

Updating XML Views Over Relational Databases

A Major Qualifying Project Report

submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Richard Omar

Richard Tamalavitch

Date: March 4, 2005

Professor Murali Mani, Major Advisor

Table of Contents

1. Background	2
1.1 Introduction	2
1.2 The TPC-H Benchmark [4]	2
1.3 What is CoT? [2]	4
1.4 Possible Solutions to Updating XML	6
1.4.1 <i>The Tatarinov, Ives, Halevy, Weld (TIHW) Algorithm [1]</i>	7
1.4.2 <i>The Braganholo, Davidson, Heuser (BDH) Algorithm [3]</i>	8
2. The Algorithm:	9
2.1 Introduction:	9
2.2 The Main Program:	10
2.2.1 <i>Overview of XPath Parser Program</i>	10
2.2.2 <i>Overview of Parser Graphical User Interface</i>	11
2.2.3 <i>Overview of Main Parser Algorithm</i>	14
2.2.4 <i>Propagating Changes of Objects from XML View</i>	14
2.2.5 <i>Insertion of a Single Element</i>	15
2.2.6 <i>Updating a Single Element</i>	16
2.2.7 <i>Insertion/Deletion of a Whole Tree</i>	17
2.3 The DTD Parser:	18
2.3.1 <i>Overview</i>	18
2.3.2 <i>The File Structure</i>	19
2.3.3 <i>The Methods</i>	20
3. Conclusions:	21
4. Appendix A	23
Bibliography	42

Table of Figures

Fig. 1: The TPC-H Benchmark.....	3
Fig. 2: Example of Tables for DTD	5
Fig. 3: Tree Built from DTD	5
Fig. 4: Schema created from CoT.....	6
Fig. 5: Example of a Tree Structure.....	7
Fig. 6: Example of a Query Tree	9
Fig. 7: Program Structure	11
Fig. 8: Parser GUI.....	12
Fig. 9: The Drop Down Menus	12
Fig. 10: Querying the Database from the Menu	13
Fig. 11 Parsed XPath Statement and Modified Table.....	13
Fig. 12: Example of XML Insert Statement	16
Fig. 13: Example of XML Update Statement.....	17
Fig. 14: Example of XML Tree Insert	18
Fig. 15: Structure of TreeNode.....	19
Fig. 16: DTDParser Functionality.....	20

1. Background

1.1 Introduction

As technology develops and the Internet becomes a more dominant source of information sharing, storage, and retrieval (using XML views onto relational databases), new methods for translating the schemas given in the relational database to the XML view must be provided. Algorithms must be created to work on making this translation faster and more efficient, and allow for an expansion to already existing algorithms for this problem. When expanding these already existent algorithms, new functions such as insert, delete, and update must come into play. This major-qualifying project will add the three aforementioned functions, by using the TPC-H Benchmark as a test case, to expand the functionality and use of the CoT algorithm. Our solution includes a parser to parse the XPath update statements, and convert the XPath statements into SQL commands to be carried out.

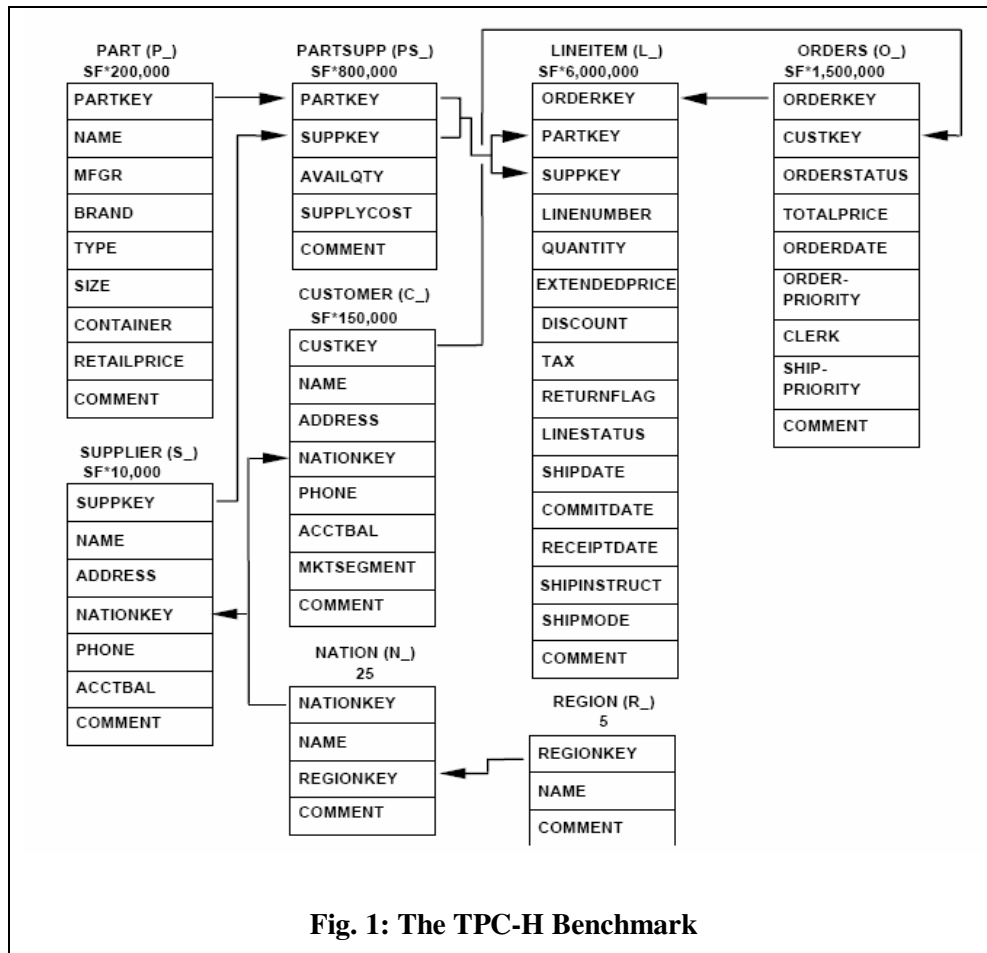
1.2 The TPC-H Benchmark [4]

The TPC-H benchmark is a database design representative of a business that needs to distribute a product worldwide, and provides an opportunity to run complex queries on this database. In a real world application of such a database, modifications and updates would be as important as queries, and would need to be simultaneously maintained. The updates modeled for the benchmark do not attempt to reflect these real world situations, as that is not the main purpose of the benchmark. In relation to XML translation and updating, more focus would probably be placed on this aspect of the database, as the structure of XML does not lend itself to representing foreign key

constraints as well as a relational database, and an update that modifies a single table would seem to result in an update of the entire XML representation of the database.

The tree structure that would need to be created to represent the database structure would also result in some branches that would result in difficult processing of the data, especially on inserts into tables where these connections back up the tree are present. Solving how to handle these problems will be one of the main tasks in creating the algorithm to insert data into these tables (Figures 2 and 3).

The schema for the TPC-H benchmark is not difficult to understand, and for the most part is a simple and obvious representation of what a business that distributed a product to customers around the world would need. (See Fig. 1)

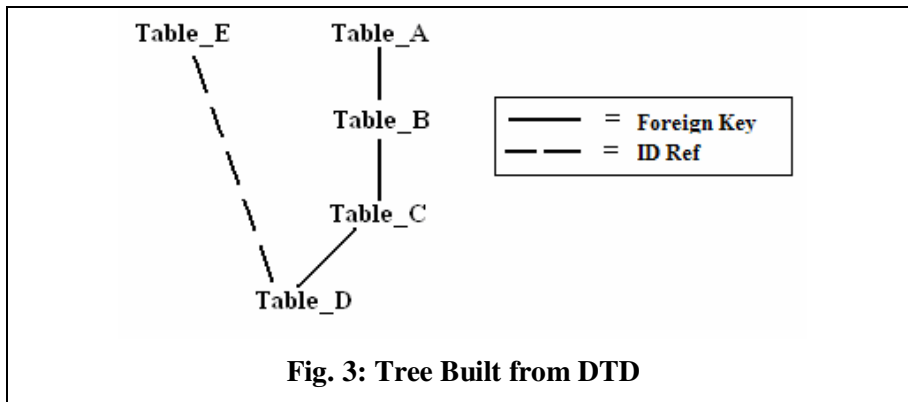
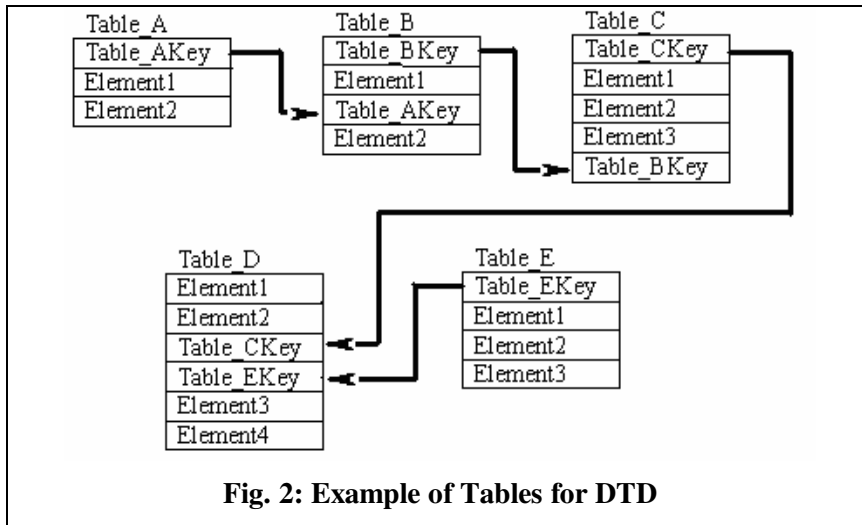


The relations that pose a problem as far as inserting and updating the tables represented in an XML view are PARTKEY in PART and PARTSUPP, as well as the connecting of ORDERKEY in ORDERS and LINEITEM. The reason for this difficulty is that these relationships use ID references, which are handled differently than the foreign key relationships in the rest of the TPC-H Benchmark. Besides that, the database structure lends itself quite well to the tree structure that would be needed to easily represent the data in an XML view, as well as update it.

While the focus of the benchmark is to model the analysis end of a business environment by computing trends and producing refined data, this does not really concern much with the XML views of the data that are produced with the CoT algorithm, or updating those views. The TPC-H benchmark will be used because of its ease of being represented as a tree structure, which is the natural form of an XML view. The structure of its schema limits the complications that result from foreign key constraints and ID references to ancestor nodes in the tree structure.

1.3 What is CoT? [2]

CoT is an algorithm for translating schemas used in relational databases to schemas used for XML views onto those relational databases. The CoT algorithm takes a relational database and looks at the tables present within it. Based on how the tables are connected, i.e. through primary keys and foreign keys, CoT builds a tree-based structure that defines the organization for the DTD, or Document Type Definition, for the XML view. CoT will make one table a parent of the other one, and so on. (See Fig. 2 & 3)



However, there are some exceptions to this rule. For instance, Table_C is a parent of Table_D, through the foreign key relationship, but Table_E can also be considered a parent to Table_D, CoT will make an IDREF from Table_D to Table_E. (See Fig. 3) This ensures that no node has more than one parent, through the foreign key relationship, but allows for the possibility of a node having one or more relationships to other nodes through foreign keys. While CoT will create these foreign keys, multiple executions of CoT might not always create the same IDREFs from node to another. For example, in Fig. 3 Table_D has an IDREF to Table_E, but if CoT were run again, there might be an

IDREF from Table_D to Table_C instead. Once the tree is developed, CoT can create the DTD (See Fig. 4), and the schema for the XML view.

```
<!ELEMENT TABLE_E (TABLE_D)>
<!ATTLIST TABLE_E ELEMENT1 ID #REQUIRED ELEMENT2 CDATA #IMPLIED ELEMENT3 CDATA #IMPLIED>

<!ELEMENT TABLE_A (TABLE_B)>
<!ATTLIST TABLE_A ELEMENT1 ID #REQUIRED ELEMENT2 CDATA #IMPLIED>

<!ELEMENT TABLE_B (TABLE_C)>
<!ATTLIST TABLE_B ELEMENT1 ID #REQUIRED ELEMENT2 CDATA #IMPLIED>

<!ELEMENT TABLE_C (TABLE_D)>
<!ATTLIST TABLE_C ELEMENT1 ID #REQUIRED ELEMENT2 CDATA #IMPLIED ELEMENT3 CDATA #IMPLIED>

<!ELEMENT TABLE_D (EMPTY)>
<!ATTLIST TABLE_D ELEMENT1 CDATA #IMPLIED ELEMENT2 CDATA #IMPLIED REF_TABLE_E IDREF
#IMPLIED ELEMENT3 CDATA #IMPLIED ELEMENT4 CDATA #IMPLIED >
```

Fig. 4: Schema created from CoT

In order to explain the functionality of CoT with a more concrete example, refer back to Section 1.2 (titled *The TPC-H Benchmark*) and use the CoT algorithm to create a tree-structure and develop the DTD.¹

1.4 Possible Solutions to Updating XML

Now that an algorithm such as CoT has been developed to make the creation of schemas quick, efficient, and easy, the problem then becomes one of increased functionality. The CoT algorithm as it stands will ONLY create schemas; it does not do any manipulation of the data stored within the XML view, or propagate those changes back to the relational databases. So, in order to have a more fully functional and useable program, some updates and new functions, such as insert(), delete(), and update(), must be added. However, there is no *one* clear way to do this. There are *many* algorithms that suggest a solution to this problem.

¹ The relational database tables and DTD for the TPC-H Benchmark can be found at <http://www.tpc.org/tpch/>

1.4.1 The Tatarinov, Ives, Halevy, Weld (TIHW) Algorithm [1]

In [1] Tatarinov, Ives, Halevy, and Weld present a language expansion to XQuery for specifying updates over XML documents. They then provide an algorithm for processing query updates. However, for our own work, it is unnecessary to go into their extension of XQuery. For an example of the basis of the algorithm derived by [1], refer to Figure 5.

```

<db lab="lalab">
  <university ID="ucla">
    <lab ID="lalab" managers="smith1 jones1">
      <name>UCLA Bio Lab</name>
      <city>Los Angeles</city>
    </lab>
  </university>
  <lab ID="baselab" managers="smith1">
    <name>Seattle Bio Lab</name>
    <location>
      <city>Seattle</city>
      <country>USA</country>
    </location>
  </lab>
  <lab ID="lab2">
    <name>PMBL</name>
    <city>Philadelphia</city>
    <country>USA</country>
  </lab>
  <paper ID="Smith991231" source="lab2"
    category="spectral" biologist="smith1">
    <title>Autocatalysis of Spectral...</title>
  </paper>
  <biologist ID="smith1">
    <lastname>Smith</lastname>
  </biologist>
  <biologist ID="jones1" age="32">
    <lastname>Jones</lastname>
  </biologist>
</db>

```

Figure 1: Sample XML document representing biology labs and publications

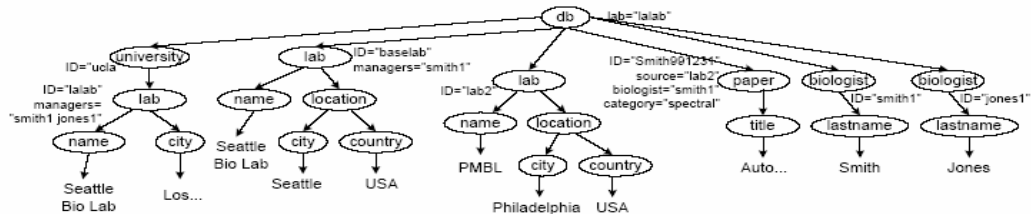


Figure 2: Data model representation for Figure 1.

Fig. 5: Example of a Tree Structure

[1] presents a robust list of operations for updating XML, but for the purposes of this paper, assume that the content does not need to be ordered and only insert, delete, and update need to be implemented.

The TIHW algorithm, using the data model described in [1] implements the following functions:

- The delete() function takes a child as an argument and then does a search of the tree checking for the object it is trying to delete. If the child is PCDATA (string)

or an attribute, then the child is just deleted, but if the child is an IDREF, just that IDREF is deleted from the list of IDs.

- The insert() function takes some content in as an argument. The content can be any of the components of XML, i.e. PCDATA, attributes, IDREFs, or elements. Simply the insert() function finds where the content is desired to be stored and inserts that content. “An attempt to insert an attribute with the same name as an existing attribute fails.” [1] However, inserting an IDREF with the same name as an existing IDREF, only adds the new IDREF into the list.
- The update() function is considered to be multiple invocations of the TIHW algorithm’s sub-update() function. The sub-update() function takes in some predicates, an update operation, and pattern-matching operation. It begins at the desired node, “invokes a new pattern-matching operation over the input, returning bindings that are filtered by the predicates. For each valid combination of bindings, recursively invokes the update operation.” [1]

1.4.2 The Braganholo, Davidson, Heuser (BDH) Algorithm [3]

The practice of representing databases through XML documents is becoming more common as XML views lend themselves to internet applications much more efficiently than an entire relational database. The practice of querying relational views has been examined for many years now, but updating XML views is an area that there is much research still to be done. An area that might be helpful to start looking at when trying to solve an XML view update problem is the collection of different methods of

updating relational views. Still, these two cases may be like apples and oranges, and the research that applies to one may completely not apply to the other.

In [2] XML views are represented as query trees. These trees are then mapped to a set of relational views.

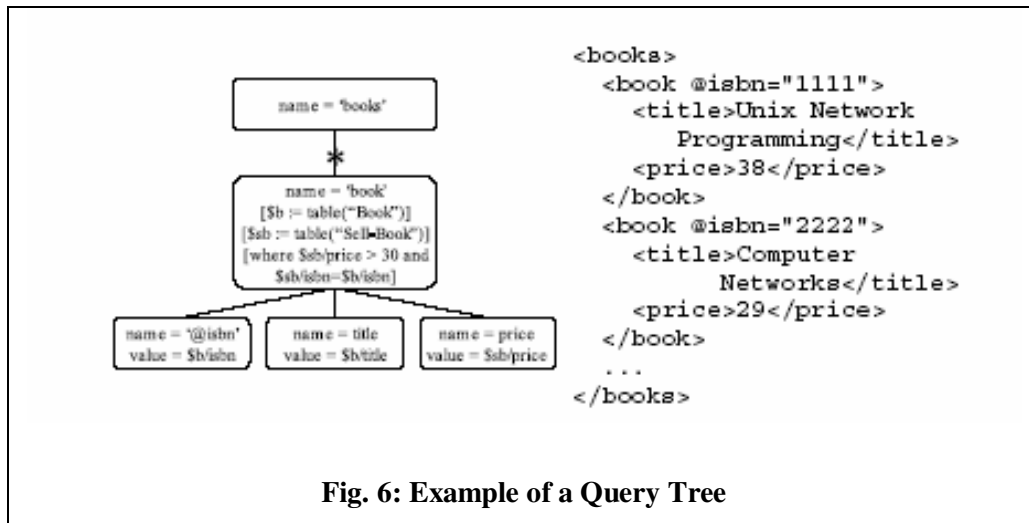


Fig. 6: Example of a Query Tree

Above is an example of a query tree and its resulting XML view. To update this, the authors first define relational views *corresponding* to this XML view. An update over the XML view is then mapped to update over the relational view. If the process can be taken that far, it is then possible to use existing relational update techniques to finish the process.

2. The Algorithm:

2.1 Introduction:

From a study of previous work done by [1] and [3] we have been able to derive our own algorithm and methods for implementation of this expansion. In order to more easily solve, and organize, this problem we have split it up into two main parts. The first

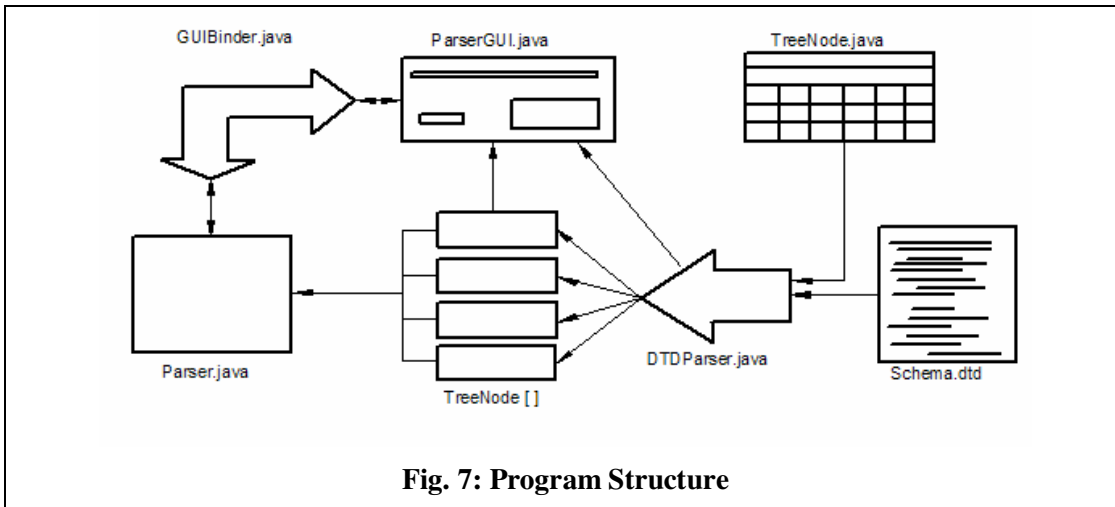
part is more in the forefront of our program and can be considered the main view, which deals with one of three types of information – an XML update statement, an SQL update statement, or an SQL query statement – all provided by the user. The second part is more of a background process and deals with the parsing and storing of the information present in the DTD (for this project we shall use the TPC-H Benchmark and the DTD that is created from that).

2.2 The Main Program:

2.2.1 Overview of XPath Parser Program

The XPath Parser program is a collection of java classes that take an object orientated approach to parsing XPath statements. The requirements of the program are that it be able to take an XPath statement, translate it to SQL, and pass it to a database which then makes the desired changes. In this instance it connects to the Oracle database at WPI. At this time this is automatic, the user cannot specify another connection. The connection is made in Parser.java, the class also responsible for parsing the XPath statements into SQL.

The difficulty in this process lies in the translation from XPath to SQL, specifically retrieving the names of the foreign keys of the tables that are being inserted into. To do this the program reads a DTD file, and passes information from that file to the parser, which then uses it to fill in information needed to form the SQL statements. The basic structure of the program is outlined in the diagram below.



The DTDParser creates an array of `TreeNode` objects, each of which contains information about one of the tables described in the DTD file. There is also a method inside `DTDParser` that returns this array, so it can be used outside of the class by calling the methods of `TreeNode` directly. The `Parser` file contains the code that translates Xpath to SQL, which it does by using the methods of `DTDParser` to retrieve the needed information about the DTD. It also contains the code that connects to the database.

2.2.2 Overview of Parser Graphical User Interface

The `ParserGUI` uses java swing components for Graphical User Interface (GUI), and creates a `GUIBinder` object which contains methods that link the buttons and menus of the GUI with the `Parser` file. The GUI also creates another `DTDParser` object and `TreeNode` array, because it uses the information returned by some of the methods to initialize some of its menus and tables. The GUI for this program is simple in functionality. It provides the user with a field to type an XPath statement, SQL Update Statement, or SQL Query, and three buttons to execute each of these options (SQL code is supported for testing purposes).

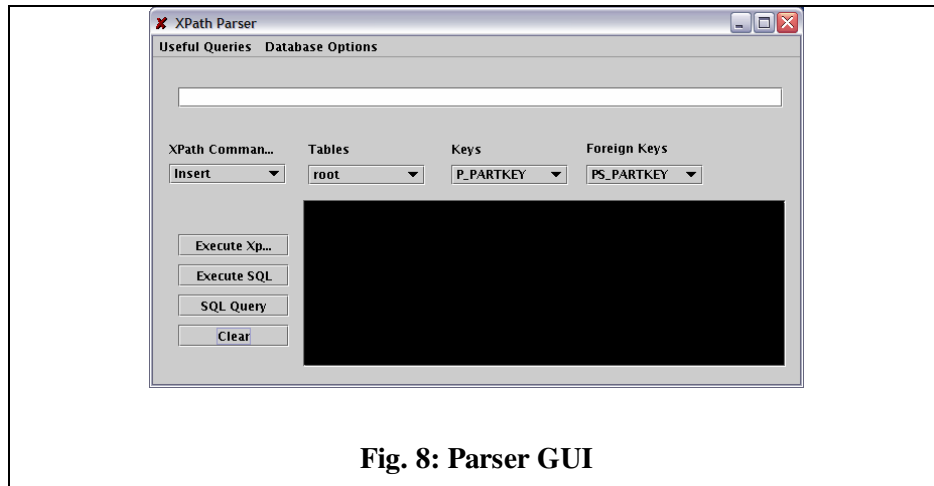


Fig. 8: Parser GUI

There are drop down menus that contain the names of the tables, keys and foreign keys of the schema outlined in the DTD file, and when the user selects an option from one of these menus, it is put into the box. The purpose of this is so that a user who is not familiar with our syntax might be able to form statements quicker.

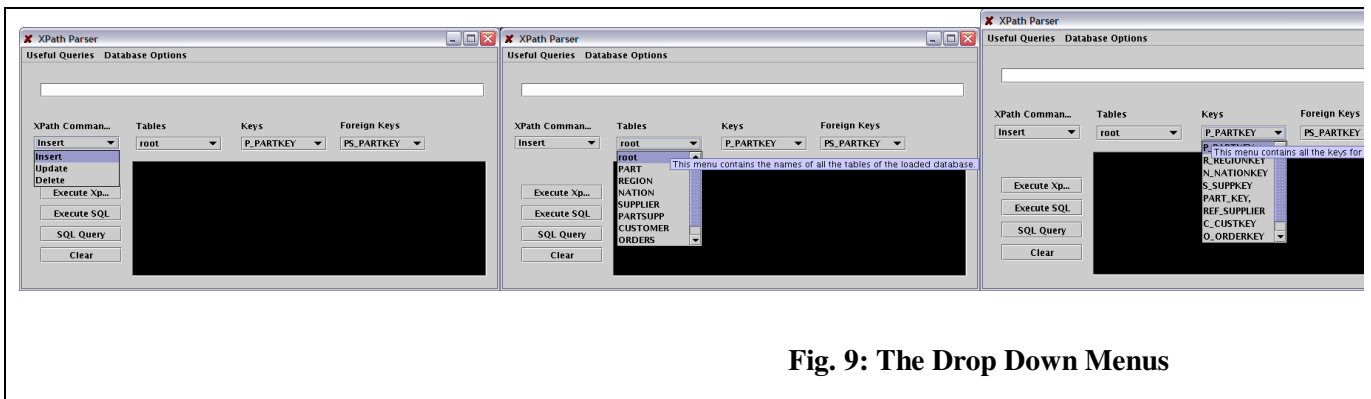


Fig. 9: The Drop Down Menus

There is also a menubar that contains a menu with some options for working with the database, as well as a menu that contains some queries that return useful information about the database (specifically, the integer keys, because if the user wants to insert an item into the table, the integer key must be unique). At this point, the menu of queries is the only component of the program that is hard coded (meaning information about the TPC-H schema not loaded from the DTD file). The database options menu contains two

choices. Rollback reverts the database to the original way it was when the program was started, and commit makes all changes permanent.

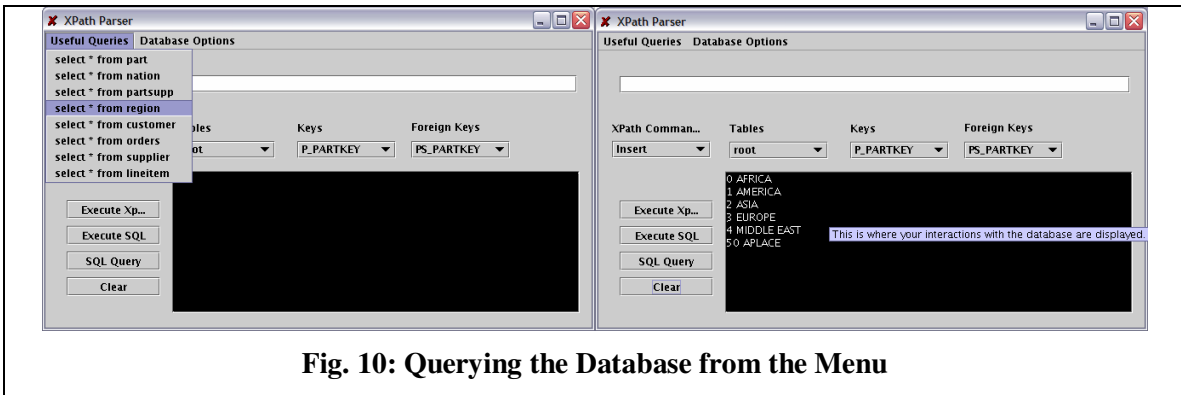


Fig. 10: Querying the Database from the Menu

All the actions of the GUI call methods of GUIBinder, which is created as an object in ParserGUI. These methods handle strings created by the user, and return strings such as query results or error messages which are printed in the text area of the GUI. For Xpath statements, the corresponding SQL statement which was passed to the database is printed if there is success and an error otherwise.

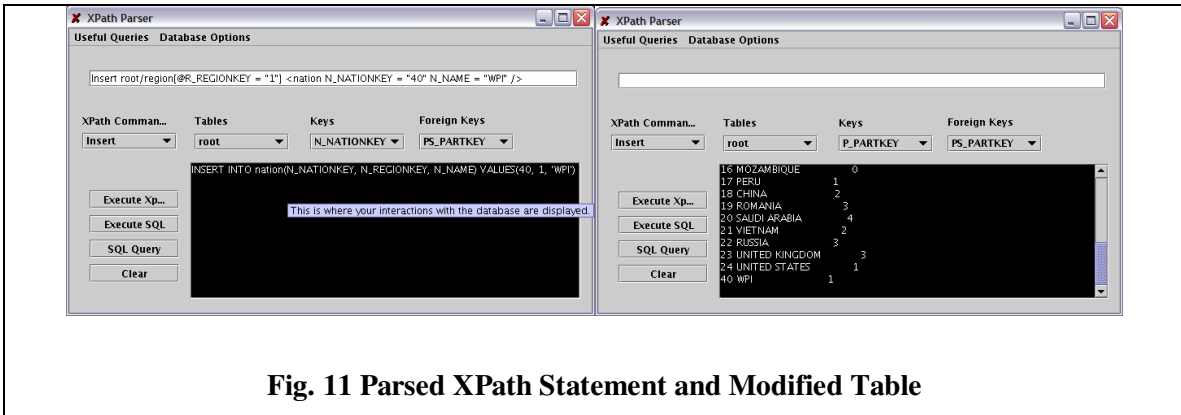


Fig. 11 Parsed XPath Statement and Modified Table

At this point another aspect of the program is being developed that will display a visual representation of the database structure by running the CoT Algorithm on it and using an XML display tool to show the result. This would either be displayed by a pop up

window or by a tabbed pane interface, where they user would be able to click a button (or tab over), and get an updated representation each time.

2.2.3 Overview of Main Parser Algorithm

If an XML update statement is chosen, the algorithm works with this statement and determines based the statement, what should be done to update the SQL database as well. The XML update statement itself consists of three parts: the desired command (insert, delete, or update), the XPath statement giving the location of the desired node in the XML view, and the information the user wants to insert – the information can either be a single piece of data or a whole tree. For deletion, the third part of the XML update statement will be empty because only the command and the location of the item to be deleted is necessary. Once the XML update statement is entered, error checking is done to ensure that the first item in the XPath statement is root. This is done to prevent any illegal insertions, deletions, or updates, and to verify that the XPath statement actually points to a valid node.

If an SQL update or SQL query is chosen, the algorithm stores this statement in a string and passes it to the relational database. Since both SQL update and query statements are in the form of SQL syntax, instead of the XPath syntax, the main algorithm has very little to do to the effect of parsing for information retrieval or error checking. The algorithm assumes that all necessary information is inserted in the statement and all error checking will be done by the relational database itself.

2.2.4 Propagating Changes of Objects from XML View

When updating objects in an XML view, the type of object you will be altering is important. Different operations and functions are called depending on whether the object

is a single element or a whole tree. The same basic algorithm can be applied to both leaf elements and trees. However different parsing is required, and the tree algorithm usually can build off of the single element algorithm.

When dealing with single elements, an insert statement and delete statement would seem rather similar but in fact are quite different based on the way the update of information is propagated through the database. Thus, a different algorithm for the insertion and deletion of objects is needed. This may be true, but the algorithm for insertion and updating *is* quite similar. The basic data stored from the XPath statement is the same for both (this will be described in more detail in the following sections).

When dealing with a whole tree, the difference between the commands is very important. Updating can only be done on single elements at a time, so the updating of a whole tree is not allowed in this algorithm. However, insertion and deletion of trees is allowed and become exact opposites, with one telescoping down the tree and the other telescoping up the tree.

2.2.5 Insertion of a Single Element

When inserting a single element, the main parser stores specific information from the XML update statement to use in building the SQL insert statement. The table in which the new information will be inserted is stored – this is known as the `primaryTable`. The primary key for this table is stored as well – this is known as the `primaryVName`. To figure out how to get from the table where the data is to be inserted to the parent of this table, the name of the foreign key must be stored – this is known as `forKey`. Another value name that needs to be stored is the names of the columns in the relational database where the actual data will be inserted – this is known as the `primaryIdent`. Once all the

variable names are known, the algorithm stores the values for each variable. The value names for the variables `primaryVName`, `forKey`, and `primaryIdent` are: `insertVal`, `foreignVal`, and `primaryIdentVal` respectively.

While most of the values for the desired variables can easily be found from the XML update statement, the name of the foreign key relation between the `primaryTable` and the “foreign table” cannot be discovered directly through this method. Instead we must look at the DTD. From the DTD we can determine whether the last table and the child table are connected by any keys at all, and if they are, what the names of those keys are. Fig. 8 shows an example of a XML insert statement and the names for the aforementioned values.

```
Enter XPath Expression:
Insert root/region[@R_REGIONKEY="1"] <nation N_NATIONKEY="60" N_NAME="XMLWorld"/>

ERROR CHECKING.....
FIRST ARGUMENT IN XPATH STATEMENT IS 'ROOT' → CORRECT

INSERT INTO nation(N_NATIONKEY, N_REGIONKEY, N_NAME) VALUES(60, 1, 'XMLWorld')

Update returned: 1

primaryTable = nation
primaryVName = N_NATIONKEY
forKey = N_REGIONKEY
primaryIdent = N_NAME
insertVal = 60
foreignVal = 1
primaryIdentVal = XMLWorld
```

Fig. 12: Example of XML Insert Statement

2.2.6 Updating a Single Element

If the user desires to update a single element, the algorithm is quite similar to the insertion algorithm, except that it does not actually perform any insertion. The format of the XPath statement stays exactly the same and the information parsed out of the

statement is quite similar as well. The only difference between the updating of an element and the insertion of one is how the SQL update statement is built. The algorithm organizes the SQL statement to fit with an SQL update command and does not need to figure out any foreign key relationships, since those are not part of an SQL update command. Fig. 9 shows an example of an XPath update statement.

```
Enter XPath Expression:
Update root/region/nation[@N_NATIONKEY="20"] <N_NAME=" XMLWorld"/>

ERROR CHECKING.....
FIRST ARGUMENT IN XPATH STATEMENT IS 'ROOT' --> CORRECT

UPDATE nation SET N_NAME = ' XMLWorld' WHERE N_NATIONKEY = 20

Update returned: 1

primaryTable = nation
primaryIdent = N_NAME
primaryIdentVal = XMLWorld
primaryVName = N_NATIONKEY
insertVal = 20
```

Fig. 13: Example of XML Update Statement

2.2.7 Insertion/Deletion of a Whole Tree

When dealing with the insertion of single elements, the algorithm only has to parse a simple path to a node and parse information out of only one element. However, this apparently simple algorithm becomes quite different when dealing with trees. Instead the algorithm must keep track of what node it needs to go to next, and remember what nodes it visited in case it needs to travel back up the tree structure. From there it creates single element XPath insert statements, and repeatedly calls the function that inserts a single element.

```
Enter XPath Expression:
InsertTree root/region[@R_REGIONKEY="1"] <nation N_NATIONKEY="50"
N_NAME="XMLWorld"> <customer C_CUSTKEY="000230" C_NAME="Bob"/> </nation>
```

```

ERROR CHECKING.....
FIRST ARGUMENT IN XPATH STATEMENT IS 'ROOT' --> CORRECT

Insert root/region[@R_REGIONKEY="1"] <nation N_NATIONKEY="50" N_NAME="XMLWorld"/>

Insert root/region/nation[@N_NATIONKEY="50"] <customer C_CUSTKEY="000230"
C_NAME="Bob"/>

INSERT INTO nation (N_NATIONKEY, N_REGIONKEY, N_NAME) values (50 , 1, 'XMLWorld')

Update returned: 1

primaryTable = nation
primaryVName = N_NATIONKEY
forKey = N_REGIONKEY
primaryIdent = N_NAME
insertVal = 50
foreignVal = 1
primaryIdentVal = XMLWorld

INSERT INTO customer (C_CUSTKEY, C_NATIONKEY, C_NAME) values (000230, 50, 'Bob')

Update returned: 1

primaryTable = customer
primaryVName = C_CUSTKEY
forKey = C_NATIONKEY
primaryIdent = C_NAME
insertVal = 000230
foreignVal = 50
primaryIdentVal = Bob

```

Fig. 14: Example of XML Tree Insert

2.3 The DTD Parser:

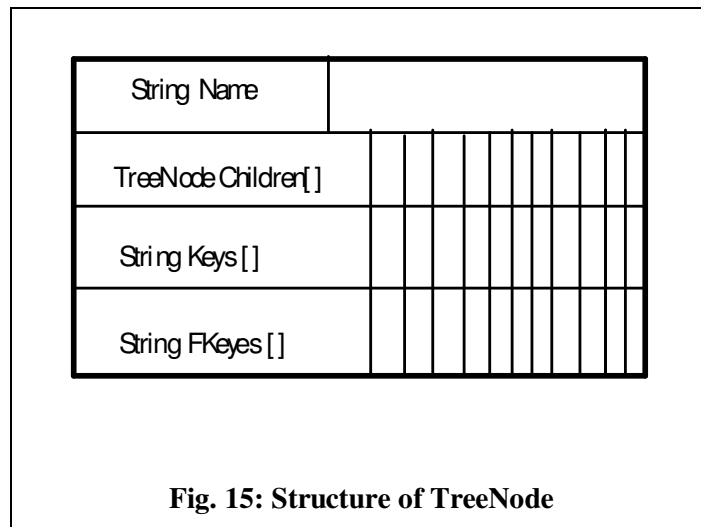
2.3.1 Overview

An important part of translating XPath insert statements into SQL is making sure they are complete and consistent with the DTD of the database. To do this, we create a JAVA data structure that can store important information about the tables, and give it methods that could be used to check the statement against the DTD. Decisions had to be made about what information was important, as well as how it would be represented.

A difficult part of this is loading information about the tables into the data structure. The information is stored in a DTD file, and that file follows a certain format. To load the information, we created a parser that would traverse through the file and take the relevant information. This information is then stored in the data structure we created.

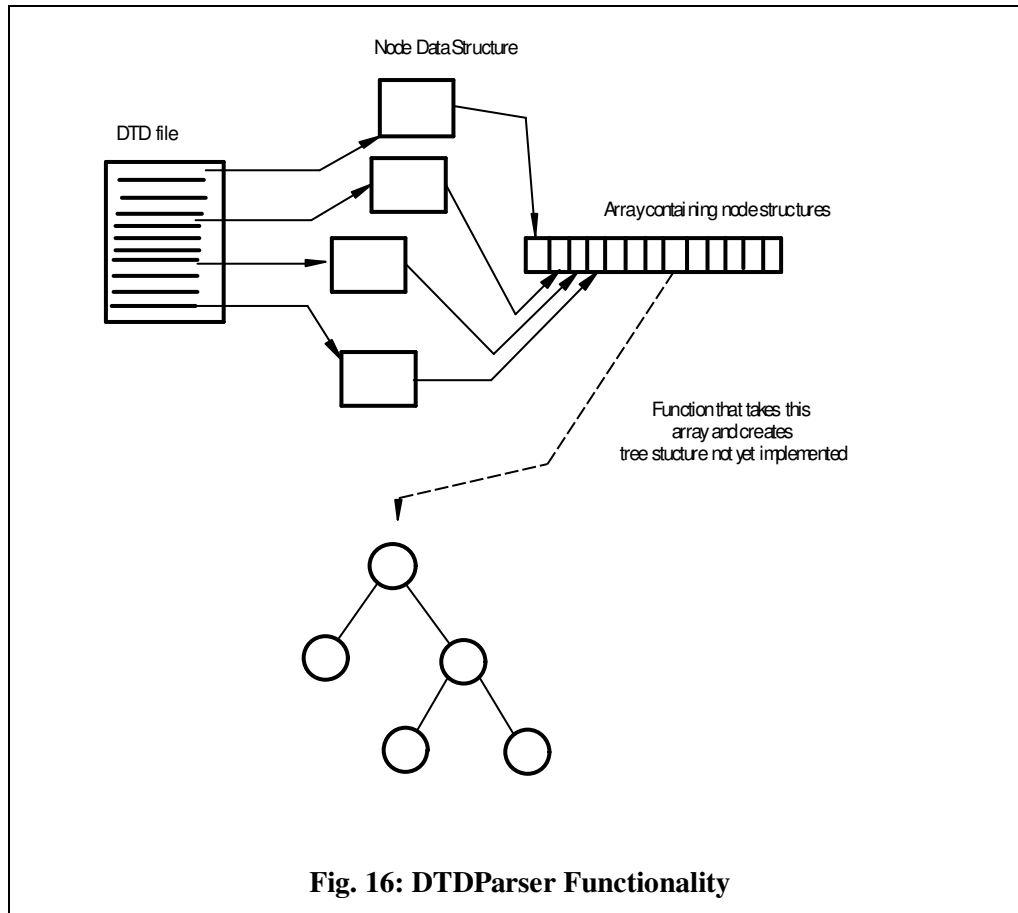
2.3.2 *The File Structure*

The file structure at this point (for just the DTD parser) is just two files, one to define a node data structure, which is then used to define the other – the parsed tree structure. The node structure contains the name, children, keys, and foreign keys. The parsed tree structure contains an array of these nodes and defines the data in the node structure by parsing the DTD file.



The data structure holds the name as a string, an array of other nodes that are its children, and arrays of strings for its keys and foreign keys. The reason that the children are stored as other nodes instead of just the names is that we initially hoped to create a function that will use the array to create a tree structure, but at this point we are not yet sure if that is completely necessary, or if all the information needed can be stored in just

the array. If that is the case, the data structure definition could be changed to store the children as simple strings for the names rather than entire other nodes.



2.3.3 The Methods

Once the information about the tables' children, keys, and foreign keys is stored in the data structure, certain methods are used to retrieve information about the tables. Usually the name will be given, and the program can check if another name is one of its children or keys. Other methods decide if one node is a child of another, and there are a few other methods that provide useful information to any program that is working with a database that uses that DTD structure.

The method that seems to be the most important for the purpose of parsing an XPath is the function that returns the names of the foreign keys that are needed to move from one table to another. The reason that this method is so important is that these attribute names are not given in the XPath statement, and they are needed to form a valid SQL statement. This method takes two table names, one being the primary table and the other being the foreign table, and returns the foreign key between the two.

The way our program uses the method that retrieves foreign key information is to make sure the XPath input statements passed into it are valid for the given database. It makes certain that the transitions from table to table are defined by using the information about keys, and also makes sure that the path is correct by using the information about the children. Other possible errors that could occur involve invalid data types, like putting a number where a word belongs. We plan on handling these errors later on by using the error returned by the SQL server.

3. Conclusions:

With the ever-growing world of information storage and retrieval, XML has become and continues to be a widely used source of representing views of relational databases. While the algorithms to translate a relational database to an XML view have been around for some time and have grown quite common, there is always room for improvement. Algorithms have been made to expand the functionality of this relationship. For example, the translation from relational database to XML view has been made faster and more efficient with the implementation of the CoT algorithm.

For this project we sought to take a different approach and expand the functionality of the relationship from XML view to relational database instead of the other way around. We created algorithms to parse and store information from a given XPath statement and a given DTD. Using these algorithms we were able to build SQL update statements of the form – insert, delete, or update from XPath update statements. These statements were then used to propagate the changes specified on the XML view back down to the relational database. This allows for a user who knows nothing about SQL queries or update statements to be able to make changes to a relational database, solely through an interaction with the XML view onto that database.

4. Appendix A

```
//Rich Omar and Rich Tamalavitch
//This program takes the DTD Structure, parses it, and stores it into a tree structure
//

import java.io.*;
import java.util.*;

class DTDParse
{
    private TreeNode SepNodes[] = new TreeNode[30];
    private int numNodes;

    public DTDParse ()
    {
        try
        {

            char[] x = parseStatement();

            int i = x.length;
            String statement= "";
            int j = 0;
            while (i > j)
            {

                if (x[j] == '/')
                {
```

```

        }
        else
        {
            if (x[j] == '[')
            {
                statement = statement + " [ ";
            }
            else
            {
                if (x[j] == '"')
                {
                    statement = statement + " * ";
                }
            }
            else
            {
                if (x[j] == ')')
                {
                    statement = statement + " ] ";
                }
            }
            else
            {
                if (x[j] == '@')
                {
                    statement = statement + " @ ";
                }
            }
            else
            {
                if (x[j] == '=')
                {
                    statement = statement + " = ";
                }
            }
            else
            {
                if (x[j] == '<')
                {
                    statement = statement + " < ";
                }
            }
            else
            {
                if (x[j] == '>')
                {
                    statement = statement + " > ";
                }
            }
            else
            {
                if (x[j] == '#')
                {
                    statement = statement + " # ";
                }
            }
            else
            {
                if (x[j] == '!')
                {
                    statement = statement + " !";
                }
            }
            else
            {
                statement = statement + x[j];
            }
        }
        j++;
    }

    String[] ss = statementStore(statement);

    Handle(ss);

    //display();

}

catch(IOException e){

}

}

public char[] parseStatement () throws IOException
{
    File inputFile = new File("TPHDTD.dtd");
    FileReader in = new FileReader(inputFile);
    char x[] = new char[10000];
    while (in.read(x) != -1){}
    in.close();

    return x;
}

public String[] statementStore(String statement)
{
    StringTokenizer z = new StringTokenizer(statement);
    String[] stored = new String[z.countTokens()];
    int i = 0;
    while (z.hasMoreTokens())
    {
        stored[i] = z.nextToken();
        i++;
    }
    return stored;
}
}

```

```

public void Handle(String[] dtd)
{
    String name = new String();
    String children[] = new String[10];
    String keys[] = new String[10];
    String fkeys[] = new String[10];
    int i = 0;
    int j = 0;
    int x = 0;
    int c = 0;
    int k = 0;
    int f = 0;
    //TreeNode Nodes[] = new TreeNode[30];
    while(i < dtd.length)
    {
        if (dtd[i].equals("!ELEMENT"))
        {
            i++;
            name = dtd[i];
            i++;
            j=0;
            //System.out.println("about to while loop");
            while(!(dtd[i].equals(">")))
            {
                children[j] = cleanUp(dtd[i]);
                j++;
                i++;
            }

            k = 0;
            while(!(dtd[i].equals("!key")) && !(dtd[i].equals("!END")))
            {
                //System.out.println("I think that !Key != " + dtd[i]);
                i++; }

            if (dtd[i].equals("!key"))
            {
                while(!(dtd[i].equals(">")))
                {
                    if(dtd[i].equals("@"))
                    {
                        // System.out.println("setting keys[" + k + "] to
" + dtd[i + 1]);

                        i++;
                        keys[k] = dtd[i];
                        k++;
                    }
                    else
                        i++;
                }
            }
            f = 0;
            while(!(dtd[i].equals("!foreignKey")) && !(dtd[i].equals("!END")))
            {
                //System.out.println("I think that !foreignKey != " + dtd[i]);
                i++;
            }

            if (dtd[i].equals("!foreignKey"))
            {
                while(!(dtd[i].equals("!END")))
                {
                    if(dtd[i].equals("@"))
                    {
                        //System.out.println("setting fkeys[" + f + "] to
" + dtd[i + 1]);

                        i++;
                        if(!(dtd[i-4].equals("key")))
                        {
                            fkeys[f] = dtd[i];
                            f++;
                        }
                    }
                    else
                        i++;
                }
            }

            i++;

            //System.out.println("name: " + name);
            SepNodes[c] = new TreeNode(name);
            int z = 0;
            for (z = 0; z < j; z++)
            {
                SepNodes[c].makeChild(new TreeNode(children[z]));
                //System.out.println(" children: " +children[z]);
            }
        }
    }
}

```

```

        for (z = 0; z < k; z++)
        {
            SepNodes[c].makeKey(keys[z]);
            //System.out.println("    keys: " + keys[z]);
        }
        for (z = 0; z < f; z++)
        {
            SepNodes[c].makeFKey(fkeys[z]);
            //System.out.println("    foreign keys: " + fkeys[z]);
        }
        c++;
    }

    i++;
}
numNodes = c;
}

public String cleanUp (String word)
{
    char brokword[] = word.toCharArray();
    String cleanword = new String();
    for(int i = 0; i < brokword.length; i++)
    {
        if ( (brokword[i] == '*') || (brokword[i] == '(') || (brokword[i] == ',') || (brokword[i]
== ')') )
        {
        }
        else
        {
            cleanword = cleanword + brokword[i];
        }
    }
    return cleanword;
}

public boolean isChild (String parent, String child)
{
    for(int i = 0; i < numNodes; i++)
    {
        if (SepNodes[i].getName().equals(parent.toUpperCase()))
        {
            //System.out.println("Checking if " + child + " is a child of " +
SepNodes[i].getName());
            return SepNodes[i].isChild(child.toUpperCase());
        }
    }
    return false;
}

public boolean hasKey (String name, String key)
{
    for(int i = 0; i < numNodes; i++)
    {
        if (SepNodes[i].getName().equals(name))
        {
            return SepNodes[i].isKey(key);
        }
    }
    return false;
}

public boolean hasFKey (String name, String fkey)
{
    for(int i = 0; i < numNodes; i++)
    {
        if (SepNodes[i].getName().equals(name))
        {
            return SepNodes[i].isFKey(fkey);
        }
    }
    return false;
}

public String getFKey (String Pname, String Fname)
{
    for(int i = 0; i < numNodes; i++)
    {
        //System.out.println("is " + SepNodes[i].getName() + " equal to " + Fname + "?");

        if ((SepNodes[i].getName()).equals(Fname.toUpperCase()))
        {
            System.out.println("Yes");
            String x[] = SepNodes[i].getFKeys();
            for (int j = 0; j < SepNodes[i].getNumFKeys(); j++)
            {
                if (x[j].charAt(0) == (Pname.toUpperCase()).charAt(0))
                {
                    return x[j];
                }
            }
        }
    }
}

```

```

        }
        }
        else {System.out.println("NO");}
    }
    return "error";
}

public String[] getKey (String name)
{
    for(int i = 0; i < numNodes; i++)
    {
        if ((SepNodes[i].getName()).equals(name))
        {
            return SepNodes[i].getKeys();
        }
    }
    return new String[1];
}

public void display()
{
    int z = 0;
    String keys[] = new String[2];
    String fkeys[] = new String[2];
    for(int i = 0; i < numNodes; i++)
    {
        System.out.println("name: " + SepNodes[i].getName());
        keys = SepNodes[i].getKeys();
        fkeys = SepNodes[i].getFKeys();
        z = 0;

        while(z < SepNodes[i].getNumKeys())
        {
            System.out.println("    keys: " + keys[z]);
            z++;
        }
        z = 0;
        while(z < SepNodes[i].getNumFKeys())
        {
            System.out.println("    foreign keys: " + fkeys[z]);
            z++;
        }
    }
}

public TreeNode[] getAll()
{
    return SepNodes;
}

public int getNumNodes()
{
    return numNodes;
}
}

```

```

class TreeNode {

    private String name;
    private TreeNode Children[] = new TreeNode[100];
    private int numChildren = 0;
    private String Keys[] = new String[2];
    int numKeys = 0;
    private String FKeys[] = new String[2];
    int numFKeys = 0;

    public TreeNode (String iname)
    {
        name = iname;
    }

    public TreeNode (String iname, TreeNode iTree)
    {
        name = iname;
        Children = new TreeNode[1];
        Children[0] = iTree;
        numChildren = 1;
    }

    public TreeNode (String iname, TreeNode[] iTree)
    {
        name = iname;
    }
}

```

```

Children = new TreeNode[iTree.length];
for(int i = 0; i < iTree.length; i++)
{
    Children[i] = iTree[i];
}
numChildren = iTree.length;
}

public String getName()
{
    return name;
}

public TreeNode[] getChildren()
{
    return Children;
}

public int getNumChildren()
{
    return numChildren;
}

public int getNumKeys()
{
    return numKeys;
}

public int getNumFKeys()
{
    return numFKeys;
}

public boolean isChild(String Child)
{
    for(int i = 0; i < numChildren; i++)
    {
        //System.out.println("trying " + i + " until " + Children.length);
        if (Children[i].getName().equals(Child))
            return true;
    }
    return false;
}

public void makeChild(TreeNode Child)
{
    Children[numChildren] = Child;
    numChildren++;
}

public void makeKey(String Key)
{
    Keys[numKeys] = Key;
    numKeys++;
}

public void makeFKey(String FKey)
{
    FKeys[numFKeys] = FKey;
    numFKeys++;
}

public boolean isKey(String Key)
{
    for(int i = 0; i < Keys.length; i++)
    {
        if (Keys[i].equals(Key))
            return true;
    }
    return false;
}

public boolean isFKey(String FKey)
{
    for(int i = 0; i < FKeys.length; i++)
    {
        if (FKeys[i].equals(FKey))
            return true;
    }
    return false;
}

public String[] getFKeys ()
{
    return FKeys;
}

```

```

public String[] getKeys()
{
    return Keys;
}

public String getFKKey (int i)
{
    return FKKeys[i];
}

public String getKey(int i)
{
    return Keys[i];
}
}

//Rich Omar and Rich Tamalavitch
//This program takes an XPath Statement and parses it into
//seperate blocks of an array

import java.io.*;
import java.util.*;
import oracle.jdbc.driver.*;
import java.sql.*;

class ParserV6 {

    private static Connection connection = dataConnect();
    private static DTDParser checker = new DTDParser();
    private static Savepoint z;
    public static void main (String[] args) throws IOException, SQLException
    {
        connection.setAutoCommit(false);
        Savepoint z = connection.setSavepoint();
    }

    public ParserV6()
    {
        try{
            connection.setAutoCommit(false);
            Savepoint z = connection.setSavepoint();
            System.out.println("Database Savepoint Created.");
        }
        catch (SQLException e)
        {
            System.out.println(e);
        }
    }

    public static char[] parseStatement () throws IOException
    {
        InputStreamReader isr = new InputStreamReader (System.in);
        BufferedReader br = new BufferedReader (isr);
        String line = br.readLine();

        char[] x= line.toCharArray();

        return x;
    }

    public static String[] statementStore(String statement)
    {
        StringTokenizer z = new StringTokenizer(statement);
        String[] stored = new String[z.countTokens()];
        int i = 0;
        while (z.hasMoreTokens())
        {
            stored[i] = z.nextToken();
            i++;
        }
        return stored;
    }

    public static void errorChecker(String[] ss)
    {
        System.out.println("\nERROR CHECKING....");

        if (ss[1].equals("root"))
        {
            System.out.println("FIRST ARGUMENT IN XPATH STATEMENT IS 'ROOT' --> CORRECT");
        }
        else
    }
}

```

```

        {
            System.out.println("INCORRECT, FIRST ARGUMENT IN XPATH STATEMENT MUST BE 'ROOT'");
            System.exit(0);
        }
    }

    public static String sqlAdd(String[] ss)
    {
        int u = 1;
        int counter = 0;

        String insertState = "";
        String insertVal = "";
        String insertInto = "";
        String primaryTable = "";
        String primaryVName = "";
        String primaryIdent = "";
        String primaryIdentVal = "";
        String parentVName = "";
        String parentTable = "";
        String foreignTable = "";
        String foreignVal = "";

        boolean insertTrue = false;
        boolean isFirstAtt = true;

        for(int p=0; p < ss.length; p++)
        {
            if(ss[p].equals("root") && insertTrue == false && ss[p+1].equals("<"))
            {
                parentTable = ss[p];          //This is root when an item is being inserted
                                              //at the highest
level.
            }
            if(ss[p].equals("@") && insertTrue == false && isFirstAtt == true)
            {
                parentVName = ss[p+1];
                parentTable = ss[p-2];
                parentTable = parentTable + ss[p-1];
                parentTable = parentTable + ss[p];
                while(!ss[u+p].equals("<"))
                {
                    parentTable = parentTable + ss[p + u];
                    u++;
                }

                isFirstAtt = false;
            }
            if(ss[p].equals("=") && insertTrue == false)
            {
                insertInto = ss[p+2];
            }
            if(ss[p].equals("<") && parentTable.equals("root"))
            {
                primaryTable = ss[p+1];          //The table to insert into
                primaryVName = ss[p+2];          //The column name for the primary key
                primaryIdent = ss[p+7];          //The column for the X_NAME
                insertVal = ss[p+5];             //The value for the primary key of the
primaryTable
                primaryIdentVal = ss[p+10];      //The value for the X_NAME option of the
primaryTable
            }
            if(ss[p].equals("<") && !parentTable.equals("root"))
            {
                insertTrue = true;
                primaryTable = ss[p+1];          //The table to insert into
                primaryVName = ss[p+2];          //The column name for the primary key
                primaryIdent = ss[p+7];          //The column for the X_NAME
                foreignTable = ss[p-9];          //The table that the primaryTable has a
foreign key for
                foreignVal = ss[p-3];           //The value for the foreign key of
primaryTable
            }
            if(ss[p].equals("=") && insertTrue == true && counter==0)
            {
                insertVal = ss[p+2];             //The value for the primary key
                                              //of the
primaryTable
                counter++;
            }
            if(ss[p].equals("=") && insertTrue == true && counter!=0)
            {
                primaryIdentVal = ss[p+2];      //The value for the X_NAME option of
//the primaryTable
            }
        }

        String forKey = "";
        forKey = checker.getFKey(primaryTable, foreignTable);

        //System.out.println(forKey);

        if(parentTable.equals("root"))
        {

```

```

        //creates the SQL insert statement if the parent table is root
insertState = insertState + "INSERT INTO ";
insertState = insertState + primaryTable;
insertState = insertState + "(";
insertState = insertState + primaryVName + ", ";
insertState = insertState + primaryIdent;
insertState = insertState + ") VALUES(";
insertState = insertState + insertVal + ", ";
insertState = insertState + "'" + primaryIdentVal + "'\n";
    }
else if(checker.isChild(foreignTable, primaryTable) == true)
{
    //creates the SQL insert statement for the parent table foreignTable
insertState = insertState + "INSERT INTO ";
insertState = insertState + primaryTable;
insertState = insertState + "(";
insertState = insertState + primaryVName + ", ";
insertState = insertState + forKey + ", " + primaryIdent;
insertState = insertState + ") VALUES(";
insertState = insertState + insertVal + ", ";
insertState = insertState + foreignVal + ", ";
insertState = insertState + primaryIdentVal + "'\n";
}

    return insertState;
}

public static String[] sqlTreeAdd(String[] treeAdd)
{
    int counter = 0;
    String[] insertStates = new String[40];

    return insertStates;
}

public static Connection dataConnect()
{
    Connection connection = null;
    try
    {
        // Load the JDBC driver
        String driverName = "oracle.jdbc.driver.OracleDriver";
        Class.forName(driverName);
        // Create a connection to the database
        String serverName = "oracle.wpi.edu";
        String portNumber = "1521";
        String sid = "CS";
        String url = "jdbc:oracle:thin:@" + serverName + ":" + portNumber + ":" + sid;
        String username = "romar05";
        String password = "romar05";
        connection = DriverManager.getConnection(url, username, password);
        if(!connection.isClosed())
        {
            System.out.println("Connected to " + serverName + " @ " + portNumber + ":" +
sid + " under user name " + username + "\n");
        }
        else { System.out.println("Database connection failed");}
    }

    catch (ClassNotFoundException e)
    {
        System.out.println(e);
        System.exit(1);
    }
    catch (SQLException e)
    {
        System.out.println(e);
        System.exit(1);
    }

    return connection;
}

public static void executeSQL(String sqlCommand)
{
    try {

        Statement sql = connection.createStatement();
        int x = sql.executeUpdate(sqlCommand);
        System.out.println("Update returned: " + x );
    }

    catch (SQLException e) {System.out.println(e);}

}

public static void query(String q) throws SQLException
{
    Statement x = connection.createStatement();
    //String query = "Select * from region";
    ResultSet z = x.executeQuery(q);
    ResultSetMetaData zmd = z.getMetaData();
}

```

```

int i = 0;
while (z.next())
{
    for(i = 1; i < zmd.getColumnCount(); i++)
    {
        System.out.print(z.getString(i) + " ");
    }
    System.out.println("");
    //System.out.println("columncount = " + zmd.getColumnCount());
}

}

public static String sqlUp(String[] ss)
{
    int u = 1;
    int counter = 0;

    String UpdateState = "";
    String insertVal = "";
    String insertInto = "";
    String primaryTable = "";
    String primaryVName = "";
    String primaryIdent = "";
    String primaryIdentVal = "";
    String parentVName = "";
    String parentTable = "";
    String foreignTable = "";
    String foreignVal = "";

    boolean insertTrue = false;
    boolean isFirstAtt = true;

    for(int p=0; p < ss.length; p++)
    {
        if(ss[p].equals("root") && insertTrue == false && ss[p+1].equals("<"))
        {
            parentTable = ss[p]; //This is root when an item is being inserted
            //at the highest level.
        }
        if(ss[p].equals("@") && insertTrue == false && isFirstAtt == true)
        {
            parentVName = ss[p+1];
            parentTable = ss[p-2];
            parentTable = parentTable + ss[p-1];
            parentTable = parentTable + ss[p];
            while(!ss[u+p].equals("<"))
            {
                parentTable = parentTable + ss[p + u];
                u++;
            }

            isFirstAtt = false;
        }
        if(ss[p].equals("=") && insertTrue == false)
        {
            insertInto = ss[p+2];
        }
        if(ss[p].equals("<") && parentTable.equals("root"))
        {
            primaryTable = ss[p+1]; //The table to insert into
            primaryVName = ss[p+2]; //The column name for the primary key
            primaryIdent = ss[p+7]; //The column for the X_NAME
            insertVal = ss[p+5]; //The value for the primary key of the
            primaryTable
            primaryTable
            primaryIdentVal = ss[p+10]; //The value for the X_NAME option of the
        }
        if(ss[p].equals("<") && !parentTable.equals("root"))
        {
            insertTrue = true;
            primaryTable = ss[p+1]; //The table to insert into
            primaryVName = ss[p+2]; //The column name for the primary key
            primaryIdent = ss[p+7]; //The column for the X_NAME
            foreignTable = ss[p-9]; //The table that the primaryTable has a
            foreign key for
            primaryTable
            foreignVal = ss[p-3]; //The value for the foreign key of

        }
        if(ss[p].equals("=") && insertTrue == true && counter==0)
        {
            insertVal = ss[p+2]; //The value for the primary key
            //of the primaryTable
            counter++;
        }
        if(ss[p].equals("=") && insertTrue == true && counter!=0)
        {
            primaryIdentVal = ss[p+2]; //The value for the X_NAME option of
            //the primaryTable
        }
    }

    //System.out.println(forKey);
}

```

```

//creates the SQL insert statement if the parent table is root
UpdateState = UpdateState + "UPDATE ";
UpdateState = UpdateState + primaryTable;
UpdateState = UpdateState + " SET ";
UpdateState = UpdateState + primaryIdent + " = ";
UpdateState = UpdateState + primaryIdentVal + " WHERE ";
UpdateState = UpdateState + primaryVName + " = ";
UpdateState = UpdateState + insertVal + "\n";

        return UpdateState;
    }

public static String XPathSender(String line)
{
    char x[] = line.toCharArray();
    int i = x.length;

        String statement= "";
        String insertSQL = "";

        int j = 0;

        while (i > j)
        {
            if (x[j] == '/')
            {
                statement = statement + " / ";
            }
            else
                if (x[j] == '[')
                {
                    statement = statement + " [ ";
                }
            else
                if (x[j] == '"')
                {
                    statement = statement + " \" ";
                }
            else
                if (x[j] == ']')
                {
                    statement =
                }
            else
                if (x[j] == '@')
                {
                    statement =
                }
            else
                if (x[j] == '=')
                {
                    statement
                }
            else
                if (x[j] == '<')
                {
                    statement
                }
            else
                if (x[j] == '>')
                {
                    statement
                }
            else
                statement
        }
        j++;
    }

    String[] ss =

        if(ss[0].equals("Insert") || ss[0].equals("insert") || ss[0].equals("INSERT"))
        {
            errorChecker(ss);

            insertSQL
        }

        = sqlAdd(ss);
}

```

```

        executeSQL(insertSQL);
insertSQL;
    }
    if(ss[0].equals("Update") || ss[0].equals("update") || ss[0].equals("UPDATE"))
    {
        errorChecker(ss);
        = sqlUp(ss);
        executeSQL(insertSQL);
insertSQL;
    }
    else
    {
        errorChecker(ss);
        ("DELETE FUNCTION IS UNDER CONSTRUCTION.");
    }
    else
    {
        return
    }
    ("INCORRECT COMMAND: " + ss[0] + " IS NOT A VALID COMMAND");
}

public static String SQLSender(String line) throws SQLException
{
    String insertSQL = new String(line);
    executeSQL(insertSQL);
    return ("SQL Statement Executed.");
}

public static String QuerySender(String line) throws SQLException
{
    String insertSQL = new String(line);
    Statement x = connection.createStatement();
    //String query = "Select * from region";
    ResultSet z = x.executeQuery(insertSQL);
    ResultSetMetaData zmd = z.getMetaData();
    int i = 0;
    String retme = "";
    while (z.next())
    {
        for(i = 1; i < zmd.getColumnCount(); i++)
        {
            retme = retme + z.getString(i) + " ";
        }
        retme = retme + "\n";
    }

    return retme;
}

public void rollBack()
{
    try
    {
        connection.rollback();
    }
    catch (SQLException e)
    {
        System.out.println(e);
    }
}

public void commit()
{
    try
    {
        connection.commit();
    }
    catch (SQLException e)
    {
        System.out.println(e);
    }
}

public void close()

```

```

    try
    {
        connection.close();
    }
    catch (SQLException e)
    {
        System.out.println(e);
    }
}

}

/*
 * Created on Feb 21, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 * @author Rich Omar
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
import java.sql.*;

public class GUIBinder {

    ParserV6 x = new ParserV6();

    public static void main(String[] args) {

public String sendXpath(String Xstatement)
{
    return x.XPathSender(Xstatement);
}

public String sendSQL(String SQLstatement)
{
    try {
        return x.SQLSender(SQLstatement);
    }
    catch (SQLException e) {return "SQLException";}
}

public String sendQuery(String Qstatement)
{
    try {
        return x.QuerySender(Qstatement);
    }
    catch (SQLException e) {return "SQLException";}
}

public void rollBack()
{
    x.rollBack();
}

public void commit()
{
    x.commit();
}

public int close()
{
    x.close();
    return 1;
}
}

import javax.swing.*;

import javax.swing.JButton;
import javax.swing.JScrollBar;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.JScrollBar;
import javax.swing.JScrollPane;
/*
 * Created on Feb 20, 2005
 *
 * TODO To change the template for this generated file go to

```

```

* Window - Preferences - Java - Code Style - Code Templates
*/

/**
 * @author Rich Omar
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class ParserGUI {

    private static JPanel jContentPane = null;
    private static JFrame jFrame = null; // @jve:decl-index=0:visual-constraint="9,11"
    private static JTextField jTextField = null;
    private static JComboBox jComboBox = null;
    private static JLabel jLabel = null;
    private static JComboBox jComboBox1 = null;
    private static JComboBox jComboBox2 = null;
    private static JComboBox jComboBox3 = null;
    private static JLabel jLabel1 = null;
    private static JLabel jLabel2 = null;
    private static JLabel jLabel3 = null;
    private static JTextArea jTextArea = null;
    private static JButton jButton = null;
    private static JButton jButton1 = null;
    private static DTDParser x = new DTDParser();
    private static TreeNode t[] = x.getAll();
    private static GUIBinder g = new GUIBinder();

    private static JButton jButton2 = null;
    private static JButton jButton3 = null;
    private static JMenuBar jMenuBar = null;
    private static JMenu jMenu = null;
    private static JMenuItem jMenuItem = null; // @jve:decl-index=0:visual-constraint="677,31"
    private static JMenu jMenu1 = null;
    private static JMenuItem jMenuItem1 = null;
    private static JMenuItem jMenuItem2 = null;
    private static JMenuItem jMenuItem3 = null;
    private static JScrollPane jScrollPane = null;
    private static JMenuItem jMenuItem4 = null;
    private static JMenuItem jMenuItem5 = null;
    private static JMenuItem jMenuItem6 = null;
    private static JMenuItem jMenuItem7 = null;
    private static JMenuItem jMenuItem8 = null;
    private static JMenuItem jMenuItem9 = null;
    /**
     * This method initializes jContentPane
     *
     * @return javax.swing.JPanel
     */
    private static JPanel getJContentPane() {
        if (jContentPane == null) {
            jContentPane = new JPanel();
            jLabel = new JLabel();
            jLabel1 = new JLabel();
            jLabel2 = new JLabel();
            jLabel3 = new JLabel();
            jContentPane.setLayout(null);
            jLabel.setBounds(16, 80, 105, 23);
            jLabel.setText("XPath Commands");
            jLabel1.setBounds(154, 80, 54, 23);
            jLabel1.setText("Tables");
            jLabel2.setBounds(296, 80, 55, 23);
            jLabel2.setText("Keys");
            jLabel3.setBounds(430, 80, 93, 20);
            jLabel3.setText("Foreign Keys");
            jContentPane.add(jTextField(), null);
            jContentPane.add(jComboBox(), null);
            jContentPane.add(jLabel, null);
            jContentPane.add(jButton2(), null);
            jContentPane.add(jButton3(), null);
            jContentPane.add(jComboBox1(), null);
            jContentPane.add(jComboBox2(), null);
            jContentPane.add(jComboBox3(), null);
            jContentPane.add(jLabel1, null);
            jContentPane.add(jLabel2, null);
            jContentPane.add(jLabel3, null);
            jContentPane.add(jButton(), null);
            jContentPane.add(jButton1(), null);
            jContentPane.add(jScrollPane(), null);
        }
        return jContentPane;
    }
    /**
     * This method initializes jFrame
     *
     * @return javax.swing.JFrame
     */
    private static JFrame getJFrame() {
        if (jFrame == null) {
            jFrame = new JFrame();
            jFrame.setTitle("XPath Parser");
            jFrame.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
        }
    }
}

```

```

        JFrame.setResizable(false);
        JFrame.setVisible(true);
        JFrame.setContentPane(getJContentPane());
        JFrame.setMenuBar(getJMenuBar());
        JFrame.setSize(650, 380);
        JFrame.setLocation(256, 256);

    }
    return JFrame;
}
/**
 * This method initializes jTextField
 *
 * @return javax.swing.JTextField
 */
private static JTextField getJTextField() {
    if (jTextField == null) {
        jTextField = new JTextField();
        jTextField.setBounds(25, 30, 600, 20);
        jTextField.setToolTipText("Enter a command here to be executed by the parser. You can
type it in by hand, or use the menus to insert a command.");
    }
    return jTextField;
}
/**
 * This method initializes jComboBox
 *
 * @return javax.swing.JComboBox
 */
private static JComboBox getJComboBox() {
    if (jComboBox == null) {
        jComboBox = new JComboBox();
        jComboBox.setBounds(16, 106, 116, 20);
        jComboBox.setToolTipText("This menu contains XPath commands that can be used above.");
        jComboBox.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                jTextField.setText(jTextField.getText() +
jComboBox.getSelectedItem() + " ");
            }
        });
        jComboBox.addItem("Insert");
        jComboBox.addItem("Update");
        jComboBox.addItem("Delete");
    }
    return jComboBox;
}
/**
 * This method initializes jComboBox1
 *
 * @return javax.swing.JComboBox
 */
private static JComboBox getJComboBox1() {
    if (jComboBox1 == null) {
        jComboBox1 = new JComboBox();
        jComboBox1.setBounds(154, 107, 116, 20);
        jComboBox1.setToolTipText("This menu contains the names of all the tables of the loaded
database.");
        jComboBox1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                jTextField.setText(jTextField.getText() +
jComboBox1.getSelectedItem() + " ");
            }
        });
        jComboBox1.addItem("root");
        for (int i = 0; i < x.getNumNodes(); i++)
        {
            jComboBox1.addItem(t[i].getName());
        }
    }
    return jComboBox1;
}
/**
 * This method initializes jComboBox2
 *
 * @return javax.swing.JComboBox
 */
private static JComboBox getJComboBox2() {
    if (jComboBox2 == null) {
        jComboBox2 = new JComboBox();
        jComboBox2.setBounds(296, 107, 116, 20);
        jComboBox2.setToolTipText("This menu contains all the keys for the loaded database.");
        jComboBox2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                jTextField.setText(jTextField.getText() +
jComboBox2.getSelectedItem() + " ");
            }
        });
        for(int i = 0; i < x.getNumNodes(); i++)
        {
            for (int j = 0; j < t[i].getNumKeys(); j++)
            {
                jComboBox2.addItem(t[i].getKey(j));
            }
        }
    }
}

```

```

        }
        return jComboBox2;
    }
    /**
     * This method initializes jComboBox3
     *
     * @return javax.swing.JComboBox
     */
    private static JComboBox getJComboBox3() {
        if (jComboBox3 == null) {
            jComboBox3 = new JComboBox();
            jComboBox3.setBounds(430, 107, 116, 20);
            jComboBox3.setToolTipText("This menu contains all the foreign keys for the loaded
database.");
            jComboBox3.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    jTextField.setText(jTextField.getText() +
jComboBox3.getSelectedItem() + " ");
                }
            });
            for(int i = 0; i < x.getNumNodes(); i++)
            {
                for (int j = 0; j < t[i].getNumFKeys(); j++)
                {
                    jComboBox3.addItem(t[i].getFKey(j));
                }
            }
            return jComboBox3;
        }
    }
    /**
     * This method initializes jTextArea
     *
     * @return javax.swing.JTextArea
     */
    private static JTextArea getJTextArea() {
        if (jTextArea == null) {
            jTextArea = new JTextArea();
            jTextArea.setBackground(java.awt.Color.black);
            jTextArea.setForeground(java.awt.Color.white);
            jTextArea.setToolTipText("This is where your interactions with the database are
displayed.");
            return jTextArea;
        }
    }
    /**
     * This method initializes jButton
     *
     * @return javax.swing.JButton
     */
    private static JButton getJButton() {
        if (jButton == null) {
            jButton = new JButton();
            jButton.setBounds(25, 176, 110, 22);
            jButton.setText("Execute Xpath");
            jButton.setToolTipText("Use this button to execute an XPath command from the field above.
");
            jButton.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    jTextArea.setText(g.sendXPath(jTextField.getText()));
                }
            });
            return jButton;
        }
    }
    /**
     * This method initializes jButton1
     *
     * @return javax.swing.JButton
     */
    private static JButton getJButton1() {
        if (jButton1 == null) {
            jButton1 = new JButton();
            jButton1.setBounds(25, 266, 110, 22);
            jButton1.setText("Clear");
            jButton1.setToolTipText("Clears the contents of all text areas.");
            jButton1.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    jTextField.setText("");
                    jTextArea.setText("");
                }
            });
            return jButton1;
        }
    }
    /**
     * This method initializes jButton2
     *
     * @return javax.swing.JButton
     */
    private static JButton getJButton2() {

```

```

        if (jButton2 == null) {
            jButton2 = new JButton();
            jButton2.setBounds(25, 206, 110, 22);
            jButton2.setText("Execute SQL");
            jButton2.setToolTipText("Use this button to execute a SQL insert, update or delete from
the field above.");
            jButton2.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    JTextArea.setText(g.sendSQL(jTextField.getText()));
                }
            });
        }
        return jButton2;
    }
    /**
     * This method initializes jButton3
     *
     * @return javax.swing.JButton
     */
    private static JButton getJButton3() {
        if (jButton3 == null) {
            jButton3 = new JButton();
            jButton3.setBounds(25, 236, 110, 22);
            jButton3.setText("SQL Query");
            jButton3.setToolTipText("Use this to query the database and display the results.");
            jButton3.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    JTextArea.setText(g.sendQuery(jTextField.getText()));
                }
            });
        }
        return jButton3;
    }
    /**
     * This method initializes jJMenuBar
     *
     * @return javax.swing.JMenuBar
     */
    private static JMenuBar getJJMenuBar() {
        if (jJMenuBar == null) {
            jJMenuBar = new JMenuBar();
            jJMenuBar.add(getJMenu());
            jJMenuBar.add(getJMenu1());
        }
        return jJMenuBar;
    }
    /**
     * This method initializes jMenu
     *
     * @return javax.swing.JMenu
     */
    private static JMenu getJMenu() {
        if (jMenu == null) {
            jMenu = new JMenu();
            jMenu.setText("Useful Queries");
            jMenu.setToolTipText("Some useful queries that will display the key numbers");
            jMenu.add(getJMenuItem());
            jMenu.add(getJMenuItem3());
            jMenu.add(getJMenuItem4());
            jMenu.add(getJMenuItem5());
            jMenu.add(getJMenuItem6());
            jMenu.add(getJMenuItem7());
            jMenu.add(getJMenuItem8());
            jMenu.add(getJMenuItem9());
        }
        return jMenu;
    }
    /**
     * This method initializes jMenuItem
     *
     * @return javax.swing.JMenuItem
     */
    private static JMenuItem getJMenuItem() {
        if (jMenuItem == null) {
            jMenuItem = new JMenuItem();
            jMenuItem.setText("select * from part");
            jMenuItem.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    JTextArea.setText(g.sendQuery("select * from part"));
                }
            });
        }
        return jMenuItem;
    }
    /**
     * This method initializes jMenu1
     *
     * @return javax.swing.JMenu
     */
    private static JMenu getJMenu1() {
        if (jMenu1 == null) {
            jMenu1 = new JMenu();
            jMenu1.setText("Database Options");
            jMenu1.setToolTipText("Commit or Rollback database modifications");
        }
    }

```

```

        jMenuItem1.add(getJMenuItem1());
        jMenuItem1.add(getJMenuItem2());
    }
    return jMenuItem;
}
/**
 * This method initializes jMenuItem1
 *
 * @return javax.swing.JMenuItem
 */
private static JMenuItem getJMenuItem1() {
    if (jMenuItem1 == null) {
        jMenuItem1 = new JMenuItem();
        jMenuItem1.setText("Commit");
        jMenuItem1.setToolTipText("Commits all changes to the database since program start.");
        jMenuItem1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                g.commit();
            }
        });
    }
    return jMenuItem1;
}
/**
 * This method initializes jMenuItem2
 *
 * @return javax.swing.JMenuItem
 */
private static JMenuItem getJMenuItem2() {
    if (jMenuItem2 == null) {
        jMenuItem2 = new JMenuItem();
        jMenuItem2.setText("Rollback");
        jMenuItem2.setToolTipText("Rolls back all changed made to the database since program
start.");
        jMenuItem2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                g.rollback();
            }
        });
    }
    return jMenuItem2;
}
/**
 * This method initializes jMenuItem3
 *
 * @return javax.swing.JMenuItem
 */
private static JMenuItem getJMenuItem3() {
    if (jMenuItem3 == null) {
        jMenuItem3 = new JMenuItem();
        jMenuItem3.setText("select * from nation");
        jMenuItem3.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                jTextArea.setText(g.sendQuery("select * from nation"));
            }
        });
    }
    return jMenuItem3;
}
/**
 * This method initializes jScrollPane
 *
 * @return javax.swing.JScrollPane
 */
private static JScrollPane getJScrollPane() {
    if (jScrollPane == null) {
        jScrollPane = new JScrollPane();
        jScrollPane.setBounds(149, 142, 479, 166);
        jScrollPane.setViewportView(getJTextArea());
    }
    return jScrollPane;
}
/**
 * This method initializes jMenuItem4
 *
 * @return javax.swing.JMenuItem
 */
private static JMenuItem getJMenuItem4() {
    if (jMenuItem4 == null) {
        jMenuItem4 = new JMenuItem();
        jMenuItem4.setText("select * from partsupp");
        jMenuItem4.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                jTextArea.setText(g.sendQuery("select * from partsupp"));
            }
        });
    }
    return jMenuItem4;
}
/**
 * This method initializes jMenuItem5
 *
 * @return javax.swing.JMenuItem
 */
private static JMenuItem getJMenuItem5() {

```

```

        if (jMenuItem5 == null) {
            jMenuItem5 = new JMenuItem();
            jMenuItem5.setText("select * from region");
            jMenuItem5.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    jTextArea.setText(g.sendQuery("select * from region"));
                }
            });
        }
        return jMenuItem5;
    }
    /**
     * This method initializes jMenuItem6
     *
     * @return javax.swing.JMenuItem
     */
    private static JMenuItem getJMenuItem6() {
        if (jMenuItem6 == null) {
            jMenuItem6 = new JMenuItem();
            jMenuItem6.setText("select * from customer");
            jMenuItem6.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    jTextArea.setText(g.sendQuery("select * from customer"));
                }
            });
        }
        return jMenuItem6;
    }
    /**
     * This method initializes jMenuItem7
     *
     * @return javax.swing.JMenuItem
     */
    private static JMenuItem getJMenuItem7() {
        if (jMenuItem7 == null) {
            jMenuItem7 = new JMenuItem();
            jMenuItem7.setText("select * from orders");
            jMenuItem7.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    jTextArea.setText(g.sendQuery("select * from orders"));
                }
            });
        }
        return jMenuItem7;
    }
    /**
     * This method initializes jMenuItem8
     *
     * @return javax.swing.JMenuItem
     */
    private static JMenuItem getJMenuItem8() {
        if (jMenuItem8 == null) {
            jMenuItem8 = new JMenuItem();
            jMenuItem8.setText("select * from supplier");
            jMenuItem8.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    jTextArea.setText(g.sendQuery("select * from supplier"));
                }
            });
        }
        return jMenuItem8;
    }
    /**
     * This method initializes jMenuItem9
     *
     * @return javax.swing.JMenuItem
     */
    private static JMenuItem getJMenuItem9() {
        if (jMenuItem9 == null) {
            jMenuItem9 = new JMenuItem();
            jMenuItem9.setText("select * from lineitem");
            jMenuItem9.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    jTextArea.setText(g.sendQuery("select * from lineitem"));
                }
            });
        }
        return jMenuItem9;
    }
}

    public static void main(String[] args) {
        String lookAndFeel = null;

        lookAndFeel = UIManager.getSystemLookAndFeelClassName();

        try {
            UIManager.setLookAndFeel(lookAndFeel);
        } catch (Exception e) {}

        init();
        jTextField.setText("");
    }

    public static void init(){
        getJFrame();
    }

```

Bibliography

- [1] I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. Updating XML.
<http://citeseer.ist.psu.edu/cache/papers/cs/25650/http:zSzzSzwww.cs.washington.eduzSz homeszSzalonzSzsitezSzfileszSzsizmod01-update.pdf/tatarinov01updating.pdf> . 2001
- [2] D. Lee, M. Mani, F. Chiu, W. W. Chu. NeT & CoT: Translating relational schemas to XML schemas using semantic constraints.
<http://delivery.acm.org/10.1145/590000/584840/p282-lee.pdf?key1=584840&key2=6715967901&coll=GUIDE&dl=GUIDE&CFID=26547628&CFTOKEN=45928531>. 2002
- [3] V. P. Braganholo, S. B. Davidson, C. A. Heuser. Propagating XML View Updates to a Relational Database.
<http://citeseer.ist.psu.edu/cache/papers/cs/31785/http:zSzzSzwww.inf.ufrgs.brzSz~vanessazSzartigoszSztrvldb.pdf/braganholo04propagating.pdf>. February 2004

[4] TPC-H Benchmark. <http://www.tpc.org/tpch/>. 2001 – 2004

[5] Eclipse Foundation. <http://www.eclipse.org>.

[6] Sun Microsystems – Java. <http://java.sun.com>.