**Important Dates**

Monday, April 17, 2017 or Wednesday, April 19, 2017  
Extra Credit: Complete any four problems and show in class (5 points)  
  
Monday, April 24, 2017  
Extra Credit: Complete Project and submit (5 points)  
  
Tuesday, April 25, 2017 by 11pm  
Project Due.

**Submission: Carefully follow all directions.**

If you decide to work by yourself, then the file name should be

YourUsername\_final.sql

If you decide to work with a partner, then the file name should be. Only one of you will submit.

YourUsername\_PartnerUsername\_final.sql

Check the file “final.sql” for guidance on how to submit your code. Upload your .sql file to Blackboard under Assignments ->Final Project. If you have any trouble submitting or understanding how to submit, consult with me before submitting the assignment.

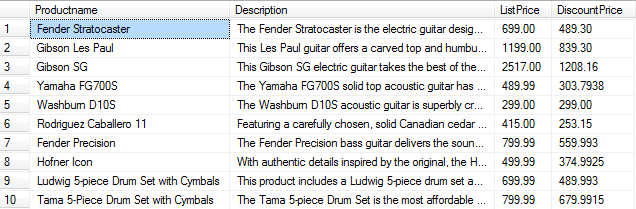
**Project Description**

**Total Points: 70 + 5 (correctly submitted using the guidance in final.sql)**

This project will require you to put together everything that you have learnt in the Database courses such as simple select queries, views, scripts, cursors, procedures, functions, security, dynamic SQL and reading text & xml data.

There are 7 questions. For the first 6, you will write procedures that will create & drop logins, roles and give proper permissions to each user of the database MyGuitarShop. To solve some of the problems, it may be easier to write a regular script before creating the procedure. The 7th question is about XQuery.

1. (10 points) Create a view called “ProductsView” from the Products table in MyGuitarShop that looks similar to the following table:



Include code before creating the view to check if “ProductsView” exists. If it does, drop it.

1. (10 points) Write a procedure called “createCustomerRole” which creates a database role *CustomerRole*. Grant select permission on “ProductsView” to this role. This procedure will have a few lines of code and accepts no parameters and returns nothing.
2. (10 points) Create a procedure called “insertFromTXT” which takes in a single parameter called *@location* of type varchar(256).

In the procedure, add code to read any text file with three columns with information about a customer’s Email, FirstName and LastName. Check the file *names.txt* for the general format of the data.

Drop the table “Logins” if it already exists and then create it to read data from a text file located at the path passed by the parameter *@location*. You have to use dynamic SQL to execute the BULK INSERT statement to read data from the text file to the *Logins* table.

To test the procedure, call the procedure with the appropriate file location path. For example,

EXEC insertFromTXT @location = 'C:\Users\MSSQL$DAY\Desktop\names.txt'

1. (10 points) Create a procedure called “insertFromXML” which takes in a single parameter called *@xmlData* of type xml.

In the procedure, add code to read any xml data with information about a customer’s email and name stored in the following format (here it’s showing data for only one customer, see names\_xml.xml for the complete set):

<customerdata>

<customer>

<email>allanSherwood@yahoo.com</email>

<name>

<first>Allan</first>

<last>Sherwood</last>

</name>

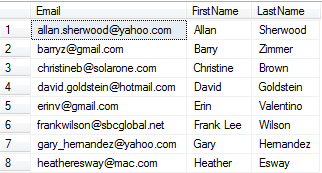
</customer>

</customerdata>

Use OPENXML for this problem. Select the data into a table called *Logins* and drop the table if it already exists. To test the procedure, call the procedure with the appropriate xml data. For example,

EXEC insertFromXML @xmlData = @data where @data contains valid xml data about customers.

This is how the *Logins* table should look like after either the “insertFromTXT” or “insertFromXML” procedures are executed:



(10 points) Write a procedure called “createLogins” which has no parameter. In the procedure, use a cursor for selecting the customer LastName and Email from the *Logins* table and create logins and database users, and add them to the database role *CustomerRole* using dynamic SQL. When you create the login, make sure to include CHECK\_POLICY = OFF and set the DEFAULT\_DATABASE = MyGuitarShop.

The login name and database user name should be the email ID of each customer (look into the LEFT function from Chapter 9 to extract the email ID from Email). For Barry Zimmer the server login and database user should be “barryz”.  
  
The login password should be last four letters of email ID + \_$17 (look into the RIGHT function from Chapter 9 to extract the last four letters of a string). For Barry Zimmer, the password should be “rryz\_$17”.

1. (10 points) Create a procedure called “dropLogins”. In this procedure, include the following code

Declare @Roles Table(DbRole varchar(20), MemberName varchar(50), MemberSID varchar(50));

insert into @Roles EXEC sp\_HelpRoleMember CustomerRole

The procedure sp\_HelpRoleMember returns a table of users for the CustomerRole (see note below). The table is then inserted into a table variable called @Roles.

Declare a cursor for “select MemberName from @Roles” where MemberName has all the logins. Using the cursor and dynamic SQL, drop the member from the CustomerRole, drop the user and then drop the login.

Note about EXEC sp\_HelpRoleMember CustomerRole

After you execute “createLogins” and then execute “sp\_HelpRoleMember”, you should get the following table:



After you execute, “dropLogins”, this table should be empty.

1. (10 points) Use the *update\_email.xml* file and the value method to write the XQueries for the following questions:
   1. Find the email of the second customer
   2. Find the email of the customer whose last name is “Valentino”
   3. Use the data in the *update\_email.xml* to update the email addresses of the customers in the Customers table. Hints: Using XQuery, get the first, last and email addresses of each customer. Then use the Update command to update that Customer in the Customers table. You cannot use CustomerID to update as there is no CustomerID in the xml file. You do not have to use loops.