# **i1.TSQLx Engine-Data Modeling Final Project**

# **COP4710-Data Modeling**

# Spring 2017

# Submission Date: 17 April 2017

# Team Members:

# Sean Domingo

# Michael Frederick

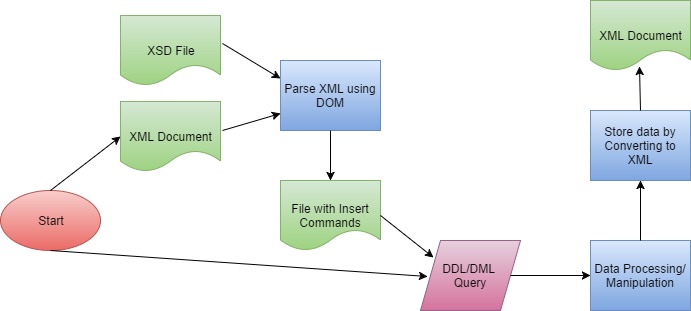
# Megan Molumby

# Mai Nguyen

# Richard(Anthony) Pratt

## i2. Description of assignment

For the Data Modeling Final Project we will be designing and implementing a temporal database for a SQL and XML software package called TSQLx. This SQL engine is able to receive SQL commands and run them. Commands included in this implementation are create database, load database, create table, insert to table, delete from table, and query table. Insertions in the database are also given a timestamp, which makes this a temporal database. An additional feature is the engine’s ability to receive an XML file, convert it to an output file of SQL commands and then with a different command prompt, inserts those SQL commands from the output file into the database. This engine is written in Java and the database is stored persistently in XML files. The reason for this design decision is it allowed our team to reutilize the code for parsing the XML to retrieve, manipulate and create data from the database, without having to write another program to manipulate the database. Our database is structured with a catalog and corresponding tables. The purpose of this TSQLx package is that it will be used for future educational and research purposes.



## i3. Class presentation outline

1. Overview/Description

* Explanation of our TSQLx engine
  + How the engine functions
  + The commands that it can run
  + How it was created
  + Intro into the design
    - Introduce DOM Parser and what it is
    - Benefits of using a DOM Parser and what it allowed us to do

1. TSQL Engine Software Design

* Go over image
* Explain how the XSD document and XML file are parsed with the dom parser.
* A file is created with insert commands which leads to the DDL/DML Query.
* During query, the data is processed or manipulated and stored after being converted into an XML document. It is finally saved as an XML document

1. Database Design

* All files are stored in XML format
* The database catalog stores all of the associated table and attributes
* The data was stored in an XML format because we utilized a Dom parser which converts XML to SQL
* The parser allowed us to reuse code without having to create another program.

1. SQL Command Implementation

* Go over image that lists all of the classes and how the all extend to the parent SQLCommand class

1. XML Dom Parser

* Dom parser allows us to manipulate XML files
* It was used to modify files, delete elements in XML, create new XML files, query XML files
* CREATE table command with options
  + How we created the command and how it will be implemented
  + Go over the format structure
  + Explain how there needs to be at least one field name declared
  + In order for the query to work the table name cannot be in the database
* DROP table command
  + How we created the command and how it will be implemented
  + Go over the format structure
  + In order for the query to work the table name must already exist
* COMMIT command
  + How we created the command and how it will be implemented
  + Go over the format structure
  + Used at the end of a transaction and makes all the changes permanent

1. Transformer

* Transformer class used to output entire XML content to stream output

1. Data Manipulation Level

* Explain how we created the commands and the structure for formatting each one.
* INSERT commands with options
  + How we created the command and how it will be implemented
  + Go over the format structure
  + At least one literal for each column must be entered that is not nullable
  + The table name must exist in the current database for the query to work
* Convert XML file to INSERT commands
  + How we created the command and how it will be implemented
  + Go over the format structure
  + The filename is what needs to be converted into SQL commands
  + In terms of the structure, explain how the filename must already exist in order for the query to work
* INPUT XML INSERT file commands
  + How we created the command and how it will be implemented
  + Go over the format structure
  + Each insert command in the file is created by the XML converter
* DELETE command with options
  + How we created the command and how it will be implemented
  + Go over the format structure
  + In the conditional statement the table-name has to exist along with the field names
* Simple SELECT command
  + How we created the command and how it will be implemented
  + Go over the format structure
  + Table-name must exist in current database for the query
* tSELECT command
  + How we created the command and how it will be implemented
  + Go over the format structure
  + “T” adds the date and time to the query
  + Similar to the regular SELECT command the table name must already exist
* [t]SELECT command with [WHERE clause]
  + How we created the command and how it will be implemented
  + Go over the format structure
  + In terms of the structure, explain how the database name cannot exist in order for the query to work

Tablename and field names must exist for the query

## i4a. Team Member Responsibilities

|  |  |
| --- | --- |
| Team Member | Responsibilities |
| Sean Domingo | * Implementation of SQL commands   + Insert Command   + Create Database   + Create Table   + Drop Database   + Drop Table   + DOMUtility   + WriteDOMtoXML |
| Michael Frederick | * User Interface * SQL Parser and Command Generator * General Help |
| Megan Molumby | * XMLtoSQL Parser * XSD Parser * Implementation of SQL commands   + Delete   + Select   + tSelect |
| Mai Nguyen | * SQL Parser and Command generator |
| Richard Pratt | * Documentation * Testing |

## i4b. Team Meetings

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Time | Topics Discussed | Team Members in Attendance |
| 3/29/17 | 1:00pm | * Database Design * Team Member's responsibilities * Future Meeting Times * Questions for Abassi:  1. Does this database need to be retrievable by others or is it more of a “personal” database?    1. Do we need to account for multiple users updating the db?    2. No. 2. Particular error control for ill formatted XML document? 3. Check syntax 4. What are the limits on libraries? I.e. Is there a limit to the number of databases available? 5. Open close one at a time 6. Ask thoughts on reuse of parser for XML to reparse into file after startup. 7. Anything you want 8. If there is an xsd does that mean the table does not exist? And do we need to make a create table command, when given an XML file for insertion? 9. XSD could be contained in XML, it is not recommended. 10. Can have multiple XML and XSD files that can load multiple times. | Sean, Michael, Megan, Mai |
| 4/4/17 | 2:00pm | * Function calls for xml parsing to the SQL parser * Github standards | Michael, Megan, Mai |
| 4/5/17 | 11:00pm | * Modifying an xml document | Sean, Megan |
| 4/5/17 | 1:30pm | Questions for team meeting on Wednesday:   1. Will the XML input file format be well-formed?    1. <https://www.w3schools.com/xml/xml_validator.asp> 2. Is the standard format for XSD?    1. <https://www.w3schools.com/xml/schema_intro.asp> 3. What data types are we account for? 4. Table is going to come from XSD and XML. There is a translation from XSD to SQL datatypes 5. Can users type multiple commands on a line? 6. It is up to you. One command with a semicolon is the usual. You can buffer the commands. Your commit could be the end of the command 7. Can we autocommit for everything? 8. Yes, typical databases are autocommit. You could buffer the commands if you want. It would be nice to turn it off. Semicolon says it commits regardless. 9. Are sql commands case sensitive? 10. They are case sensitive when checking the table structure. 11. What are the table headers for the JAG database and what are the data types? 12. XSD to SQL datatypes   If record already existing still insert with new data time  Date: is xsd format mmddyy  (When parsing the xsd, current implementation is such that we will be ignoring all extra attributes that we are not concerned with.) | Sean, Michael, Mai. Richard, Megan |
| 4/5/17 | 2:00 pm | Assigned specific tasks to each member | Michael, Mai, Richard, Sean, Megan |
| 4/10/17 | 1:45 pm | General Body Meeting: Update on each person’s current progress. | Richard, Megan Michael, Sean, Mai |

## i5. Catalog and Data Structure

Catalog File Structure Diagram

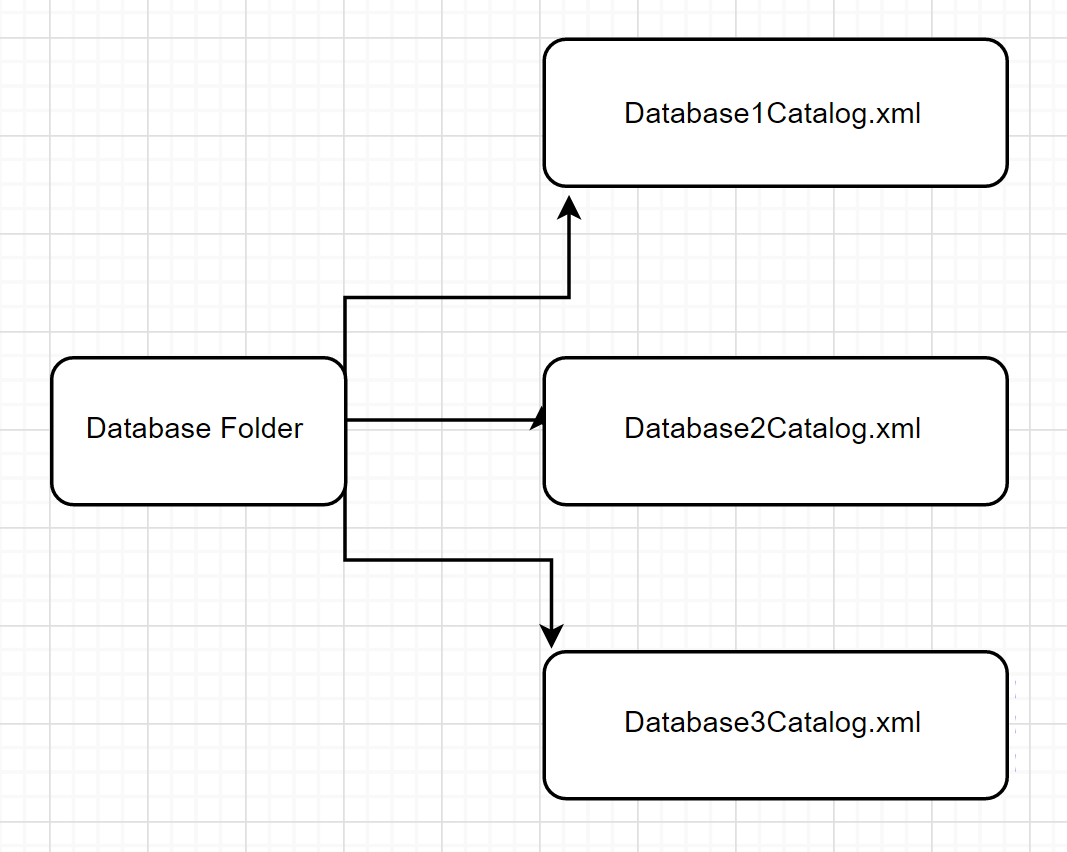
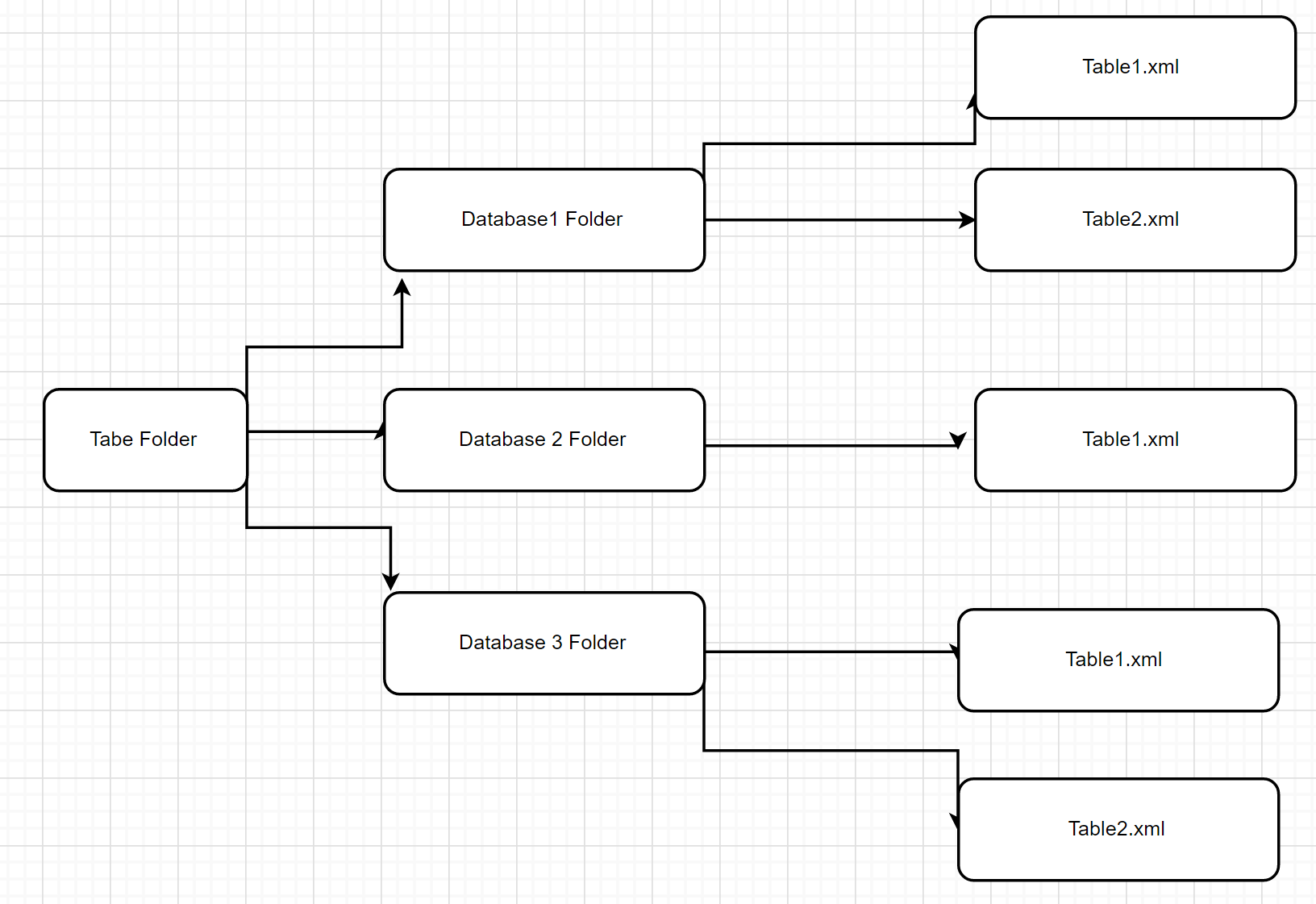
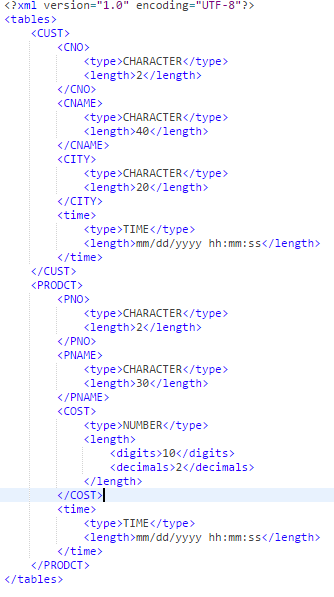
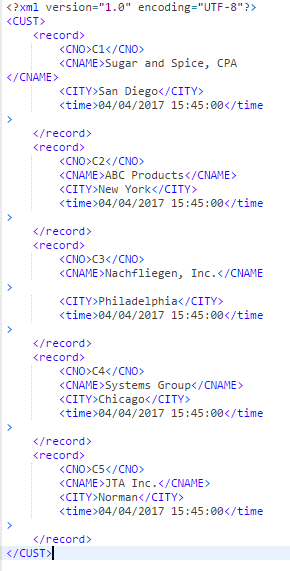


Table File Structure Diagram

We decided to use the XML file format for the structure of our catalog and datafiles. The database catalog will store the tables associated to it along with their attribute structure. The JAG database catalog file would look like this:



With all of the database catalog files being in a “database” directory. For the records, we decided to place the record values in an xml file for the table name. So the CUST table’s records from the JAG database would be stored in CUST.xml file. The XML format is similar to:



We choose to use this because we are able to use same code for reading an xml file for conversion of an xml file into sql commands for the reading and writing of the data files.

## i6. Algorithm

1. Get User Input
2. Determine User command based on first word
   1. Create

*Check next word*

* + 1. Table
    2. Database
  1. Drop

*Check next word*

* + 1. Table
    2. Database
  1. Insert
  2. Delete
  3. Convert
  4. Input
  5. Select
  6. TSelect
  7. Help
  8. Exit

1. Check Syntax and Semantics of User Command

*User’s command is:*

* 1. Create table
     1. Get table name
        1. Check that the current database does not contain that table
     2. Get column definition
        1. Get column name
        2. Get column type
        3. Get column length
        4. Go to 3.1.2 if another column is defined
     3. Append to Database Catalog XML
        1. Add Table name tag
        2. Add Column definition for each column definition
  2. Create Database
     1. Get Database Name
     2. Check if Database Catalog XML exists
     3. Build Database Catalog XML
     4. Create Database Table directory
     5. Save Database Catalog XML
     6. Change the current database to new database
  3. Drop Table
     1. Get table name
     2. Check if table exists in current database
     3. Remove table from database catalog xml
     4. Remove table data xml
  4. Drop Database
     1. Get database name
     2. Check if database catalog xml exists
     3. Delete database table directory
     4. Delete database catalog xml
  5. Insert
     1. Check if columns selected
        1. If none, assume all columns for insert
     2. Get table name
     3. Check insert values
        1. Find catalog entry for table’s column
        2. Check insert value’s type
        3. Check insert value’s length
     4. Check that all non-nullable columns have values
     5. Open table data xml
     6. Append new record
  6. Delete
     1. Finds records from given table that validate the given delete WHERE condition
        1. If valid:
           1. Delete record from table
        2. If not valid:
           1. Skip
        3. Display to user number of record deleted.
  7. Convert
     1. Get xml and xsd filenames
     2. Read xsd for xml constraints
     3. For each xml element
        1. Check that value matches xsd constraint
        2. Add to Insert Command File
        3. Save Insert Command File
  8. Input
     1. Get SQL input filename
     2. Read input line by line
        1. Pass line to parser to get SqlCommand
        2. Execute SqlCommand
     3. Show notification that input is done
  9. Select
     1. Takes requested attributes from specific table, prints those attributes with their corresponding values for each record.
     2. If the attribute value is \*
        1. Prints all records with all attributes.
     3. If there is a WHERE condition:
        1. For each record, finds if the condition clause is valid
           1. If valid:

Print requested attributes of that record

* + - * 1. If not valid:

Skip

* + 1. Displays results to user.
  1. TSelect
     1. Takes requested attributes from specific table, prints those attributes with their corresponding value for each record with the time attribute at time of insertion.
        1. If the attribute value is \*
           1. Prints all records with all attributes
        2. If there is a WHERE condition
           1. For each record, finds if the condition clause is valid

If valid:

Print requested attributes of that record and time of insertion attribute.

If not valid:

Skip

* + - 1. Display results to user.
  1. Help
     1. Check if a database is selected
        1. Show Database Catalog level commands
        2. Show Database level commands
  2. Exit
     1. End Program

1. Create SqlCommand object corresponding to the User Command
2. Execute SqlCommand
3. Return to 1.

## i7. How to Run:

1. Run ‘make’ command to compile program.
2. Execute start engine script
   1. In Linux this is: ./startEngine
3. The Program will start and you will be at a command line with “sql>”
4. You will be starting at the Database level and can issue any queries
   1. You may use the HELP command to get a list of Database level queries
5. Once a database has been loaded or created, you will be Database Catalog level. You can issue queries to manipulate the table inside the database.
   1. Again, you may use the HELP command to get a list of Database Catalog queries. HELP also shows the current database loaded
6. You can leave the program at any time with EXIT

*Included in deliverable is a Sample Query.txt with a list of all SQL Queries that can be issued at each level.*

## i8. Program ListingDM-ObjectClasses.jpg

**Commit**

Commit is a class and holds the data field description relating to database name from its extension to SQLCommand.

1. **executeCommand()**

The program will print a message to notify the user that changes have been committed to the specified database.

**CreateDatabase**

This class contains the WriteDomtoFilewriter, DOMUtility, Path, Document, and File objects as attributes. This class is used to instantiate the createDatabase object in order to create a database catalog file in /database/ folder as well as a folder in the table folder in the form /tables/<databaseName>/databaseName.xml. This class extends SQLCommand class.

1. **executeCommand()**

Creates the database catalog xml in the databases directory. Then creates a database directory in the table’s directory to hold table data.

**CreateTable**

CreateTable is a class and holds the data field description relating to database name from its extension to SQLCommand. Additionally, CreateTable also contains data field description relating to the table name, an array of field names (columnNames), an array of field types (columnTypes), an array of field max lengths (columnLengths), and array that determines whether a field can be null (Nullable). The arrays “columnNames”, “columnTypes”, and “columnLengths” are of type String, while array “Nullable” is of type boolean. Other objects within CreateTable includes a WriteDOMtoFile method, DOMUtility constructor, Path class (tablePath and dbPath), Document interface (tableFileDOM and dbFileDOM), and File class (tableFile and databaseFile). WriteDOMtoFile is a method within DOMUtility. Its primary function is to write a DOM document to a file. DOMUtility is a constructor with various methods relating to parsing an XML document. Path is an instance used to specify file location or directory and locate a file. The Document interface serves as the root of a document tree and as the primary access to the data within the XML document. The class File is used for the creation of files and directories.

1. **executeCommand()**

Adds to the Database catalog the table and column attributes. Then creates the table data xml in the tables/database directory using the createFile() method

1. **createFile()**

Creates the file that was given to it.

**Delete**

Delete is a class and holds the data field description relating to database name from its extension to SQLCommand. Additionally, Delete also contains data field description relating to the table name (of type String), a DOMUtility object for accessing and modifying the XML document. A Path object to find the specific table in the database. It also contains the condition that follows a “where” (“whereConditional”) in the form of a String array. .

1. **executeCommand()**

Iterates through the table data xml looking for records that satisfy the where conditional. For each record that it finds, it removes from the table data xml.

1. **fileExist(Path tablePath)**

Verifies that the table exists in the file system.

**DropDatabase**

DropDatabase class used to find and drop the database catalog file as well as the associated table folder which contains the table directory which holds the various table within the database.

1. **executeCommand()**

Removes the database table directory and all able data files. Then removes the database catalog xml

**DropTable**

DropTable is an object class and holds the data field description relating to database name from its extension to SQLCommand. Additionally, DropTable also contains data field description relating to the table name.

1. **executeCommand()**

Remove the table data xml and the table entry in the database catalog xml.

**Insert**

Insert is an object class and holds the data field description relating to database name from its extension to SQLCommand. Additionally, Insert also contains data field description relating to the table name, an array of field names (columnNames) of type String, an array of field values (columnValues) of type String, a WriteDOMtoFile object, a method called “writer”, a DOMUtility object called DOMUtil, a Path object called tablePath, a Document interface called tableDOM and a File object called outputFile. The WriteDOMtoFile object’s primary function is to write a DOM document to a file. DOMUtility is a class with various methods relating to parsing an XML document. Path is an instance used to specify file location or directory and locate a file. The Document interface serves as the root of a document tree and as the primary access to the data within the XML document. The class File is used for the creation of files and directories.

1. **executeCommand()**

Appends the record the the table data xml.

**LoadDatabase**

LoadDatabase is an object class and holds the data field description relating to database name from its extension to SQLCommand.

1. **executeCommand()**

executeCommand() loads the specified database and puts the user at a database catalog level.

**SQLCommand**

Abstract class to hold the database name of what the child command is operating on. The executeCommand() method is abstract and to be implemented in each of the SqlCommand children.

1. **getDatabaseName()**

getDatabaseName() returns the name of the database.

1. **executeCommand()**

executeCommand() is an abstract class that is to be defined by the children.

**SQLParser**

The class SQL Parser has 2 global arraylists of type token. The first list holds strings created through the StringTokenizer and the other list holds strings and operands that were separated from one another. Arraylists “columnNames”, “columnTypes”, “columnLength”, and “columnNullable”, and array “whereConditional” are global variables that contains information that is used to create command object classes.

1. **executeSQLParser()**

Information regarding the commandline input and current database is passed to this class. Its primary function is to execute the main methods to parse input from the command line.

1. **resetParser()**

The primary function of resetParser() is to assign each global variables an empty object to make sure that previous data is cleared and that the variable does not give false information to the parser.

1. **generateTokens()**

The purpose of the generateTokens() method is to use StringTokenizer to break the string from command line into smaller strings, according to whitespaces. It calls the method seperateOperands() and passes the current token to break strings into simple form by seperating operands from other parts of the string. Then it converts all keywords defined in the global array (“keywords”) into uppercase using the method convertUpperCase() and stores in an arrayList called finalTokens.

1. **checkFirstChar()**

The primary function of checkFirstChar() is to check the first character of every string passed to this method to make sure it is either an uppercase letter or lowercase letter.

1. **convertUpperCase()**

The primary function of convertUpperCase() is to convert all keywords into uppercase to have a consistent command and allow users to ignore cases in keywords when forming queries.

1. **seperateOperands()**

The primary function of seperateOperands() is to simplify strings formed by the StringTokenizer and form a separate token for the operand.

1. **isLegal()**

The primary function of isLegal() is to check every character in every token for illegal characters that do not appear in our SQL grammar.

1. **isOperand()**

The primary function for isOperand() is to check the string and compare it to our global array of operands to return true if it matches an operand in the array or false if it does not match any operand in the array.

1. **parseTokens()**

The primary function of parseTokens() is to determine which parsing path to take. The implementation of a switch allows the program to determine which path to take by comparing the first token to each of the cases. Some cases will have another switch within to take into consideration commands beginning with two keywords.

1. **generateCreateTable()**

The primary function of generateCreateTable() is to create a new CreateTable object and to assign values to the data fields defined in the CreateTable object. This program will check the first character of the table name by calling checkFirstChar() and check if the table already exists using makeTableCheck(). Upon the success of the evaluation, the program will assign appropriate values to the data fields to CreateTable.

1. **generateCreateDatabase()**

The primary function of generateCreateDatabase() is to create a new CreateDatabase object and to assign it the database name specified in the command. The program will only create a new CreateDatabase object if the name of the database does not already exist.

1. **generateDropTable()**

The primary function of generateDropTable() is to check if the table exists within the current database before creating a new DropTable object and to assign values to the data fields defined in the DropTable object.

1. **generateDropDatabase()**

The primary function of generateDropDatabase() is to check if the database exists before creating a new DropDatabase object and assigning values to its data field.

1. **generateSaveDatabase()**

The primary function of generateSaveDatabase() is to create a SaveDatabase object only if the user is in the database specified in the command.

1. **generateLoadDatabase()**

The primary function of generateLoadDatabase() check if the database exists before creating a LoadDatabase object and assigning values to its data field.

1. **generateCommit()**

The primary function of generateCommit() is to create a Commit object.

1. **generateSelect()**

The primary function of generateSelect() is to create a Select object and assigning values to its data fields. It checks for whether the command lists field names or if there is a “\*” to assign values to an array. It also checks the where conditional statement to assign appropriate values to each array. Upon the success of these criteria, it will create the Select object with the correct assigned values.

1. **generateTSelect()**

The primary function of generateTSelect() is to create the TSelect object and assign values to its data fields. The implementation of generateTSelect() is similar to generateSelect().

1. **generateInsert()**

The primary function of generateInsert() is to check if the specified table exists and if the number of field names match the number of literals defined in the command before creating and Insert object and with the correct field values.

1. **generateDelete()**

The primary function of generateDelete() is to check if the records exist on an existing table in the current database before creating the Delete object with appropriate values for data fields.

1. **checkIfFileExists(String str)**

checkIfFileExists() checks if the file pointed at by the string exists in the system.

1. **checkEndOfCommand()**

checkEndOfCommand() checks if the token passed is a semicolon.

1. **badEndofCommand()**

badEndofCommand() throws an exception to notify the user that the program is not able to determine the end of the command.

1. **getColumnDefinition()**

getColumnDefinition() checks the field type defined and adds it to the appropriate data structure based on whether it is a number, integer, character, or date.

1. **convertObjectArrayString()**

convertObjectArrayString() converts objects to strings and assigns the strings to a new array.

1. **convertObjectArrayBoolean()**

convertObjectArrayBoolean() converts objects to booleans and assigns the boolean to a new array.

1. **makeDatatypeCheck()**

makeDatatypeCheck() checks the data passed to make sure it matches follows the data fields defined for field types within the tables.

1. **makeColumnCheck()**

makeColumnCheck() checks the field names to make sure the specified field name exists on the current table.

1. **makeTableCheck()**

makeTableCheck() checks the table names to make sure the specified table exists within the current database.

1. **makeWhereConditional()**

makeWhereConditional() creates a three tuple array.

1. **getTableColumns()**

getTableColumns() retrieves all columns for the table as specified.

1. **isColumnNullable()**

isColumnNullable() returns true if a column is nullable and returns false otherwise.

**SaveDatabase**

SaveDatabase is an object class and holds the data field description relating to database name from its extension to SQLCommand.

1. **executeCommand()**

Saves the database to the file system. Then places user in the database level.

**Schema**

Schema is a class, created to hold the data retrieved from an XSD document. One schema object represents one column or attribute definition in a table. It contains getters and setters for each of the potential elements of an attribute definition. For example, its name, data type, length of characters whether it can be null, it date, the table name it belongs to and if it is a decimal how long it can be.

1. **setName()**

Allows the user to assign a value to name.

1. **getName()**

Allows the user to retrieve the value of name.

1. **setDataType()**

Allows the user to assign a value to dataType.

1. **getDataType()**

Allows the user to retrieve the value of dataType.

1. **setLength()**

Allows the user to assign a value to length.

1. **getLength()**

Allows the user to retrieve the value of length.

1. **setMin()**

Allows the user to assign a value to min.

1. **getMin()**

Allows the user to retrieve the value of min.

1. **setFraction()**

Allows the user to assign a value to fraction.

1. **getFraction()**

Allows the user to retrieve the value of fraction.

1. **setTableName()**

Allows the user to assign a value to tableName.

1. **getTableName()**

Allows the user to retrieve the value of tableName.

1. **setDate()**

Allows the user to assign a value to date.

1. **getDate()**

Allows the user to retrieve the value of date.

**Select**

The Select class handles queries from the user during a SELECT command. A select command can also contain a WHERE condition, which this class handles as well. Additionally, Select contains data field, table name (of type String), a DOMUtility object for accessing and modifying the XML document. A Path object to find the specific table in the database. It also contains the condition that follows a “where” (“whereConditional”) in the form of a String array and String array of names, which correspond to the requested columns.

1. **executeCommand()**

Iterates through the records in the table data xml. Pulling all records that match the where conditional.

1. **fileExist(Path tablePath)**

Check if the table data file exists.

**TSQLEngine**

Class that holds the main driver for Java applications. Creates the DataEngine object and calls the runEngine() method.

1. **main()**

Creates the DataEngine object and start program loop.

**tSelect**

tSelect is a class that handles Select queries with a time signature. The constructor holds the data field description relating to database name from its extension to SQLCommand. Additionally, tSelect also contains data field description relating to the table name (of type String), a DOMUtility object for accessing and modifying the XML document. A Path object to find the specific table in the database. It also contains the condition that follows a “where” (“whereConditional”) in the form of a String array and String array of names, which correspond to the requested columns.

1. **executeCommand()**

Iterate through the records in the table data xml. Pulling all records that match the where conditional. Also shows the internal time column.

1. **fileExist(Path tablePath)**

Checks if the table data file exists

**UserInterface**

1. **showProgramHeader()**

Show to the User the introduction to the program.

1. **getInput()**

Gets a query from the user

1. **showUser()**

Shows the user the specified message.

1. **showDatabaseHelp()**

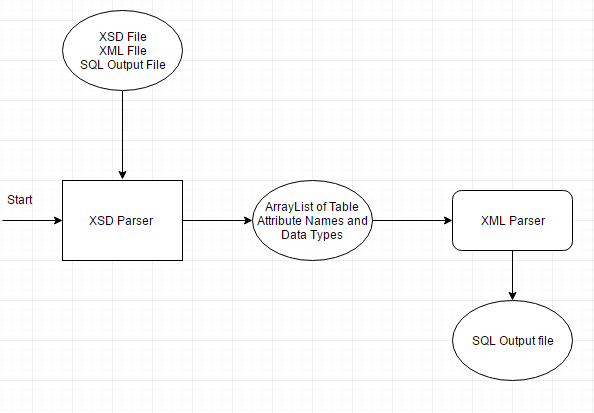
Shows available commands at the Database level.

1. **showTableHelp()**

Shows the available commands at the Database Catalog level.

1. **showUserError()**

Shows a generic error when the program get give a more specific error.

**XMLToSQLParser**

XMLToSQLParser class that takes care of parsing the XML, converting it to SQL insert commands and writing them to an output file to be used later in the SQL engine. This class’ only method is parse, which uses an Array List of Schema to identify the elements in the XML document, in conjunction with the classes from the javax.xml.parsers package. The javax.xml.parsers package has multiple classes which help to modify, access and write xml documents. It then accesses the XML input file and a proceeds to use the values

1. **parse(ArrayList<Schema> table, String xmlFileName, String outputFilename )**

**XSDParser**

The XSDParser utilizes the Dom packages as well to access the nodes in the XSD file. This class also created an Array List of Schema objects, which represents a table. This class is strictly for pulling the data types and names of the accompanying XML file. This information is then passed to the XMLtoSQLParser.

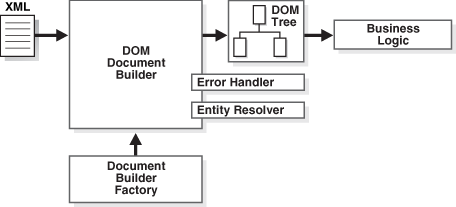
1. **parseXSD(String xmlFilename, String xsdFilename, String outputFilename)**

**DataEngine**

Main class that contains the command interface to get queries from user.

1. **runEngine()**
2. **convertXmlToSql()**
3. **parseCommand()**
4. **inputFileIntoDatabase()**
5. **checkFilesystem()**
6. **xsdParser()**

**DOMUtility**



DOMUtility is a utility class used to facilitate DOM operations which are repetitively used throughout various file manipulation operations the program. The DOMUtility contains the attributes DocumentBuilderFactory, DocumentBuilder, and a Document attributes which are required objects utilized build DOM Document objects. The DOMUtility contains the methods XMLtoDOM and createDOM.

1. **DOMUtility()**

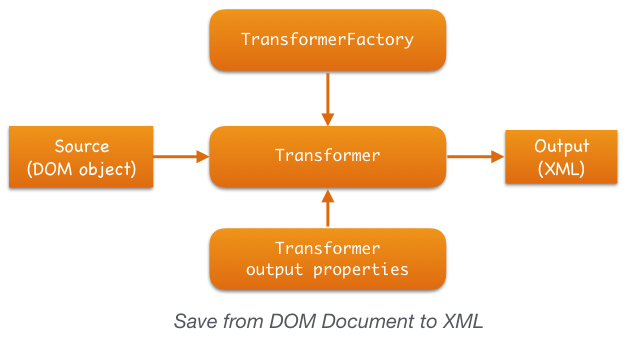
The DOMUtility constructor instantiates a new instance of DocumentBuilderFactor and DocumentBuilder.

1. **XMLtoDOM()**

The XMLtoDOM method receives an XML file as a File object, instantiates a new Document object from the Document builder, and utilizing the DocumentBuilder’s parse methods returns a parsed XML file as a document object. This method then returns the Document object. This method is utilized on all operations which retrieve the preexisting database catalog files and/or the table files and performs manipulations on them.

1. **createDOM()**

createDOM is a method that instantiates a Document object using the DocumentBuilder object and returns an empty DOM document object. This method is utilized when creating database catalog and table files.

**WriteDOMtoFile**

WriteDOMtoFile is a class which is used to write the DOM objects to file. This class is used often throughout database catalog and table file manipulation. WriteDOMtoFile contains DOMSource, StringWriter, StreamResult, TransformerFactory, and Transformer objects as attributes. WriteDOMtoFile is utilized each time the DOMUtility object performs write operations back to file from DOM objects.

1. **WriteDOMtoFile()**

The WriteDOMtoFIle constructor points transformer to null.

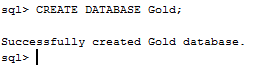
1. **write()**

## i9. Input and Output

A1. Create Database

Before:





After:



A2. Drop Database

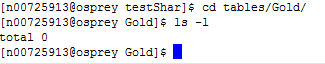


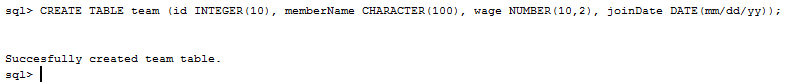
A3/A4. Save and Load Database



B1. Create Table

Before:





After:



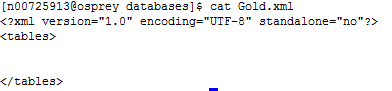
B2. Drop Table

Before:





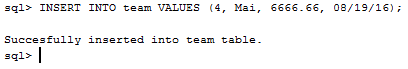
After:



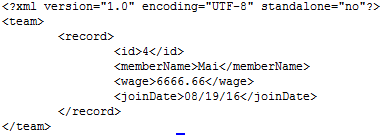
C1. Insert

Before:





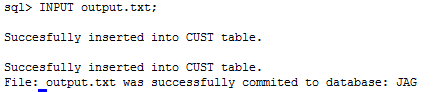
After:



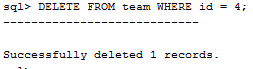
C2. Convert XML To Insert Commands



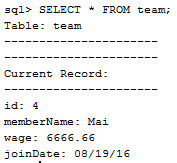
C3. INPUT output



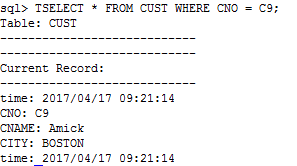
C4.Delete



C5. Select



C6. tSelect



## i10. Supporting Material

**Document Object Model (DOM) Parser**

The DOM Parser defines an application programming interface that allows programs to access and update the style, structure, and contents of a well-formed XML file. It is very useful in events where information in a XML document is accessed more than once. When a user uses the DOM parser to parse an XML document, the program returns a tree structure with all of the elements relating the document. Common DOM methods allow the user to retrieve the root element of an XML file and returns children and siblings of nodes. The resource below was used as a reference for creating the DOM parser and for understanding all of its functions and capabilities.

<https://www.tutorialspoint.com/java_xml/java_dom_parse_document.htm>