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

```
5   USE AccountPayables;
6   IF OBJECT_ID('spInvoiceReport') IS NOT NULL
7     DROP PROC spInvoiceReport;
8   GO
9
10  CREATE PROC spInvoiceReport
11  AS
12
13  SELECT VendorName, InvoiceNumber, InvoiceDate, InvoiceTotal
14  FROM Invoices JOIN Vendors
15      ON Invoices.VendorID = Vendors.VendorID
16  WHERE InvoiceTotal - CreditTotal - PaymentTotal > 0
17  ORDER BY VendorName;
18  GO
```

Test the procedure

```
22  USE AccountPayables;
23  EXEC spInvoiceReport;
24  GO
```

A stored procedure with input and output parameters

Create the procedure

```
28  USE AccountPayables;
29  IF OBJECT_ID('spInvTotal3') IS NOT NULL
30      DROP PROC spInvTotal3;
31  GO
32
33  CREATE PROC spInvTotal3
34      @InvTotal money OUTPUT,
```

```sql
35          @DateVar smalldatetime = NULL,
36          @VendorVar varchar(40) = '%'
37     AS
38
39     IF @DateVar IS NULL
40         SELECT @DateVar = MIN(InvoiceDate) FROM Invoices;
41
42     SELECT @InvTotal = SUM(InvoiceTotal)
43     FROM Invoices JOIN Vendors
44         ON Invoices.VendorID = Vendors.VendorID
45     WHERE (InvoiceDate >= @DateVar) AND
46           (VendorName LIKE @VendorVar);
```

Test the procedure with parameters passed by position

```sql
50     USE AccountPayables;
51     DECLARE @MyInvTotal money;
52     EXEC spInvTotal3 @MyInvTotal OUTPUT, '2016-02-01', 'P%';
53
54     PRINT '$' + CONVERT(varchar,@MyInvTotal,1);
```

Test the procedure with parameters passed by name

```sql
58     USE AccountPayables;
59     DECLARE @MyInvTotal money;
60     EXEC spInvTotal3 @DateVar = '2016-02-01', @VendorVar = 'P%',
61         @InvTotal = @MyInvTotal OUTPUT;
62
63     PRINT '$' + CONVERT(varchar,@MyInvTotal,1);
```

A stored procedure with return

Create the procedure

```sql
67     USE AccountPayables;
68     IF OBJECT_ID('spInvCount') IS NOT NULL
69         DROP PROC spInvCount;
70     GO
71
72     CREATE PROC spInvCount
73          @DateVar smalldatetime = NULL,
74          @VendorVar varchar(40) = '%'
75     AS
76
77     IF @DateVar IS NULL
78         SELECT @DateVar = MIN(InvoiceDate) FROM Invoices;
79
```

```
80   DECLARE @InvCount int;
81
82   SELECT @InvCount = COUNT(InvoiceID)
83   FROM Invoices JOIN Vendors
84       ON Invoices.VendorID = Vendors.VendorID
85   WHERE (InvoiceDate >= @DateVar) AND
86         (VendorName LIKE @VendorVar);
87
88   RETURN @InvCount;
```

Test the procedure

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

A stored procedure for inserting invoices with data validation

Create the procedure

```
100  USE AccountPayables;
101  IF OBJECT_ID('spInsertInvoice') IS NOT NULL
102      DROP PROC spInsertInvoice;
103  GO
104
105  CREATE PROC spInsertInvoice
106          @VendorID        int = NULL,
107          @InvoiceNumber   varchar(50) = NULL,
108          @InvoiceDate     smalldatetime = NULL,
109          @InvoiceTotal    money = NULL,
110          @PaymentTotal    money = NULL,
111          @CreditTotal     money = NULL,
112          @TermsID         int = NULL,
113          @InvoiceDueDate  smalldatetime = NULL,
114          @PaymentDate     smalldatetime = NULL
115  AS
116
117  IF NOT EXISTS (SELECT * FROM Vendors WHERE VendorID = @VendorID)
118      THROW 50001, 'Invalid VendorID.', 1;
119  IF @InvoiceNumber IS NULL
120      THROW 50001, 'Invalid InvoiceNumber.', 1;
121  IF @InvoiceDate IS NULL OR @InvoiceDate > GETDATE()
122          OR DATEDIFF(dd, @InvoiceDate, GETDATE()) > 30
123      THROW 50001, 'Invalid InvoiceDate.', 1;
124  IF @InvoiceTotal IS NULL OR @InvoiceTotal <= 0
```

```
125    THROW 50001, 'Invalid InvoiceTotal.', 1;
126  IF @PaymentTotal IS NULL
127      SET @PaymentTotal = 0;
128  IF @CreditTotal IS NULL
129      SET @CreditTotal = 0;
130  IF @CreditTotal > @InvoiceTotal
131      THROW 50001, 'Invalid CreditTotal.', 1;
132  IF @PaymentTotal > @InvoiceTotal - @CreditTotal
133      THROW 50001, 'Invalid PaymentTotal.', 1;
134  IF NOT EXISTS (SELECT * FROM Terms WHERE TermsID = @TermsID)
135      IF @TermsID IS NULL
136          SELECT @TermsID = DefaultTermsID
137          FROM Vendors
138          WHERE VendorID = @VendorID;
139      ELSE  -- @TermsID IS NOT NULL
140          THROW 50001, 'Invalid TermsID.', 1;
141  IF @InvoiceDueDate IS NULL
142      SET @InvoiceDueDate = @InvoiceDate +
143          (SELECT TermsDueDays FROM Terms WHERE TermsID = @TermsID);
144  ELSE  -- @InvoiceDueDate IS NOT NULL
145      IF @InvoiceDueDate < @InvoiceDate OR
146              DATEDIFF(dd, @InvoiceDueDate, @InvoiceDate) > 180
147          THROW 50001, 'Invalid InvoiceDueDate.', 1;
148  IF @PaymentDate < @InvoiceDate OR
149          DATEDIFF(dd, @PaymentDate, GETDATE()) > 14
150      THROW 50001, 'Invalid PaymentDate.', 1;
151
152  INSERT Invoices
153  VALUES (@VendorID, @InvoiceNumber, @InvoiceDate, @InvoiceTotal,
154          @PaymentTotal, @CreditTotal, @TermsID, @InvoiceDueDate,
155          @PaymentDate);
156  RETURN @@IDENTITY;
157  GO
```

Test the procedure

```
161  USE AccountPayables;
162  BEGIN TRY
163      DECLARE @InvoiceID int;
164      EXEC @InvoiceID = spInsertInvoice
165          @VendorID = 799,
166          @InvoiceNumber = 'RZ99381',
167          @InvoiceDate = '2016-04-12',
168          @InvoiceTotal = 1292.45;
169      PRINT 'Row was inserted.';
170      PRINT 'New InvoiceID: ' + CONVERT(varchar, @InvoiceID);
171  END TRY
```

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

Passing tables to stored procedures

Create the procedure

```
181  USE AccountPayables;
182  -- drop stored procedure if it exists already
183  IF OBJECT_ID('spInsertLineItems') IS NOT NULL
184      DROP PROC spInsertLineItems;
185  GO
186
187  -- drop table type if it exists already
188  IF  EXISTS (SELECT * FROM sys.types WHERE name = 'LineItems')
189      DROP TYPE LineItems;
190  GO
191
192  -- create the user-defined table type named LineItems
193  CREATE TYPE LineItems AS
194  TABLE
195  (InvoiceID        INT           NOT NULL,
196  InvoiceSequence   SMALLINT      NOT NULL,
197  AccountNo         INT           NOT NULL,
198  ItemAmount        MONEY         NOT NULL,
199  ItemDescription   VARCHAR(100)  NOT NULL,
200  PRIMARY KEY (InvoiceID, InvoiceSequence));
201  GO
202
203  -- create a stored procedure that accepts the LineItems type
204  CREATE PROC spInsertLineItems
205      @LineItems LineItems READONLY
206  AS
207      INSERT INTO InvoiceLineItems
208      SELECT *
209      FROM @LineItems;
210  GO
211  -- start snippet sp_table_passing_test
212  USE AccountPayables;
213  -- delete old line item data
214  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
```

```sql
-- 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
307         ON Vendors.VendorName = TopVendors.VendorName;
308 -- end snippet udf_simple_table_test
309
310 -- start snippet udf_multis_table
311 USE AccountPayables;
312 GO
313
314 IF OBJECT_ID('fnCreditAdj') IS NOT NULL
315     DROP FUNCTION fnCreditAdj;
316 GO
317
318 CREATE FUNCTION fnCreditAdj (@HowMuch money)
319     RETURNS @OutTable table
320             (InvoiceID int, VendorID int, InvoiceNumber varchar(50),
321              InvoiceDate smalldatetime, InvoiceTotal money,
322              PaymentTotal money, CreditTotal money)
323 BEGIN
324     INSERT @OutTable
325         SELECT InvoiceID, VendorID, InvoiceNumber, InvoiceDate,
326                 InvoiceTotal, PaymentTotal, CreditTotal
327         FROM Invoices
328         WHERE (InvoiceTotal - CreditTotal - PaymentTotal) > 0;
329     WHILE (SELECT SUM(InvoiceTotal - CreditTotal - PaymentTotal)
330            FROM @OutTable) >= @HowMuch
331         UPDATE @OutTable
332         SET CreditTotal = CreditTotal + .01
333         WHERE (InvoiceTotal - CreditTotal - PaymentTotal) > 0;
334     RETURN;
335 END;
336 -- end snippet udf_multis_table
337
338 -- start snippet udf_multis_table_test
339 USE AccountPayables;
340
341 SELECT VendorName, SUM(CreditTotal) AS CreditRequest
342 FROM Vendors JOIN dbo.fnCreditAdj(25000) AS CreditTable
343     ON Vendors.VendorID = CreditTable.VendorID
344 GROUP BY VendorName;
345 -- end snippet udf_multis_table_test
346
347 -- start snippet tg_after_01
348 USE AccountPayables;
349 IF OBJECT_ID('Vendors_INSERT_UPDATE') IS NOT NULL
350     DROP TRIGGER Vendors_INSERT_UPDATE;
351 GO
352
```

9

```sql
353   CREATE TRIGGER Vendors_INSERT_UPDATE
354       ON Vendors
355       AFTER INSERT,UPDATE
356   AS
357       UPDATE Vendors
358       SET VendorState = UPPER(VendorState)
359       WHERE VendorID IN (SELECT VendorID FROM Inserted);
360   -- end snippet tg_after_01
361
362   -- start snippet tg_after_01_test
363
364   USE AccountPayables;
365   INSERT Vendors
366   VALUES ('Peerless Uniforms, Inc.', '785 S Pixley Rd', NULL,
367           'Piqua', 'Oh', '45356', '(937) 555-8845', NULL, NULL, 4,550);
368   -- end snippet tg_after_01_test
369
370   -- start snippet tg_after_02
371   USE AccountPayables;
372   GO
373   IF OBJECT_ID('Invoices_DELETE') IS NOT NULL
374       DROP TRIGGER Invoices_DELETE;
375   GO
376
377   CREATE TRIGGER Invoices_DELETE
378           ON Invoices
379           AFTER DELETE
380   AS
381   INSERT INTO InvoiceArchive
382           (InvoiceID, VendorID, InvoiceNumber, InvoiceDate, InvoiceTotal,
383               PaymentTotal, CreditTotal, TermsID, InvoiceDueDate, PaymentDate)
384           SELECT InvoiceID, VendorID, InvoiceNumber, InvoiceDate, InvoiceTotal,
385               PaymentTotal, CreditTotal, TermsID, InvoiceDueDate, PaymentDate
386           FROM Deleted;
387   -- end snippet tg_after_02
388
389   -- start snippet tg_after_02_test
390   USE AccountPayables;
391   DELETE Invoices
392   WHERE VendorID = 37;
393
394   SELECT * FROM InvoiceArchive;
395   -- end snippet tg_after_02_test
396
397   -- start snippet tg_instead_01
398   USE AccountPayables;
```

```sql
399    GO

400

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

404

405    CREATE VIEW IBM_Invoices
406    AS
407    SELECT InvoiceNumber, InvoiceDate, InvoiceTotal
408    FROM Invoices
409    WHERE VendorID = (SELECT VendorID FROM Vendors WHERE VendorName = 'IBM');
410    GO

411

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

415

416    CREATE TRIGGER IBM_Invoices_INSERT
417        ON IBM_Invoices
418        INSTEAD OF INSERT
419    AS
420    DECLARE @InvoiceDate smalldatetime, @InvoiceNumber varchar(50),
421            @InvoiceTotal money, @VendorID int,
422            @InvoiceDueDate smalldatetime, @TermsID int,
423            @DefaultTerms smallint, @TestRowCount int;
424    SELECT @TestRowCount = COUNT(*) FROM Inserted;
425    IF @TestRowCount = 1
426        BEGIN
427            SELECT @InvoiceNumber = InvoiceNumber, @InvoiceDate = InvoiceDate,
428                @InvoiceTotal = InvoiceTotal
429            FROM Inserted;
430            IF (@InvoiceDate IS NOT NULL AND @InvoiceNumber IS NOT NULL AND
431                @InvoiceTotal IS NOT NULL)
432                BEGIN
433                    SELECT @VendorID = VendorID, @TermsID = DefaultTermsID
434                    FROM Vendors
435                    WHERE VendorName = 'IBM';

436

437                    SELECT @DefaultTerms = TermsDueDays
438                    FROM Terms
439                    WHERE TermsID = @TermsID;

440

441                    SET @InvoiceDueDate = @InvoiceDate + @DefaultTerms;

442

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

```
445                        TermsID, InvoiceDueDate, PaymentDate)
446                VALUES (@VendorID, @InvoiceNumber, @InvoiceDate,
447                    @InvoiceTotal, @TermsID, @InvoiceDueDate, NULL);
448            END;
449        END;
450    ELSE
451            THROW 50027, 'Limit INSERT to a single row.', 1;
452    -- end snippet tg_instead_01
453
454    -- start snippet tg_instead_01_test
455    USE AccountPayables;
456    INSERT IBM_Invoices
457    VALUES ('RA23988', '2016-05-09', 417.34);
458    -- end snippet tg_instead_01_test
459
460    -- start snippet tg_dc
461    /*
462    updates the InvoiceLineItems table
463    to have an incorrect value for one of the line
464    items for InvoiceID 100.
465    */
466    USE AccountPayables;
467    UPDATE InvoiceLineItems
468    SET InvoiceLineItemAmount = 477.79
469    WHERE InvoiceID = 98 AND InvoiceSequence = 1;
470    GO
471
472
473    USE AccountPayables;
474    IF OBJECT_ID('Invoices_UPDATE') IS NOT NULL
475        DROP TRIGGER Invoices_UPDATE;
476    GO
477
478    CREATE TRIGGER Invoices_UPDATE
479        ON Invoices
480        AFTER UPDATE
481    AS
482    IF EXISTS            --Test whether PaymentTotal was changed
483      (SELECT *
484       FROM Deleted JOIN Invoices
485         ON Deleted.InvoiceID = Invoices.InvoiceID
486       WHERE Deleted.PaymentTotal <> Invoices.PaymentTotal)
487      BEGIN
488        IF EXISTS          --Test whether line items total and InvoiceTotal match
489         (SELECT *
490          FROM Invoices JOIN
```

12

```
491        (SELECT InvoiceID, SUM(InvoiceLineItemAmount) AS SumOfInvoices
492         FROM InvoiceLineItems
493         GROUP BY InvoiceID) AS LineItems
494       ON Invoices.InvoiceID = LineItems.InvoiceID
495     WHERE (Invoices.InvoiceTotal <> LineItems.SumOfInvoices) AND
496           (LineItems.InvoiceID IN (SELECT InvoiceID FROM Deleted)))
497     BEGIN
498       ;
499       THROW 50113, 'Correct line item amounts before posting payment.', 1;
500       ROLLBACK TRAN;
501     END;
502   END;
503 -- end snippet tg_dc
504
505 -- start snippet tg_dc_test
506 USE AccountPayables;
507
508 UPDATE Invoices
509 SET PaymentTotal = 662, PaymentDate = '2016-05-09'
510 WHERE InvoiceID = 98;
511 -- end snippet tg_dc_test
512
513 -- start snippet tg_modify
514 USE AccountPayables;
515 GO
516
517 ALTER TRIGGER Vendors_INSERT_UPDATE
518     ON Vendors
519     AFTER INSERT,UPDATE
520 AS
521     UPDATE Vendors
522     SET VendorState = UPPER(VendorState),
523         VendorAddress1 = LTRIM(RTRIM(VendorAddress1)),
524         VendorAddress2 = LTRIM(RTRIM(VendorAddress2))
525     WHERE VendorID IN (SELECT VendorID FROM Inserted);
526 -- end snippet tg_modify
```

Test the procedure

```
212 USE AccountPayables;
213 -- delete old line item data
214 DELETE FROM InvoiceLineItems WHERE InvoiceID = 114;
215
216 -- declare a variable for the LineItems type
217 DECLARE @LineItems LineItems;
218
219 -- insert rows into the LineItems variable
```

13

```
220   INSERT INTO @LineItems VALUES (114, 1, 553, 127.75, 'Freight');
221   INSERT INTO @LineItems VALUES (114, 2, 553, 29.25, 'Freight');
222   INSERT INTO @LineItems VALUES (114, 3, 553, 48.50, 'Freight');
223
224   -- execute the stored procedure
225   EXEC spInsertLineItems @LineItems;
```

Modify stored procedures

```
229   -- create a store procedure
230   USE AccountPayables;
231   IF OBJECT_ID('spVendorState') IS NOT NULL
232       DROP PROC spVendorState;
233   GO
234
235   CREATE PROC spVendorState
236         @State varchar(20)
237   AS
238   SELECT VendorName
239   FROM Vendors
240   WHERE VendorState = @State;
241
242   EXEC sp_HelpText spVendorState
243
244   -- modify it
245   USE AccountPayables;
246   GO
247
248   ALTER PROC spVendorState
249         @State varchar(20) = NULL
250   AS
251   IF @State IS NULL
252       SELECT VendorName
253       FROM Vendors;
254   ELSE
255       SELECT VendorName
256       FROM Vendors
257       WHERE VendorState = @State;
258
259   EXEC sp_HelpText spVendorState
```

Exercise 01

Create a procedure named spBalancedRange:
- Output: - 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

```
263  USE AccountPayables;
264  IF OBJECT_ID('fnBalanceDue') IS NOT NULL
265      DROP FUNCTION fnBalanceDue;
266  GO
267
268  CREATE FUNCTION fnBalanceDue()
269      RETURNS money
270  BEGIN
271      RETURN (SELECT SUM(InvoiceTotal - PaymentTotal - CreditTotal)
272              FROM Invoices
273              WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0);
274  END;
```

Test the function

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

15

A simple table-valued function

Create the function

```
283  USE AccountPayables;
284  IF OBJECT_ID('fnTopVendorsDue') IS NOT NULL
285      DROP FUNCTION fnTopVendorsDue;
286  GO
287
288  CREATE FUNCTION fnTopVendorsDue
289      (@CutOff money = 0)
290      RETURNS TABLE
291  RETURN
292          (SELECT VendorName, SUM(InvoiceTotal) AS TotalDue
293          FROM Vendors JOIN Invoices ON Vendors.VendorID = Invoices.VendorID
294          WHERE InvoiceTotal - CreditTotal - PaymentTotal > 0
295          GROUP BY VendorName
296          HAVING SUM(InvoiceTotal) >= @CutOff);
```

Test the function

```
300  USE AccountPayables;
301  SELECT * FROM dbo.fnTopVendorsDue(5000);
302
303
304  USE AccountPayables;
305  SELECT Vendors.VendorName, VendorCity, TotalDue
306  FROM Vendors JOIN dbo.fnTopVendorsDue(DEFAULT) AS TopVendors
307      ON Vendors.VendorName = TopVendors.VendorName;
```

A multi-statement table-valued function

Create the function

```
311  USE AccountPayables;
312  GO
313
314  IF OBJECT_ID('fnCreditAdj') IS NOT NULL
315      DROP FUNCTION fnCreditAdj;
316  GO
317
318  CREATE FUNCTION fnCreditAdj (@HowMuch money)
319      RETURNS @OutTable table
320              (InvoiceID int, VendorID int, InvoiceNumber varchar(50),
321               InvoiceDate smalldatetime, InvoiceTotal money,
322               PaymentTotal money, CreditTotal money)
323  BEGIN
```

16

```
324      INSERT @OutTable
325          SELECT InvoiceID, VendorID, InvoiceNumber, InvoiceDate,
326                  InvoiceTotal, PaymentTotal, CreditTotal
327          FROM Invoices
328          WHERE (InvoiceTotal - CreditTotal - PaymentTotal) > 0;
329      WHILE (SELECT SUM(InvoiceTotal - CreditTotal - PaymentTotal)
330              FROM @OutTable) >= @HowMuch
331          UPDATE @OutTable
332          SET CreditTotal = CreditTotal + .01
333          WHERE (InvoiceTotal - CreditTotal - PaymentTotal) > 0;
334      RETURN;
335  END;
```

Test the function

```
339  USE AccountPayables;
340
341  SELECT VendorName, SUM(CreditTotal) AS CreditRequest
342  FROM Vendors JOIN dbo.fnCreditAdj(25000) AS CreditTable
343      ON Vendors.VendorID = CreditTable.VendorID
344  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

```
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.

17

Triggers

AFTER triggers

A trigger to correct mixed-case state names

Create the trigger

```
348  USE AccountPayables;
349  IF OBJECT_ID('Vendors_INSERT_UPDATE') IS NOT NULL
350      DROP TRIGGER Vendors_INSERT_UPDATE;
351  GO
352
353  CREATE TRIGGER Vendors_INSERT_UPDATE
354      ON Vendors
355      AFTER INSERT,UPDATE
356  AS
357      UPDATE Vendors
358      SET VendorState = UPPER(VendorState)
359      WHERE VendorID IN (SELECT VendorID FROM Inserted);
```

Test the trigger

```
363
364  USE AccountPayables;
365  INSERT Vendors
366  VALUES ('Peerless Uniforms, Inc.', '785 S Pixley Rd', NULL,
367          'Piqua', 'Oh', '45356', '(937) 555-8845', NULL, NULL, 4,550);
```

A trigger that archive deleted data

Create the trigger

```
371  USE AccountPayables;
372  GO
373  IF OBJECT_ID('Invoices_DELETE') IS NOT NULL
374      DROP TRIGGER Invoices_DELETE;
375  GO
376
377  CREATE TRIGGER Invoices_DELETE
378          ON Invoices
379          AFTER DELETE
380  AS
381  INSERT INTO InvoiceArchive
382          (InvoiceID, VendorID, InvoiceNumber, InvoiceDate, InvoiceTotal,
383              PaymentTotal, CreditTotal, TermsID, InvoiceDueDate, PaymentDate)
384          SELECT InvoiceID, VendorID, InvoiceNumber, InvoiceDate, InvoiceTotal,
```

```
385              PaymentTotal, CreditTotal, TermsID, InvoiceDueDate, PaymentDate
386          FROM Deleted;
```

Test the trigger

```
390  USE AccountPayables;
391  DELETE Invoices
392  WHERE VendorID = 37;
393
394  SELECT * FROM InvoiceArchive;
```

INSTEAD OF triggers

An insert trigger for a view

Create the trigger

```
398  USE AccountPayables;
399  GO
400
401  IF OBJECT_ID('IBM_Invoices') IS NOT NULL
402          DROP VIEW IBM_Invoices
403  GO
404
405  CREATE VIEW IBM_Invoices
406  AS
407  SELECT InvoiceNumber, InvoiceDate, InvoiceTotal
408  FROM Invoices
409  WHERE VendorID = (SELECT VendorID FROM Vendors WHERE VendorName = 'IBM');
410  GO
411
412  IF OBJECT_ID('IBM_Invoices_INSERT') IS NOT NULL
413      DROP TRIGGER IBM_Invoices_INSERT;
414  GO
415
416  CREATE TRIGGER IBM_Invoices_INSERT
417      ON IBM_Invoices
418      INSTEAD OF INSERT
419  AS
420  DECLARE @InvoiceDate smalldatetime, @InvoiceNumber varchar(50),
421          @InvoiceTotal money, @VendorID int,
422          @InvoiceDueDate smalldatetime, @TermsID int,
423          @DefaultTerms smallint, @TestRowCount int;
424  SELECT @TestRowCount = COUNT(*) FROM Inserted;
425  IF @TestRowCount = 1
426      BEGIN
427          SELECT @InvoiceNumber = InvoiceNumber, @InvoiceDate = InvoiceDate,
```

19

```
428              @InvoiceTotal = InvoiceTotal
429          FROM Inserted;
430          IF (@InvoiceDate IS NOT NULL AND @InvoiceNumber IS NOT NULL AND
431              @InvoiceTotal IS NOT NULL)
432              BEGIN
433                  SELECT @VendorID = VendorID, @TermsID = DefaultTermsID
434                  FROM Vendors
435                  WHERE VendorName = 'IBM';

437                  SELECT @DefaultTerms = TermsDueDays
438                  FROM Terms
439                  WHERE TermsID = @TermsID;

441                  SET @InvoiceDueDate = @InvoiceDate + @DefaultTerms;

443                  INSERT Invoices
444                      (VendorID, InvoiceNumber, InvoiceDate, InvoiceTotal,
445                       TermsID, InvoiceDueDate, PaymentDate)
446                  VALUES (@VendorID, @InvoiceNumber, @InvoiceDate,
447                      @InvoiceTotal, @TermsID, @InvoiceDueDate, NULL);
448              END;
449      END;
450  ELSE
451          THROW 50027, 'Limit INSERT to a single row.', 1;
```

Test the trigger

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

Use triggers for enforcing data consistency

Create the trigger

```
461  /*
462  updates the InvoiceLineItems table
463  to have an incorrect value for one of the line
464  items for InvoiceID 100.
465  */
466  USE AccountPayables;
467  UPDATE InvoiceLineItems
468  SET InvoiceLineItemAmount = 477.79
469  WHERE InvoiceID = 98 AND InvoiceSequence = 1;
470  GO

471
```

```
472
473  USE AccountPayables;
474  IF OBJECT_ID('Invoices_UPDATE') IS NOT NULL
475      DROP TRIGGER Invoices_UPDATE;
476  GO
477
478  CREATE TRIGGER Invoices_UPDATE
479      ON Invoices
480      AFTER UPDATE
481  AS
482  IF EXISTS              --Test whether PaymentTotal was changed
483   (SELECT *
484    FROM Deleted JOIN Invoices
485      ON Deleted.InvoiceID = Invoices.InvoiceID
486    WHERE Deleted.PaymentTotal <> Invoices.PaymentTotal)
487    BEGIN
488      IF EXISTS          --Test whether line items total and InvoiceTotal match
489       (SELECT *
490        FROM Invoices JOIN
491            (SELECT InvoiceID, SUM(InvoiceLineItemAmount) AS SumOfInvoices
492             FROM InvoiceLineItems
493             GROUP BY InvoiceID) AS LineItems
494          ON Invoices.InvoiceID = LineItems.InvoiceID
495        WHERE (Invoices.InvoiceTotal <> LineItems.SumOfInvoices) AND
496            (LineItems.InvoiceID IN (SELECT InvoiceID FROM Deleted)))
497        BEGIN
498          ;
499          THROW 50113, 'Correct line item amounts before posting payment.', 1;
500          ROLLBACK TRAN;
501        END;
502    END;
```

Test the trigger

```
506  USE AccountPayables;
507
508  UPDATE Invoices
509  SET PaymentTotal = 662, PaymentDate = '2016-05-09'
510  WHERE InvoiceID = 98;
```

Modify triggers

```
514  USE AccountPayables;
515  GO
516
517  ALTER TRIGGER Vendors_INSERT_UPDATE
```

```
518     ON Vendors
519     AFTER INSERT,UPDATE
520  AS
521     UPDATE Vendors
522     SET VendorState = UPPER(VendorState),
523         VendorAddress1 = LTRIM(RTRIM(VendorAddress1)),
524         VendorAddress2 = LTRIM(RTRIM(VendorAddress2))
525     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 NoDup-Name 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.