Intermediate SQL

STORED PROCEDURES

What is a stored procedure?

- A stored procedure is a set of SQL statements that is saved as a database object
- They can accept input parameters
- They can return values in the form of output parameters
- They are precompiled, which means that the executuion plan for the SQL code is compiled the first time a procedure is executed and is then saved in its compiled form

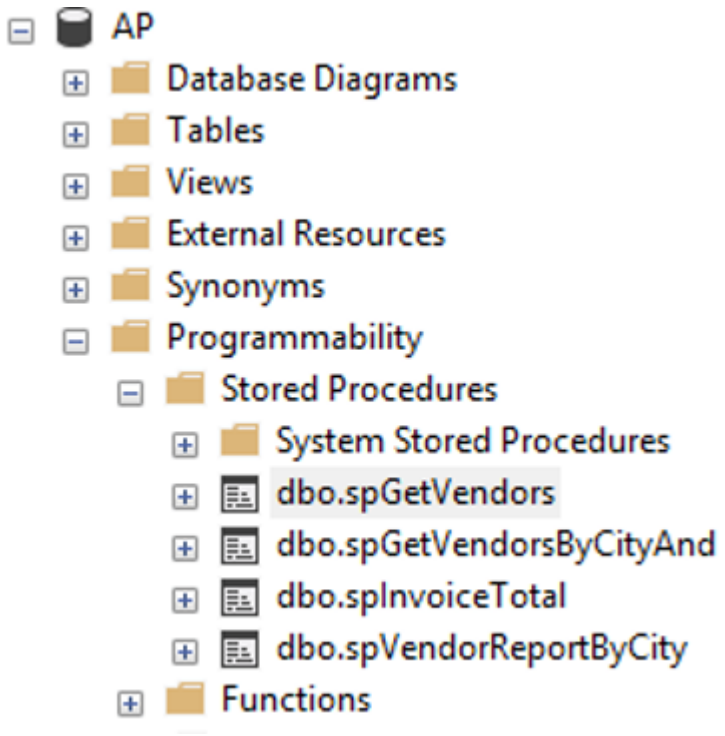
Benefits of Stored Procedures

- **Performance** – Since stored procedures are precompiled, they will execute faster than the equivalent code running individually every time
- **Encapsulation** – Programmers and end users do not need to know the structure of the database or what the code and logic look like…they just need to know what it will do and what parameters they need to pass when calling it
- **Improve Maintainability** – If a coding change needs to be made, we can make the change in the stored procedure instead of every application that is calling that stored procedure
- **Reduce client/server network traffic** – Instead of applications passing queries to the database server, they can just call a stored procedure
- **Security** – We can restrict access to data by allowing users to manipulate the data only through stored procedures that they have been giving access to

Naming Conventions

- There is no standard naming convention to follow, however, it is good practice to come up with one and follow it.
- You should prefix your stored procedure names with “sp”. For example, **spGetVendors**
- Do NOT use “sp_”. This is the naming convention that is used in the master database
- I like to name my stored procedures based on the action that they are performing. Ie: spInsertVendor, spDeleteVendor, spUpdateVendor

Creating a Stored Procedure



- To create a stored procedure, we use the CREATE PROCEDURE command
- We can shorten that command to CREATE PROC
- To modify a stored procedure, we use the ALTER PROCEDURE (or ALTER PROC) command
- We can also combine the two by using the CREATE OR ALTER PROC command. This will CREATE the procedure if it does not already exist, and modify it if it does
- When a stored procedure is created, it is stored as an object in the database…just like a table is (see image to left)

Basic Syntax

- This stored procedure contains a select statement to return all vendors

```
CREATE PROCEDURE spGetVendors
AS

BEGIN
    SELECT
        VendorName, VendorCity, VendorState
    FROM
        Vendors
    ORDER BY
        VendorName;
END
GO
```

Calling a Stored Procedure

- We can call a stored procedure using the **EXEC** statement

```
EXEC spGetVendors
```

Input Parameters

- Stored procedures can contain parameters which we can pass arguments into. This enables stored procedures to handle each call to it differently
- A stored procedure can have multiple parameters
- Parameters are created similar to variables – they are prefixed with @ and are assigned a data type
- Multiple parameters are separated by a comma

Using Input Parameters

- This stored procedure has two input parameters and will retrieve invoices based on the vendor and invoice total passed into it

```
CREATE OR ALTER PROCEDURE spInvoiceReport
    @VendorID INT,
    @InvoiceTotal MONEY
AS

BEGIN
    SELECT InvoiceID, InvoiceDate
    FROM Invoices
    WHERE VendorID = @VendorID AND InvoiceTotal < @InvoiceTotal
END
GO
```

Calling a Stored Procedure with input parameters

- To call spGetVendors procedure (by position):

```
EXEC spInvoiceReport 122, 3000
```

- To call spGetVendors procedure (by name):

```
EXEC spInvoiceReport @VendorID = 122, @InvoiceTotal = 3000
```

Optional Parameters

- You can assign a default value to a parameter, similar to how you would do it in C#

```
CREATE PROCEDURE spVendorReportByCity
    @City VARCHAR(100) = NULL
AS

BEGIN
    IF @City IS NOT NULL
        SELECT * FROM Vendors WHERE VendorCity = @City;
    ELSE
        SELECT * FROM Vendors;
END
GO
```

Calling a Stored Procedure with optional parameters

- To call spVendorReportByCity procedure and pass in a city:

```
EXEC spVendorReportByCity 'Washington'
```

- To call spVendorReportByCity procedure and pass in a city. The stored procedure would use the default value set for the city parameter:

```
EXEC spVendorReportByCity
```

Output Parameters

- Output parameters are used to return values to the calling code
- Output parameters are defined the same way as input parameters. The only difference is that we use the **OUTPUT** clause after the parameter name to specify that it should return a value.
- You can have multiple output parameters, enabling you to return multiple values from your stored procedures to your calling code

Using Output Parameters

- This stored procedure one input parameter and one output parameter. It will return the sum of all the invoices for the invoice date that gets passed in

```
CREATE PROCEDURE spInvoiceTotal
    @InvoiceDate SMALLDATETIME,
    @InvoiceTotal MONEY OUTPUT
AS

BEGIN
    SELECT @InvoiceTotal = (
        SELECT SUM(InvoiceTotal)
        FROM Invoices
        WHERE InvoiceDate > @InvoiceDate
    )
END
GO
```

Calling stored procedures with Output parameters

- To call the spInvoiceTotal procedure

```
-- first declare a variable to store the
-- value that is being returned in the output parameter
DECLARE @InvTotal MONEY;
-- execute the stored procedure, specifying that
-- the parameter is an output parameter
EXEC spInvoiceTotal '2019-02-01', @InvTotal OUTPUT;
-- use the variable that is stored in the output parameter
SELECT @InvTotal;
```

Return Values

- In addition to returning values to the calling code with output parameters, stored procedures also pass back a return value
- We do this by using a **RETURN** statement
- A **RETURN** statement immediately exits the procedure and returns an optional integer value to the calling program. If you do not specify the integer value, the default is zero
- If your stored procedure only needs to return a single integer value, you can use the **RETURN** statement to send that value back to the calling code. However, if your stored procedure needs to return a non-integer value or needs to return multiple values, then you need to use OUTPUT parameters. (I always just use output parameters)

NOCOUNT ON/OFF

- When SQL statements are executed, a message is generated that shows the number of rows affected by the SQL statement
- We can prevent that from happening by using **SET NOCOUNT ON** command
- Doing so can improve performance by reducing the amount of traffic between the database server and the application calling the stored procedure
- Setting NOCOUNT ON has no affect on @@ROWCOUNT function. Selecting @@ROWCOUNT will still return the number of rows affected event if NOCOUNT IS ON

Deleting Stored Procedures

- To delete a stored procedure, we use the DROP PROCEDURE command

```
DROP PROCEDURE spVendorReportByCity;
```

Modifying Stored Procedures

- To modify a stored procedure we use the ALTER statement
- When we ALTER a stored procedure all the code in the stored procedure is replaced by the new code written in the ALTER statement