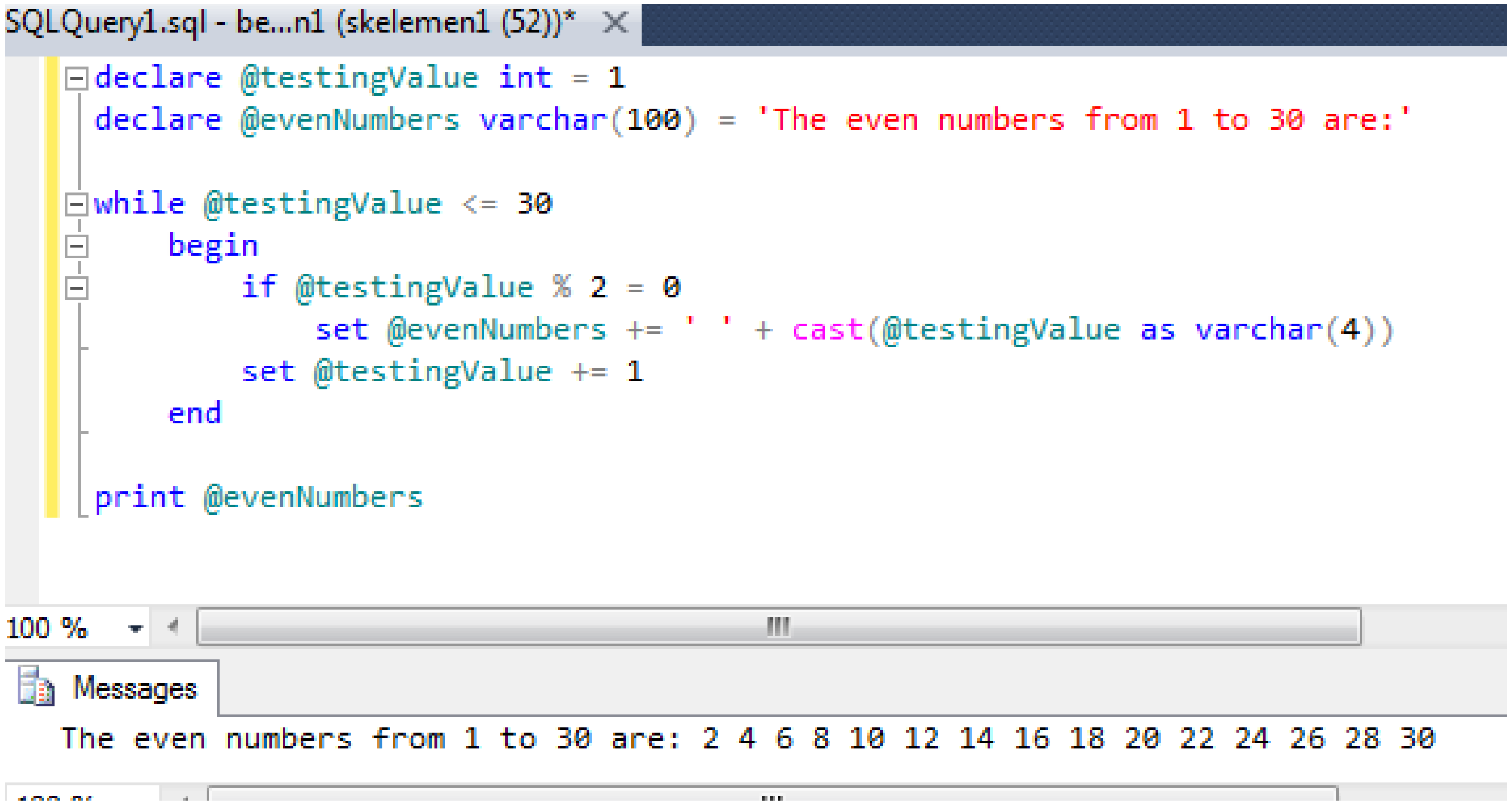
Loops containing a single statement.

```
while    booleanExpression
• statement
```

Loops containing multiple statements.

```
while    booleanExpression
begin
• multipleStatements
end
```

WHILE Loop Example


SQLQuery1.sql - be...n1 (skelemen1 (52))*

```
declare @testingValue int = 1
declare @evenNumbers varchar(100) = 'The even numbers from 1 to 30 are:'

while @testingValue <= 30
    begin
        if @testingValue % 2 = 0
            set @evenNumbers += ' ' + cast(@testingValue as varchar(4))
        set @testingValue += 1
    end

print @evenNumbers
```

100 %

Messages

```
The even numbers from 1 to 30 are: 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
```

Exercise 2

- Consider the sequence of values (2, 7, 9, 12, 14, 17, 18, 20)
- Write a segment of code that generates 20 random whole numbers in the range of 1-20 (inclusive).
- For each value generated:
  - if it is in the sequence given display:

    generated: <value> - in sequence
  - otherwise display:

    generated: <value> - not in sequence
- Eg. **generated: 9 - in sequence**

Exercise 3

- A new credit monitoring company updates your credit score every 40 days and will start on 01 April 2022. Write a segment of SQL code that will show the day of the week and the date of all the updates until the end of 2024.
- Your outputs should be in the form:

  Friday 01 Apr 2022

  Hint: Use datename to obtain the name of the day and convert with style 113 for the date format.

Exercise 4

- For this question, you will use a loop so as to develop the practice of working with loops
- Write a program that performs the following:
  - It obtains the productids of all products with ids 1-20 (one by one) from the NorthwindTraders products table and also obtains their productname, unitsinstock, unitprice and categoryname.
  - It inserts each of these records into the products table created previously.
  - When it inserts the products into the new table, the product id should start at 1 value higher than the last product id in the table and increments by 1 for each inserted product.