NBCC

INFORMATION TECHNOLOGY

Intermediate SQL

USER-DEFINED FUNCTIONS

What is a User-Defined Function (UDF)?

- An executable database object that contains SQL statements
- We've learned about built in SQL Server functions throughout the book such as SUBSTRING, DATEADD, PATINDEX, etc. UDF's are functions we create ourselves.
- Naming Convention: prefix with "fn"

Differences between Stored Procedures and UDF’s

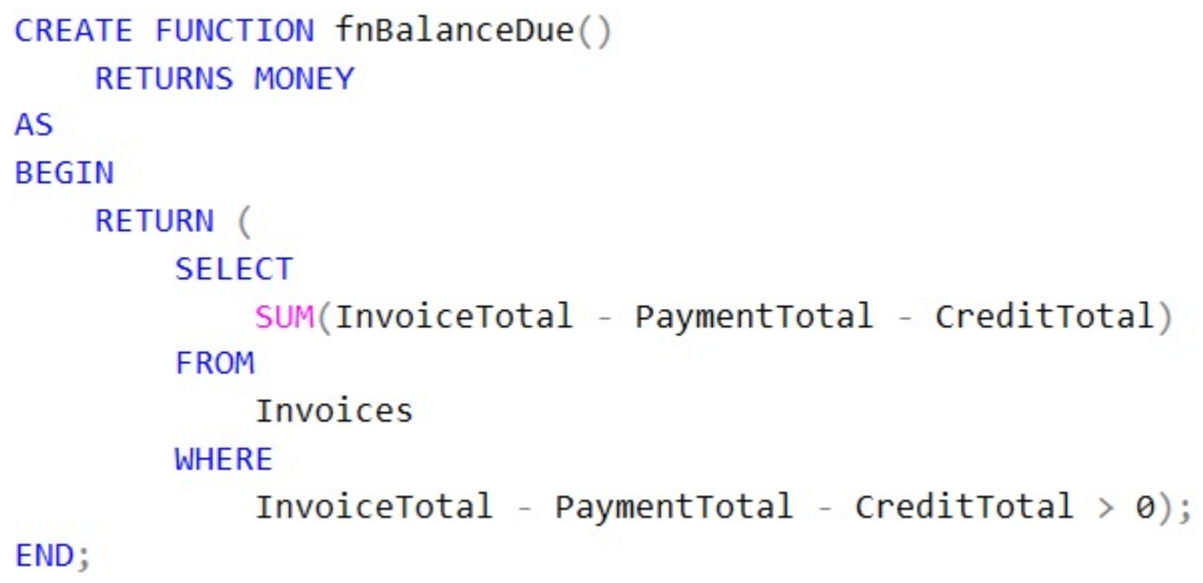
- Both can accept input parameters, but only stored procedures can have output parameters
- UDF’s MUST return a value. With a stored procedure it is optional.
- UDF’s can return a value of any data type, while stored procedures return an integer
- UDF’s can be called from stored procedures, but stored procedures cannot be called from UDF’s
- UDF’s cannot make permanent changes to objects in a database. Ie: they cannot INSERT, UPDATE and DELETE. However, they can create tables, temporary tables and table variables inside the function and modify them.
- UDF’s can be used in SELECT/WHERE/HAVING statements. Stored procedures cannot.
- UDF’s cannot have try/catch blocks, while stored procedures can.

Two Types of UDF’s

- Scalar-valued functions that return a single value. These are like built int functions that we have learned
- Table-value functions – return an entire table

Example –Scalar-valued function

- This function returns the total of all invoices that have a balance due



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

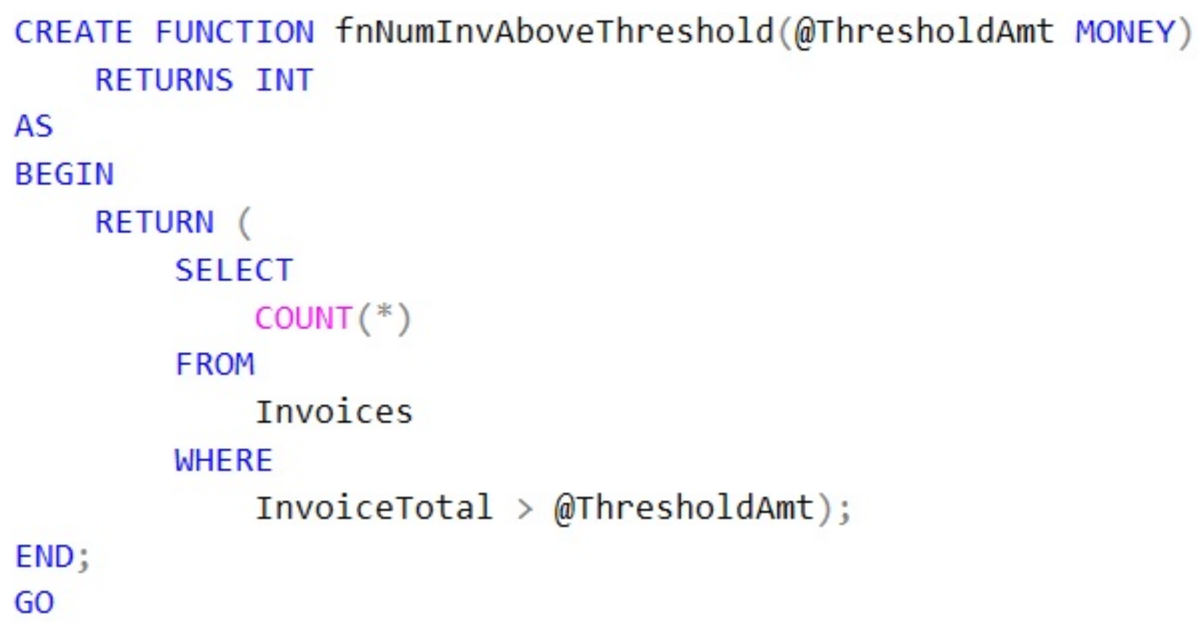
Invoking a Scalar-valued function

- When we invoke a function, we MUST specify the schema name

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

Example –Scalar-valued function with parameter

- This function returns the number of invoices with a total above a specified threshold



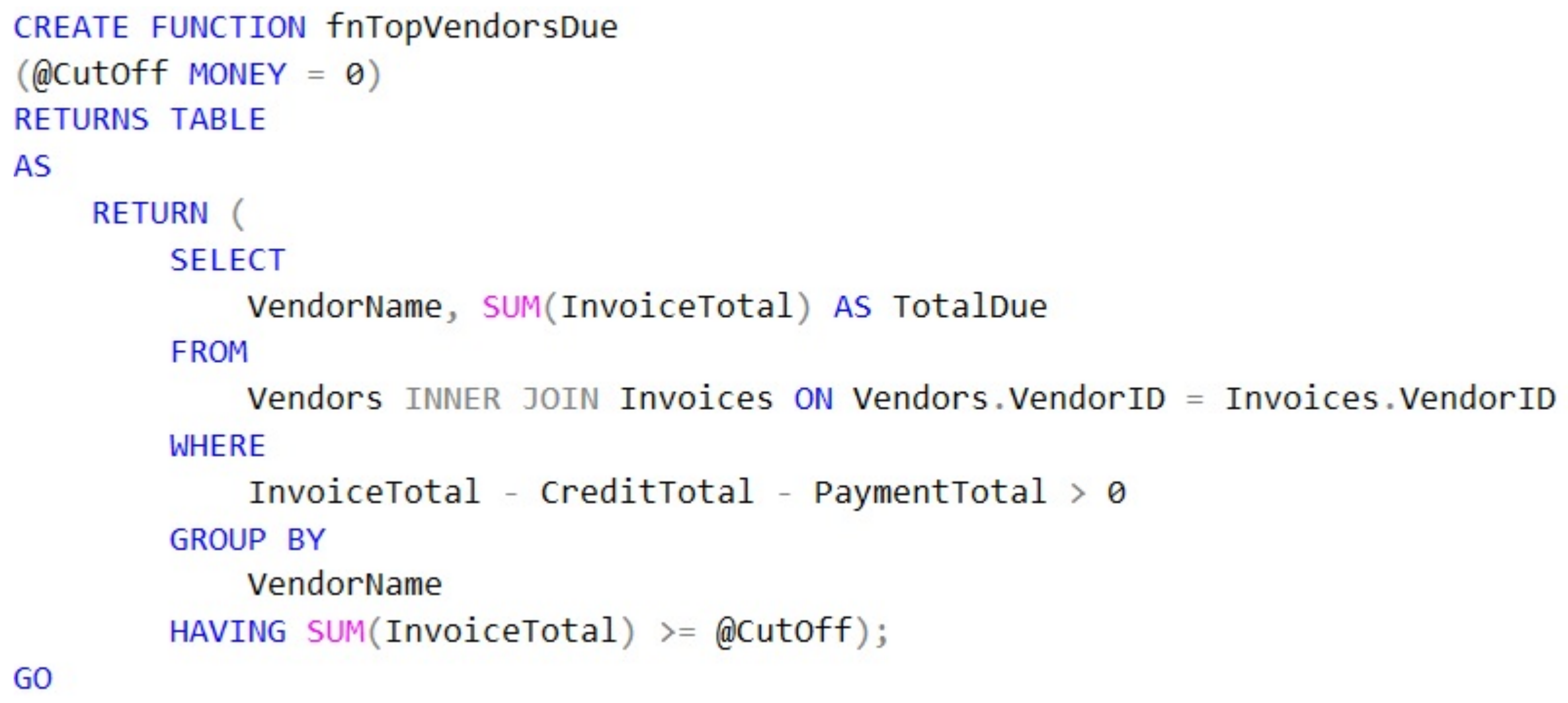
```sql
CREATE FUNCTION fnNumInvAboveThreshold(@ThresholdAmt MONEY)
    RETURNS INT
AS
BEGIN
    RETURN (
        SELECT
            COUNT(*)
        FROM
            Invoices
        WHERE
            InvoiceTotal > @ThresholdAmt);
END;
GO
```

Invoking a Scalar-valued function with parameter

```
  PRINT 'Num Invoices ' +
CONVERT(varchar,dbo.fnNumInvAboveThreshold(8000));
```

Example –Table-valued function

- This function returns a table



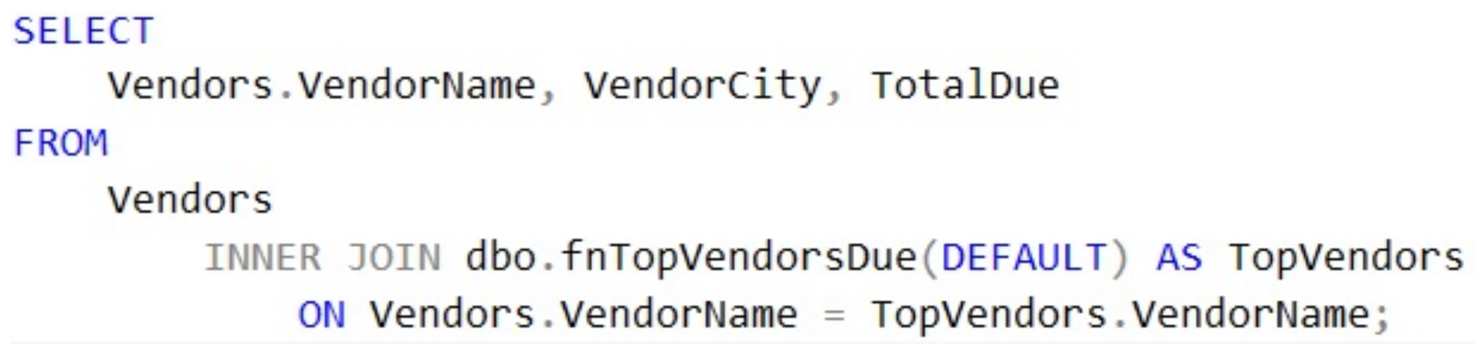
```sql
CREATE FUNCTION fnTopVendorsDue
(@CutOff MONEY = 0)
RETURNS TABLE
AS
    RETURN (
        SELECT
            VendorName, SUM(InvoiceTotal) AS TotalDue
        FROM
            Vendors INNER JOIN Invoices ON Vendors.VendorID = Invoices.VendorID
        WHERE
            InvoiceTotal - CreditTotal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

Invoking a Table-valued function in a SELECT statement

- Invoking in a SELECT statement

```sql
SELECT * FROM dbo.fnTopVendorsDue(5000);
```

- Using the function in a join operation



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

DELETE OR MODIFYING A FUNCTION

- Deleting a function

```
DROP FUNCTION fnTopVendorsDue;
```

- Modifying a function

  - You can use CREATE OR ALTER or ALTER. We do it the same way as we do a stored procedure. All of the code in the function is replaced by the code in the ALTER statement