CSE 581 : INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS
LAB : 9
DATE : 11/9/2022

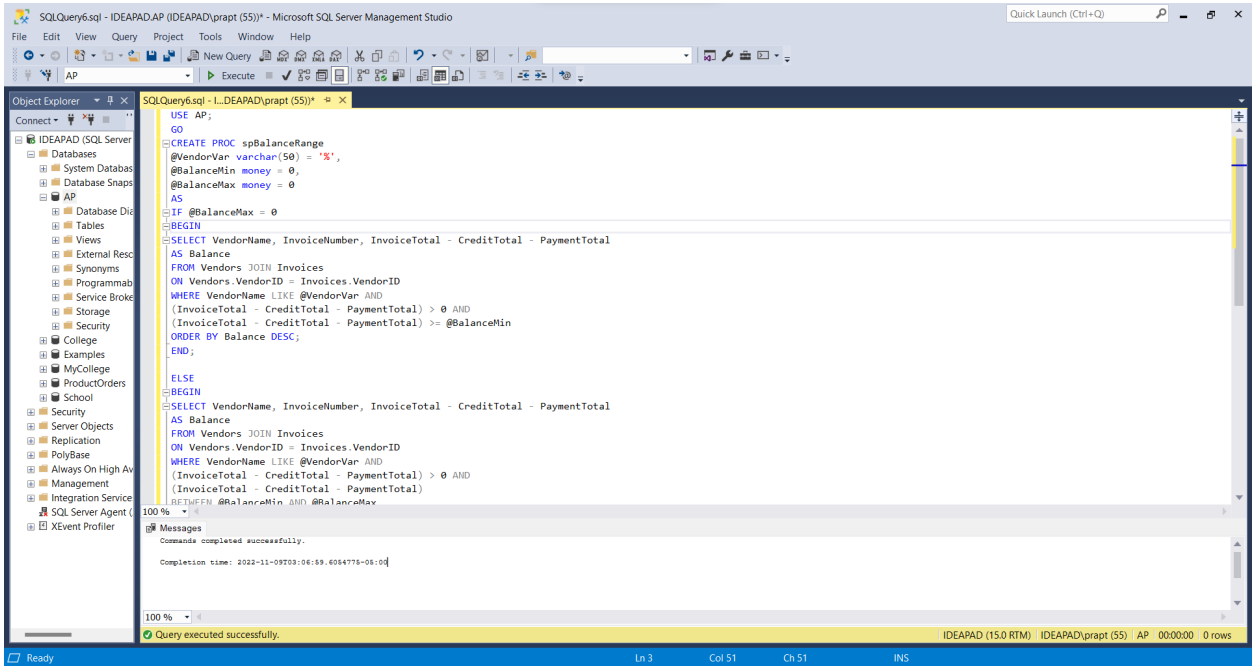
Use AP database throughout.
Balance column's definition: InvoiceTotal – CreditTotal – PaymentTotal

Q1) Create a stored procedure named spBalanceRange that accepts three optional parameters. The procedure should return a result set consisting of VendorName, InvoiceNumber, and Balance for each invoice with a balance due, sorted with the largest balance due first. The parameter @VendorVar is a mask that's used with a LIKE operator to filter by vendor name. @BalanceMin and @BalanceMax are parameters used to specify the requested range of balances due. If called with no parameters or with a maximum value of 0, the procedure should return all invoices with a balance due.

Ans:

```
USE AP;
GO
CREATE PROC spBalanceRange
@VendorVar varchar(50) = '%',
@BalanceMin money = 0,
@BalanceMax money = 0
AS
IF @BalanceMax = 0
BEGIN
SELECT VendorName, InvoiceNumber, InvoiceTotal - CreditTotal - PaymentTotal
AS Balance
FROM Vendors JOIN Invoices
ON Vendors.VendorID = Invoices.VendorID
WHERE VendorName LIKE @VendorVar AND
(InvoiceTotal - CreditTotal - PaymentTotal) > 0 AND
(InvoiceTotal - CreditTotal - PaymentTotal) >= @BalanceMin
ORDER BY Balance DESC;
END;
ELSE
BEGIN
SELECT VendorName, InvoiceNumber, InvoiceTotal - CreditTotal - PaymentTotal
AS Balance
FROM Vendors JOIN Invoices
ON Vendors.VendorID = Invoices.VendorID
WHERE VendorName LIKE @VendorVar AND
(InvoiceTotal - CreditTotal - PaymentTotal) > 0 AND
```

```
(InvoiceTotal - CreditTotal - PaymentTotal)
BETWEEN @BalanceMin AND @BalanceMax
ORDER BY Balance DESC;
END;
```



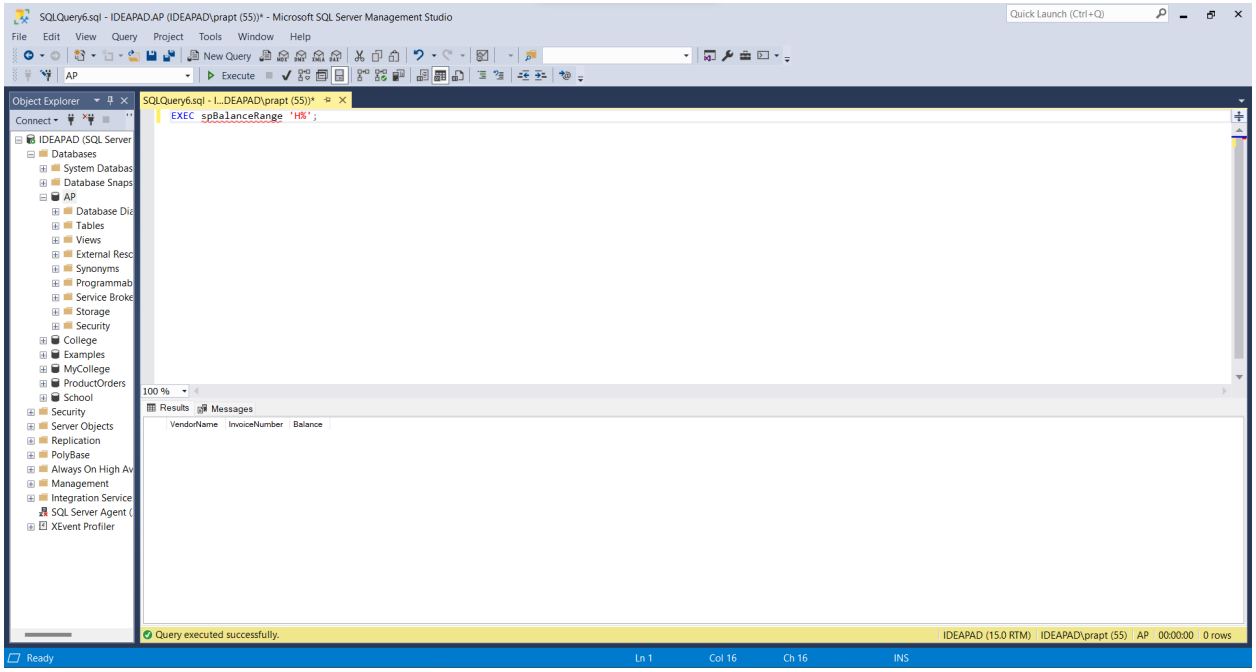
Comment: Here, CREATE PROC is used to create a new stored procedure and multiple parameters are set up namely @VendorVar with datatype varchar(50) used with %, @BalanceMin with datatype money and @Balance Max with datatype money. IF is used to condition that if there are no parameters or @BalanceMax = 0 then return all invoices with a balance due which is done using a SELECT statement. The SELECT statement retrieves the VendorName, InvoiceNumber, and the calculated value of InvoiceTotal - CreditTotal - PaymentTotal from Vendors and Invoices using the JOIN keyword on the VendorID column. A condition is made using the WHERE clause to return only those records whose Balance is due i.e., more than 0 and Balance is more than or equal to the @BalanceMin. This is sorted in descending order to show the largest balance due first on Balance.

Q2) Code three calls to the procedure created in question 1:

A. passed by position with @VendorVar = 'H%' and no balance range
B. passed by name with @VendorVar omitted and a balance range from $100 to $500
C. passed by position with a balance due from $400 to $600, filtering for vendors whose names begin with H or M.

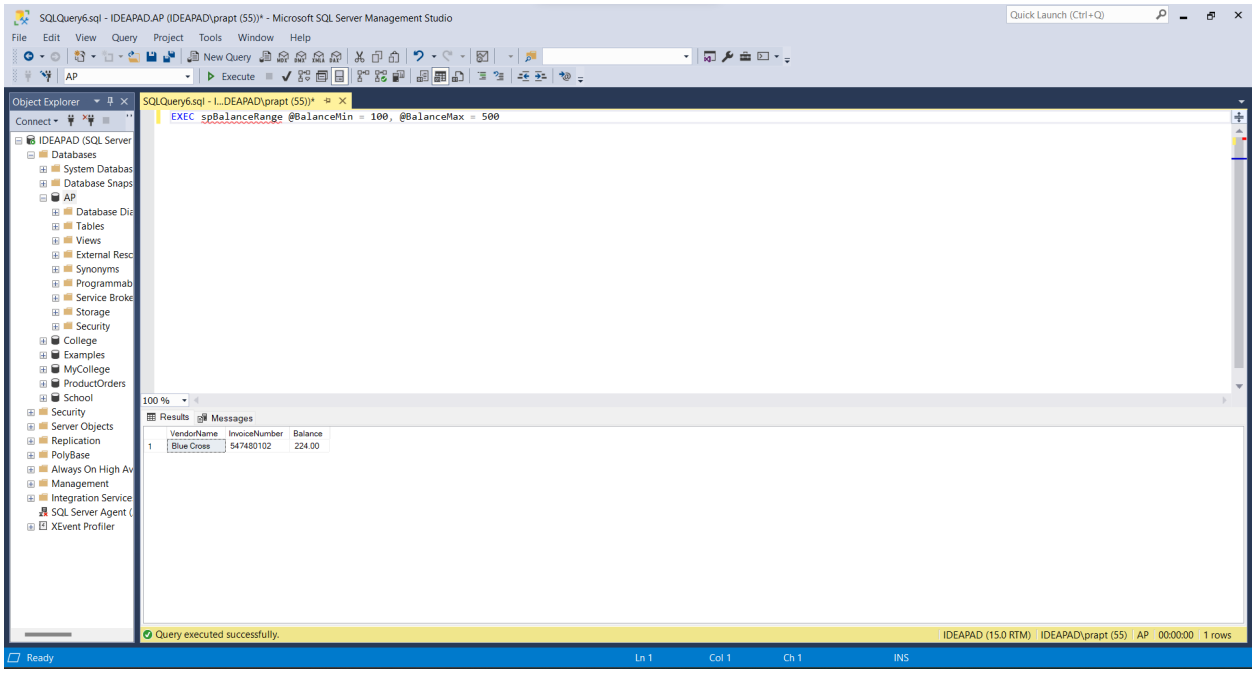
Ans:

A. EXEC spBalanceRange 'H%';



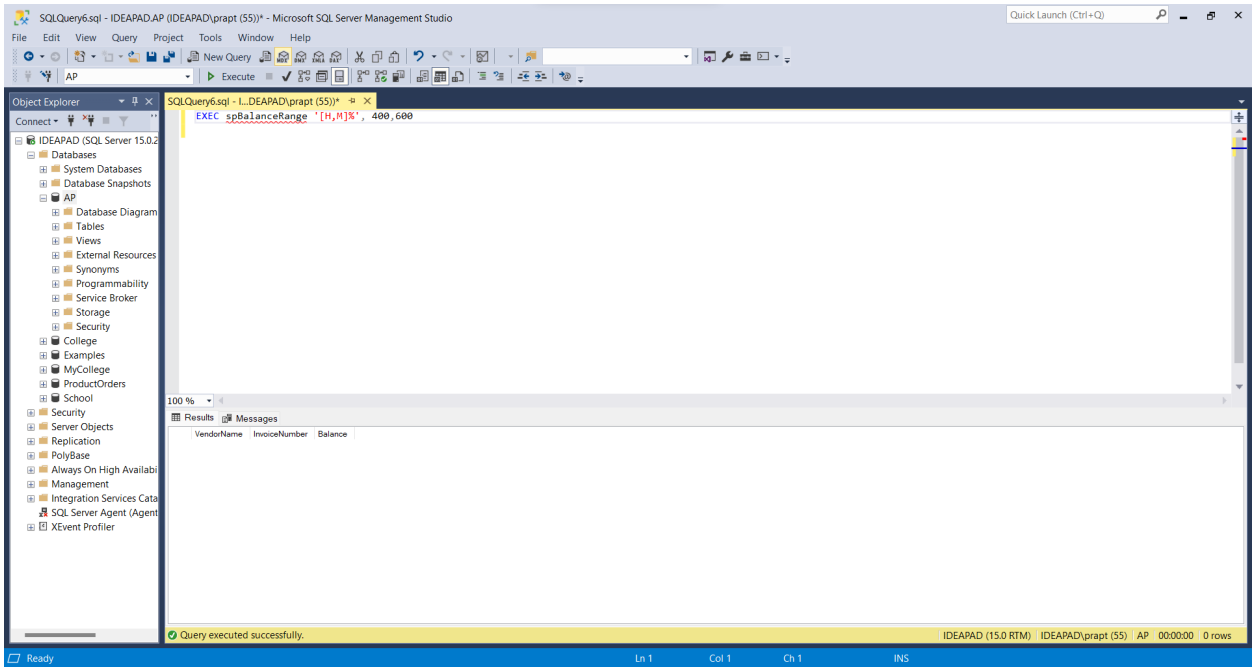
Comment A: Here, EXEC is used to execute the stored procedure that has already been created named spBalanceRange and @VendorVar as 'H%' and no balance.

B. EXEC spBalanceRange @BalanceMin = 100, @BalanceMax = 500



Comment B: Here, EXEC is used to execute the stored procedure spBalanceRange that has already been created. VendorName is omitted and @BalanceMin is 100, @BalanceMax is 500 as the balance range is $100 to $500.

C. EXEC spBalanceRange '[H,M]%', 400,600



Comment C: Here, EXEC is used to execute the stored procedure spBalanceRange that has already been created. VendorName is any name that begin with letters from H to M i.e., '[H-M]%' and @BalanceMin is 400, @BalanceMax is 600 as the balance range is $400 to $600.

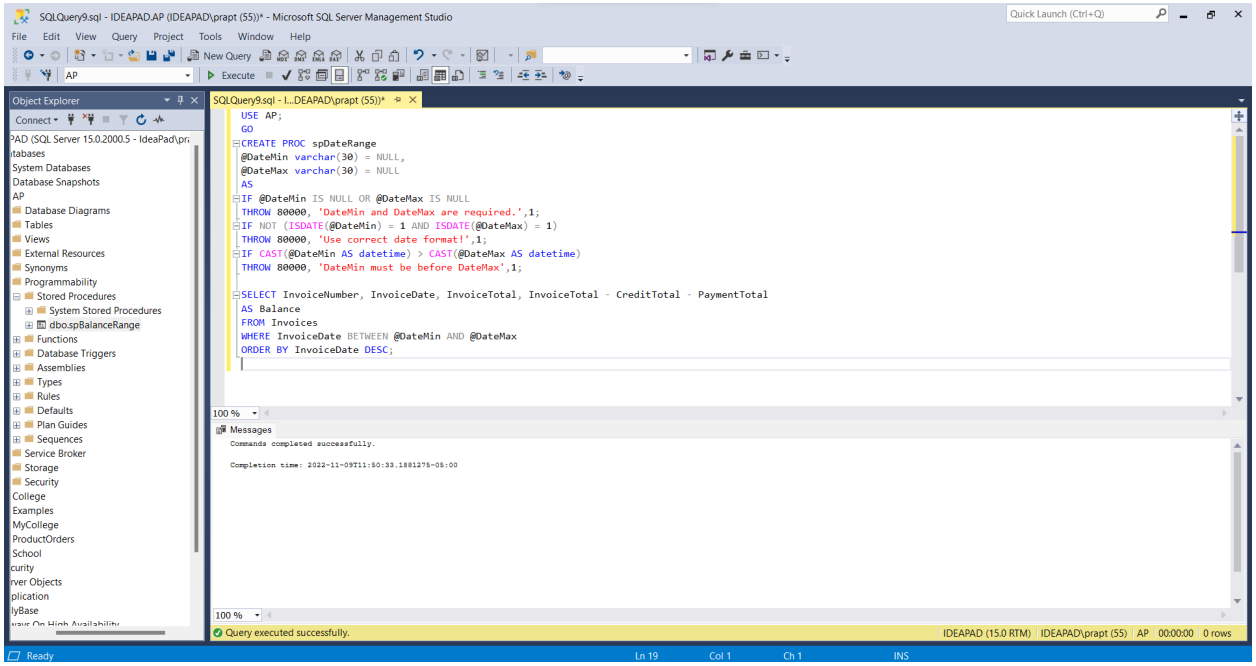
Q3) Create a stored procedure named spDateRange that accepts two parameters, @DateMin, and @DateMax, with data type varchar and default value null. If called with no parameters or with null values, raise an error that describes the problem. If called with non-null values, validate the parameters. Test that the literal strings are valid dates and test that @DateMin is earlier than @DateMax. If the parameters are valid, return a result set that includes the InvoiceNumber, InvoiceDate, InvoiceTotal, and Balance for each invoice for which the InvoiceDate is within the date range, sorted with the latest invoice first.

Ans:

```
USE AP;
GO
CREATE PROC spDateRange
@DateMin varchar(30) = NULL,
@DateMax varchar(30) = NULL
AS
IF @DateMin IS NULL OR @DateMax IS NULL
THROW 80000, 'DateMin and DateMax are required.',1;
IF NOT (ISDATE(@DateMin) = 1 AND ISDATE(@DateMax) = 1)
THROW 80000, 'Use correct date format!',1;
IF CAST(@DateMin AS datetime) > CAST(@DateMax AS datetime)
```

```
THROW 80000, 'DateMin must be before DateMax',1;

SELECT InvoiceNumber, InvoiceDate, InvoiceTotal, InvoiceTotal - CreditTotal - PaymentTotal
AS Balance
FROM Invoices
WHERE InvoiceDate BETWEEN @DateMin AND @DateMax
ORDER BY InvoiceDate DESC;
```



Comment: Here, CREATE PROC is used to create a stored procedure named as spDateRange and given parameters like @DateMin with the datatype varchar(30) and set as NULL, @DateMax with data type varchar(30) and set as NULL. IF is used to condition that if @DateMin or @DateMax is null then throw an error message which is done using the THROW statement which is used to raise an exception and show the message that dates are required. THROW statement is followed by an error number which can be any number greater than 50000 followed by an error message and state. IF NOT is used to condition whether the dates are in correct format and if not then error message is thrown to use the correct date format. Finally, IF is used to check whether the DateMin is greater than the DateMax and if yes then error is raised and message is shown that the DateMin must be greater than the DateMax. SELECT statement is used to display the InvoiceNumber, InvoiceDate, InvoiceTotal, calculated value as Balance from Invoices table and condition is made using the WHERE clause to get the records whose InvoiceDate is between the min and max which is done using BETWEEN operator and is sorted in descending order on InvoiceDate.

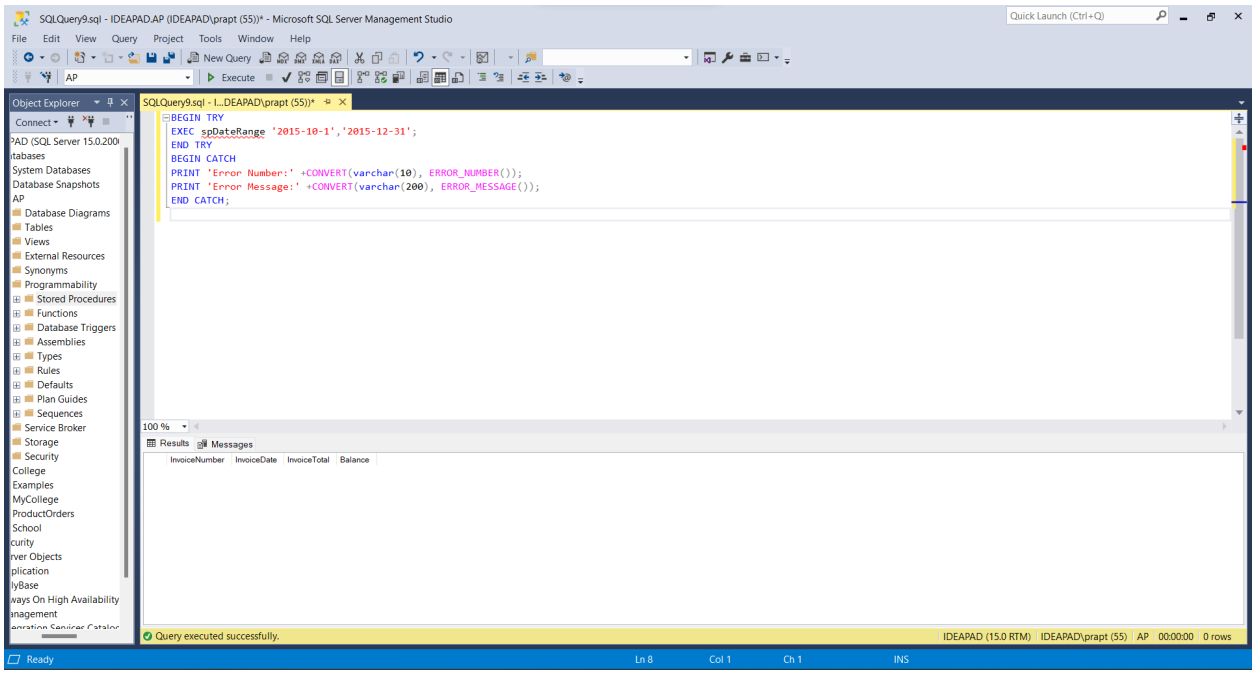
Q4) Code (1) A call to the stored procedure created in question 3 that returns invoices with an InvoiceDate between October 1 and December 31, 2015, (2) A call to the stored procedure again

that returns invoices with an @DateMin is December 10. These calls should also catch any errors that are raised by the procedure and print the error number and description.

Ans:

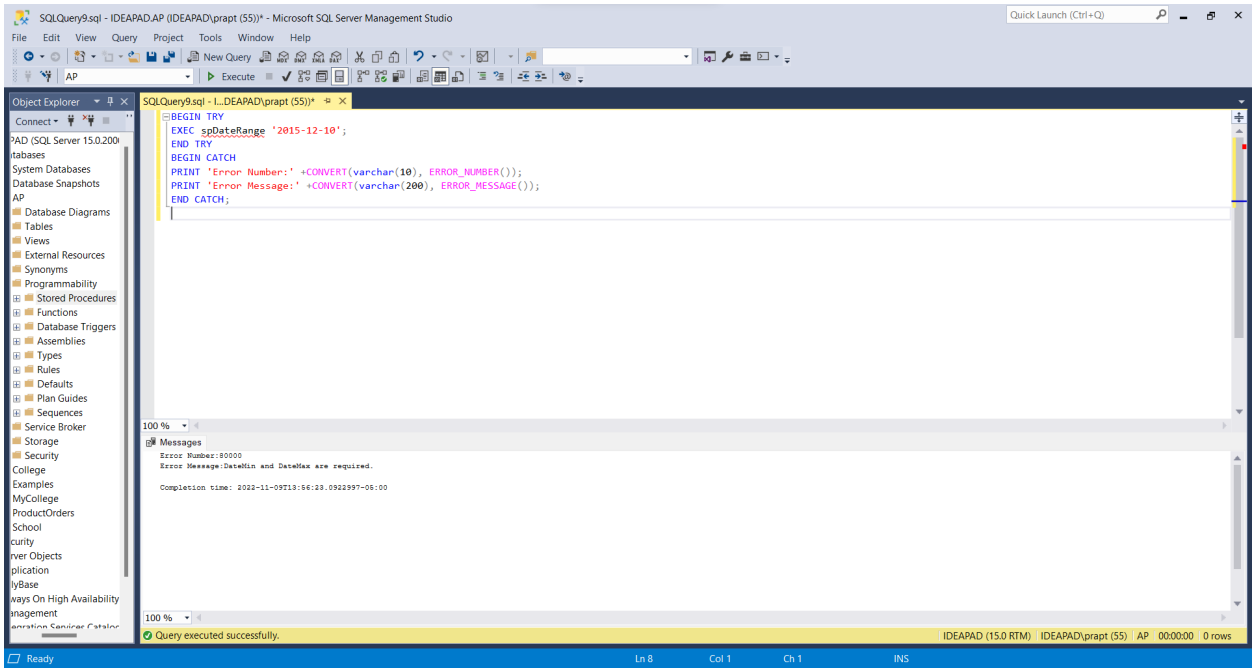
(1)

```
BEGIN TRY
EXEC spDateRange '2015-10-1','2015-12-31';
END TRY
BEGIN CATCH
PRINT 'Error Number:' +CONVERT(varchar(10), ERROR_NUMBER());
PRINT 'Error Message:' +CONVERT(varchar(200), ERROR_MESSAGE());
END CATCH;
```



(2)

```
BEGIN TRY
EXEC spDateRange '2015-12-10';
END TRY
BEGIN CATCH
PRINT 'Error Number:' +CONVERT(varchar(10), ERROR_NUMBER());
PRINT 'Error Message:' +CONVERT(varchar(200), ERROR_MESSAGE());
END CATCH;
```



Comment: (1)Here, BEGIN and END is used to indicate a block so the statements inside are executed together. BEGIN and END block is used to write a TRY block. EXEC is used to execute the stored procedure that has been previously created named spDateRange ranging between '2015-10-1' and '2015-12-31'. BEGIN and END block is used to write a CATCH block immediately after the TRY block. Error number and Error message are printed otherwise.
(2)Here, BEGIN and END is used to indicate a block so the statements inside are executed together. BEGIN and END block is used to write a TRY block. EXEC is used to execute the stored procedure that has been previously created named spDateRange where DateMin is '2015-12-10'. BEGIN and END block is used to write a CATCH block immediately after the TRY block. Error number and Error message are printed otherwise.

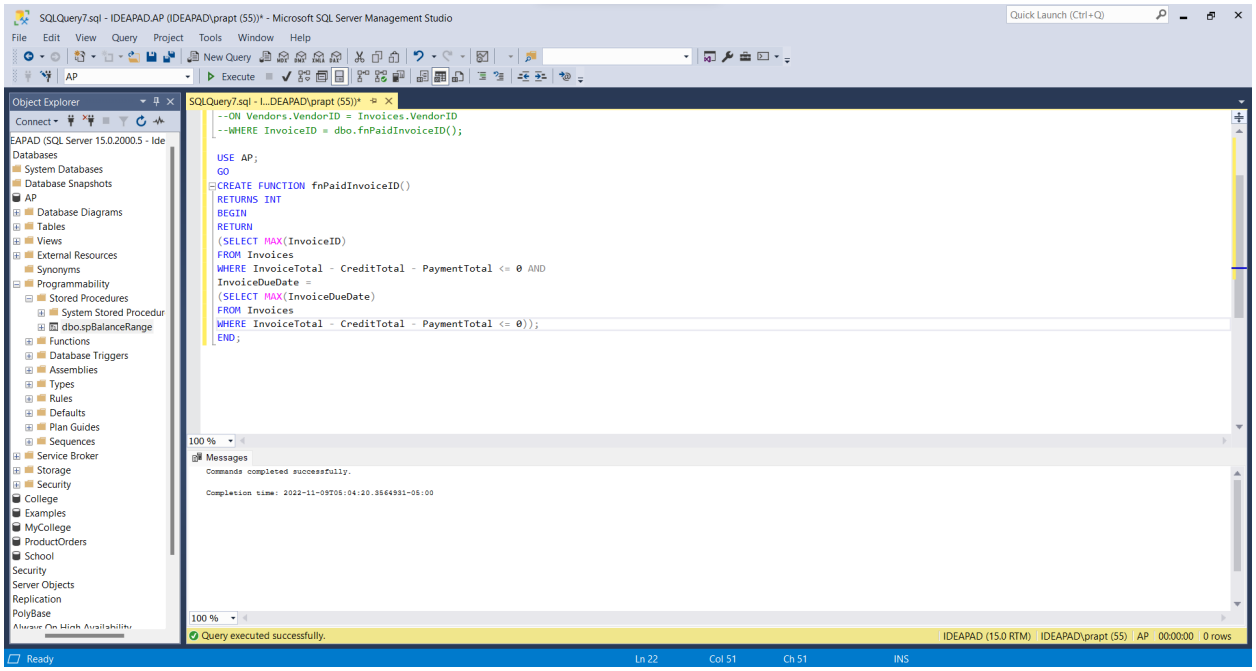
Q5) Create a scalar-valued function named fnPaidInvoiceID that returns the InvoiceID of the latest invoice with an paid balance. Test the function in the following SELECT statement:

```
SELECT VendorName, InvoiceNumber, InvoiceDueDate,
        InvoiceTotal - CreditTotal - PaymentTotal AS Balance
FROM Vendors JOIN Invoices
ON Vendors.VendorID = Invoices.VendorID
WHERE InvoiceID = dbo.fnPaidInvoiceID();
```

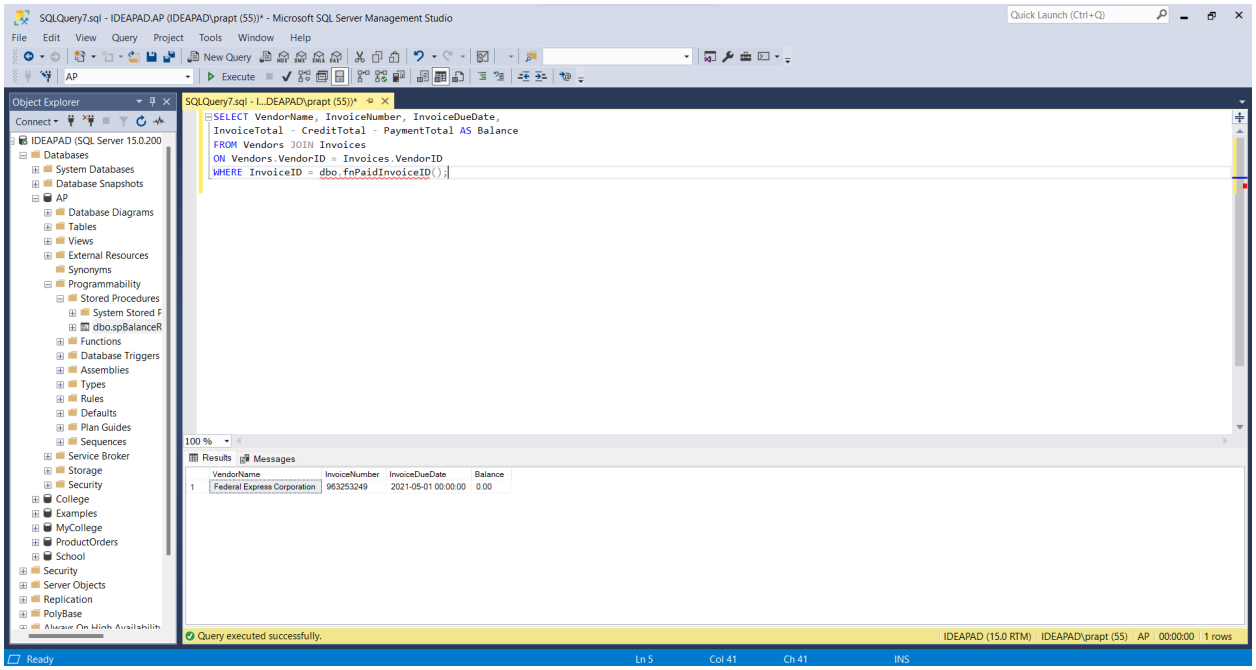
Ans:

```
USE AP;
GO
CREATE FUNCTION fnPaidInvoiceID()
```

```
RETURNS INT
BEGIN
RETURN
(SELECT MAX(InvoiceID)
FROM Invoices
WHERE InvoiceTotal - CreditTotal - PaymentTotal <= 0 AND
InvoiceDueDate =
(SELECT MAX(InvoiceDueDate)
FROM Invoices
WHERE InvoiceTotal - CreditTotal - PaymentTotal <= 0));
END;
```




```
SELECT VendorName, InvoiceNumber, InvoiceDueDate,
InvoiceTotal - CreditTotal - PaymentTotal AS Balance
FROM Vendors JOIN Invoices
ON Vendors.VendorID = Invoices.VendorID
WHERE InvoiceID = dbo.fnPaidInvoiceID();
```

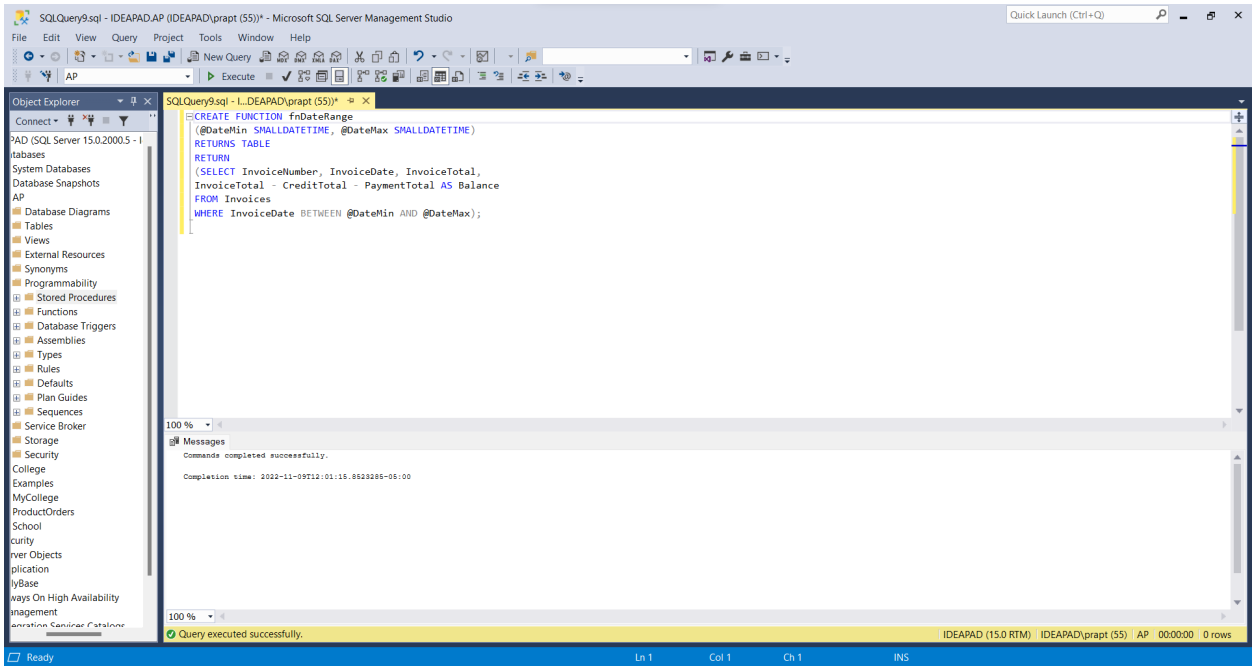


Comment: Here, scalar-valued is a function that returns a single value. CREATE FUNCTION statement is used to create a new function followed by its name that is fnPaidIvoiceID which returns an integer. TO return the InoiveID of the latestinvoice with a paid balance, SELECT statement is used to display the maximum value of the InvoiceID column calculatedd using the MAX() aggregate function from the Invoices table. WHERE clause is used to find the maximum InvoiceID from only those records whose Balance is not due and InvoiceDueDate is maximum from the dates with no balance due which is done using a subquery.

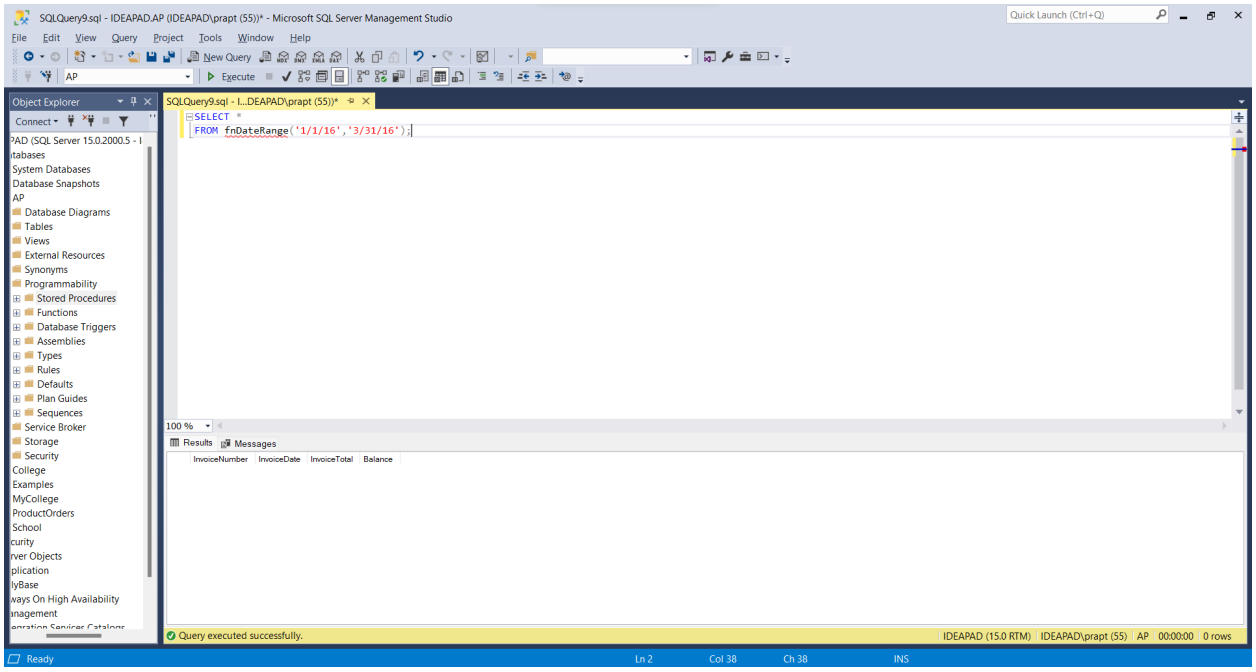
Q6) Create a table-valued function named fnDateRange, similar to the stored procedure of question 3. The function requires two parameters of data type smalldatetime. Don't validate the parameters. Return a result set that includes the InvoiceNumber, InvoiceDate, InvoiceTotal, and Balance for each invoice for which the InvoiceDate is within the date range. Invoke the function from within a SELECT statement to return those invoices with InvoiceDate between January 1 and March 31, 2016.

Ans:

```
CREATE FUNCTION fnDateRange
(@DateMin SMALLDATETIME, @DateMax SMALLDATETIME)
RETURNS TABLE
RETURN
(SELECT InvoiceNumber, InvoiceDate, InvoiceTotal,
InvoiceTotal - CreditTotal - PaymentTotal AS Balance
FROM Invoices
WHERE InvoiceDate BETWEEN @DateMin AND @DateMax);
```



SELECT *
FROM fnDateRange('1/1/16','3/31/16');



Comment:

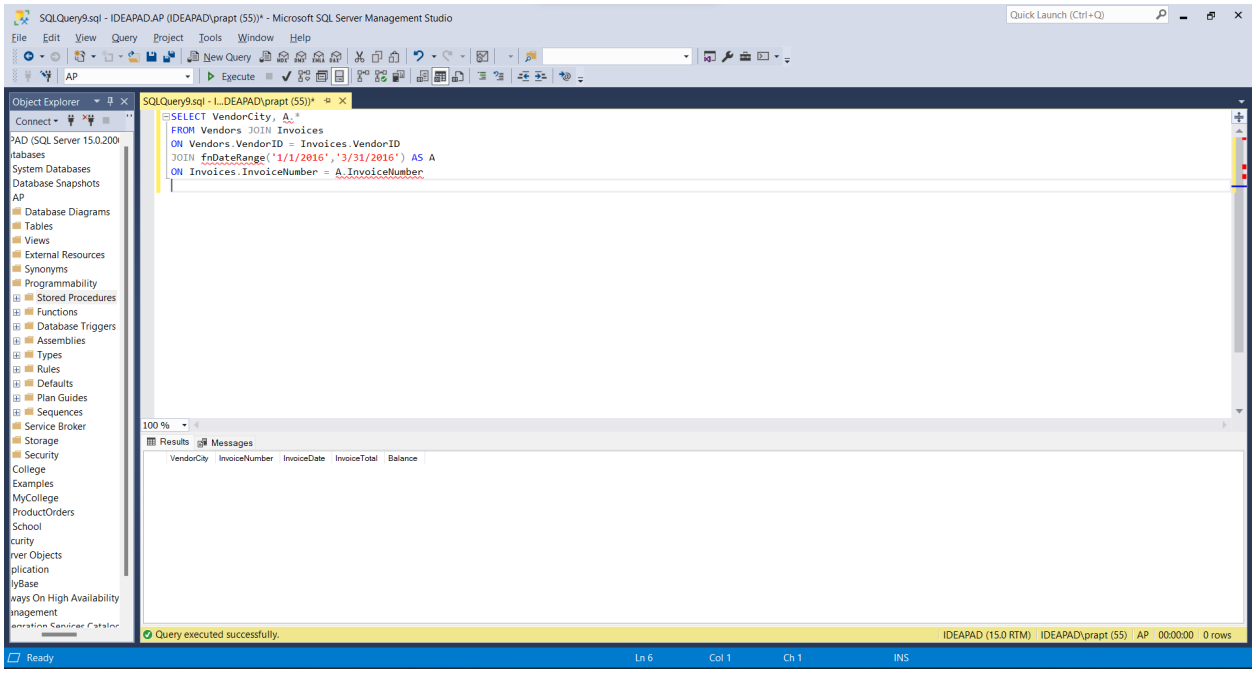
Here, a table-valued function is a function that return a table. CREATE FUNCTION is used to create a table-valued function named fnDateRange with two parameter @DateMin and @DateMax both of smalldatetime type and this function returns a table. The table returned is shown using a SELECT statement that retrieves InvoiceNumber, InvoiceDate, InvoiceTotal,

Balance from Invoices. WHERE clause is used to condition to get the records for which the InvoiceDate is between the DateMin and DateMax range. To invoke the function, a SELECT statement is used where the DateMin is January 1,2016 and DateMax is March 31,2016.

Q7) Use the function you created in question 6 in a SELECT statement that returns five columns: VendorCity and the four columns returned by the function.

Ans:

```
SELECT VendorCity, A.*
FROM Vendors JOIN Invoices
ON Vendors.VendorID = Invoices.VendorID
JOIN fnDateRange('1/1/2016','3/31/2016') AS A
ON Invoices.InvoiceNumber = A.InvoiceNumber
```



Comment: Here, a SELECT statement is used to retrieve VendorCity and four other columns returned by the fnDateRange function done using table aliasing from the Vendors and Invoices table using JOIN keyword on VendorID and joining fnDateRange result set on InvoiceNumber.

Remarks: In this lab, concepts related to stored procedures, errors, TRY, CATCH, THROW blocks, error messages, types of functions are used.