Information and Instructional Documentation

for

SQL to XML Generator

COP 4710 – Data Modeling

Dr. Behrooz Seyed-Abbassi

University of North Florida

Created: Fall 2016

By:

Brandon White

Collin McClure

David Timmers

Michael Dormann

**Table of Contents**

**a.0: Team Information**

a.0.1: Team Meetings and Member Responsibilities

**a.1: XML and SQL Explanation**

a.1.1: Extensible Markup Language

a.1.2: Structured Query Language

a.1.3: General Overview of SQL to XML Conversion

a.1.4: SQL to XML Conversion with the JAVA Application and Oracle Database

**a.2: Description of Software Design**

a.2.1: General Description

a.2.2: SQL to XML Generator

a.2.3: SQL Parser

a.2.4: XML Formatter

**a.3: Physical Development**

a.3.1: General Description

a.3.2: Finite State Machine Diagram

a.3.3: Brainstorming and Design Pictures

**a.4: Software Samples**

a.4.1: Definition 1 Simple Query

a.4.2: Definition 1 Advanced Query

a.4.3: Definition 2 Simple Query

a.4.4: Definition 2 Advanced Query

a.4.5: Definition 3 Simple Query

a.4.5: Definition 3 Advanced Query

**a.5: Software and Database Setup**

**a.6: How to Utilize the Software Sample**

**a.7: Student User Manual**

**a.8: Special Features**

**a.9: Software Demo and Presentation**

**a.0: Team Information**

**a.0.1: Team Meetings and Member Responsibilities**

Our group met several times over the course of three weeks to design and develop the SQL to XML Generator. These meetings started with general planning and proceeded through design plans, development, testing, and concluded with documentation.

Our first meeting took place on November 8th where we proceeded to briefly discuss the project and how we would handle communication and development between the team members. We decided the best development strategy for our team would be to develop the software in JAVA using Eclipse as our IDE. We also chose to use GitHub as our code repository as it has many features that are useful when several people are working on the same project.

The following team session did not take place until November 15th where we put together our basic design of the software and set team member tasks. We broke down the different SQL query possibilities and decided on a parsing technique. A finite state machine was designed to handle the different specifics of the SQL queries that could be executed by the software. We broke the software into two main components to allow for modularity and to easier divide tasks between team members. Brandon and Michael were responsible from developing the SQL parser that would read the query from the user, parser specific data from the query, execute the query against the SQL database and pass the returned data to the XML formatter. Collin and David were responsible for developing the XML formatter which took the data generated by the SQL parser and displayed XML output. The XML formatter would handle various types of tags specified within the SQL query such as grouping, alternate group names, and compression. The XML formatter would also generate a DTD file and an XSD file.

We met twice on November 17th to begin development of both the SQL parser and the XML generator. By making our software modular we were able to develop both the SQL parser and the XML generator simultaneously. Brandon and Michael proceeded with the design of the SQL parser by creating various classes to handle the queries that could be entered into the software. Collin and David began to develop the handling of the data that would be passed to the XML generator.

The following meeting took place on November 22nd and was dedicated to the continuing development of both main pieces of the software. Brandon and Michael were continuing their development by writing conditions and specifics to tokenize the input from the user and obtain the correct data from the SQL database. Collin and David were working on finalizing the XML output from the data received from the SQL parser. These final obstacles for the XML generator included grouping and compression handling.

We met again on November 28th to combine the SQL parser and the XML generator, conduct testing and debugging, make final improvements, and create all the documentation for the SQL to XML generator. This meeting lasted the longest as we had to pull all of the modules together and create the interface between them. We then had to do all the testing and debugging to ensure everything was operating correctly. This also accounted for time spent making corrections and improvement to errors that were discovered during testing. Once testing was completed we created the documentation.

**a.1: XML and SQL Explanation**

**a.1.1: Extensible Markup Language (XML)**

Extensible Markup Language, commonly referred to as XML, is a markup language used to store data in a specific format so it can be read by both humans and machines across different entities such as the internet or within a corporate network. XML is the recommended markup language to format data as recognized by the World Wide Web Consortium (W3C).

**a.1.2: Structured Query Language (SQL)**

Structured Query Language, commonly referred to as SQL, is the standard programming language used to create, manipulate, and manage relational databases. SQL allows users, specifically the database administrator, the ability to manipulate the architecture of the database to increase efficiency and make general improvements across the system. SQL was adopted by the American National Standards Institute for database systems.

**a.1.3: General Overview of SQL to XML Conversion**

The conversion of SQL to XML is the process of retrieving data from an SQL database via a query command and formatting the data returned by the query into XML format. Upon connecting to the appropriate database, the user issues a SQL query command via the SELECT statement. This query is passed to the database via the executing application and the query is executed within the database platform. The data returned from the database platform via the SQL query is returned to the executing application as an object. The data object is parsed for its individual data stored from the SQL database and that data is formatted within the executing application to display XML formatted data to the console.

**a.1.4: SQL to XML Conversion with the JAVA Application and Oracle Database**

Our application performs SQL to XML conversion as specified in the previous section. Our executing application is written in JAVA and the database platform is Oracle. Our application prompts the user with a meu. This menu allows the user to perform several tasks such as input a query or view an example query. The user can input a SQL query via the SELECT statement to issue a command against the Oracle database. The SQL query is parsed by the JAVA application to ensure it is syntactically and semantically correct. As the JAVA application parses through the query creating tokens and ensuring correctness, it also creates an ArrayList of attributes that will be queried from the database. This ArrayList will be used to display the correctly formatted XML tags with the appropriate data. The SQL query is stored as a variable and the command is executed against the Oracle database, returning the data using the ResultSet object from an existing library within JAVA. The ResultSet object contains all the information that was queried from the database. The JAVA application then executes through the ArrayList of attributes generating the appropriates XML tags to be displayed to the console. As the JAVA application generates the XML tags it parses the data stored in the ResultSet object to display the appropriate data in the correct XML tag for the user. Once the application has executed against all the attributes in the ArrayList and read all the data from the ResultSet, it generates the closing tags for the XML document and displays the closing tags to the user via the console. Finally, the user is prompted with the original menu to allow for another query to be entered.

**a.2: Description of Software Design**

**a.2.1: General Description**

The SQL to XML generator takes an SQL SELECT query, parses it for specific information to be used by the XML generator. The SQL parser is responsible for parsing through the query and storing this information. Once this is completed the SQL parser executes the query against the database specified and the data pertaining to that query is returned to the parser. This data along with the query information parsed earlier is passed to the XML formatter. The XML formatter takes the information parsed from the query and the data returned from the database and generates XML output as well as DTD output and XSD output. All of which can be displayed directly to the console for the user or written to a file to be used later.

**a.2.2: SQL to XML Generator**

The executing class, or main class, for the SQL to XML Generator calls the other classes to execute the SQL query and generate the XML output. The main class presents the user with a menu and prompts the user to select an option from the menu. The menu has three options to choose from. One option to enter and SQL query, one option to view some examples of SQL queries, and one option to quit the program. The menu is on a loop so that once a query is executed or examples are viewed, the user is prompted again to make a selection from the menu. If the user chooses to quit, the software exits. If a user chooses to view some example SQL queries, some examples are displayed to the screen. When a user chooses to enter a query, they are prompted for the query. Once the query is entered, the main class calls the SQL parser to handle the query and return the data to the main class. The main class passes this data to the XML formatter to generate XML output.

**a.2.2: SQL Parser**

The SQL parser is passed the query entered by the user in the main class and proceeds to parse through that query storing Attributes in an ArrayList before executing the query against the SQL database. The SQL query retrieves the metadata for the Attribute and with this metadata it pulls the relevant information for the SQL query and stores it in an Attribute object. All the Attribute objects are stored in the ArrayList and passed to the main method to be sent to the XML formatter. After all this information is gather, the SQL query is executed against the SQL database and the results are returned to the SQL parser in the form of a ResultSet object. The ResultSet object stores all the information from all the records that were accessed as a result of the initial SQL query. The ResultSet is also passed to the main method to be used in execution by the XML formatter class.

**a.2.3: XML Formatter**

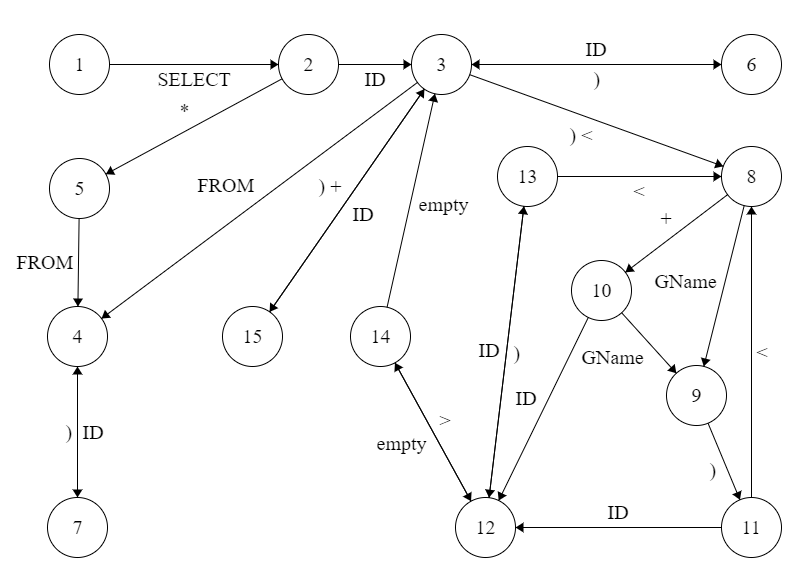
The XML formatter is executed by the main method. It is passed two parameters, one parameter is the ArrayList of Attributes and one parameter is the ResultSet that was returned to the SQL parser from the database. The XML formatter presents the user with a menu that allows the user to display the XML to the console, save the XML to a file, or both. The formatter then executes using the information stored in each Attribute object from the ArrayList and the data from the ResultSet. The XML formatter uses a nested loop design to move through the ArrayList and through the ResultSet to generate the correct XML output for the user. The Attributes stored in the ArrayList contain specific information to each attribute from the SQL database such as table name, grouping, and compression. This information is used in combination with the data from the ResultSet to generate XML tags and output. Once the XML formatter has moved through the entire ArrayList and displayed all the information in XML output to the user, another menu is prompted to the user. This menu allows the user to choose if they would like the DTD output and the XSD output displayed to the console, written to a file, or both. Based on the user selection the output is displayed to the console or saved to a file. Once the XML formatter method has generated the XML, the DTD, and the XSD it is returned to the main method. From the main method the user will see the original menu and be prompted to make another selection.

**a.3: Physical Development**

**a.3.1: General Description**

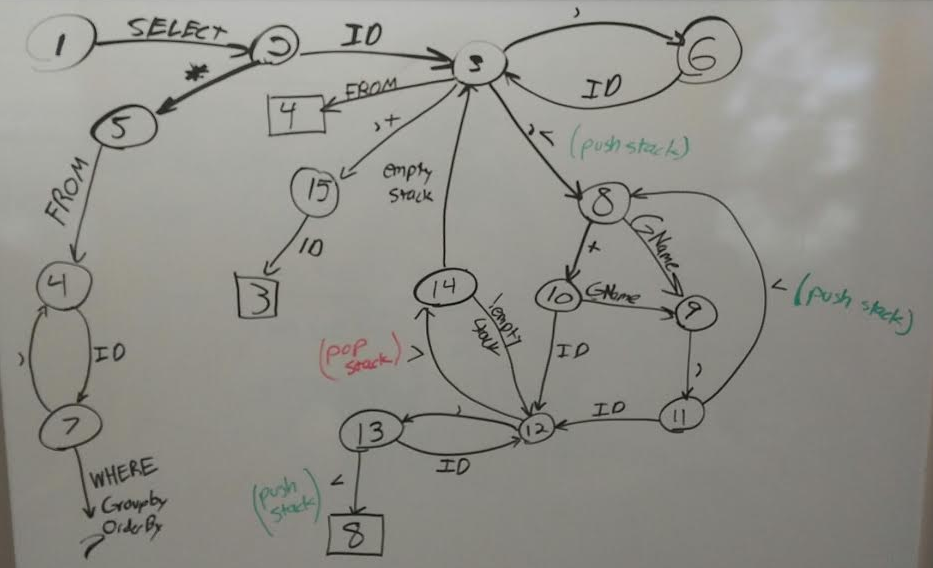
The development for the SQL to XML Generator revolved primarily around the finite state machine that was designed to handle the SQL queries passed to the application. As everything is issued to the application from the command line, there are no development diagrams or documents for a graphical user interface because one was not created. A diagram was designed to provide insight on the finite state machine that was created to handle the SQL queries that would be executed against the database. Included in this section are the original pictures of the design from the first brainstorming session on how to the team would handle the project and specifications

**a.3.2: Finite State Machine Diagram**

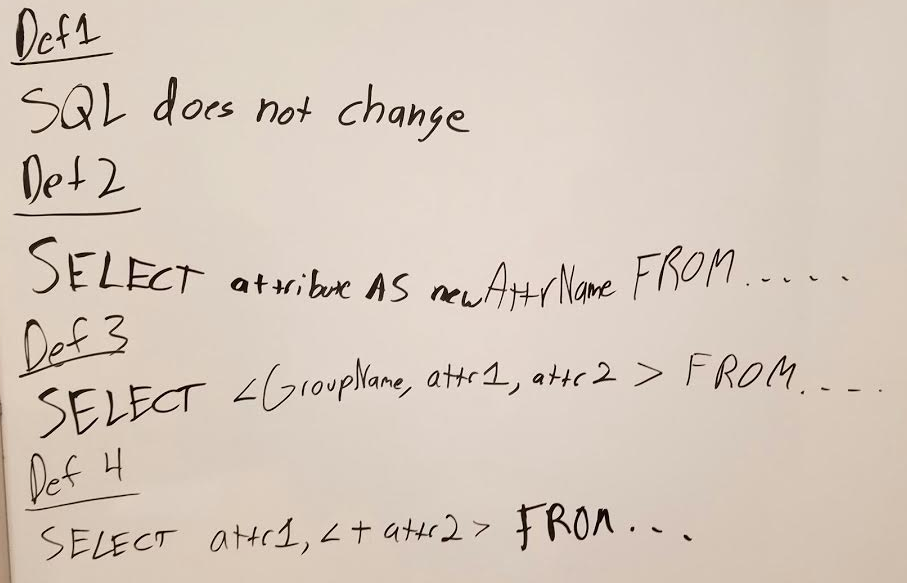
****

**a.3.3: Brainstorming and Design Pictures**

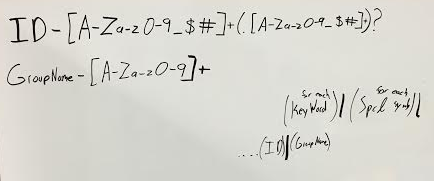
Original Finite State Machine Diagram



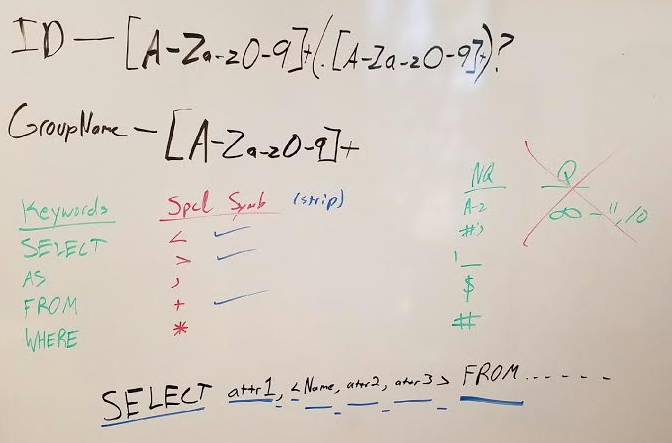
Original SQL to XML Generator Definitions



Original Grammer and Input Rules



More In-depth Input String Design



**a.4: Software Samples**