Lab 05: Stored Procedures, User-define Functions ,and Triggers

※\(^o^)/※

September, 2019

Contents

Introduction

This lab aims to help students get used to stored procedures, user-defined functions and triggers in T-SQL.

Lab Activities

Stored Procedures

A stored procedure with no parameter

Create the procedure

```
USE AccountPayables;
IF OBJECT_ID('spInvoiceReport') IS NOT NULL
  DROP PROC spInvoiceReport;
GO

CREATE PROC spInvoiceReport
AS

SELECT VendorName, InvoiceNumber, InvoiceDate, InvoiceTotal
FROM Invoices JOIN Vendors
     ON Invoices.VendorID = Vendors.VendorID
WHERE InvoiceTotal - CreditTotal - PaymentTotal > 0
ORDER BY VendorName;
GO
```

Test the procedure

```
USE AccountPayables;
EXEC spInvoiceReport;
GO
```

A stored procedure with input and output parameters

Create the procedure

```
USE AccountPayables;
IF OBJECT_ID('spInvTotal3') IS NOT NULL
    DROP PROC spInvTotal3;
GO

CREATE PROC spInvTotal3
       @InvTotal money OUTPUT,
       @DateVar smalldatetime = NULL,
       @VendorVar varchar(40) = '%'
AS

IF @DateVar IS NULL
   SELECT @DateVar = MIN(InvoiceDate) FROM Invoices;

SELECT @InvTotal = SUM(InvoiceTotal)
FROM Invoices JOIN Vendors
    ON Invoices.VendorID = Vendors.VendorID
WHERE (InvoiceDate >= @DateVar) AND
      (VendorName LIKE @VendorVar);
```

Test the procedure with parameters passed by position

```
USE AccountPayables;
DECLARE @MyInvTotal money;
EXEC spInvTotal3 @MyInvTotal OUTPUT, '2016-02-01', 'P%';

PRINT '$' + CONVERT(varchar,@MyInvTotal,1);
```

Test the procedure with parameters passed by name

```
USE AccountPayables;
DECLARE @MyInvTotal money;
EXEC spInvTotal3 @DateVar = '2016-02-01', @VendorVar = 'P%',
    @InvTotal = @MyInvTotal OUTPUT;

PRINT '$' + CONVERT(varchar,@MyInvTotal,1);
```

A stored procedure with return

Create the procedure

```
USE AccountPayables;
IF OBJECT_ID('spInvCount') IS NOT NULL
    DROP PROC spInvCount;
GO

CREATE PROC spInvCount
       @DateVar smalldatetime = NULL,
       @VendorVar varchar(40) = '%'
AS

IF @DateVar IS NULL
   SELECT @DateVar = MIN(InvoiceDate) FROM Invoices;

```

```sql
DECLARE @InvCount int;

SELECT @InvCount = COUNT(InvoiceID)
FROM Invoices JOIN Vendors
    ON Invoices.VendorID = Vendors.VendorID
WHERE (InvoiceDate >= @DateVar) AND
       (VendorName LIKE @VendorVar);

RETURN @InvCount;
```

Test the procedure

```sql
USE AccountPayables;
DECLARE @InvCount int;
EXEC @InvCount = spInvCount '2016-02-01', 'P%';
PRINT 'Invoice count: ' + CONVERT(varchar, @InvCount);
```

A stored procedure for inserting invoices with data validation

Create the procedure

```sql
USE AccountPayables;
IF OBJECT_ID('spInsertInvoice') IS NOT NULL
    DROP PROC spInsertInvoice;
GO

CREATE PROC spInsertInvoice
       @VendorID       int = NULL,
       @InvoiceNumber  varchar(50) = NULL,
       @InvoiceDate    smalldatetime = NULL,
       @InvoiceTotal   money = NULL,
       @PaymentTotal   money = NULL,
       @CreditTotal    money = NULL,
       @TermsID        int = NULL,
       @InvoiceDueDate smalldatetime = NULL,
       @PaymentDate    smalldatetime = NULL
AS

IF NOT EXISTS (SELECT * FROM Vendors WHERE VendorID = @VendorID)
    THROW 50001, 'Invalid VendorID.', 1;
IF @InvoiceNumber IS NULL
    THROW 50001, 'Invalid InvoiceNumber.', 1;
IF @InvoiceDate IS NULL OR @InvoiceDate > GETDATE()
        OR DATEDIFF(dd, @InvoiceDate, GETDATE()) > 30
    THROW 50001, 'Invalid InvoiceDate.', 1;
IF @InvoiceTotal IS NULL OR @InvoiceTotal <= 0
```

```
    THROW 50001, 'Invalid InvoiceTotal.', 1;
IF @PaymentTotal IS NULL
    SET @PaymentTotal = 0;
IF @CreditTotal IS NULL
    SET @CreditTotal = 0;
IF @CreditTotal > @InvoiceTotal
    THROW 50001, 'Invalid CreditTotal.', 1;
IF @PaymentTotal > @InvoiceTotal - @CreditTotal
    THROW 50001, 'Invalid PaymentTotal.', 1;
IF NOT EXISTS (SELECT * FROM Terms WHERE TermsID = @TermsID)
    IF @TermsID IS NULL
        SELECT @TermsID = DefaultTermsID
        FROM Vendors
        WHERE VendorID = @VendorID;
    ELSE  -- @TermsID IS NOT NULL
        THROW 50001, 'Invalid TermsID.', 1;
IF @InvoiceDueDate IS NULL
    SET @InvoiceDueDate = @InvoiceDate +
        (SELECT TermsDueDays FROM Terms WHERE TermsID = @TermsID);
ELSE  -- @InvoiceDueDate IS NOT NULL
    IF @InvoiceDueDate < @InvoiceDate OR
            DATEDIFF(dd, @InvoiceDueDate, @InvoiceDate) > 180
        THROW 50001, 'Invalid InvoiceDueDate.', 1;
IF @PaymentDate < @InvoiceDate OR
        DATEDIFF(dd, @PaymentDate, GETDATE()) > 14
    THROW 50001, 'Invalid PaymentDate.', 1;

INSERT Invoices
VALUES (@VendorID, @InvoiceNumber, @InvoiceDate, @InvoiceTotal,
        @PaymentTotal, @CreditTotal, @TermsID, @InvoiceDueDate,
        @PaymentDate);
RETURN @@IDENTITY;
GO
```

Test the procedure

```
USE AccountPayables;
BEGIN TRY
    DECLARE @InvoiceID int;
    EXEC @InvoiceID = spInsertInvoice
         @VendorID = 799,
         @InvoiceNumber = 'RZ99381',
         @InvoiceDate = '2016-04-12',
         @InvoiceTotal = 1292.45;
    PRINT 'Row was inserted.';
    PRINT 'New InvoiceID: ' + CONVERT(varchar, @InvoiceID);
END TRY
```

```sql
BEGIN CATCH
    PRINT 'An error occurred. Row was not inserted.';
    PRINT 'Error number: ' + CONVERT(varchar, ERROR_NUMBER());
    PRINT 'Error message: ' + CONVERT(varchar, ERROR_MESSAGE());
END CATCH;
GO
```

Passing tables to stored procedures

Create the procedure

```sql
USE AccountPayables;
-- drop stored procedure if it exists already
IF OBJECT_ID('spInsertLineItems') IS NOT NULL
    DROP PROC spInsertLineItems;
GO

-- drop table type if it exists already
IF  EXISTS (SELECT * FROM sys.types WHERE name = 'LineItems')
    DROP TYPE LineItems;
GO

-- create the user-defined table type named LineItems
CREATE TYPE LineItems AS
TABLE
(InvoiceID         INT            NOT NULL,
InvoiceSequence    SMALLINT       NOT NULL,
AccountNo          INT            NOT NULL,
ItemAmount         MONEY          NOT NULL,
ItemDescription    VARCHAR(100)   NOT NULL,
PRIMARY KEY (InvoiceID, InvoiceSequence));
GO

-- create a stored procedure that accepts the LineItems type
CREATE PROC spInsertLineItems
    @LineItems LineItems READONLY
AS
    INSERT INTO InvoiceLineItems
    SELECT *
    FROM @LineItems;
GO
-- start snippet sp_table_passing_test
USE AccountPayables;
-- delete old line item data
DELETE FROM InvoiceLineItems WHERE InvoiceID = 114;
```

```sql

-- declare a variable for the LineItems type
DECLARE @LineItems LineItems;

-- insert rows into the LineItems variable
INSERT INTO @LineItems VALUES (114, 1, 553, 127.75, 'Freight');
INSERT INTO @LineItems VALUES (114, 2, 553, 29.25, 'Freight');
INSERT INTO @LineItems VALUES (114, 3, 553, 48.50, 'Freight');

-- execute the stored procedure
EXEC spInsertLineItems @LineItems;
-- end snippet sp_table_passing_test

-- start snippet sp_modify
-- create a store procedure
USE AccountPayables;
IF OBJECT_ID('spVendorState') IS NOT NULL
    DROP PROC spVendorState;
GO

CREATE PROC spVendorState
       @State varchar(20)
AS
SELECT VendorName
FROM Vendors
WHERE VendorState = @State;

EXEC sp_HelpText spVendorState

-- modify it
USE AccountPayables;
GO

ALTER PROC spVendorState
      @State varchar(20) = NULL
AS
IF @State IS NULL
   SELECT VendorName
   FROM Vendors;
ELSE
   SELECT VendorName
   FROM Vendors
   WHERE VendorState = @State;

EXEC sp_HelpText spVendorState
-- end snippet sp_modify

-- start snippet udf_scalar
USE AccountPayables;
IF OBJECT_ID('fnBalanceDue') IS NOT NULL
    DROP FUNCTION fnBalanceDue;
GO

CREATE FUNCTION fnBalanceDue()
    RETURNS money
BEGIN
    RETURN (SELECT SUM(InvoiceTotal - PaymentTotal - CreditTotal)
            FROM Invoices
            WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0);
END;
-- end snippet udf_scalar

-- start snippet udf_scalar_test
USE AccountPayables;
PRINT 'Balance due: $' + CONVERT(varchar, dbo.fnBalanceDue(), 1);
-- end snippet udf_scalar_test

-- start snippet udf_simple_table
USE AccountPayables;
IF OBJECT_ID('fnTopVendorsDue') IS NOT NULL
    DROP FUNCTION fnTopVendorsDue;
GO

CREATE FUNCTION fnTopVendorsDue
    (@CutOff money = 0)
    RETURNS TABLE
RETURN
        (SELECT VendorName, SUM(InvoiceTotal) AS TotalDue
        FROM Vendors JOIN Invoices ON Vendors.VendorID = Invoices.VendorID
        WHERE InvoiceTotal - CreditTotal - PaymentTotal > 0
        GROUP BY VendorName
        HAVING SUM(InvoiceTotal) >= @CutOff);
-- end snippet udf_simple_table

-- start snippet udf_simple_table_test
USE AccountPayables;
SELECT * FROM dbo.fnTopVendorsDue(5000);


USE AccountPayables;
SELECT Vendors.VendorName, VendorCity, TotalDue
FROM Vendors JOIN dbo.fnTopVendorsDue(DEFAULT) AS TopVendors
```

```sql
     ON Vendors.VendorName = TopVendors.VendorName;
-- end snippet udf_simple_table_test

-- start snippet udf_multis_table
USE AccountPayables;
GO

IF OBJECT_ID('fnCreditAdj') IS NOT NULL
    DROP FUNCTION fnCreditAdj;
GO

CREATE FUNCTION fnCreditAdj (@HowMuch money)
    RETURNS @OutTable table
           (InvoiceID int, VendorID int, InvoiceNumber varchar(50),
            InvoiceDate smalldatetime, InvoiceTotal money,
            PaymentTotal money, CreditTotal money)
BEGIN
    INSERT @OutTable
        SELECT InvoiceID, VendorID, InvoiceNumber, InvoiceDate,
               InvoiceTotal, PaymentTotal, CreditTotal
        FROM Invoices
        WHERE (InvoiceTotal - CreditTotal - PaymentTotal) > 0;
    WHILE (SELECT SUM(InvoiceTotal - CreditTotal - PaymentTotal)
           FROM @OutTable) >= @HowMuch
        UPDATE @OutTable
        SET CreditTotal = CreditTotal + .01
        WHERE (InvoiceTotal - CreditTotal - PaymentTotal) > 0;
    RETURN;
END;
-- end snippet udf_multis_table

-- start snippet udf_multis_table_test
USE AccountPayables;

SELECT VendorName, SUM(CreditTotal) AS CreditRequest
FROM Vendors JOIN dbo.fnCreditAdj(25000) AS CreditTable
     ON Vendors.VendorID = CreditTable.VendorID
GROUP BY VendorName;
-- end snippet udf_multis_table_test

-- start snippet tg_after_01
USE AccountPayables;
IF OBJECT_ID('Vendors_INSERT_UPDATE') IS NOT NULL
    DROP TRIGGER Vendors_INSERT_UPDATE;
GO

```

```sql
CREATE TRIGGER Vendors_INSERT_UPDATE
    ON Vendors
    AFTER INSERT,UPDATE
AS
    UPDATE Vendors
    SET VendorState = UPPER(VendorState)
    WHERE VendorID IN (SELECT VendorID FROM Inserted);
-- end snippet tg_after_01

-- start snippet tg_after_01_test

USE AccountPayables;
INSERT Vendors
VALUES ('Peerless Uniforms, Inc.', '785 S Pixley Rd', NULL,
        'Piqua', 'Oh', '45356', '(937) 555-8845', NULL, NULL, 4,550);
-- end snippet tg_after_01_test

-- start snippet tg_after_02
USE AccountPayables;
GO
IF OBJECT_ID('Invoices_DELETE') IS NOT NULL
    DROP TRIGGER Invoices_DELETE;
GO

CREATE TRIGGER Invoices_DELETE
        ON Invoices
        AFTER DELETE
AS
INSERT INTO InvoiceArchive
        (InvoiceID, VendorID, InvoiceNumber, InvoiceDate, InvoiceTotal,
            PaymentTotal, CreditTotal, TermsID, InvoiceDueDate, PaymentDate)
        SELECT InvoiceID, VendorID, InvoiceNumber, InvoiceDate, InvoiceTotal,
            PaymentTotal, CreditTotal, TermsID, InvoiceDueDate, PaymentDate
        FROM Deleted;
-- end snippet tg_after_02

-- start snippet tg_after_02_test
USE AccountPayables;
DELETE Invoices
WHERE VendorID = 37;

SELECT * FROM InvoiceArchive;
-- end snippet tg_after_02_test

-- start snippet tg_instead_01
USE AccountPayables;
```

```
GO

IF OBJECT_ID('IBM_Invoices') IS NOT NULL
        DROP VIEW IBM_Invoices
GO

CREATE VIEW IBM_Invoices
AS
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal
FROM Invoices
WHERE VendorID = (SELECT VendorID FROM Vendors WHERE VendorName = 'IBM');
GO

IF OBJECT_ID('IBM_Invoices_INSERT') IS NOT NULL
    DROP TRIGGER IBM_Invoices_INSERT;
GO

CREATE TRIGGER IBM_Invoices_INSERT
    ON IBM_Invoices
    INSTEAD OF INSERT
AS
DECLARE @InvoiceDate smalldatetime, @InvoiceNumber varchar(50),
        @InvoiceTotal money, @VendorID int,
        @InvoiceDueDate smalldatetime, @TermsID int,
        @DefaultTerms smallint, @TestRowCount int;
SELECT @TestRowCount = COUNT(*) FROM Inserted;
IF @TestRowCount = 1
    BEGIN
        SELECT @InvoiceNumber = InvoiceNumber, @InvoiceDate = InvoiceDate,
            @InvoiceTotal = InvoiceTotal
        FROM Inserted;
        IF (@InvoiceDate IS NOT NULL AND @InvoiceNumber IS NOT NULL AND
            @InvoiceTotal IS NOT NULL)
            BEGIN
                SELECT @VendorID = VendorID, @TermsID = DefaultTermsID
                FROM Vendors
                WHERE VendorName = 'IBM';

                SELECT @DefaultTerms = TermsDueDays
                FROM Terms
                WHERE TermsID = @TermsID;

                SET @InvoiceDueDate = @InvoiceDate + @DefaultTerms;

                INSERT Invoices
                   (VendorID, InvoiceNumber, InvoiceDate, InvoiceTotal,
```

```sql
                    TermsID, InvoiceDueDate, PaymentDate)
                VALUES (@VendorID, @InvoiceNumber, @InvoiceDate,
                   @InvoiceTotal, @TermsID, @InvoiceDueDate, NULL);
            END;
    END;
ELSE
        THROW 50027, 'Limit INSERT to a single row.', 1;
-- end snippet tg_instead_01

-- start snippet tg_instead_01_test
USE AccountPayables;
INSERT IBM_Invoices
VALUES ('RA23988', '2016-05-09', 417.34);
-- end snippet tg_instead_01_test

-- start snippet tg_dc
/*
updates the InvoiceLineItems table
to have an incorrect value for one of the line
items for InvoiceID 100.
*/
USE AccountPayables;
UPDATE InvoiceLineItems
SET InvoiceLineItemAmount = 477.79
WHERE InvoiceID = 98 AND InvoiceSequence = 1;
GO


USE AccountPayables;
IF OBJECT_ID('Invoices_UPDATE') IS NOT NULL
    DROP TRIGGER Invoices_UPDATE;
GO

CREATE TRIGGER Invoices_UPDATE
    ON Invoices
    AFTER UPDATE
AS
IF EXISTS           --Test whether PaymentTotal was changed
 (SELECT *
  FROM Deleted JOIN Invoices
    ON Deleted.InvoiceID = Invoices.InvoiceID
  WHERE Deleted.PaymentTotal <> Invoices.PaymentTotal)
  BEGIN
    IF EXISTS       --Test whether line items total and InvoiceTotal match
     (SELECT *
      FROM Invoices JOIN
```

```sql
          (SELECT InvoiceID, SUM(InvoiceLineItemAmount) AS SumOfInvoices
           FROM InvoiceLineItems
           GROUP BY InvoiceID) AS LineItems
        ON Invoices.InvoiceID = LineItems.InvoiceID
      WHERE (Invoices.InvoiceTotal <> LineItems.SumOfInvoices) AND
            (LineItems.InvoiceID IN (SELECT InvoiceID FROM Deleted))
      BEGIN
        ;
        THROW 50113, 'Correct line item amounts before posting payment.', 1;
        ROLLBACK TRAN;
      END;
  END;
-- end snippet tg_dc

-- start snippet tg_dc_test
USE AccountPayables;

UPDATE Invoices
SET PaymentTotal = 662, PaymentDate = '2016-05-09'
WHERE InvoiceID = 98;
-- end snippet tg_dc_test

-- start snippet tg_modify
USE AccountPayables;
GO

ALTER TRIGGER Vendors_INSERT_UPDATE
    ON Vendors
    AFTER INSERT,UPDATE
AS
    UPDATE Vendors
    SET VendorState = UPPER(VendorState),
        VendorAddress1 = LTRIM(RTRIM(VendorAddress1)),
        VendorAddress2 = LTRIM(RTRIM(VendorAddress2))
    WHERE VendorID IN (SELECT VendorID FROM Inserted);
-- end snippet tg_modify
```

Test the procedure

```sql
USE AccountPayables;
-- delete old line item data
DELETE FROM InvoiceLineItems WHERE InvoiceID = 114;

-- declare a variable for the LineItems type
DECLARE @LineItems LineItems;

-- insert rows into the LineItems variable
```

```sql
INSERT INTO @LineItems VALUES (114, 1, 553, 127.75, 'Freight');
INSERT INTO @LineItems VALUES (114, 2, 553, 29.25, 'Freight');
INSERT INTO @LineItems VALUES (114, 3, 553, 48.50, 'Freight');

-- execute the stored procedure
EXEC spInsertLineItems @LineItems;
```

Modify stored procedures

```sql
-- create a store procedure
USE AccountPayables;
IF OBJECT_ID('spVendorState') IS NOT NULL
    DROP PROC spVendorState;
GO

CREATE PROC spVendorState
       @State varchar(20)
AS
SELECT VendorName
FROM Vendors
WHERE VendorState = @State;

EXEC sp_HelpText spVendorState

-- modify it
USE AccountPayables;
GO

ALTER PROC spVendorState
      @State varchar(20) = NULL
AS
IF @State IS NULL
   SELECT VendorName
   FROM Vendors;
ELSE
   SELECT VendorName
   FROM Vendors
   WHERE VendorState = @State;

EXEC sp_HelpText spVendorState
```

Exercise 01

Create a procedure named spBalancedRange:

- Outputs:
  - the procedure should return a result set consisting of VendorName, InvoiceNumber, and Balance for each invoice with a balance due (InvoiceTotal - PaymentTotal - CreditTotal > 0).

  - Results should be sorted with largest balance due first.
- Inputs: three optional parameters
  - @VendorVar is a mask that's used with a LIKE operator to filter by VendorName, e.g. @VendorVar = 'K%'
  - @BalanceMin and @BalanceMax are parameters used to specify the requested range of balances due. If called with no parameters or with @BalanceMax = 0, the procedure should return all invoices with a balance due.

Exercise 02

Call the procedure from exercise 01 for the following situations:

- Passed by position with @VendorVar='M%' and no balance range
- Passed by name with @VendorVar omitted a balance range from $200 to $500
- Passed by position with a balance due that's less than $200, filtering for vendors whose name begin with C or F

User-defined Functions

A scalar-valued functions

Create the function

```sql
USE AccountPayables;
IF OBJECT_ID('fnBalanceDue') IS NOT NULL
    DROP FUNCTION fnBalanceDue;
GO

CREATE FUNCTION fnBalanceDue()
    RETURNS money
BEGIN
    RETURN (SELECT SUM(InvoiceTotal - PaymentTotal - CreditTotal)
            FROM Invoices
            WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0);
END;
```

Test the function

```
USE AccountPayables;
PRINT 'Balance due: $' + CONVERT(varchar, dbo.fnBalanceDue(), 1);
```

A simple table-valued function

Create the function

```
USE AccountPayables;
IF OBJECT_ID('fnTopVendorsDue') IS NOT NULL
    DROP FUNCTION fnTopVendorsDue;
GO

CREATE FUNCTION fnTopVendorsDue
    (@CutOff money = 0)
    RETURNS TABLE
RETURN
        (SELECT VendorName, SUM(InvoiceTotal) AS TotalDue
        FROM Vendors JOIN Invoices ON Vendors.VendorID = Invoices.VendorID
        WHERE InvoiceTotal - CreditTotal - PaymentTotal > 0
        GROUP BY VendorName
        HAVING SUM(InvoiceTotal) >= @CutOff);
```

Test the function

```
USE AccountPayables;
SELECT * FROM dbo.fnTopVendorsDue(5000);


USE AccountPayables;
SELECT Vendors.VendorName, VendorCity, TotalDue
FROM Vendors JOIN dbo.fnTopVendorsDue(DEFAULT) AS TopVendors
     ON Vendors.VendorName = TopVendors.VendorName;
```

A multi-statement table-valued function

Create the function

```
USE AccountPayables;
GO

IF OBJECT_ID('fnCreditAdj') IS NOT NULL
    DROP FUNCTION fnCreditAdj;
GO

CREATE FUNCTION fnCreditAdj (@HowMuch money)
    RETURNS @OutTable table
```

```sql
            (InvoiceID int, VendorID int, InvoiceNumber varchar(50),
             InvoiceDate smalldatetime, InvoiceTotal money,
             PaymentTotal money, CreditTotal money)
BEGIN
    INSERT @OutTable
        SELECT InvoiceID, VendorID, InvoiceNumber, InvoiceDate,
               InvoiceTotal, PaymentTotal, CreditTotal
        FROM Invoices
        WHERE (InvoiceTotal - CreditTotal - PaymentTotal) > 0;
    WHILE (SELECT SUM(InvoiceTotal - CreditTotal - PaymentTotal)
           FROM @OutTable) >= @HowMuch
        UPDATE @OutTable
        SET CreditTotal = CreditTotal + .01
        WHERE (InvoiceTotal - CreditTotal - PaymentTotal) > 0;
    RETURN;
END;
```

Test the function

```sql
USE AccountPayables;

SELECT VendorName, SUM(CreditTotal) AS CreditRequest
FROM Vendors JOIN dbo.fnCreditAdj(25000) AS CreditTable
     ON Vendors.VendorID = CreditTable.VendorID
GROUP BY VendorName;
```

Exercise 03

Create a scalar-valued function name fnUnpaidInvoiceID that returns the InvoiceID of the earliest invoice with an unpaid balance.
Use the following statement to test the function

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

Exercise 04

Create a table-valued function named fnDateRange

- Two inputs @DateMin and @DateMax, type smalldatetime
- 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 the function from within a select statement to return those invoices with InvoiceDate between December 10 and December 20, 2015.

Triggers

AFTER triggers

A trigger to correct mixed-case state names

Create the trigger

```sql
USE AccountPayables;
IF OBJECT_ID('Vendors_INSERT_UPDATE') IS NOT NULL
    DROP TRIGGER Vendors_INSERT_UPDATE;
GO

CREATE TRIGGER Vendors_INSERT_UPDATE
    ON Vendors
    AFTER INSERT,UPDATE
AS
    UPDATE Vendors
    SET VendorState = UPPER(VendorState)
    WHERE VendorID IN (SELECT VendorID FROM Inserted);
```

Test the trigger

```sql

USE AccountPayables;
INSERT Vendors
VALUES ('Peerless Uniforms, Inc.', '785 S Pixley Rd', NULL,
        'Piqua', 'Oh', '45356', '(937) 555-8845', NULL, NULL, 4,550);
```

A trigger that archive deleted data

Create the trigger

```sql
USE AccountPayables;
GO
IF OBJECT_ID('Invoices_DELETE') IS NOT NULL
    DROP TRIGGER Invoices_DELETE;
GO

CREATE TRIGGER Invoices_DELETE
        ON Invoices
        AFTER DELETE
AS
INSERT INTO InvoiceArchive
```

```sql
        (InvoiceID, VendorID, InvoiceNumber, InvoiceDate, InvoiceTotal,
            PaymentTotal, CreditTotal, TermsID, InvoiceDueDate, PaymentDate)
        SELECT InvoiceID, VendorID, InvoiceNumber, InvoiceDate, InvoiceTotal,
            PaymentTotal, CreditTotal, TermsID, InvoiceDueDate, PaymentDate
        FROM Deleted;
```

Test the trigger

```sql
USE AccountPayables;
DELETE Invoices
WHERE VendorID = 37;

SELECT * FROM InvoiceArchive;
```

INSTEAD OF triggers

An insert trigger for a view

Create the trigger

```sql
USE AccountPayables;
GO

IF OBJECT_ID('IBM_Invoices') IS NOT NULL
        DROP VIEW IBM_Invoices
GO

CREATE VIEW IBM_Invoices
AS
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal
FROM Invoices
WHERE VendorID = (SELECT VendorID FROM Vendors WHERE VendorName = 'IBM');
GO

IF OBJECT_ID('IBM_Invoices_INSERT') IS NOT NULL
    DROP TRIGGER IBM_Invoices_INSERT;
GO

CREATE TRIGGER IBM_Invoices_INSERT
    ON IBM_Invoices
    INSTEAD OF INSERT
AS
DECLARE @InvoiceDate smalldatetime, @InvoiceNumber varchar(50),
        @InvoiceTotal money, @VendorID int,
        @InvoiceDueDate smalldatetime, @TermsID int,
        @DefaultTerms smallint, @TestRowCount int;
SELECT @TestRowCount = COUNT(*) FROM Inserted;
```

```
IF @TestRowCount = 1
    BEGIN
        SELECT @InvoiceNumber = InvoiceNumber, @InvoiceDate = InvoiceDate,
            @InvoiceTotal = InvoiceTotal
        FROM Inserted;
        IF (@InvoiceDate IS NOT NULL AND @InvoiceNumber IS NOT NULL AND
            @InvoiceTotal IS NOT NULL)
            BEGIN
                SELECT @VendorID = VendorID, @TermsID = DefaultTermsID
                FROM Vendors
                WHERE VendorName = 'IBM';

                SELECT @DefaultTerms = TermsDueDays
                FROM Terms
                WHERE TermsID = @TermsID;

                SET @InvoiceDueDate = @InvoiceDate + @DefaultTerms;

                INSERT Invoices
                   (VendorID, InvoiceNumber, InvoiceDate, InvoiceTotal,
                    TermsID, InvoiceDueDate, PaymentDate)
                VALUES (@VendorID, @InvoiceNumber, @InvoiceDate,
                   @InvoiceTotal, @TermsID, @InvoiceDueDate, NULL);
            END;
    END;
ELSE
        THROW 50027, 'Limit INSERT to a single row.', 1;
```

Test the trigger

```
USE AccountPayables;
INSERT IBM_Invoices
VALUES ('RA23988', '2016-05-09', 417.34);
```

Use triggers for enforcing data consistency

Create the trigger

```
/*
updates the InvoiceLineItems table
to have an incorrect value for one of the line
items for InvoiceID 100.
*/
USE AccountPayables;
UPDATE InvoiceLineItems
SET InvoiceLineItemAmount = 477.79
```

```
WHERE InvoiceID = 98 AND InvoiceSequence = 1;
GO


USE AccountPayables;
IF OBJECT_ID('Invoices_UPDATE') IS NOT NULL
    DROP TRIGGER Invoices_UPDATE;
GO

CREATE TRIGGER Invoices_UPDATE
    ON Invoices
    AFTER UPDATE
AS
IF EXISTS           --Test whether PaymentTotal was changed
 (SELECT *
  FROM Deleted JOIN Invoices
    ON Deleted.InvoiceID = Invoices.InvoiceID
  WHERE Deleted.PaymentTotal <> Invoices.PaymentTotal)
  BEGIN
    IF EXISTS       --Test whether line items total and InvoiceTotal match
     (SELECT *
      FROM Invoices JOIN
          (SELECT InvoiceID, SUM(InvoiceLineItemAmount) AS SumOfInvoices
           FROM InvoiceLineItems
           GROUP BY InvoiceID) AS LineItems
        ON Invoices.InvoiceID = LineItems.InvoiceID
      WHERE (Invoices.InvoiceTotal <> LineItems.SumOfInvoices) AND
            (LineItems.InvoiceID IN (SELECT InvoiceID FROM Deleted))
      BEGIN
        ;
        THROW 50113, 'Correct line item amounts before posting payment.', 1;
        ROLLBACK TRAN;
      END;
  END;
```

Test the trigger

```
USE AccountPayables;

UPDATE Invoices
SET PaymentTotal = 662, PaymentDate = '2016-05-09'
WHERE InvoiceID = 98;
```

Modify triggers

```
USE AccountPayables;
GO

ALTER TRIGGER Vendors_INSERT_UPDATE
    ON Vendors
    AFTER INSERT,UPDATE
AS
    UPDATE Vendors
    SET VendorState = UPPER(VendorState),
        VendorAddress1 = LTRIM(RTRIM(VendorAddress1)),
        VendorAddress2 = LTRIM(RTRIM(VendorAddress2))
    WHERE VendorID IN (SELECT VendorID FROM Inserted);
```

Exercise 05

Create a trigger for the Invoices table that automatically inserts the vendor name and address for a paid invoice into a table named ShippingLabels.
The trigger should fire any time the PaymentTotal column of the Invoices table is updated.
The ShippingLabels table could be defined as:

```
CREATE TABLE ShippingLabels
(
VendorName varchar(50),
VendorAddress1 varchar(50),
VendorAddress2 varchar(50),
VendorCity varchar(50),
VendorState char(2),
VendorZipCode varchar(20)
);
```

Test the trigger with the following statement:

```
UPDATE Invoices
SET PaymentTotal = 67.92, PaymentDate = '2016-04-23'
WHERE InvoiceID = 100;
```

Exercise 06

Write a trigger that prohibits duplicate values except for nulls in the NoDupName column of the following table:

```
CREATE TABLE TestUniqueNulls
(
RowID int IDENTITY NOT NULL,
NoDupName varchar(20) NULL
);
```

If an INSERT of UPDATE statement creates a duplicate value, rollback the statement and return an error message.
Write some INSERT statements to test that duplicate null values are allowed but duplicates of other values are not.